Computer Science Logic 2012

26th International Workshop
21th Annual Conference of the EACSL
CSL'12, September 3–6, 2012, Fontainebleau, France

Edited by

Patrick Cégielski
Arnaud Durand

LIPICS

Editors

Patrick Cégielski
LACL EA 4219
IUT Sénart-Fontainebleau
Université Paris Est Créteil
cegielski@u-pec.fr

Arnaud Durand
Institut de Mathématiques de Jussieu, CNRS UMR 7586
UFR de mathématiques
Université Paris Diderot
durand@math.univ-paris-diderot.fr

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 978-3-939897-42-2

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/978-3-939897-42-2.

Publication date
September, 2012

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.2012.i

ISBN 978-3-939897-42-2 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

www.dagstuhl.de/lipics

Contents

Report on the Ackermann Award 2012

Abstracts of Invited Talks

Contributed Papers

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Editors's Preface

The annual conference of the European Association for Computer Science Logic (EACSL), CSL 2012, was held in Fontainebleau, France, from 3 to 6 September 2012. CSL started as a series of international workshops on Computer Science Logic, and became at its sixth meeting the Annual Conference of the EACSL. This conference was the 26th workshop and 21th EACSL conference; it was organized by the Department of Computer Science of the Institut Universitaire de Technologie, IUT, of the university Paris Est Créteil (UPEC).

The Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science. This year, the jury decided to give the Ackermann Award for 2012 to two recipients : Andrew Polonsky and Szymon Toruńczyk. The awards were officially presented at the conference (September, 4). The citation of the awards, an abstract of the thesis, and a biographical sketch of the recipients written by Thierry Coquand, Anuj Dawar and Damian Niwiński may be found in the proceedings.

This is the second year that the CSL proceedings are not published as a Springer LNCS but in the LIPIcs series. A total of 102 abstracts were registered and 80 of these were followed by full papers submitted to CSL 2012. After a two weeks electronic meeting, the Program Committee (PC) selected 35 papers for presentation at the conference and publication in these proceedings. Each paper was assigned to at least three PC members. In the call for papers, authors were encouraged to include a well written introduction accessible to a general audience in computer science logic. The program committee has paid a careful attention to assess the merits of the submissions regarding this criterion too.

The overall quality of the submissions was really high. The program committee did not fix a strict a priori limit on the number of accepted papers and wished to accept as many good papers as possible. However, at the end some of them had to be rejected due to lack of space. In addition to the contributed talks, CSL 2012 had four invited speakers: Serge Abiteboul (Collège de France, INRIA Saclay and ENS Cachan) Stephen A. Cook (University of Toronto), Inês Lynce (Technical University of Lisbon), Johann A. Makowsky (Technion, Haifa). Abstracts of the invited talks are included in the proceedings.

We wish to warmly thank the PC and all external reviewers for their precious help in reviewing the papers. Our thanks also go to the members of the Organizing Committee, in particular Pierre Valarcher (co-chair), for their considerable efforts in organizing the conference and to Tristan Crolard for his great work in preparing the proceedings. The conference received support from the Université Paris-Est Créteil, the IUT Sénart-Fontainebleau, the LACL (Laboratoire d'Algorithmique, Complexité et Logique, EA 4219), and INRIA. We thank these organizations for their generous supports.

September 2012 Patrick Cégielski, Arnaud Durand

Conference Organization

Program Committee

Jeremy Avigad	Carnegie Mellon, Pittsburgh, USA
Arnold Beckmann	Swansea, UK
Nikolaj Bjorner	Microsoft Research, Redmond, USA
Julian Bradfield	Edimburgh, UK
Thomas Brihaye	Mons, Belgium
Patrick Cégielski	UPEC, France (co-chair)
Victor Dalmau	Pompeu Fabra, Barcelona, Spain
Josee Desharnais	Laval, Canada
Mariangiola Dezani-Ciancaglini	Torino, Italy
Gilles Dowek	INRIA, Paris-Rocquencourt, France
Arnaud Durand	Paris-Diderot, Paris, France (co-chair)
Nicola Galesi	Roma, Italy
Laura Kovacs	TU Vienna, Austria
Antonin Kucera	Brno, Czech Republic
Viktor Kuncak	EPFL, Lausanne, Switzerland
Daniel Leivant	Indiana, USA
Markus Lohrey	Leipzig, Germany
Alexandre Miquel	ENS Lyon, France
Filip Murlak	Warsaw, Poland
Prakash Panangaden	McGill, Montreal, Canada
Nicole Schweikardt	Frankfurt, Germany
Sam Staton	Cambridge, UK
Mirek Truszczynski	Kentucky, USA
Helmut Veith	TU Vienna
Frank Wolter	Liverpool, UK

Organizing Committee

Alexis Bès
Régis Brouard
Patrick Cégielski (co-chair)
Julien Cervelle
Tristan Crolard
Patricia Crouan-Véron
Lilian Dubau
Arnaud Durand
Frédéric Gervais
Yoann Marquer
Denis Monnerat
Pierre Valarcher (co-chair)
Konstantin Verchinine

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Alessi, Fabio
Aschieri, Federico
Atkey, Robert
Atserias, Albert
Baaz, Matthias
Barany, Vince
Barnat, Jiri
Ben-Amram, Amir
Bertrand, Nathalie
Beyersdorff, Olaf
Bollig, Benedikt
Brazdil, Tomas
Cave, Andrew
Chatterjee, Krishnendu
Coppo, Mario
Crolard, Tristan
Dal Lago, Ugo
Danielsson, Nils Anders
Dragoi, Cezara
Falke, Stephan
Fearnley, John
Ferreira, Francisco
Figueira, Diego
Gabbay, Murdoch
Gaboardi, Marco
Garner, Richard
Gauld, David
Ghelli, Giorgio
Gore, Rajeev
Goubault-Larrecq, Jean
Gutierrez, Julian
Harwath, Frederik
Hetzl, Stefan
Hirschowitz, Tom
Hofmann, Martin
Hofstra, Pieter
Holzer, Andreas
Hoshino, Naohiko
Huth, Michael
Hyvernat, Pierre
Iemhoff, Rosalie
Ilik, Danko
Iosif, Radu
Kara, Ahmet
Keller, Chantal
Klima, Ondrej
Krebs, Andreas
Kretinsky, Jan
Krishnaswami, Neelakantan
Kuske, Dietrich
Lamarche, Francois
Lange, Martin
Lauria, Massimo
Libkin, Leonid
Lisitsa, Alexei
Liu, Jiamou
Longley, John
Lumsdaine, Peter
Macedonio, Damiano
Marek, Victor
Martini, Simone
Meinecke, Ingmar
Murawski, Andrzej
Müller, Sebastian
Nguyen, Phuong
Obdrzalek, Jan
Oliva, Paulo
Oliveras, Albert
Padovani, Luca
Pambuccian, Victor
Paperman, Charles
Pechoux, Romain
Pfenning, Frank
Piccolo, Mauro
Piskac, Ruzica
Polonsky, Andrew
Pratt-Hartmann, Ian
Riba, Colin
Ritter, Eike
Roux, Cody
Rubin, Sasha
Schmidt-Schauss, Manfred
Schoepp, Ulrich
Schwoon, Stefan
Simonsen, Jakob Grue
Strejcek, Jan
Terui, Kazushige
Tzameret, Iddo
Tzevelekos, Nikos
van Raamsdonk, Femke

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Vollmer, Heribert
Waldmann, Uwe
Wooldridge, Mike
Zeilberger, Noam
Zeume, Thomas
Zhang, Lijun
Zuleger, Florian

The Ackermann Award 2012

Thierry Coquand[1], Anuj Dawar[2], and Damian Niwiński[3]

1,2,3 Members of EACSL Jury of the Ackermann Award*

The eighth Ackermann Award is presented at this CSL'12, held in Fontainebleau, France. This is the sixth year in which the EACSL Ackermann Award is generously sponsored. Our sponsor for the period 2011–2013 is the Kurt Gödel Society. Besides providing financial support for the Ackermann Award, the KGS also committed itself to inviting recipients of the Award for a special lecture to be given in Vienna.

Eligible for the 2012 Ackermann Award were PhD dissertations in topics specified by the EACSL and LICS conferences, which were formally accepted as PhD theses at a university or equivalent institution between 1.1. 2010 and 31.12. 2011. The Jury received 7 nominations for the Ackermann Award 2012. The candidates came from 7 different nationalities from Europe, Australia, and South America, and received their PhDs in 6 different countries in Europe (2 in UK). Only two candidates received the PhD in the country of their origin.

All the submissions were of very high standard and contained remarkable 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 finally, to give for the year 2012 two awards, one for work in *lambda calculus*, and one for work in *automata theory*. The **2012 Ackermann Award** winners are, in alphabetical order

- Andrew Polonsky from Ukraine, for his thesis
  Proofs, Types and Lambda Calculus
  issued by University of Bergen, Norway, in 2011,
  supervised by Marc Bezem,
- Szymon Toruńczyk from Poland, for his thesis
  Languages of profinite words and the limitedeness problem
  issued by University of Warsaw, Poland, in 2011,
  supervised by Mikołaj Bojańczyk.

Andrew Polonsky

Citation. Andrew Polonsky receives the *2012 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Proofs, Types and Lambda Calculus.

His thesis brings a number of valuable results in λ-calculus. In particular, it solves in a negative way the range property problem for the theory $\mathcal{H}$, stated by Barendregt in 1976.

Background of the Thesis. The λ-calculus, introduced by Church in the 1930s, is the first mathematical model of effectively computable functions and it was for this model that Church's Thesis was originally formulated. This is surprising if one considers the fact that λ-terms were first introduced to represent logical formulae, and this tight connection between logic and computations gives to λ-calculus a special status in computer science.

* We would like to thank H. Barendregt, J.-E. Pin, and R. Statman for their help in preparing the citations.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 1–5
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One important issue has been when two λ-terms should be identified, since this corresponds to the issue of when two algorithms can be considered to be the same. This question has given rise to a rich lattice of lambda theories. The first notion of equality is β-conversion, for which the confluence result known as the Church-Rosser property was first formulated. Church himself believed that meaningless computations in the λ-calculus are represented by terms without normal forms, and so that all these terms should be identified. For the type-free λ-calculus however this approach has a number of disadvantages and it became clear in the 1970s, through the work of Scott, Wadsworth, Barendregt and others, that a more appropriate notion to capture non-terminating computations was that of terms without a head normal form, known as *unsolvable terms.* Barendregt defined the theory $\mathcal{H}$ as the least equational theory extending β-conversion and equating all unsolvable terms. This theory $\mathcal{H}$ has a number of interesting properties and occupies an important place in the lattice of different ways of identifying λ-terms.

To deeper analyze these various theories, Barendregt formulated the *range property.* This is a kind of zero-one law: it states that for any closed λ-term F the collection of all closed applications $F\ M$ has, modulo the given theory, either exactly one or infinitely many elements. This expresses the fact that a given term cannot classify the λ-terms into finitely many sets with given properties. The first instance of this property, for β-conversion, was conjectured in a classical paper of Böhm (1968). This conjecture was proved later by Barendregt and Myhill, as a consequence of Scott's Theorem that a non trivial set of λ-terms closed under β-conversion cannot be recursive. It was then realized that most lambda theories ($\lambda\beta\eta$, the theory of models D_∞,the Böhm-tree model, and many others) satisfy the range property. In 1976, Barendregt conjectured the range property for the theory $\mathcal{H}$. In his classic monograph *The lambda calculus, its syntax and semantics*, North-Holland 1984, Barendregt collected several arguments in favour of this conjecture. The main one is that a putative counter-example would have non-computable properties, and hence cannot be represented by a λ-term. Since then, several strong researchers have been trying to settle the range property for $\mathcal{H}$. Indeed Statman and Intrigila were recently able to show that the corresponding conjecture holds for combinatory logic.

Polonsky's Thesis. The main result of Polonsky's thesis is a surprising negative solution to the range property of the lambda theory $\mathcal{H}$. This failure of the range property indicates a subtle unexpected difference between the theory $\mathcal{H}$ and most other lambda theories. For this, Polonksy defines a lambda term, having modulo $\mathcal{H}$ a range of cardinality 2. This uses previous works by Statman and Barendregt who outlined what kind of construction was needed in order to get such a counterexample. The required behaviour of such lambda term was however so complex that this was then used as a positive argument in favour of the conjecture. This construction is carried out in Polonsky's thesis as the "devil's tunnel". The exact details are quite ingenious and the devil's tunnel construction produces a recursively enumerable Böhm tree with a unique infinite path which is not recursively enumerable. This path is constructed by a Putnam-Gold-like trial and error procedure. All this is very elegantly presented in the thesis. The solution of the range problem for $\mathcal{H}$ can be considered to be one of the strong achievements in the field of λ-calculus. It is in some way reminiscent of Plotkin (1974) fundamental and unexpected discovery of the failure of ω-completeness of λ-calculus (exhibiting two closed terms F and Z such that $F\ M =_\beta Z$ for any closed term M and $F\ x \neq_\beta Z$).

Polonsky's thesis contains several other significant results. Smullyan (1994) had given an interesting axiomatization of enumerators, but left three open problems having to do with the independence of the given axiomatic system. Polonsky solved these by characterizing when an enumerator is "standard". He did this again by using results of computability theory, making clever use of non-computable functions. Another elegant result of Polonsky characterizes when an untyped lambda term can be typed in a type algebra. This contribution is already incorporated in the recent book *Lambda calculus with types* by Barendregt, Cambridge University Press (to appear in 2013). The first part of the thesis contains an interesting analysis of the connection between coherent logic and general first-order logic.

The thesis is of high level of original creativity. Not only are strong results presented, but Polonsky also shows creativity stating open problems, which undoubtedly are fruitful for future research.

Biographical Sketch. Andrew Polonsky was born on 9 June 1984 in Ukraine. In 2002-2005 he studied Mathematics in the University of Storss, Connecticut, U.S. In 2006-2007 he attend a Master Class in Logic at the Mathematical Research Institute of Utrecht, The Netherlands. Between 2007 and 2010 he was a PhD student at the University of Bergen, Norway, under the supervision of Marc Bezem. In 2011-2012 he has been a postdoc at the University of Amsterdam, The Netherlands, and is now a postdoc at the Radboud University Nijmegen, The Netherlands.

Szymon Toruńczyk

Citation. Szymon Toruńczyk receives the *2012 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Languages of profinite words and the limitedeness problem.

His thesis builds a new framework to study limitedeness problems of finite automata, based on the topology of profinite words.

Background of the Thesis. Research currents in computer science are often driven by decidability questions. Even when a problem is solved, it often stimulates theoretical work explaining the solution. Toruńczyk's thesis is related to two decidability questions: the limitedeness problem for distance automata originally solved by Hashigushi in 1982, and the still unsolved decidability status of the logic MSO + B. The actual motivation arose from a connection between the two problems.

In 2004, Mikołaj Bojańczyk raised the question whether the decidability of some instances of the finite model property can be strengthened to decidability of a suitable logic. He proposed an extension of the monadic second order logic by a quantifier $BX\,\varphi(X)$, expressing the existence of a finite bound on the cardinality of sets X satisfying φ. The decidability status of the new logic remains unknown. In particular, the MSO + B theory of $\langle \omega, \leq \rangle$ is a plausible extension of Büchi's arithmetic S1S, but its decidability remains open. Approaching the problem, Bojańczyk and Thomas Colcombet captured a fragment of this theory by automata with blind counters, which can be incremented and reset, but not read. The technical core of their proof is a duality result claiming that two variants of the model, called ωB and ωS-automata respectively, actually complemented each other. It was only afterwards that the authors realized that some of their ideas occurred previously in research on the *star height* problem.

This celebrated automata-theoretic decision problem asks for the minimal nesting of the star operator needed to define a given regular language. The problem, raised by Lawrence C. Eggan in 1963, was solved positively by Kosaburo Hashiguchi only 25 years later. As a step toward the main solution, Hashiguchi solved in 1982 a related decision problem of limitedeness of distance automata. In the original setting, the problem concerns finite (non-deterministic) automata with non-negative costs over transitions, and asks if any acceptable word can be accepted with a cumulative cost within some finite bound. This can be generalized to other concepts of cost. The difficult techniques of Hashiguchi were further studied and developed in particular by Imre Simon and Hing Leung. An insightful new solution was given in 2005 by Daniel Kirsten, who reduced the star height question to the limitedeness problem of what he called distance desert automata. A surprising discovery made by Thomas Colcombet revealed that the ωB and ωS-automata introduced in the context of the logic MSO + B, were in fact the infinite computation variants of Kirsten's automata, and the aforementioned duality theorem yielded yet another path to the solution of the star height problem. Colcombet pursued this connection by cleaning away infinite computations, and concentrating on B and S-automata on finite words. He further developed a theory of cost functions, where the limitedeness problem appears as a special case of an equivalence between two functions. This theory constitutes a powerful quantitative generalization of the theory of regular languages, involving automata, algebra, and logic.

Toruńczyk's Thesis. Compared to the previous work, the thesis of Szymon Toruńczyk reveals a metric aspect of the limitedeness problem. It was first noticed by Hing Leung who in his alternative proof of Hashiguchi's result (from 1988) used a concept of limit, but it was not pursued very far. The reason perhaps is that a topological structure of finite words is not as apparent as it is in the case of ω-words, which can be just identified with reals. A topological structure of finite words is revealed however by their metric completion to *profinite* words, which moreover preserves the algebraic structure of semigroup. This is an alternative extension of the free monoid of finite words, just as the p-adic numbers form an extension of the rational field, which is alternative to reals. In his thesis, throughout a long series of results, Toruńczyk argues that the topological semigroup of profinite words constitutes a suitable framework to consider the limitedeness problem, cost functions, as well as the quantifier B.

The first main result is a self-contained proof of the decidability of the limitedeness problem for the (most general) case of B-automata, based on topological ideas. The results by Kirsten and Colcombet follow there from a more general, new result, on the structure of a certain profinite semigroup generalizing the tropical semiring considered by Simon and Leung.

The second main result consists of a multiple characterization of cost functions computable by B-automata (or S-automata) in terms of the associated sets of profinite words. It is a basic fact of the theory of profinite words that the completions of the ordinary regular languages form the family of *clopen* (closed and open) sets of profinite words. Toruńczyk makes the next step by showing that the profinite languages associated with B-automata (or S-automata) correspond to the *open* (resp. *closed*) sets with finite syntactic stabilization monoids. This yields a machine independent characterization of the cost functions of Colcombet. Another characterization is given in terms of logic, which is a profinite counterpart of the logic MSO + B. The decidability of the satisfiability problem for this logic remains open, but the author manages to show that satisfiability of propositional combinations of the formulas corresponding to cost functions is decidable.

Biographical Sketch. Szymon Toruńczyk was born in 1983. He obtained all his degrees from the University of Warsaw: B.S. degree in computer science in 2006, and M.Sc. degree in mathematics in 2006. He wrote his doctoral dissertation under the supervision of Mikołaj Bojańczyk and obtained the Ph.D. degree in computer science (with honors) in 2011. Throughout 2009-2011 he visited ENS Cachan, working under supervision of Luc Segoufin. He is currently an assistant professor at the Warsaw University.

Jury

The Jury for the **Ackermann Award 2012** 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

- Thierry Coquand (University of Gothenburg),
- Anuj Dawar (University of Cambridge), the vice-president of EACSL,
- Thomas A. Henzinger (IST Austria),
- Jean-Pierre Jouannaud (École Polytechnique, Paris),
- Daniel Leivant (Indiana University, Bloomington),
- Damian Niwiński (University of Warsaw), the president of EACSL,
- Luke Ong (University of Oxford), LICS representative,
- Wolfgang Thomas (RWTH, Aachen).

Previous winners

Previous winners of the Ackermann Award were

2005, Oxford
Mikołaj Bojańczyk,
Konstantin Korovin,
Nathan Segerlind,

2006, Szeged
Balder ten Cate,
Stefan Milius,

2007, Lausanne
Dietmar Berwanger,
Stéphane Lengrand,
Ting Zhang,

2008, Bertinoro
Krishnendu Chatterjee,

2009, Coimbra
Jakob Nordström,

2010, Brno
no award given,

2011, Bergen
Benjamin Rossman.

Detailed reports on their work appeared in the CSL proceedings, and are also available *via* the EACSL homepage.

Sharing Distributed Knowledge on the Web

Serge Abiteboul

Collège de France, INRIA Saclay & ENS Cachan, France

Abstract

To share information, we propose to see the Web as a knowledge base consisting of distributed logical facts and rules. Our objective is to help Web users finding information, as well as controlling their own, using automated reasoning over this knowledge base towards improving the quality of service and of data. For this, we introduce Webdamlog, a Datalog-style language with rule delegation. We mention the implementation of a system to support this language as well as standard communications and security protocols.

1998 ACM Subject Classification H.2 Database

Keywords and phrases Knowledge base, distributed data, world wide web, deduction.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.6

Category Invited Talk

1 Overview

Information of interest may be found on the Web in a variety of forms, in many systems, and with different access protocols. Today, the control and management of the diversity of data and tasks in this setting are beyond the skills of casual users. Facing similar issues, companies see the cost of managing and integrating information skyrocketing. We are interested here in the management of Web data. Our focus is not on harvesting all the data of a particular user or a particular application domain and then managing it in a centralized manner. Instead, we are concerned with the management of Web data *in place* in a distributed manner, with a possibly large number of autonomous, heterogeneous systems collaborating to support certain tasks. We describe ongoing works concerned with the foundations of such data management based on *declarative distributed data management.*

Centralized data management has matured with relational database systems, enabled by the combination and cooperation of a very active research community and a very successful industry. The success of the field rests on solid formal foundations that combine existing tools, e.g., first-order logic for specifying queries and dependencies, with others that were developed from scratch, e.g., query optimization or concurrency control. As a result, centralized data management systems are now very reliable and the corresponding science is well-developed.

Now consider a user of the Web today. Such a user typically needs to manage the following kinds of information: data (documents, photos, music, etc.), metadata (personal ontologies, other's, etc.), data localization (e.g., where friends place their photos), access rights (e.g., list of friends who have access to private photos), credentials (login, passwords, etc.), temporal and provenance information, other kinds of knowledge (replication policy, trust in others, beliefs, etc.). The information resides on many devices (smartphone, laptop, TV box, etc.), many systems (mailers, blogs, Web sites, etc.), many social networks (Facebook, Picasa, etc.) as well as in the "data rings" [5] of family members, friends, associations, companies, and health, tax or other organizations.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 6–8
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Clearly, relational databases are not providing adequate support for managing the diversity of such information. First, the exchange standard for the Web is based on data trees (HTML, XML, JSON) and not on relations. Then, the information is by nature distributed between huge number of autonomous systems, unlike in relational systems where it is centralized or distributed within a few tightly controlled machines. Finally, a critical dimension of the problem is the imprecise, uncertain, noisy, possibly contradicting nature of data in this setting.

Our thesis is that managing the richness and diversity of data residing on the Web can be tamed using a holistic approach based on a *distributed knowledge base*. Our approach is to represent all Web information as *logical facts*, and Web data management tasks as *logical rules*. A variety of complex data management tasks that currently require intense work and deep expertise may then greatly benefit from the automatic reasoning provided by inference engines, operating over the distributed Web knowledge base.

The kinds of reasoning tasks we are envisioning, and that are to be captured by *rules*, therefore include:

- Accessing information. Knowledge is used to localize data, e.g., find which systems hold the information of interest. Also, when a new source of information is discovered, some simple reasoning may be required to understand how it should be used, and how to obtain access rights.
- Peer's policy. Each peer specifies its own policy, which includes choices such as where to store/search for particular information, which data to serve to other peers, and which data to replicate. Such policies in social networks are typically defined based on information such as the composition of user groups ("circles" in Google+ terminology).
- Ontology processing. A particular source may structure its information in a particular manner or even describe it using RDF or RDFS. Reasoning is necessary to query this information. In particular, when accessing different information sources, knowledge is needed to align their concepts and relations.
- Quality management. Reasoning may be needed to assess the truthfulness of some data or to choose between contradicting information. This is related to evaluating the confidence one has in some data, the trust in sources, and, more generally, the beliefs of a particular peer or user.
- Knowledge acquisition and dissemination. These are central issues in this context. Knowledge acquisition, i.e., acquiring new facts and rules and evaluating their quality, should provide principled mechanisms that protect against (i) accepting any kind of information that is published by anyone on the Web and (ii) revising opinions too easily and in an ad-hoc manner (e.g., believing the last person who spoke).

We propose to express the peer's logic in Webdamlog, a datalog-style rule-based language. In WebdamLog, peers exchange facts (for information) and rules (in place of code). The use of declarative rules provides the following advantages:

- Peers may perform automatic reasoning using the available knowledge;
- Because the model is formally defined, it becomes possible to prove (or disprove) desirable properties in the spirit of [1] that uses logic to describe access control protocols;
- Because the model is based on a Datalog-style language, it can benefit from optimization techniques, as in [7, 9];
- Because our model represents provenance [3] and time, we can better control the quality of data; and
- Because the model is general, it can represent wide variety of scenarios and protocoles, which is the reality of today's Web.

Acknowledgment. This work has been partially funded by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013); ERC grant Webdam, agreement 226513. A more detailed presentations of the results of the project may be found on the Webdam Web site at `http://webdam.inria.fr/`.

References

1 Martín Abadi. Logic in Access Control (Tutorial Notes). In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V*, volume 5705, chapter 5, pages 145–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

2 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Emilien Antoine. A rule-based language for Web data management. In *Proceedings of the Symposium on Principles of Database Systems*, 2011.

3 Serge Abiteboul, Alban Galland, and Neoklis Polyzotis. A model for web information management with access control. In *Proceedings of the International Workshop on the Web and Databases*, 2011.

4 Emilien Antoine, Alban Galland, Kristian Lyngbaek, Amélie Marian, and Neoklis Polyzotis. [Demo] Social Networking on top of the WebdamExchange System. In *Proceedings of the International Conference on Data Engineering*, 2011.

5 Serge Abiteboul, Neoklis Polyzotis, The Data Ring: Community Content Sharing, Conference on Innovative Data Systems Research, 2007.

6 Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. Corroborating Information from Disagreeing Views. In *Proceedings of the International Conference on Web Search and Data Mining*, 2010.

7 Joseph M. Hellerstein. Datalog redux: experience and conjecture. In *Proceedings of the Symposium on Principles of Database Systems*, 2010.

8 Serge Abiteboul, T-H. Hubert Chan, Evgeny Kharlamov, Werner Nutt, and Pierre Senellart, Capturing Continuous Data and Answering Aggregate Queries in Probabilistic XML. In *ACM Transactions on Database Systems, vol. 36, n^o 4*, 2011.

9 Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: language, execution and optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 97–108, New York, NY, USA, 2006. ACM.

Connecting Complexity Classes, Weak Formal Theories, and Propositional Proof Systems

Stephen A. Cook

University of Toronto
sacook@cs.toronto.edu

Abstract

This is a survey talk explaining the connection between the three items mentioned in the title.

1998 ACM Subject Classification F.1.3, F.4.1

Keywords and phrases Complexity Classes, Weak Formal Theories, Propositional Proof Systems

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.9

Category Invited Talk

1 Overview

I've been interested in this three-way connection since my 1975 paper [7] in which I introduced the formal theory PV to capture polynomial time reasoning, and showed how each of its theorems can be translated into a polynomial size family of Extended Resolution proofs. The corresponding triple is $(\mathsf{P}, PV, \mathsf{ERes})$. I originally defined PV as an equational theory in which the function symbols range over all polynomial time functions on $\mathbb{N}$. The axioms consist of equations defining the polynomial time functions, based on Cobham's Theorem [6], and a rule giving "induction on notation"; i.e. induction based on binary notation for numbers. (Later Martin Dowd and others pointed out that ordinary induction on $\mathbb{N}$ can be derived from this rule, using binary search.) My idea for PV came from Skolem's 1923 equational theory based on function symbols for all primitive recursive functions.

The Extended Resolution proof system ERes was introduced by Tseitin [16], and is equivalent to Extended Frege systems [10]. ERes can be characterized roughly as the strongest propositional proof system whose soundness can be proved in PV. See [13] for much more on these ideas.

The theory PV can formalize the proofs of many theorems useful in computer science, such as the Pigeonhole Principle, Extended Euclidean Algorithm, Hall's Theorem, Menger's Theorem, and properties of integer (or rational) determinants. Each of these corresponds to a family of tautologies with polynomial size Extended Frege proofs. However it follows from the witnessing theorem for PV that no polynomial time algorithm for prime recognition (such as [1]) can be proved correct in PV unless there is a polynomial time algorithm for integer factorization.

Buss's influential 1986 book *Bounded Arithmetic* [4] introduced a hierarchy of first-order theories S^i_2 corresponding to the polynomial hierarchy. The functions Σ^b_1-definable in the base theory S^1_2 are the polynomial time functions, and Buss proved that $\mathsf{S}^1_2(PV)$ is Σ^b_1-conservative over PV (where now PV is regarded as a first-order theory axiomatized by the theorems of the original equational theory). Later [12] proved that (first-order) PV is properly included in $\mathsf{S}^1_2(PV)$, unless the polynomial hierarchy collapses. In particular $\mathsf{S}^1_2(PV)$ proves that integers can be factored as a product of primes, which is unlikely to be a consequence of PV.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 9–11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are plenty of interesting complexity classes included in P which have been studied extensively, including

$$\mathsf{AC}^0 \subset \mathsf{AC}^0(2) \subset \mathsf{AC}^0(6) \subseteq \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{LogSpace} \subseteq \mathsf{NL} \subseteq \mathsf{P}$$

The famous open question in complexity theory is $\mathsf{P} =?\mathsf{NP}$, but a more embarrassing open question is whether

$$\mathsf{NP} = \mathsf{P} = \mathsf{NL} = \mathsf{LogSpace} = \mathsf{NC}^1 = \mathsf{TC}^0 = \mathsf{AC}^0(6),$$

where as far as we know, the smallest class $\mathsf{AC}^0(6)$ cannot count the number of 1-bits in an input bit string. This is one motivation for studying small complexity classes: we need to separate them from NP before separating P from NP.

In [9] Phuong Nguyen and I develop a uniform way of associating each of these classes C with a theory VC and a suitable propositional proof system, which are connected like $(\mathsf{P}, PV, \mathsf{ERes})$ mentioned above. (The design of the propositional translation is inspired by [15].) The triple connecting NC^1 is especially interesting here since the associated propositional proof system is a 'Frege system'; i.e. a standard Hilbert style proof system for the propositional calculus. (Earlier Arai [3] connected NC^1 to Frege systems in a similar way, using a theory AID which is syntactically very different but logically equivalent to our theory VNC^1 [8].)

The theories in [9] use the two-sorted vocabulary developed by Zambella [17], in which variables of the number sort $x, y, z, \ldots$ range over $\mathbb{N}$, and those of the string sort $X, Y, Z, \ldots$ range over finite subsets of $\mathbb{N}$, interpreted as binary bit strings. The base theory V^0 corresponds to the complexity class AC^0. The two-sorted setting is ideal here, because (using the descriptive complexity characterization $\mathsf{AC}^0 = \mathsf{FO}$ [11]) we have the convenient fact that the bounded two-sorted Σ_0^B formulas represent precisely the AC^0 relations. Part of the interest here is that of 'bounded reverse mathematics' [14], where the goal is to find the smallest complexity class C such that the corresponding theory VC proves a given combinatorial theorem. The standard example here is the Pigeonhole Principle, which can be proved in VTC^0 (and in VNC^1) but not in V^0, and in fact the corresponding tautology family does not have polynomial size proofs in the corresponding propositional proof system Bounded-Depth Frege [2] (but does have polynomial size Frege proofs [5]). There are many other interesting examples, but in most cases the lower bounds remain conjectures.

References

1 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

2 Ajtai. The complexity of the pigeonhole principle. *Combinatorical*, 14(4):417–433, 1994.

3 Toshiyasu Arai. A bounded arithmetic AID for Frege systems. *Annals of Pure and Applied Logic*, 103(1–3):155–199, 2000.

4 Samuel Buss. *Bounded Arithmetic.* Bibliopolis, 1986.

5 Samuel Buss. Polynomial size proofs of the propositional pigeonhole principle. *Journal of Symbolic Logic*, 52:916–927, 1987.

6 A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Congress Logic, Methodology, and Philosophy of Science*, pages 24–30. North Holland, 1965.

7 Stephen Cook. Feasibly constructive proofs and the propositional calculus. *Proceedings of the 7th Annual ACM Symposium on Theory of computing*, pages 83–97, 1975.

8 Stephen Cook and Tsuyoshi Morioka. Quantified Propositional Calculus and a Second-Order Theory for NC^1. *Archive for Mathematical Logic*, 44(6):711–749, 2005.

9 Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity.* Cambridge University Press, 2010. Draft available from URL `http://www.cs.toronto.edu/~sacook`.

10 Stephen Cook and Robert Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44(1):36–50, 1979.

11 Neil Immerman. *Descriptive Complexity.* Springer, 1999.

12 J. Krajíček, P. Pudlák, and G. Takeuti. Bounded arithmetic and the polynomial hierarchy. *Annals of Pure and Applied Logic*, 52:143–153, 1991.

13 Jan Krajíček. *Bounded Arithmetic, Propositional Logic and Computational Complexity.* Cambridge University Press, 1995.

14 Phuong Nguyen. *Bounded Reverse Mathematics.* PhD thesis, University of Toronto, 2008. `http://www.cs.toronto.edu/~pnguyen/`.

15 Jeff B. Paris and Alex J. Wilkie. Counting problems in bounded arithmetic. In *Methods in Mathematical Logic*, number 1130 in Lecture Notes in Mathematics, pages 317–340. Springer, 1985.

16 G. S. Tseitin. On the complexity of derivation in propositional calculus. In A. O. Slisenko (Translated from Russian), editor, *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125. Consultants Bureau, New Yor, London, 1970.

17 D. Zambella. Notes on polynomially bounded arithmetic. *Journal of Symbolic Logic*, 61(3):942–966, 1996.

Satisfiability: where Theory meets Practice*

Inês Lynce

INESC-ID/IST,
Technical University of Lisbon, Portugal
`ines@sat.inesc-id.pt`

Abstract

Propositional Satisfiability (SAT) is a keystone in the history of computer science. SAT was the first problem shown to be NP-complete in 1971 by Stephen Cook [4]. Having passed more than 40 years from then, SAT is now a lively research field where theory and practice have a natural intermixing. In this talk, we overview the use of SAT in practical domains, where SAT is thought in a broad sense, i.e. including SAT extensions such as Maximum Satisfiability (MaxSAT), Pseudo-Boolean Optimization (PBO) and Quantified Boolean Formulas (QBF).

1998 ACM Subject Classification I.2.8 Problem Solving, Control Methods, and Search

Keywords and phrases Propositional Satisfiability, SAT solvers, Applications

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.12

Category Invited Talk

1 Overview

Given a Boolean formula, the SAT problem is to find an assignment to the Boolean variables such that the formula is satisfied or prove that no such assignment exists. Whereas SAT is used for solving decision problems, extensions of SAT are used for solving optimization problems. Interestingly, many implementations of SAT extensions make iterative calls to a SAT solver. This means that SAT research has a direct impact on SAT extensions. Also interesting is the fact that SAT solvers are also used by or have been an inspiration to other paradigms. Well-known examples are Satisfiability Modulo Theories (SMT) [15], Answer Set Programming (ASP) [11], model checking [3] and planning [10].

Currently, SAT is well known not only for its theoretical interest but also for the effectiveness of modern SAT solvers. SAT solvers are reliable and easy to use as the result of more than one decade accessing the status of SAT solvers in international competitions [9]. Modern SAT solvers find their roots in the seminal contributions made in the 60s with the resolution based DP procedure [6] and the backtrack search based DPLL procedure [5]. DPLL has later been enhanced with a few techniques: clause learning [17] to reduce the search space, search restarts [7] to diversify the search and lazy data structures [14] to reduce the cost of propagation, among others.

SAT and SAT extensions are clearly application driven research fields. Advances are most often motivated by real problems requiring a solution. Successful examples are haplotype inference [12] and biological networks [8] in the field of biology, as well as upgrades in software packages [1] in the field of software engineering, among many others. SAT is now used not only by SAT researchers but also by other researchers who use a SAT solver as a black box.

* This work was partially supported by national funds through FCT research project ASPEN (PTDC/EIA-CCO/110921/2009) and INESC-ID multiannual funding from the PIDDAC program funds.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 12–13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SAT solvers are not limited to providing a solution or proving that no such solution exists. Another interesting topic in SAT is to reason over formulas. The motivation is to better understand the *structure* of the formula either to improve the SAT solvers performance or to better perceive the problem being encoded into SAT. For example, we may want to eliminate redundant clauses, i.e. to get a minimal equivalent subformula (MES) [2]. In terms of variables, one may be interested in identifying the backbones [13], i.e. the value assignments that are common to all solutions, and the backdoors, i.e. a set of value assignments such that the simplified formula can be solved by a poly-time algorithm [18]. As for unsatisfiable formulas, it may be relevant to identify a minimal unsatisfiable subformula (MUS) [16].

References

1 Josep Argelich, Inês Lynce, and João P. Marques Silva. On solving boolean multilevel optimization problems. In *IJCAI*, pages 393–398, 2009.

2 Anton Belov, Mikoláš Janota, Inês Lynce, and João Marques-Silva. On computing minimal equivalent subformulas. In *CP*, 2012. To appear.

3 Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *TACAS*, pages 193–207, 1999.

4 Stephen A. Cook. The complexity of theorem-proving procedures. In *ACM Symposium on Theory of Computing*, pages 151–158, 1971.

5 Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

6 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

7 Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *AAAI*, pages 431–437, 1998.

8 João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In *CP*, 2012. To appear.

9 Matti Järvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The international SAT solver competitions. *AI Magazine*, 33(1), 2012.

10 Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

11 Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

12 Inês Lynce and João Marques-Silva. Efficient haplotype inference with boolean satisfiability. In *AAAI*, pages 104–109, 2006.

13 João Marques-Silva, Mikoláš Janota, and Inês Lynce. On computing backbones of propositional theories. In *ECAI*, pages 15–20, 2010.

14 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535, 2001.

15 Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(t). *J. ACM*, 53(6):937–977, 2006.

16 João P. Marques Silva and Inês Lynce. On improving MUS extraction algorithms. In *SAT*, pages 159–173, 2011.

17 João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

18 Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.

Definability and Complexity of Graph Parameters

Johann A. Makowsky

Faculty of Computer Science, Technion–Israel Institute of Technology
Haifa Israel
`{janos}@cs.technion.ac.il`

Abstract

In this talk we survey definability and complexity results of graph parameters which take values in some ring or field $\mathcal{R}$.

1998 ACM Subject Classification F.1.1, F.2.2, F.4.1, G.2.2

Keywords and phrases Model theory, finite model theory, graph invariants

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.14

Category Invited Talk

1 Overview

Partition functions and related graph parameters have found many applications in computer science, combinatorics, physics, chemistry, biology and even the mathematics of finance.

In this talk we survey definability and complexity results of graph parameters which take values in some ring or field $\mathcal{R}$. For this purpose we introduced the classes $\mathrm{SOLEVAL}_{\mathcal{R}}$ and $\mathrm{MSOLEVAL}_{\mathcal{R}}$ of graph parameters with values in a ring or a field $\mathcal{R}$ which are definable in Second Order Logic SOL and Monadic Second Order Logic MSOL respectively, [6, 12, 13, 9, 10].

Partition functions are special cases of such parameters. They all are in $\mathrm{MSOLEVAL}_{\mathcal{R}}$. Many classes of partition functions have been characterized in a series of papers by M. Freedman, L. Lovasz, A. Schrijver and B. Szegedy [5, 15, 1], Their characterizations all use algebraic properties of connection matrices, which are a generalization of Hankel matrices over words. Partition functions can also be viewed as weighted constraint satisfaction problems (CSP) over relational structures. However, characterizing those functions which map labeled relational structures into $\mathcal{R}$ which are representable as CSP problems, seems harder. It was shown in [6] that for all functions in $\mathrm{MSOLEVAL}_{\mathcal{R}}$ and for a wide class of connection matrices the rank of these connection matrices is finite.

A classical result of J.W. Carlyle and A. Paz [3] used Hankel matrices to characterize functions $f : \Sigma^* \to \mathcal{R}$ of words over a finite alphabet Σ recognizable by multiplicity automata (aka weighted automata). We use this to give a complete characterization of $\mathcal{R}$-valued functions over words in terms of their Hankel matrix. We discuss how to extend such characterization to labeled trees, edge-labeled graphs, and, more generally, to relational structures, [14]. This contrasts and complements the approach given in [4], which uses weighted formulas of MSOL rather than functions in $\mathrm{MSOLEVAL}_{\mathcal{R}}$.

Studying the complexity of functions in $\mathrm{SOLEVAL}_{\mathcal{R}}$ and $\mathrm{MSOLEVAL}_{\mathcal{R}}$ poses some problems. To capture the complexity of their combinatorial nature, the Turing model of computation and Valiant's notion of counting complexity classes $\sharp\mathbf{P}$ seem most natural. To capture the algebraic and numeric nature of partition functions as real or complex valued functions, the Blum-Shub-Smale (BSS) model of computation seems more natural.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 14–15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, in BSS there are various analogues of $\sharp\mathbf{P}$ based on discrete counting, but there is no established complexity class suitable for hard to compute graph parameters. As a result most papers use a naive hybrid approach in discussing their complexity or restrict their considerations to sub-fields of $\mathbb{C}$ which can be coded in a way to allow dealing with Turing computability. Polynomial time computability is formulated in BSS but hardness is formulated resorting to $\sharp\mathbf{P}$. Pioneered by F. Jaeger and D.L. Vertigan and D.J.A. Welsh [8], and A. Bulatov and M. Grohe [2], dichotomy theorems for a wide class of partition functions were proven which were all formulated in this hybrid language, see also [7].

In the second part of this talk we discuss a unified natural framework for the study of computability and complexity of partition functions and graph parameters and show how classical results can be cast in this framework, cf. [11].

The emphasis of this talk is conceptual and includes a list of open problems and a discussion further directions of research.

(Partially based on joint work with T. Kotek, N. Labai and E.V. Ravve)

References

1 A.Schrijver. Graph invariants in the spin model. *J. Comb. Theory Series B*, 99:502–511, 2009.

2 A. Bulatov and M. Grohe. The complexity of partition functions. *Theoretical Computer Science*, 348:148–186, 2005.

3 J.W. Carlyle and A. Paz. Realizations by stochastic finite automata. *J. Comp. Syst. Sc.*, 5:26–40, 1971.

4 M. Droste and P. Gastin. Weighted automata and weighted logics. In *ICALP 2005*, pages 513–525, 2005.

5 M. Freedman, László Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphisms of graphs. *Journal of AMS*, 20:37–51, 2007.

6 B. Godlin, T. Kotek, and J.A. Makowsky. Evaluation of graph polynomials. In *34th International Workshop on Graph-Theoretic Concepts in Computer Science, WG08*, volume 5344 of *Lecture Notes in Computer Science*, pages 183–194, 2008.

7 M. Grohe and M. Thurley. Counting homomorphisms and partition functions. In M. Grohe and J.A. Makowsky, editors, *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 243–292. American Mathematical Society, 2011.

8 F. Jaeger, D.L. Vertigan, and D.J.A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Math. Proc. Camb. Phil. Soc.*, 108:35–53, 1990.

9 T. Kotek. *Definability of combinatorial functions.* PhD thesis, Technion - Israel Institute of Technology, Haifa, Israel, March 2012. Submitted.

10 T. Kotek and J.A. Makowsky. Connection matrices and the definability of graph parameters. In *CSL 2012*, this volume, 2012.

11 T. Kotek, J.A. Makowsky, and E.V. Ravve. A computational framework for the study of partition functions and graph polynomials. Preprint, 2012.

12 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.

13 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 207–242. American Mathematical Society, 2011.

14 N. Labai and J.A. Makowsky. Recognizable power series and msol-definable functions of words. Preprint, 2012.

15 B. Szegedy. Edge coloring models and reflection positivity. *Journal of the American Mathematical Society*, 20.4:969–988, 2007.

A Syntactical Approach to Weak ω-Groupoids

Thorsten Altenkirch[1] **and Ondřej Rypáček**[2]

[1] **Functional Programming Laboratory**
School of Computer Science
University of Nottingham, UK
`thorsten.altenkirch@nottingham.ac.uk`

[2] **Department of Computer Science**
University of Sheffield, UK
`ondrej.rypacek@gmail.com`

Abstract

When moving to a Type Theory without proof-irrelevance the notion of a setoid has to be generalized to the notion of a weak ω-groupoid. As a first step in this direction we study the formalisation of weak ω-groupoids in Type Theory. This is motivated by Voevodsky's proposal of univalent type theory which is incompatible with proof-irrelevance and the results by Lumsdaine and Garner/van de Berg showing that the standard eliminator for equality gives rise to a weak ω-groupoid.

1998 ACM Subject Classification F.4.1. Lambda calculus and related systems, F.4.1. Mechanical theorem proving, G.0 Mathematics of Computing – General

Keywords and phrases Type Theory, Category Theory, Higher dimensional structures

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.16

1 Introduction

The main motivation for the present work is the development of Univalent Type Theory by Voevodsky [23, 5]. In a nutshell, Univalent Type Theory is a variant of Martin-Löf's Type Theory where we give up the principle of uniqueness of identity proofs (UIP) to make it possible to treat equivalence of structures (e.g. isomorphism of sets) as equality. While Voevodsky's motivation comes from Homotopy Theory, Univalent Type Theory has an intrinsic type theoretic motivation in enabling us to treat abstract structures as first class citizens making it possible to combine high level reasoning and concrete applications without unnecessary clutter.

The central principle of Univalent Type Theory is the Univalence Axiom which states that equality of types is weakly equivalent to weak equivalence. Here weak equivalence is a notion motivated by homotopy theoretic models of type theory which can be alternatively understood as a refinement of the notion of isomorphism in the absence of UIP. The Univalence axiom can be viewed as a strong extensionality principle and indeed it implies functional extensionality. As with functional extensionality, univalence doesn't easily fit within the computational understanding of Type Theory, since it does not fit into the common pattern of introduction and elimination rules. The first author has suggested a solution of this problem for functional extensionality [1]: we can justify extensionality by a translation based on the setoid model. This approach was later refined [2] to *Observational Type Theory* which is the base for the development of Epigram 2 [9].

However, Observational Type Theory relies essentially on UIP and hence is incompatible with Univalent Type Theory. To address this we need to replace setoids with a structure able to model non-unique identity proofs. A first step in this direction is the groupoid model [11] but this forces UIP on the next level, i.e. for equality between equality proofs. [1] To be able to model Type Theory without UIP at any level we need to move to ω-groupoids. Moreover, the equalities we need to assume are in general non-strict (i.e. they are not definitional equalities in the sense of Type Theory) and hence we need to look at weak ω-groupoids. Indeed, as [8] and [17] have shown: Type Theory with Identity types gives rise to a weak ω-groupoid.

Our goal is to eliminate univalence by formalizing a weak ω-groupoid model of Type Theory in Type Theory. As a first step we need to implement the notion of a weak ω-groupoid in Type Theory and this is what we do in the present paper. An obvious possibility would have been to implement a categorical notion of weak ω-groupoids (e.g. based on globular operads) in Type Theory. However, this forces us to implement many categorical notions first, generating an avoidable overhead. It also seems that a structure with a more type theoretic flavour is more manageable in type theory and more accessible from a naive point of view. Hence, we are looking for a more direct type theoretic formulation of weak ω-groupoids. In the present paper we attempt this by defining a weak ω-groupoid to be a globular set with additional structure where this structure is given by interpreting a syntactic theory in the globular set. This approach is new and remains to be proved in our future work that it is equivalent to an established definition. The material presented here has been formalised in Agda [19]. We present it in a natural deduction style for the reader unfamiliar with Agda syntax.

In the following text, we first discuss and define globular sets in type theory in Section 2. In Section 3 we start introducing the syntax of ω-groupoids by defining the general syntactical framework which includes variables contexts, categories, and objects. We also define interpretation of the framework into a globular set. Sections 4, 5 and 6 are concerned with definition of the syntax of objects. Section 4 describes the construction of all units and objects; Section 5 defines the construction of all coherence cells witnessing the left and right unit laws, associativity of composition and interchange. In Section 6, we complete the definition of coherence by introducing all coherence cells between coherence cells. In Section 7 we summarise the results, and provide a rough comparison to other (categorical) approaches to weak ω-categories.

2 Globular Sets

In Type Theory we use the notion of a setoid to describe a set with a specific equality. That is a A : Setoid is given by the following data:

$$\begin{array}{lll} \mathsf{obj}_A & : & \mathsf{Set} \\ \mathrm{hom}_A & : & \mathsf{obj}_A \to \mathsf{obj}_A \to \mathbf{Prop} \end{array}$$

and proof objects $\mathsf{id}, -^{-1}, -;-$ witnessing that hom is an equivalence relation. Here we write **Prop** to denote the class of sets which have at most one inhabitant. This restriction is important when showing that the category of setoids has certain structure, in particular forms a model of Type Theory. That is setoids can model a type theory with a proof-irrelevant equality. To model proof-relevant equality we need to generalize the notion of a setoid so that the hom-sets are generalized setoids again. It is not enough to just postulate the laws of an equivalence relation at each level, we also need some laws how these proofs interact. On

[1] [16] have shown that the appropriate restriction of Univalence can be eliminated in this setting.

the first level we require the laws of a groupoid, e.g. we want that $\mathsf{id};\alpha$ is equal to α. Here *is equal* means that they are related by the equality relation of a setoid again. As demonstrated in [8, 17], the structure we are looking for is a weak ω-groupoid. It is the goal of this paper to develop a formalisation of this structure. As a first step let's ignore the proof objects (i.e. the data of an equivalence relation and the groupoid laws etc). We end up with a coinductive definition of a *globular set* $G : \mathsf{Glob}$ given by

$$\begin{array}{lcl} \mathsf{obj}_G & : & \mathsf{Set} \\ \mathrm{hom}_G & : & \mathsf{obj}_G \to \mathsf{obj}_G \to \infty\,\mathsf{Glob} \end{array}$$

Here we use ∞ to indicate that Glob is defined coinductively. More formally, Glob is the terminal coalgebra of $\Sigma\mathsf{obj} : \mathsf{Set}.\mathsf{obj} \to \mathsf{obj} \to -$. Given globular sets A, B a morphism $f : \mathsf{Glob}(A, B)$ between them is given by

$$\begin{array}{lcl} \mathrm{obj}_{\vec{f}} & : & \mathsf{obj}_A \to \mathsf{obj}_B \\ \mathrm{hom}_{\vec{f}} & : & \Pi a, b : \mathsf{obj}_A.\mathsf{Glob}(\mathrm{hom}_A\ a\,b, \mathrm{hom}_B(\mathrm{obj}_{\vec{f}}\ a, \mathrm{obj}_{\vec{f}}\ b)) \end{array}$$

Note that this definition exploits the coinductive character of Glob. Identity and composition can be defined easily by iterating the set-theoretic definitions ad infinitum. As an example we can define the terminal object in $\mathbf{1}_{\mathsf{Glob}} : \mathsf{Glob}$ by the equations

$$\begin{array}{lcl} \mathsf{obj}_{\mathbf{1}_{\mathsf{Glob}}} & = & \mathbf{1}_{\mathsf{Set}} \\ \mathrm{hom}_{\mathbf{1}_{\mathsf{Glob}}}\ x\,y & = & \mathbf{1}_{\mathsf{Glob}} \end{array}$$

More interestingly, the globular set of identity proofs over a given set A, $\mathrm{Id}^{\omega}\ A : \mathsf{Glob}$ can be defined as follows:

$$\begin{array}{lcl} \mathsf{obj}_{\mathrm{Id}^{\omega}\ A} & = & A \\ \mathrm{hom}_{\mathrm{Id}^{\omega}\ A}\ a\,b & = & \mathrm{Id}^{\omega}\ (a = b) \end{array}$$

Our definition of globular sets is equivalent to the usual one as a presheaf category over the diagram:

$$\begin{tikzcd} 0 \arrow[r, shift left, "s_0"] \arrow[r, shift right, "t_0"'] & 1 \arrow[r, shift left, "s_1"] \arrow[r, shift right, "t_1"'] & 2 \cdots n \arrow[r, shift left, "s_n"] \arrow[r, shift right, "t_n"'] & (n+1)\cdots \end{tikzcd}$$

with the globular identities:

$$\begin{array}{lcl} s_{i+1} \circ s_i & = & t_{i+1} \circ s_i \\ t_{i+1} \circ t_i & = & s_{i+1} \circ t_i \end{array}$$

In the example of $\mathrm{Id}^{\omega}\ A$ the presheaf is given by a family $F^A : \mathbb{N} \to \mathsf{Set}$:

$$\begin{array}{lcl} F^A\,0 & = & A \\ F^A\,1 & = & \Sigma a, b : A, a = b \\ F^A\,2 & = & \Sigma a, b : A, \Sigma\alpha, \beta : a = b, \alpha = \beta \\ \vdots & \vdots & \vdots \\ F^A\,(n+1) & = & \Sigma a, b : A, F^{a=b}\,n \end{array}$$

and source and target maps $s_i, t_i : F^A\,(n+1) \to F^A\,n$ satisfying the globular identities.

$$\begin{array}{lcllcl} s_0(a, b, -) & = & a & \qquad s_{n+1}\,(a, b, \alpha) & = & (a, b, s_n\,\alpha) \\ t_0(a, b, -) & = & b & \qquad t_{n+1}\,(a, b, \alpha) & = & (a, b, t_n\,\alpha) \end{array}$$

3 Syntax

Our goal is to specify the conditions under which a globular set is a weak ω-groupoid. This means we need to require the existence of certain objects in various object sets within the structure. A natural way would be to generalize the definition of a setoid and add these components to the structure. However, it is not clear how to capture the coherence condition which basically says that any two morphisms which just represent identities in the strict case should be equal. Instead we will follow a different approach which can be compared to the definition of environment models for the λ-calculus: we shall define a syntax for weak ω-groupoids and then define a weak ω-groupoid as a globular set in which this syntax can be interpreted.

3.1 The syntactical framework

We start by presenting a syntactical framework which is a syntax for globular sets. This syntax could be used to identify any globular set with structure (e.g. weak or strict ω-categories), the specific aspects of a weak ω-groupoid will be introduced later by adding additional syntax for objects and other auxiliary syntactic components.

Our framework consists of the following main components which are defined by mutual induction[2]:

Contexts

$$\mathsf{Con} : \mathsf{Set}$$

Contexts serve to formalize the existence of hypothetical objects which are specified by the globular set in which they live. E.g. to formalize ordinary composition we have to assume that objects a, b, c and 1-cells $f : a \longrightarrow b$ and $g : b \longrightarrow c$ exist to be able to form $g \circ f : a \longrightarrow c$.

Categories

$$\frac{\Gamma \;:\; \mathsf{Con}}{\mathsf{VarCat}\ \Gamma \;:\; \mathsf{Set}} \qquad \frac{\Gamma : \mathsf{Con}}{\mathsf{Cat}\ \Gamma : \mathsf{Set}}$$

In order to define the valid compositions of cells one needs to know their boundaries, i.e. iterated domains and codomains in the globular case. Category expressions record this data. We define two kinds of categories: VarCats are categories which contain only variables, while Cats contain all cells freely generated from variables. The set of expressions for both kinds of categories depends on a context, e.g. we need at least to assume that there is one object in the top-level category to be able to form any other categories.

Variables & Objects

$$\frac{G : \mathsf{VarCat}\ \Gamma}{\mathsf{Var}\ G : \mathsf{Set}} \qquad \frac{C : \mathsf{Cat}\ \Gamma}{\mathsf{Obj}\ C : \mathsf{Set}}$$

VarCats contain only variables, which are projections out of the context Γ. On the other hand, given a category we define all expressions which identify objects lying within the category. As indicated above this is the main focus of the forthcoming sections.

[2] This is an instance of an inductive-inductive definition in Type Theory, see [3].

We now specify the constructors for the various sets (apart from objects). We use unnamed variables ala deBruijn, hence contexts are basically sequences of categories. However, note that this is a dependent context since the well-formedness of a category expression may depend on the previous context. At the same time we build globular sets from nameless variables in contexts.

$$\frac{}{\varepsilon : \mathsf{Con}} \qquad \frac{G : \mathsf{VarCat}\ \Gamma}{(\Gamma, G) : \mathsf{Con}} \qquad \frac{}{\bullet : \mathsf{VarCat}\ \Gamma} \qquad \frac{G : \mathsf{VarCat}\ \Gamma \quad a, b : \mathsf{Var}\ G}{G[a, b] : \mathsf{VarCat}\ \Gamma}$$

$$\frac{}{\mathsf{vz} : \mathsf{Var}\ (\mathsf{wk}\ G)} \qquad \frac{v : \mathsf{Var}\ G}{\mathsf{vs}\ v : \mathsf{Var}\ (\mathsf{wk}\ G\ G')}$$

where wk is weakening defined for categories by induction on the structure in the obvious way:

$$\frac{G, G' : \mathsf{VarCat}\ \Gamma}{\mathsf{wk}\ G\ G' : \mathsf{VarCat}\ (\Gamma, G')} \qquad \begin{aligned} \mathsf{wk}\ \bullet\ G' &= \bullet \\ \mathsf{wk}\ (G[a, b])\ G' &= (\mathsf{wk}\ G\ G')[\mathsf{vs}\ a, \mathsf{vs}\ b] \end{aligned}$$

There are two ways to form category expressions: there is the top level category denoted by $\bullet$ and given any two objects a, b in a category C we can form the hom category $C[a, b]$.

$$\frac{}{\bullet : \mathsf{Cat}\ \Gamma} \qquad \frac{C : \mathsf{Cat}\ \Gamma \quad a,\ b : \mathsf{Obj}\ C}{C[\,a\ ,\ b\,] : \mathsf{Cat}\ \Gamma}$$

Variables become objects by the following constructor of Obj, which mutually extends to VarCats:

$$\frac{v : \mathsf{Var}\ G}{\mathsf{var}\ v : \mathsf{Obj}\ (\mathsf{var}\ G)} \qquad \frac{G : \mathsf{VarCat}\ \Gamma}{\mathsf{var}\ G : \mathsf{Cat}\ \Gamma} \quad \text{where} \quad \begin{aligned} \mathsf{var}\ \bullet &= \bullet \\ \mathsf{var}\ G[a, b] &= (\mathsf{var}\ G)[\mathsf{var}\ a, \mathsf{var}\ b] \end{aligned}$$

We use the usual arrow notation for categories and objects. For instance, $\bullet[a, b]$, $\bullet[a, b][f, g]$ and $\alpha : \mathsf{Obj}\ (\bullet[a, b][f, g])$ are pictured respectively as follows:

$$\begin{tikzcd} a & b \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, bend left, "f"] \arrow[r, bend right, "g"'] & b \end{tikzcd} \qquad \begin{tikzcd} a \arrow[r, bend left, "f", ""{name=U, below}] \arrow[r, bend right, "g"', ""{name=D}] & b \arrow[Rightarrow, from=U, to=D, "\alpha"] \end{tikzcd}$$

We also write, as usual, $x : a_n \longrightarrow b_n : \cdots : a_0 \longrightarrow b_0$ for an $x : \mathsf{Obj}\ (\bullet[a_0, b_0] \cdots [a_n, b_n])$. Note that it is essential to first introduce VarCats and then Cats with an inclusion

$$\mathsf{var} : \Sigma(\mathsf{VarCat}\ \Gamma)\ \mathsf{Var} \longrightarrow \Sigma(\mathsf{Cat}\ \Gamma)\ \mathsf{Obj}$$

In this way we make sure that variables alone form a globular set, i.e. that the domain and codomain of a variable is a variable. In particular, that it is not possible to introduce a variable between syntactically constructed coherence cells. In this way we can talk about the ω-category freely generated by a globular set.

3.2 Interpretation

Given a globular set we define what we mean by an interpretation of the syntax. Once we have specified all the constructors for objects a weak ω-groupoid is given by such an interpretation. The interpretation of the structural components given in the present section is fixed. Again this is reminiscent of environment models.

An *interpretation* in a globular set G : Glob is given by the following data:

1. An assignment of sets to contexts:

$$\frac{\Gamma : \mathsf{Con}}{[\![\Gamma]\!] : \mathsf{Set}}$$

2. An assignment of globular sets to VarCat and Cat expressions:

$$\frac{G : \mathsf{VarCat}\ \Gamma \qquad \gamma : [\![\Gamma]\!]}{[\![G]\!]\ \gamma : \mathsf{Glob}} \qquad \frac{C : \mathsf{Cat}\ \Gamma \qquad \gamma : [\![\Gamma]\!]}{[\![C]\!]\ \gamma : \mathsf{Glob}}$$

3. An assignment of elements of object sets to object expressions and variables

$$\frac{G : \mathsf{VarCat}\ \Gamma \qquad x : \mathsf{Var}\ G \qquad \gamma : [\![\Gamma]\!]}{[\![x]\!]\ \gamma : \mathsf{obj}_{[\![G]\!]\ \gamma}} \qquad \frac{C : \mathsf{Cat}\ \Gamma \qquad A : \mathsf{Obj}\ C \qquad \gamma : [\![\Gamma]\!]}{[\![A]\!]\ \gamma : \mathsf{obj}_{[\![C]\!]\ \gamma}}$$

subject to the following conditions:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
[\![\varepsilon]\!] & = & 1 & [\![\mathsf{var}\ x]\!]\ \gamma & = & [\![x]\!]\ \gamma \\
[\![\Gamma, G]\!] & = & \Sigma\gamma : [\![\Gamma]\!], [\![G]\!]\ \gamma & [\![\mathsf{vz}]\!]\ (\gamma, a) & = & a \\
[\![\bullet]\!]\ \gamma & = & G & [\![\mathsf{vs}\,x]\!]\ (\gamma, a) & = & [\![x]\!]\ \gamma \\
[\![C[a,b]]\!]\ \gamma & = & \mathsf{hom}_{[\![C]\!]\gamma}\ ([\![a]\!]\ \gamma)\ ([\![b]\!]\ \gamma) & & &
\end{array} ,$$

where the last case applies both to VarCats and Cats.

4 Structure

A category, strict or weak, is a globular set with additional structure. The difference between the strict and the weak case is whether we adorn the structure with (equational) constraints or whether one instead of axioms introduces more structure, which witnesses rather than postulates the constraints; so-called *coherence cells*. In this section we introduce the syntax for the structure of composition and units giving rise to syntax for what one could call a *pre-monoidal globular category*, where composites and units are expressible but unconstrained by coherence cells.

4.1 Composition

In the ordinary case, a category, $\mathcal{C}$, defines an indexed operation of composition on its arrows. Explicitly, for a, b, c objects of $\mathcal{C}$, f in $\mathcal{C}[a,b]$, g in $\mathcal{C}[b,c]$, there is a gf in $\mathcal{C}[a,c]$. In the higher-dimensional case, $\mathcal{C}(a,b)$ and $\mathcal{C}(b,c)$ are not mere sets but ω-categories and composition extends from sets the whole hom-categories. Informally: for a, b, c as before, f, g, n-cells of homcategories $\mathcal{C}(a,b)$ and $\mathcal{C}(b,c)$, respectively, one requires an n-cell $g \circ f$ of $\mathcal{C}(a,c)$. The fact that both f and g are of the same relative depth with respect to $\mathcal{C}$ is important, as well the fact the homcategories of f and g meet at a common object, b, of $\mathcal{C}$. Following are some examples of valid compositions for increasing n:

$$\begin{tikzcd}
a \arrow[r, "f"] & b \arrow[r, "g"] & c & \mapsto & a \arrow[r, "gf"] & c & & a \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & b \arrow[r, bend left, "g"] \arrow[r, bend right, "g'"'] \arrow[r, phantom, "\Downarrow\beta"] & c & \mapsto & a \arrow[r, bend left, "gf"] \arrow[r, bend right, "g'f'"'] \arrow[r, phantom, "\Downarrow\beta\alpha"] & c
\end{tikzcd} \qquad (1)$$

$$\begin{tikzcd}[row sep=large]
a \arrow[r, bend left=60, "f"] \arrow[r] \arrow[r, bend right=60, "f''"'] \arrow[r, phantom, shift left=3, "\alpha\Downarrow \overset{\gamma}{\Rrightarrow} \Downarrow\alpha'"] \arrow[r, phantom, shift right=3, "\beta\Downarrow \underset{\delta}{\Rrightarrow} \Downarrow\beta'"] & b & \mapsto & a \arrow[r, bend left, "f"] \arrow[r, bend right, "f''"'] \arrow[r, phantom, "\beta\alpha\Downarrow \overset{\delta\gamma}{\Rrightarrow} \Downarrow\beta'\alpha'"] & c
\end{tikzcd} \qquad (2)$$

We formalise this as follows.

4.1.1 Telescopes

The type $\mathsf{Obj} : \mathsf{Cat} \longrightarrow \mathsf{Set}$ represents the set of syntactical objects lying directly in any category. In order to talk about arbitrary n-cells of a category, for instance to define their compositions, we must introduce *telescopes*. Informally, a telescope is a category in a category. Formally, telescopes, Tel, are defined below at the same time as their *concatenation* onto a category, $+\!\!+$, which takes a telescope to a category, and therefore allows us to put objects into a telescope:

$$\frac{C : \mathsf{Cat}\ \Gamma \qquad n : \mathbb{N}}{\mathsf{Tel}\ C\ n : \mathsf{Set}} \qquad \frac{t : \mathsf{Tel}\ C\ n}{C \mathbin{+\!\!+} t\ :\ \mathsf{Cat}\ \Gamma}$$

Telescopes are like categories except that the base case is an arbitrary category C rather than $\bullet$:

$$\frac{}{\bullet : \mathsf{Tel}\ C\ 0} \qquad \frac{t : \mathsf{Tel}\ C\ n \quad a, b : \mathsf{Obj}\,(C \mathbin{+\!\!+} t)}{t[a,b] : \mathsf{Tel}\ C\ (n+1)} \qquad \begin{array}{lcl} C\ \mathbin{+\!\!+}\ \bullet & = & C \\ C\ \mathbin{+\!\!+}\ t[a,b] & = & (C \mathbin{+\!\!+} t)[a,b] \end{array}$$

Here, we call n the *length of* t, and we say any $x : \mathsf{Obj}(C \mathbin{+\!\!+} t)$ to be *of depth* n.

We say that t *lies in* C. Note that only the left associative reading of $+\!\!+$ makes sense so expressions like $C \mathbin{+\!\!+} t \mathbin{+\!\!+} u$ are unambiguous.

We say that an object $x : \mathsf{Obj}\ (C \mathbin{+\!\!+} t)$ *lies in (the telescope)* t. When t lies in C, x is called an *object relative to* C. Alternatively, when the category t lies in is not important we use the following syntactical shorthand:

$$\frac{C : \mathsf{Cat}\ \Gamma \quad t : \mathsf{Tel}\ C\ n}{t \Downarrow : \mathsf{Cat}\ \Gamma \qquad t \Downarrow = C \mathbin{+\!\!+} t}$$

For example, given the category $$\begin{tikzcd} a \arrow[r, bend left, "f"{name=F}] \arrow[r, bend right, "g"'{name=G}] \arrow[from=F, to=G, phantom, "\varphi\Downarrow\quad\Downarrow\gamma"] & b \end{tikzcd}$$, one has:

$$\bullet[\varphi,\gamma] \Downarrow\ =\ \bullet[a,b][f,g] \mathbin{+\!\!+} \bullet\ [\varphi,\gamma]\ =\ \bullet[a,b][f,g][\varphi,\gamma]\ .$$

4.1.2 Back to composition

We use telescopes to define syntax for all compositions of an ω-category. These are defined mutually recursively with their extensions to telescopes:

$$\frac{t : \mathsf{Tel}\ (C[a,b])\ n \qquad u : \mathsf{Tel}\ (C[b,c])\ n}{u \circ t : \mathsf{Tel}\ (C[a,c])} \qquad \frac{\alpha : \mathsf{Obj}(C[a,b] \mathbin{+\!\!+} t) \qquad \beta : \mathsf{Obj}(C[b,c] \mathbin{+\!\!+} u)}{\beta \circ \alpha : \mathsf{Obj}(C[a,c] \mathbin{+\!\!+} (u \circ t))}$$

Where $\circ$ is a new constructor of Obj and $\circ$ for telescopes is a function defined by cases

$$\begin{array}{lcl} \bullet \circ \bullet & = & \bullet \\ u[a',b'] \circ t[a,b] & = & (u \circ t)[a' \circ a, b' \circ b] \end{array}$$

Any α and β as above are said to be *composable.* Note that for a fixed category C, $\circ$ always defines the composition in C, called *horizontal* in the 2-categorical case, which can be applied to all composable $(n+1)$-cells of C, where n is the length of the telescopes t and u. To compose cells "vertically", one moves to a homcategory. In 2-category theory, horizontal composition is usually denoted $\circ$ or $*$ or is left out, whereas vertical composition by $\cdot$. In our case, we always use $\circ$ and the level we mean is contained in the (implicit) parameter C. For example, examples in (1) are both horizontal compositions where $C = \bullet$, while (2) is a vertical composition where $C = \bullet[a,b]$.

4.2 Units

We generate all higher units from a single constructor id defined as follows:

$$\frac{a : \mathsf{Obj}\ C}{\mathsf{id}\ a : \mathsf{Obj}\ C[a, a]}$$

By iteration we obtain the unit for horizontal composition of n-cells:

$$\frac{a : \mathsf{Obj}\ C \qquad n : \mathbb{N}}{\mathsf{idTel}\, a\, n : \mathsf{Tel}\ C\ n \qquad \mathsf{id}^n a : \mathsf{Obj}\ (\mathsf{idTel}\, a\, n \Downarrow)}$$

Again, an iterated unit is defined at the same time as its telescope.

$$\begin{array}{lcl@{\qquad}lcl} \mathsf{idTel}\, a\, 0 & = & \bullet & \mathsf{id}^0 a & = & a \\ \mathsf{idTel}\, a\, (n+1) & = & (\mathsf{idTel}\, a\, n)[\mathsf{id}^n a, \mathsf{id}^n a] & \mathsf{id}^{(n+1)} a & = & \mathsf{id}\, (\mathsf{id}^n a) \end{array}$$

5 Laws

In a strict ω-category composition and identities – structure – are accompanied by axioms expressing their fundamental properties. Namely, composition should be *associative*, and it should satisfy the so-called *interchange law*; identities should be the *units* of composition. The fact that the axioms are equations and one can therefore replace equals for equals in expressions has the pleasant consequence that the complexity of axioms doesn't increase with dimension. Indeed, the whole theory for strict ω-categories can be generalised without much difficulty to categories enriched in an arbitrary monoidal category [12]. However, once the equational axioms are replaced by data – *coherence cells* – their complexity rises steeply with dimension.

The combinatorial complexity of coherence cells has been a major obstacle in the development of weak ω-categories. It has led to the development of many diverse approaches to weak ω-categories, e.g. [21, 7, 6, 22, 20, 15, 17]. Comprehensive surveys and comparisons can be found in [14, 10]. However the development of Type Theory has made it possible to express all coherence cells in a closed form. In this section we start to describe how in Type Theory all coherence cells can be generated by induction on their depth.

For example, the 1-categorical left-unity law:

$$\mathsf{id}_b \circ f \quad = \quad f \ , \tag{3}$$

for all $f : a \longrightarrow b$, is replaced in a weak ω-category by a pair of 2-cells $\lambda_f : \mathsf{id}_b \circ f \Longrightarrow f$ and $\lambda_f^{-1} : f \Longrightarrow \mathsf{id}_b \circ f$. A similar law should hold for $\circ$ and higher cells. I.e. it should also hold in the strict case that for any $\alpha : f \Longrightarrow f'$:

$$\mathsf{id}_b^2 \circ \alpha \quad = \quad \alpha \ , \tag{4}$$

where $\mathsf{id}_b^2 = \mathsf{id}_{\mathsf{id}_b}$. Note that (4) makes sense because (3) holds. In the weak case, it is not the case that the boundary of $\mathsf{id}_b^2 \circ \alpha$ is equal to the boundary of α and it is simply not possible to categorify (4) by introducing a pair of 3-cells between the left and right side of (4). However, we can use λ_f and λ_f^{-1} to coerce the boundary of the former, $\mathsf{id}_b \circ f$ and $\mathsf{id}_b \circ f'$, to the boundary of the latter, f and f', respectively. The following figure illustrates this idea:

$$\begin{tikzcd}
\lambda_\alpha \ : & a \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] \arrow[rr, bend left=70, "f"] \arrow[rr, bend right=70, "f'"'] & b \arrow[r, bend left, "\mathsf{id}_b"] \arrow[r, bend right, "\mathsf{id}_b"'] \arrow[r, phantom, "\Downarrow\mathsf{id}_b^2"] \arrow[u, phantom, "\Downarrow\lambda_f^{-1}"] \arrow[d, phantom, "\Downarrow\lambda_{f'}"] & b & \Rrightarrow & a \arrow[r, bend left, "f"] \arrow[r, bend right, "f'"'] \arrow[r, phantom, "\Downarrow\alpha"] & b
\end{tikzcd} \tag{5}$$

The reader is invited to try to write down the fourth iteration, i.e. the domain and codomain of λ_γ for $\gamma : \alpha \Rrightarrow \alpha' : f \Longrightarrow' f : a \longrightarrow a'$. Note that each higher pair of λ's can be seen as expressing the naturality of the preceding lower lambda.

Similarly one must introduce ρ's to witness the right unit law, χ's to witness interchange, and α's to witness associativity. In the case of groupoids where every arrow has an inverse there are ι's and κ's to witness the left and right cancellation properties. The example of λ has been chosen because of its relative simplicity.

Moreover, all such coherence cells must satisfy a coherence property basically saying that any pair of n-cells from d to d' involving only coherence cells and units[3] must have a mediating n+1-cell connecting d and d'. Intuitively, as the coherence cells λ, ρ, α and χ we have just described witness axioms, the higher coherence cells witness their closure under composition and identity.

5.1 Formalising left units

In (5) we made the boundaries of the left- and right-hand sides match by applying the function:

$$\Phi \quad \equiv \quad (l, l') \quad \mapsto \quad x \quad \mapsto \quad l' \cdot x \cdot l$$

to $(\lambda_f^{-1}, \lambda_{f'})$ and $\mathsf{id}_b^2 \circ \alpha$. The 3-cells λ_α and λ_α^{-1} are then introduced as

$$\begin{array}{lcl} \lambda_\alpha & : & \mathsf{Obj}\ (\bullet[a,b][f,f'][\Phi\,(\lambda_f^{-1}, \lambda_{f'})\,(\mathsf{id}_b^2 \circ \alpha), \alpha]) \\ \lambda_\alpha^{-1} & : & \mathsf{Obj}\ (\bullet[a,b][f,f'][\alpha, \Phi\,(\lambda_f^{-1}, \lambda_{f'})\,(\mathsf{id}_b^2 \circ \alpha)]) \end{array} \quad .$$

These arrows should be natural in 3-cells $\gamma : \alpha \Rrightarrow \alpha'$. In a diagram:

$$\begin{tikzcd}
\mathsf{id}_b \circ f \arrow[d, "\mathsf{id}_b^2 \circ \alpha"'] & & f \arrow[ll, "\lambda_f^{-1}"'] \arrow[d, "\alpha"] \\
\mathsf{id}_b \circ f' \arrow[rr, "\lambda_{f'}"'] \arrow[u, phantom, shift left=8, "\overset{\lambda_\alpha^{-1}}{\Lleftarrow}\ \overset{\lambda_\alpha}{\Rrightarrow}"] & & f'
\end{tikzcd}
\qquad
\begin{tikzcd}
\lambda_{f'} * (\mathsf{id}_b^2 \circ \alpha) * \lambda_f^{-1} \arrow[d, "\mathsf{id}_{\lambda_{f'}} * (\mathsf{id}_b^3 \circ \gamma) * \mathsf{id}_{\{\lambda_f^{-1}}"'] & & \alpha \arrow[ll, "\lambda_\alpha^{-1}"'] \arrow[d, "\gamma"] \\
\lambda_{f'} * (\mathsf{id}_b^2 \circ \alpha') * \lambda_f^{-1} \arrow[rr, "\lambda_{\alpha'}"'] \arrow[u, phantom, shift left=20, "\overset{\lambda_\gamma^{-1}}{\Lleftarrow}\ \overset{\lambda_\gamma}{\Rrightarrow}"] & & \alpha'
\end{tikzcd}$$

Note that going top-left-bottom around the square one gets

$$\Phi\,(\lambda_\alpha^{-1}, \lambda_{\alpha'})\,(\Phi\,(\lambda_f^{-1}, \lambda_{f'})\,\gamma)) \ .$$

This is the basic idea of the recursion generating all higher λ's. A similar pattern occurs in the definition of the other coherence cells.

[3] Identity cells can be seen as coherence cells witnessing reflexivity of equality.

5.2 Formalising all coherence cells

To summarise and generalise, we want to introduce for each α in a telescope t of length n and each β in a telescope u of length n a cell $\Phi\, m\, \alpha \longrightarrow \beta$ where m is the data necessary to define a function $\mathsf{Obj}\ (t \Downarrow) \to \mathsf{Obj}\ (u \Downarrow)$. We will call such an m a *telescope morphism from t to u*; formally $m : t \rightrightarrows u$. Then Φ has type $t \rightrightarrows u \to \mathsf{Obj}(t \Downarrow) \to \mathsf{Obj}(u \Downarrow)$. Formally, we define telescope morphisms as follows; in mutual recursion with their application to telescopes and objects in telescopes:

$$\frac{t, u : \mathsf{Tel}\, C\, n}{t \rightrightarrows u : \mathsf{Set}} \qquad \frac{}{\bullet : \bullet \rightrightarrows \bullet} \qquad \frac{m : t \rightrightarrows u \quad \alpha : \mathsf{Obj}\,(u \Downarrow [a', m@a]) \quad \mathsf{Obj}\,(u \Downarrow [m@b, b'])}{m[\alpha, \beta] : t[a,b] \rightrightarrows u[a', b']}$$

where

$$\frac{m : t \rightrightarrows u \quad t' : \mathsf{Tel}\,(t \Downarrow)\, n}{m \overrightarrow{@} t' : \mathsf{Tel}\,(u \Downarrow)\, n} \qquad \frac{m : t \rightrightarrows u \quad a : \mathsf{Obj}\,(t \Downarrow +\!\!+\, t')}{m@a : \mathsf{Obj}\,(u \Downarrow +\!\!+\, m \overrightarrow{@} t')}$$

$$\begin{aligned} \bullet \overrightarrow{@} t &= t \\ m'[\alpha, \beta] \overrightarrow{@} t &= (m' \overrightarrow{@} t)[m@\alpha, m@\beta]) \end{aligned}$$

Φ is then a special case of $@$ for $t' = \bullet$. To define $@$ we need the following auxiliary function, among others, which extends a telescope on the left.

$$\frac{t : \mathsf{Tel}\ (C[a,b])\ n}{[a,b]t : \mathsf{Tel}\ C\ (n+1)} \qquad \text{where} \qquad \begin{aligned} [a,b]\bullet &= \bullet[a,b] \\ [a,b](t[c,d]) &= ([a,b]t)[c,d] \end{aligned}$$

Note that here c and d don't actually fit into the telescope $[a,b]t$ because the latter is definitionally different from t. However, it is straightforward to prove by induction that

$$t \Downarrow \quad \equiv \quad [a,b]t \Downarrow\ , \tag{6}$$

and use the proof to make c and d fit. However, in the interest of clarity we left the details out above. The full details can be found in [4].

We are now in the position do define $@$ as follows: The base case is trivial:

$$\bullet @ x \;=\; x$$

The hom-case follows the pattern outlined in Section 5.1.

$$\frac{m'[\alpha, \beta] : t[a,b] \rightrightarrows u[a', b'] \qquad t' : \mathsf{Tel}\ (t[a,b] \Downarrow)\ n \qquad x : \mathsf{Obj}\ (t' \Downarrow)}{m'[\alpha,\beta]@x \;=\; \mathsf{id}^n \beta \circ (m'@x) \circ \mathsf{id}^n \alpha}$$

In summary, $m@x$ is defined by induction on m where in each step the length of m decreases by one and the depth of x increases by one. To make the levels match the category of x has to be whiskered by the morphisms α, β for $m = m'[\alpha, \beta]$. When $m = \bullet$, the recursion stops. The meticulous reader will have noticed that the expression $m'@x$ above is not well typed as x lives in $t[a,b] +\!\!+\, t'$ and we need an object in $t +\!\!+ [a,b]t'$. But this is easily fixed by substituting using (6). Other similar inaccuracies are dealt with similarly.

Here is an illustration for $m = \bullet[\varphi, \gamma][\alpha, \beta]$, $t' = \bullet$, $t = \bullet[a,b][f,g]$, $u = \bullet[a', b'][f', g']$:

$$\begin{tikzcd}
a' \arrow[r, bend left, "\varphi"] \arrow[r, bend right, "\varphi"'] \arrow[r, phantom, "\Downarrow \mathsf{id}_\varphi"] \arrow[rrr, bend left=60, "f'"] \arrow[rrr, bend right=60, "g'"'] \arrow[rrr, phantom, bend left=35, "\Downarrow \alpha"] \arrow[rrr, phantom, bend right=35, "\Downarrow \beta"] & m@a \arrow[r, bend left, "m@f"] \arrow[r, bend right, "m@g"'] \arrow[r, phantom, "\Downarrow m@x"] & m@b \arrow[r, bend left, "\gamma"] \arrow[r, bend right, "\gamma"'] \arrow[r, phantom, "\Downarrow \mathsf{id}_\gamma"] & b'
\end{tikzcd}$$

Having defined telescope morphisms, it is relatively easy to define λ's of all depths relative to an arbitrary category. All that is needed is a telescope morphism, $\overrightarrow{\lambda}$, together with a new constructor, λ, of Obj:

$$\frac{t : \mathsf{Tel}\ C[a,b]\ n}{\overrightarrow{\lambda}\, t : (\mathsf{idTel}\,(\,\mathsf{id}\, b)\, n) \circ t \rightrightarrows t} \qquad \frac{t : \mathsf{Tel}\ C[a,b]\, n \quad f : \mathsf{Obj}(t \Downarrow)}{\lambda\, t\, f : \mathsf{Obj}\, ((t \Downarrow)[\overrightarrow{\lambda}\, t@(\mathsf{id}^n b \,\circ\, f), f])}$$

where

$$\begin{array}{lcl} \overrightarrow{\lambda}\, \bullet & = & \bullet \\ \overrightarrow{\lambda}\, (t'[a,b]) & = & (\overrightarrow{\lambda}\, t')[(\lambda\, t'\, a)^{-1}, \lambda\, t'\, b] \end{array}$$

Here we could define a pair of constructors λ and λ^{-1} for the two opposite directions of λ. Instead, as we are interested in groupoids, we define a generic constructor ${}^{-1}$ on all cells of a homcategory:

$$\frac{f : \mathsf{Obj}\ (C[a,b])}{f^{-1} : \mathsf{Obj}\, (C[b,a])}$$

The introduction of formal inverses forces the introduction of coherence cells witnessing their being left and right inverses. See the next section.

5.3 Right units, associativity and interchange

Similarly to λ's we define the remaining coherence cells, i.e. ρ's to witness right units, α's to witness associativity of composition, ι's and κ's to witness inverses and χ's to witness interchange. These are defined analogically to λ's.

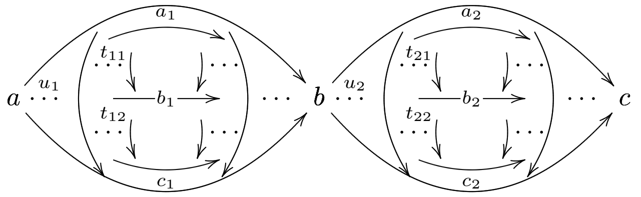
To this end, note that everything in the definition of λ is forced by the type of $\overrightarrow{\lambda}$. In general it is enough to give for ρ and α the type of the telescope morphism. Just as in the case of λ, it is in each case just a "telescopisation" of the ordinary case.

$$\frac{t : \mathsf{Tel}\, C[a,b]\, n}{\overrightarrow{\rho}\, t : t \circ (\mathsf{idTel}\ \ (\mathsf{id}\, a)\, n) \rightrightarrows t} \qquad \frac{t : \mathsf{Tel}\ C[a,b]\ m \quad u : \mathsf{Tel}\ C[b,c]\ n \quad v : \mathsf{Tel}\ C[c,d]\ o}{\overrightarrow{\alpha}\, t\, u\, v : (v \circ u) \circ t \rightrightarrows v \circ (u \circ t)}$$

Because of the way we introduce identities, the laws of inverses are also simple:

$$\frac{f : \mathsf{Obj}\ (C[a,b])}{\iota\ f : \mathsf{Obj}\ (C[a,a][f^{-1} \circ f, \mathsf{id}\ a]) \qquad \kappa\ f : \mathsf{Obj}(C[b,b][f \circ f^{-1}, \mathsf{id}\ b])}$$

The coherence cells witnessing interchange, $\overrightarrow{\chi}$ are in the ω case is more complicated. In the simple 2-categorical case, the interchange law states that $(\gamma' {\cdot} \gamma) {*} (\varphi' {\cdot} \varphi) \ = \ (\gamma' {*} \varphi') {\cdot} (\gamma {*} \varphi)$. In the ω-case the law remains syntactically the same but we consider each of φ, φ', γ and γ' in their telescopes with the generalised notion of composability. The following picture illustrates the idea:



Where $\cdots$ indicate telescopes of arbitrary depth where u_1 and u_2 have to be of the same length; and $t_{ij},\ i,j \in \{1,2\}$ have to be of the same length. In this situation, it is possible, up to definitional equality of telescopes, to form both the composition $(t_{22} \circ t_{21}) \circ (t_{12} \circ t_{11})$ and also $(t_{22} \circ t_{12}) \circ (t_{21} \circ t_{11})$. A telescope morphism from the former to the latter telescope induces a coherence cell for interchange. This is formalised as follows:

$$\frac{\begin{array}{c} u_1 : \mathsf{Tel}\ \mathcal{C}[a,b]\ n \quad u_2 : \mathsf{Tel}\ \mathcal{C}[b,c]\ n \quad t_{11} : \mathsf{Tel}\ (C[a,b] +\!\!+\, u_1)\ m \\ t_{12} : \mathsf{Tel}\ (C[a,b] +\!\!+\, u_1)\ m \quad t_{21} : \mathsf{Tel}\ (C[b,c] +\!\!+\, u_2)\ m \quad t_{22} : \mathsf{Tel}\ (C[b,c] +\!\!+\, u_2)\ m \end{array}}{\chi\, t_{11}\, t_{12}\, t_{21}\, t_{22} \ :\ (t_{22} \circ t_{21}) \circ (t_{12} \circ t_{11}) \rightrightarrows (t_{22} \circ t_{12}) \circ (t_{21} \circ t_{11})}$$

6 Coherence

6.1 The need of more coherence

In the previous sections we showed how to define all coherence cells witnessing the axioms of a strict ω-groupoid. These can be composed to witnesses identity of cells which don't exactly match the sides of either of the axioms.

For instance, to witness the equality

$$(g\,\mathsf{id}_b)\,f \quad = \quad g\,f$$

we can compose λ after α to obtain witnesses such as:

$$\mathsf{id}_g\lambda_f \cdot \alpha_{g,\mathsf{id}_g,f} \ :\ (g\,\mathsf{id}_b)\,f \longrightarrow g\,f \qquad \text{and} \qquad \rho\,\mathsf{id}_f \ :\ (g\,\mathsf{id}_b)\,f \longrightarrow g\,f \quad .$$

In the strict case equality of 1-cells is a proposition and therefore all proofs of equality of 1-cells are equal. In the weak 2-categorical case, equality of 1-cells is not propositional but equality of 2-cells is. In other words, equality of 1-cells is witnessed by 2-cells which must satisfy new axioms. For instance:

$$\begin{tikzcd}
(g\,\mathsf{id}_b)\,f \arrow[r, "\alpha_{g,\mathsf{id}_g,f}"] \arrow[dr, "\rho\,\mathsf{id}_f"'] & g\,(\mathsf{id}_b\,f) \arrow[d, "\mathsf{id}_g\ \lambda_f"] \\
& g\,f
\end{tikzcd} \tag{7}$$

In our case, when all levels are weakened the equality is replaced be a new 3-cell

$$\mathsf{id}_g\lambda_f \cdot \alpha_{g,\mathsf{id}_g,f} \longrightarrow \rho\,\mathsf{id}_f$$

Moreover, for any pair of such 3-cells, p, q : $\mathsf{id}_g\lambda_f \cdot \alpha_{g,\mathsf{id}_g,f} \longrightarrow \rho\,\mathsf{id}_f$, there must be 4-cell $p \longrightarrow q$, etc., all the way up to ω. This is a weakening of propositionality of equality of n-cells in the strict setting. The following diagram illustrates this up to level 4:

$$\begin{tikzcd}
(g\,\mathsf{id}_b)\,f \arrow[r, "\alpha_{g,\mathsf{id}_g,f}"] \arrow[dr, "\rho\,\mathsf{id}_f"', ""{name=D}] & g\,(\mathsf{id}_b\,f) \arrow[d, "\mathsf{id}_g\ \lambda_f", ""{name=U, left}] \\
& g\,f
\arrow[from=U, to=D, bend right, "p"', ""{name=P}]
\arrow[from=U, to=D, bend left, "q", ""{name=Q, above}]
\arrow[from=P, to=Q, dotted]
\end{tikzcd} \tag{8}$$

In summary and full generality:

> *For any pair of coherence cells with the same domain and codomain, there must be a mediating coherence cell.*

6.2 Formalising coherence cells between coherence cells

We formalise the above principle as follows. We define a predicate thin on objects such that $\mathsf{thin}\,(\mathsf{id}\,f) = \top$ where $\top$ is the one element Set. Moreover each coherence cell is thin

$$\mathsf{thin}\,(\lambda\,\_\,\_) = \top \quad \mathsf{thin}(\rho\,\_\,\_) \ = \ \top \quad \mathsf{thin}\,(\alpha,\_\,\_\,\_) = \top \quad \mathsf{thin}\ (\chi\,\_\,\_\,\_\,\_\,\_) = \top$$
$$\mathsf{thin}\,(\iota\ f) = \top \quad \mathsf{thin}\,(\kappa\ f) = \top$$

Further we close thin under weakening, composition and inverses. Finally, we introduce a constructor of Obj and a clause of thin:

$$\frac{f\ g : \mathsf{Obj}\ C[a,b] \qquad p : \mathsf{thin}\ f \qquad q : \mathsf{thin}\ g}{\mathsf{coh}\ p\ q : \mathsf{Obj}\ C[a,b][f,g]} \qquad \mathsf{thin}\,(\mathsf{coh}\,p\,q) = \top$$

The last remaining case are variables, which are not thin:

$$\mathsf{thin}\,(\mathsf{var}\,v) = \bot$$

6.3 The problem with coherence

The question of coherence in weak ω-categories is subtle. On the one hand, one needs enough coherence to make weakly equivalent all cells that should be identities in the strict case. On the other hand, if we shouldn't add too many equations not to loose any of our intended models. We believe our definition is correct as we are saying in the definition of coh only that all *formal* diagrams of coherence cells commute (see [18], §VII.2, for a related discussion). The fact that not all diagrams commute rests on the fact that one is not allowed to postulate equations between nonvariables, which is achieved by the separation of VarCat and Cat and the fact that contexts are built from VarCats. Otherwise it would be for instance possible[4] to assume a 0-cell a, and a pair of 2-cells: $x, y : \mathsf{id}\ a \to \mathsf{id}\ a : a \to a$. Then it is be possible to construct, by the Eckerman-Hilton argument, a cell $\eta : x \circ y \to y \circ x : \mathsf{id}\ a \to \mathsf{id}\ a : a \to a$ which is thin. As $\mathsf{id}\ (x \circ y)$ is also thin, $\eta \circ \eta$ and $\mathsf{id}(x \circ y)$ would be equivalent by coh. However, not every monoidal braided category is symmetrical. However it is not possible to construct this example in our syntactical framework as $(\mathsf{id}\ a)$ is not a variable.

7 Conclusions and Further Work

7.1 Summary

We have presented a novel approach to defining weak ω-groupoids which is based on ideas from Type Theory. The central idea is to define the syntax of weak ω-groupoids and then define a weak ω-groupoid as a globular set with an interpretation of the syntax, which is where Type Theory has its greatest strength. Our approach to formalization of coherence is natural, in a way naive, since it is a natural generalisation of the corresponding first order laws.

We have formalized all of the material presented here in Agda [19], except for the definition of the telescope morphism for χ. We believe this is a technicality, albeit a difficult one. The Agda source file is available from [4].

[4] We are indebted to an anonymous referee for pointing out this example to us.

7.2 Related work

There exists an abundance of categorical definitions of weak ω-categories and groupoids. Although a direct comparison is a slippery road, we would like to give a rough comparison of the key similarities and differences between our and other definitions. Most importantly, our definition is fundamentally different in that it is formulated in Type Theory rather than Category Theory. This means that we couldn't just formalise any of the approaches [20, 7, 15] because the notion a strict ω-category is central in them in that it drives the definition of coherence cells. However, it is unclear to how define *strict* ω-categories in Type Theory without quotient types. This forces some of the choices we have made. Nevertheless, on an intuitive level there are similarities of our approach to some categorical approaches, in particular to Batanin's definition [7] which we briefly discuss below:

- Batanin's *spans* are essentially our telescopes. And as noted by Batanin, Cartier called Batanin's spans telescopes in 1994.
- Batanin's definition, same as ours, is globular and works with a system of units and binary compositions.
- We conjecture that for a $C : \mathsf{Cat}\ \Gamma$, the mapping $n \mapsto \Sigma(t : \mathsf{Tel}\ C\ n)(\mathsf{Obj}(t \Downarrow))$ forms a strict monoidal globular category freely generated by the globular set determined by Γ.
- It remains to show that our syntax defines a contractible ω-operad and our notion of interpretation defines its algebra.

7.3 Further work

The current formalisation is still quite complicated and we hope to find ways to simplify it. One interesting idea may be to use the syntactical approach to define opetopes based on dependent polynomial functors (i.e. indexed containers) [13], which has a very type-theoretic flavour.

It remains to prove that the definition proposed in this paper is a sensible one. This seems to be most easily done by showing that any interpretation of the syntax defines an algebra for Batanin's universal contractible operad.

We would like to use our framework to provide a formalisation of a variation of the results in [17, 8] by showing that Id^{ω} is a weak ω-groupoid. Such a formalisation would be different form their results because we are working inside Type Theory, rather than on a meta-level.

The main challenge ahead is to formalize the notion of a ω-groupoid model of Type Theory. Once this has been done we will be able to eliminate the univalence axiom and provide a computational interpretation of this principle.

Acknowledgments

We would like to thank Peter Lumsdaine and Darin Morrison who contributed to initial attempts to formalize weak ω-groupoids in Agda. We would like to acknowledge interesting and useful discussions on topics related to this paper with Steve Awodey, Thierry Coquand, Robert Harper, Kris Kapulkin, Nicolai Krauss, Dan Licata, Thomas Streicher and Vladimir Voevodsky. We would also like to thank the annonymous referees for providing valuable comments and indeed pointing out a shortcoming with our previous definition of coherence.

References

1 Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Symposium on Logic in Computer Science*, pages 412 – 420, 1999.

2 Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV '07: Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68, New York, NY, USA, 2007. ACM.

3 Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science*, volume 6859 of *Lecture Notes in Computer Science*, pages 70–84. Springer Berlin / Heidelberg, 2011.

4 Thorsten Altenkirch and Ondřej Rypáček. A syntactical approach to weak ω-groupoids: Agda implementation. `https://github.com/txa/OmegaCats/blob/master/Syntactical/WeakOmegaCat/Core.agda`.

5 Steve Awodey. Type theory and homotopy. arXiv:1010.1810v1.

6 J. Baez and J. Dolan. Higher-dimensional algebra iii. n-categories and the algebra of opetopes. *Advances in Mathematics*, 135(2):145–206, 1998.

7 Michael Batanin. Monoidal globular categories as natural environment for the theory of weak n-categories. *Advances in Mathematics*, 136:39–103, 1998.

8 Benno Van Den Berg and Richard Garner. Types are weak ω-groupoids, 2008.

9 James Chapman, Pierre-Évariste Dagand, Conor McBride, and Peter Morris. The gentle art of levitation. *SIGPLAN Not.*, 45:3–14, September 2010.

10 Eugenia Cheng and Aaron Lauda. Higher-dimensional categories: an illustrated guide book.

11 Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.

12 G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *Lecture Notes in Mathematics*. Cambridge University Press, 1982.

13 Joachim Kock, André Joyal, Michael Batanin, and Jean-François Mascari. Polynomial functors and opetopes. *Advances in Mathematics*, 224(6):2690 – 2737, 2010.

14 T. Leinster. A survey of definitions of n-category. *Th. Appl. Cat. 10*, 10:1–70, 2002.

15 Tom Leinster. *Operads in higher-dimensional category theory.* PhD thesis, University of Cambridge, Cambridge, 2000.

16 Daniel R. Licata and Robert Harper. 2-dimensional directed type theory. *Electr. Notes Theor. Comput. Sci.*, 276:263–289, 2011.

17 Peter Lefanu Lumsdaine. Weak ω-categories from intensional type theory. *Logical Methods in Computer Science*, 6:1–19, 2010.

18 Saunders MacLane. *Categories for the Working Mathematician.* Springer.

19 Ulf Norell. *Towards a practical programming language based on dependent type theory.* PhD thesis, Chalmers University of Technology, 2007.

20 Jacques Penon. Approche polygraphique des ∞-categories non strictes. *Cahiers de Topologie et Géométrie Différentielle*, 40(1):31–80, 1999.

21 Ross Street. The algebra of oriented simplexes. *Journal of Pure and Applied Algebra*, 49(3):283 – 335, 1987.

22 Todd Trimble. What are 'fundamental n-groupoids'? Cambridge, August 1999. seminar at DPMMS.

23 Vladimir Voevodsky. Univalent foundations of mathematics. In Lev Beklemishev and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, volume 6642 of *Lecture Notes in Computer Science*, pages 4–4. Springer Berlin / Heidelberg, 2011.

Interactive Realizability for Classical Peano Arithmetic with Skolem Axioms

Federico Aschieri

Laboratoire PPS, équipe PI.R2, Université Paris 7, INRIA & CNRS

Abstract

Interactive realizability is a computational semantics of classical Arithmetic. It is based on interactive learning and was originally designed to interpret excluded middle and Skolem axioms for simple existential formulas. A realizer represents a proof/construction depending on some state, which is an approximation of some Skolem functions. The realizer interacts with the environment, which may provide a counter-proof, a counterexample invalidating the current construction of the realizer. But the realizer is always able to turn such a negative outcome into a positive information, which consists in some new piece of knowledge learned about the mentioned Skolem functions. The aim of this work is to extend Interactive realizability to a system which includes classical first-order Peano Arithmetic with Skolem axioms. For witness extraction, the learning capabilities of realizers will be exploited according to the paradigm of learning by levels. In particular, realizers of atomic formulas will be update procedures in the sense of Avigad and thus will be understood as stratified-learning algorithms.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Interactive realizability, learning, classical Arithmetic, witness extraction

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.31

1 Introduction

Do classical proofs have some constructive content? If yes, what is a construction in classical logic? On a first thought one is inclined to think that these questions cannot have any interesting answers in terms of effective computer programs.

Surely, a classical proof is a *mental construction*, for it is a succession of constructive steps interleaved with some ineffective considerations, which however appear to have a clear mental constructive significance. Indeed, when we use the excluded middle, we can clearly picture ourselves ideally deciding whether in a situation something holds or does not. When we use an axiom of comprehension, we employ a definite law in order to construct in our minds a perfectly determined collection of elements. When we use the axiom of choice, we may imagine ourselves to make arbitrary choices as long as it is needed.

Even if a classical proof seems a legitimate mental construction, it is still a long way to yield some effective computer program. Nevertheless, from the beginning of proof theory many results have been obtained in that direction, which clearly showed that classical proofs have a constructive content. Of course, we refer to the seminal results obtained by Hilbert's epsilon substitution method (see e.g [18]) and Gentzen's cut elimination [14]. Then, several other techniques have been introduced: among them, Gödel's double negation translation followed either by the Gödel functional interpretation [13] or Kreisel's modified realizability [16] and Friedman's translation [12]; finally, Curry-Howard correspondence (see e.g. [19]).

The Curry-Howard correspondence, first introduced for intuitionistic logic and finally extended to higher-order classical logic [17], clearly shows that a classical proof not only is a mental construction but has the very same syntactic structure of a program. In other words, a proof is an *effective program.* Thus the problem of explaining what is a construction in classical logic acquires a perfectly sound mathematical sense.

Though all these constructive interpretations may seem very different from each other, a deeper study shows that they are all based on the same concept: learning. As suggestively showed by Coquand [11], a classical proof yields a learning strategy in some class of games in which players can erase their moves and backtrack to earlier positions of the game.

Now, the most important problem to solve in order to *understand* and *implement efficiently* the constructive content of classical proofs, is to provide an accurate description of the nature and the structure of the knowledge that a program extracted from a classical proof gathers during its execution. Only after a precise description of learning has been completed, one can think of how to build efficient programs. This remains a complex task, but it is of central importance to start by defining a semantics for classical proofs explicitly based on learning.

In order to provide answers to those issues, Interactive realizability [1, 4, 6] has been developed: it is a realizability semantics based on states of knowledge and learning, designed for a system of Heyting Arithmetic with excluded middle and choice principles (Skolem axioms) restricted to Σ^0_1-formulas. In hindsight, it can be seen as modern evolution of the epsilon substitution method, refined and rebuilt around the Curry-Howard correspondence for classical logic and game semantics. The aim of this paper is to take the theory of Interactive realizability to the next level: we extend it in order to interpret a full classical system, which includes first-order Peano Arithmetic with Skolem axioms, that is, excluded middle, comprehension and choice over all first-order formulas.

The theory of Interactive realizability makes precise the intuitive considerations that we have made above. In particular, it explains how to interpret a classical proof, how to pass from the ideal *mental construction* it represents to a concrete *effective construction* (a program). The main concepts are indeed the following:

- $\mathcal{T}_{\text{Class}}$ *and mental constructions.* An interactive realizer is in the first place a term of $\mathcal{T}_{\text{Class}}$, a version of Gödel's system T enriched with Skolem function symbols for every arithmetical formulas. The terms of $\mathcal{T}_{\text{Class}}$ represent the mental constructions that one can obtain directly and intuitively from a classical proof.
- *States of Knowledge as Approximations.* Terms of $\mathcal{T}_{\text{Class}}$ are ineffective and, let to themselves, useless. Therefore, interactive realizers are always computed with respect to *states*, i.e. *approximations* of the Skolem functions they contain and thus effectiveness is recovered.
- *Learning.* Since finite approximations may be inadequate, results of computations may be wrong. But an interactive realizer is also a *self-correcting* program, able to spot incorrect values of the approximations used during computations and to correct them with *right* values. The new values are *learned*, remarkably, by realizers of classical principles and all the oracle values needed during each particular computation are acquired through learning processes. Here is the fundamental insight: classical principles may be computationally interpreted as learning devices.

All these ideas work very smoothly for the most simple instance of the excluded middle

$$\mathsf{EM}_1 := \forall x^{\mathtt{N}}.\ \exists y^{\mathtt{N}} P(x,y) \vee \forall y^{\mathtt{N}} \neg P(x,y)\ \ (P \text{ computable atomic predicate})$$

A realizer of this principle uses some approximation s of a Skolem function Φ for P in order to decide which side of EM_1 is true. If for some particular n, $P(n, s(n))$ holds, then the realizer has a witness for the left side; otherwise, it declares the right side to be true. However, if the environment asks the realizer for a construction of $\neg P(n,m)$ and actually m is a counterexample to its belief (i.e. $P(n,m)$ is true), then the realizer *corrects* the approximation s as to output m on input n. We observe that this correction is sound on *absolute* grounds: m is a correct value for Φ on argument n.

In the case of a general instance of the excluded middle

$$\mathsf{EM} := \forall x^{\mathtt{N}}.\ \exists y^{\mathtt{N}} A(x,y) \vee \forall y^{\mathtt{N}} \neg A(x,y)\ \ (A \text{ first-order formula})$$

one stumbles across a serious problem: even if one has an approximation s of a Skolem function Φ for $A(x,y)$, how to compute whether $A(n, s(n))$ holds? In general, one cannot know for sure the truth value of a complex formula. A realizer thus may assume that $s(n)$ is not a witness and declare the right side to be true. However, a problem remains because learning is going to be by counterexamples: how to test whether an m given by a computational environment is such that $A(n,m)$ is true? The solution is to "compute" the truth value of $A(n,m)$ by using the approximations of the Skolem functions for the sub-formulas of A: for eliminating quantifiers, it suffices to use the equivalences $\exists y\, B(x,y) \equiv B(x, \psi(x))$, where ψ is a Skolem function for B. Since these approximations refer to Skolem functions for formulas of lower logical complexity than that of A, a realizer can determine on *relatively* firm grounds whether m is a correct value for Φ on argument n.

An important concept that we shall introduce is therefore that of *truth value of a formula in a state* and we shall study how it relates to realizability. Another novelty is that the self-corrections of realizers will not be absolute, but relative, and learning will be by *levels*. By this we mean that whenever a realizer gains some knowledge concerning a Skolem function for some formula, it may falsify a knowledge about another Skolem function for a formula of higher complexity than the first. In this case, one is forced to remove the falsified knowledge and all its consequences of greater level from the current state. The termination of this add-and-remove process is not at all evident, yet it may be proved by well-established techniques: we shall be able to see that a realizer of an atomic formula is an update procedure in the sense of Avigad [8], and that will be enough for witness extraction. As in [4], one can see Interactive realizability as new use of Friedman's translation and modified realizability, that allows to extract update procedures from classical proofs without transforming them in quantifier-free form, as in the epsilon substitution method or Herbrand analysis.

Plan of the Paper. In section §2 we introduce the term calculus $\mathcal{T}_{\text{Class}}$ in which Interactive, learning-based realizers are written, namely an extension of Gödel's system T plus Skolem function symbols for a denumerable collection of Skolem functions. In section §3, we extend Interactive realizability, as described in [1, 4], to $\mathsf{HA}^{\omega} + \mathsf{EM} + \mathsf{SK}$, an arithmetical system with functional variables which includes first-order classical Peano Arithmetic and Skolem axioms. In section §4 we show how to perform witness extraction with Interactive realizability. Full proofs of all results of this paper may be found in [7].

Acknowledgments. We thank Stefano Berardi for valuable comments and suggestions.

2 The Term Calculus $\mathcal{T}_{\text{Class}}$

In this section we follow the approach of [1, 4] and describe the typed lambda calculi $\mathcal{T}$ and $\mathcal{T}_{\text{Class}}$ in which interactive realizers are written. $\mathcal{T}$ is an extension of Gödel's system T (see Girard [15]) with some syntactic sugar. The basic objects of $\mathcal{T}$ are numerals, booleans, and its basic computational constructs are primitive recursion at all types, if-then-else, pairs, as in Gödel's T. $\mathcal{T}$ also includes as basic objects finite partial functions over $\mathbb{N}$ and simple primitive recursive operations over them. $\mathcal{T}_{\text{Class}}$ is obtained from $\mathcal{T}$ by adding on top of it a collection of Skolem function symbols $\Phi_0, \Phi_1, \Phi_2 \ldots$, of type $\mathtt{N} \rightarrow \mathtt{N}$, one for each arithmetical formula. The symbols are inert from the computational point of view and realizers are always computed with respect to some approximation of the Skolem maps represented by $\Phi_0, \Phi_1, \ldots$.

2.1 Updates

In order to define $\mathcal{T}$, we start by introducing the concept of "update", which is nothing but a finite function over $\mathbb{N}$ (i.e. a map over $\mathbb{N}$ with finite domain). Realizers of atomic formulas will return these finite functions, or "updates", as new pieces of information that they have learned about the Skolem function $\Phi_0, \Phi_1, \ldots$. Skolem functions, in turn, are used as "oracles" during computations in the system $\mathcal{T}_{\text{Class}}$. Updates are new associations input-output that are intended to correct, and in this sense, to *update*, wrong oracle values used in a computation.

▶ **Definition 1** (Updates and Consistent Union)**.** We define:

1. An update set U, shortly an *update*, is a finite set of triples of natural numbers representing a finite function from $\mathbb{N}^2$ to $\mathbb{N}$.
2. Two triples (a, n, m) and (a', n', m') of numbers are *consistent* if $a = a'$ and $n = n'$ implies $m = m'$. Two updates U_1, U_2 are consistent if $U_1 \cup U_2$ is an update.
3. $\mathbb{U}$ is the set of all updates.
4. The *consistent union* $U_1 \,\mathcal{U}\, U_2$ of $U_1, U_2 \in \mathbb{U}$ is $U_1 \cup U_2$ minus all triples of U_2 which are inconsistent with some triple of U_1.

The consistent union $U_1 \,\mathcal{U}\, U_2$ is an non-commutative operation: whenever a triple of U_1 and a triple of U_2 are inconsistent, we arbitrarily keep the triple of U_1 and we reject the triple of U_2, therefore for some U_1, U_2 we have $U_1 \,\mathcal{U}\, U_2 \neq U_2 \,\mathcal{U}\, U_1$. The operator $\mathcal{U}$ represents a way of selecting a consistent subset of $U_1 \cup U_2$, such that $U_1 \mathcal{U} U_2 = \emptyset \implies U_1 = U_2 = \emptyset$. Any operator $\mathcal{U}$ with that property would produce an alternative Realizability Semantics.

2.2 The System $\mathcal{T}$

$\mathcal{T}$ is formally described in figure 1. Terms of the form $\mathsf{if}_A\, t_1\, t_2\, t_3$ will be written in the more legible form $\mathsf{if}\ t_1\ \mathsf{then}\ t_2\ \mathsf{else}\ t_3$. A *numeral* is a term of the form $\mathsf{S}(\ldots \mathsf{S}(0) \ldots)$. For every update $U \in \mathbb{U}$, there is in $\mathcal{T}$ a constant $\overline{U} : \mathtt{U}$, where $\mathtt{U}$ is a new base type representing $\mathbb{U}$. We write $\varnothing$ for $\overline{\emptyset}$. In $\mathcal{T}$, there are four operations involving updates (see figure 1):

1. The first operation is denoted by the constant $\mathsf{min} : \mathtt{U} \to \mathtt{N}$. min takes as argument an update constant $\overline{U}$; it returns the minimum numeral a such that $(a, n, m) \in U$ for some $n, m \in \mathtt{N}$, if any exists; it returns 0 otherwise.
2. The second operation is denoted by the constant $\mathsf{get} : \mathtt{U} \to \mathtt{N}^3 \to \mathtt{N}$. get takes as arguments an update constant $\overline{U}$ and three numerals a, n, l; it returns m if $(a, n, m) \in U$ for some $m \in \mathtt{N}$ (i.e. if (a, n) belongs to the domain of the partial function U); it returns l otherwise.
3. The third operation is denoted by the constant $\mathsf{mkupd} : \mathtt{N}^3 \to \mathtt{U}$. mkupd takes as arguments three numerals a, n, m and transforms them into (the constant coding in $\mathcal{T}$) the update $\{(a, n, m)\}$.
4. The fourth operation is denoted by the constant $\Cup : \mathtt{U}^2 \to \mathtt{U}$. $\Cup$ takes as arguments two update constants and returns the update constant denoting their consistent union.

We observe that the constants $\mathsf{min}, \mathsf{get}, \mathsf{mkupd}$ are just syntactic sugar and may be avoided by coding finite partial functions into natural numbers. System $\mathcal{T}$ may thus be coded in Gödel's T. As proved in [1, 4], $\mathcal{T}$ is strongly normalizing, has the uniqueness-of-normal-form property and the following normal form theorem also holds.

▶ **Lemma 2** (Normal Form Property for $\mathcal{T} + C + \mathcal{R}$)**.** *Assume that $\mathcal{R}$ is a functional set of reduction rules for C. Assume A is either an atomic type or a product type. Then any closed normal term $t \in \mathcal{T}$ of type A is: a numeral $n : \mathtt{N}$, or a boolean $\mathtt{True}, \mathtt{False} : \mathtt{Bool}$, or an update constant $\overline{U} : \mathtt{U}$, or a constant of type A, or a pair $\langle u, v \rangle : B \times C$.*

Types

$$\sigma, \tau ::= \mathtt{N} \mid \mathtt{Bool} \mid \mathtt{U} \mid \sigma \to \tau \mid \sigma \times \tau$$

Constants

$$c ::= \mathsf{R}_\tau \mid \mathsf{if}_\tau \mid 0 \mid \mathsf{S} \mid \mathtt{True} \mid \mathtt{False} \mid \mathsf{min} \mid \mathsf{get} \mid \mathsf{mkupd} \mid \Cup \mid \overline{U}\ (\forall U \in \mathbb{U})$$

Terms

$$t, u ::= \ c \mid x^\tau \mid tu \mid \lambda x^\tau u \mid \langle t, u\rangle \mid \pi_0 u \mid \pi_1 u$$

Typing Rules for Variables and Constants

$$x^\tau : \ \tau \mid 0 : \ \mathtt{N} \mid \mathsf{S} : \ \mathtt{N} \to \mathtt{N} \mid \mathtt{True} : \ \mathtt{Bool} \mid \mathtt{False} : \ \mathtt{Bool} \mid \overline{U} : \ \mathtt{U}\ (\text{for every } U \in \mathbb{U}) \mid \Cup : \ \mathtt{U} \to \mathtt{U} \to \mathtt{U}$$

$$\mid \mathsf{min} : \ \mathtt{U} \to \mathtt{N} \mid \mathsf{get} : \ \mathtt{U} \to \mathtt{N} \to \mathtt{N} \to \mathtt{N} \to \mathtt{N} \mid \mathsf{mkupd} : \ \mathtt{N} \to \mathtt{N} \to \mathtt{N} \to \mathtt{U}$$

$$\mid \mathsf{if}_\tau : \ \mathtt{Bool} \to \tau \to \tau \to \tau \mid \mathsf{R}_\tau : \ \tau \to (\mathtt{N} \to (\tau \to \tau)) \to \mathtt{N} \to \tau$$

Typing Rules for Composed Terms

$$\frac{t : \sigma \to \tau \qquad u : \sigma}{tu : \tau} \qquad \frac{u : \tau}{\lambda x^\sigma u : \sigma \to \tau} \qquad \frac{u : \sigma \qquad t : \tau}{\langle u, t\rangle : \sigma \times \tau} \qquad \frac{u : \tau_0 \times \tau_1}{\pi_i u : \tau_i}\ i \in \{0, 1\}$$

Reduction Rules All the usual reduction rules for simply typed lambda calculus (see Girard [15]) plus the rules for recursion, if-then-else and projections

$$\mathsf{R}_\tau uv0 \mapsto u \quad \mathsf{R}_\tau uv\mathsf{S}(t) \mapsto vt(\mathsf{R}_\tau uvt) \qquad \mathsf{if}_\tau\, \mathtt{True}\, u\, v \mapsto u \quad \mathsf{if}_\tau\, \mathtt{False}\, u\, v \mapsto v \qquad \pi_i\langle u_0, u_1\rangle \mapsto u_i, i = 0, 1$$

plus the following ones, assuming a, n, m, l be numerals:

$$\mathsf{min}\,\overline{U} \mapsto \begin{cases} a & \text{if } \exists m, n.\ (a, n, m) \in U \wedge \forall (b, i, j) \in U.\ a \leq b \\ 0 & \text{otherwise} \end{cases} \qquad \overline{U}_1 \Cup \overline{U}_2 \mapsto \overline{U_1 \mathcal{U} U_2}$$

$$\mathsf{get}\,\overline{U}\, a\, n\, l \mapsto \begin{cases} m & \text{if } \exists m.\ (a, n, m) \in U \\ l & \text{otherwise} \end{cases} \qquad \mathsf{mkupd}\, a\, n\, m \mapsto \overline{\{(a, n, m)\}}$$

Figure 1 The extension $\mathcal{T}$ of Gödel's system T.

2.3 The System $\mathcal{T}_{\mathsf{Class}}$

We now define a classical extension of $\mathcal{T}$, that we call $\mathcal{T}_{\mathrm{Class}}$, with a Skolem function symbol for each arithmetical formula. The elements of $\mathcal{T}_{\mathrm{Class}}$ will represent (non-computable) realizers.

▶ **Definition 3** (The System $\mathcal{T}_{\mathsf{Class}}$)**.** Define $\mathcal{T}_{\mathrm{Class}} = \mathcal{T} + \mathcal{SC}$, where $\mathcal{SC}$ is a countable set of Skolem function constants, each one of type $\mathtt{N} \to \mathtt{N}$. We assume to have an enumeration $\Phi_0, \Phi_1, \Phi_2, \ldots$ of all the constants in $\mathcal{SC}$ (while generic elements of $\mathcal{SC}$ will be denoted with letters $\Phi, \Psi, \sigma, \tau, \ldots$).

Every $\Phi \in \mathcal{SC}$ represents a *Skolem function* for some arithmetical formula $\exists y^{\mathtt{N}}\, A(x, y)$, taking as argument a number x and returning some y such that $A(x, y)$ is true if any exists, and an arbitrary value otherwise. In general, there is no set of computable reduction rules for the constants in $\mathcal{SC}$, and therefore no set of computable reduction rules for $\mathcal{T}_{\mathrm{Class}}$. Each (in general, non-computable) term $t \in \mathcal{T}_{\mathrm{Class}}$ is associated to a set $\{t[s]\ | s \in \mathcal{T}, s : \mathtt{N}^2 \to \mathtt{N}\} \subseteq \mathcal{T}$ of computable terms we call its "approximations", one for each term $s : \mathtt{N}^2 \to \mathtt{N}$ of $\mathcal{T}$, which is thought as a sequence $s_0, s_1, s_2, \ldots$ of computable approximations of the oracles $\Phi_0, \Phi_1, \Phi_2, \ldots$ (with s_i we denote $s(i)$).

▶ **Definition 4** (Approximation at state s)**.** We define:

1. A *state* is a closed term of type $\mathtt{N}^2 \to \mathtt{N}$ of $\mathcal{T}$. If i is a numeral, with s_i we denote $s(i)$.
2. Assume $t \in \mathcal{T}_{\mathrm{Class}}$ and s is a state. The "approximation of t at state s" is the term $t[s]$ of $\mathcal{T}$ obtained from t by replacing each constant Φ_i with s_i.

3 An Interactive Learning-Based Notion of Realizability for $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$

In this section we introduce a learning-based notion of realizability for $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$, Heyting Arithmetic in all finite types (see e.g. Troelstra [20]) plus Excluded Middle and Skolem axiom schemes for all arithmetical formulas. Then we prove our main Theorem, the Adequacy Theorem: *"if a closed formula is provable in* $\mathsf{HA}^\omega+\mathsf{EM}+\mathsf{SK}$*, then it is realizable"*.

We first define the formal system $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$. We represent atomic predicates of HA^ω+EM+SK with closed terms of $\mathcal{T}_{\text{Class}}$ of type $\mathtt{Bool}$. Terms of HA^ω+EM+SK are elements of $\mathcal{T}_{\text{Class}}$ and thus may include the function symbols in $\mathcal{SC}$. We assume having in Gödel's T some terms $\Rightarrow_{\mathtt{Bool}}\colon \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool}, \neg_{\mathtt{Bool}} : \mathtt{Bool} \to \mathtt{Bool}, \vee_{\mathtt{Bool}} : \mathtt{Bool} \to \mathtt{Bool} \to \mathtt{Bool} \ldots$, implementing boolean connectives. As usual, we shall use infix notation: for example, we write $t_1 \Rightarrow_{\mathtt{Bool}} t_2$ in place of $\Rightarrow_{\mathtt{Bool}} t_1 t_2$ and similarly for the other connectives.

3.1 Language of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$

We now define the language of the arithmetical theory $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$.

▶ **Definition 5** (Language of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$). The language $\mathcal{L}_{\text{Class}}$ of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$ is defined as follows.

1. The terms of $\mathcal{L}_{\text{Class}}$ are all $t \in \mathcal{T}_{\text{Class}}$.
2. The atomic formulas of $\mathcal{L}_{\text{Class}}$ are all $Q \in \mathcal{T}_{\text{Class}}$ such that $Q : \mathtt{Bool}$.
3. The formulas of $\mathcal{L}_{\text{Class}}$ are built from atomic formulas of $\mathcal{L}_{\text{Class}}$ by the connectives $\vee$, $\wedge$, $\to$, $\setminus$, $\forall$, $\exists$ as usual, with quantifiers possibly ranging over variables $x^\tau, y^\tau, z^\tau, \ldots$ of arbitrary finite type τ of $\mathcal{T}_{\text{Class}}$.
4. A formula of $\mathcal{L}_{\text{Class}}$ is said *arithmetical* if it does not contain constants in $\mathcal{SC}$ and all its quantifiers range over the type $\mathtt{N}$, i.e. it has one of the following forms: $\forall x^{\mathtt{N}} A$, $\exists x^{\mathtt{N}} A$, $A \vee B$, $A \wedge B$, $A \to B$, $A \setminus B$, P, with A, B arithmetical and P atomic formula of $\mathcal{T}$.

We denote with $\perp$ the atomic formula $\mathtt{False}$ and with $\neg A$ the formula $A \to \perp$. The connective $\setminus$ is the dual of implication as in bi-intuitionistic logic and means "A and the opposite of B". If F is a formula of $\mathcal{L}_{\text{Class}}$ in the free variables $x_1^{\tau_1}, \ldots, x_n^{\tau_n}$ and $t_1 : \tau_1, \ldots, t_n : \tau_n$ are terms of $\mathcal{L}_{\text{Class}}$, with $F(t_1, \ldots, t_n)$ we shall denote the formula $F[t_1/x_1, \ldots, t_n/x_n]$. Sequences of variable $x_1^{\mathtt{N}}, \ldots, x_k^{\mathtt{N}}$ will be written as $\vec{x}$. We denote with $\langle \vec{x} \rangle$ a term of $\mathcal{T}$ in the free numeric variables $\vec{x}$ representing a injection of $\mathtt{N}^k$ into $\mathtt{N}$. Moreover, for every sequence of numerals $\vec{n} = n_1, \ldots, n_k$, we define $\langle \vec{n} \rangle := \langle \vec{x} \rangle[\vec{n}/\vec{x}]$ and assume that the function $\vec{n} \mapsto \langle \vec{n} \rangle$ is a bijection.

The *Excluded Middle axiom scheme* EM is defined as the set of all formulas of the form:

$$\forall \vec{x}^{\mathtt{N}}.\ A(\vec{x}) \vee \neg A(\vec{x})$$

where A is an arithmetical formula.

The *Skolem axiom scheme* SK contains for each arithmetical formula $A(\vec{x}, y)$ an axiom:

$$\forall \vec{x}^{\mathtt{N}}.\ \exists y^{\mathtt{N}} A(\vec{x}, y) \to A(\vec{x}, \Phi\langle \vec{x} \rangle)$$

with $\Phi \in \mathcal{SC}$. We assume that for every $\Phi \in \mathcal{SC}$ there is in SK one and only one formula in which Φ occurs. Such unique formula A is said to be the *formula associated to* Φ and Φ will be sometimes written as Φ_A. If s is a state and $\Phi_i = \Phi_A$, with s_A we denote s_i and with $\mathsf{mkupd}\, A\, u\, t$ we denote $\mathsf{mkupd}\, i\, u\, t$. With $\mathsf{lev}(\Phi)$ we denote the number measuring the logical complexity of the formula A associated to Φ, i.e. the number of quantifiers occurring in A.

If s, s' are two states and n a numeral, we write $s \equiv s'\ \mathsf{lev}(n)$ if for every A of complexity strictly less than n, one has $s_A(m) = s'_A(m)$ for all numerals m.

For each formula F of $\mathcal{L}_{\text{Class}}$, its involutive negation $F^\perp$ is defined by induction on F. First, we say that an atomic formula P is positive if it is of the form $\neg_{\texttt{Bool}} \dots \neg_{\texttt{Bool}} Q$, Q is not of the form $\neg_{\texttt{Bool}} R$, and the number of $\neg_{\texttt{Bool}}$ in front of Q is even. Then we define:

$$\begin{aligned}
(\neg_{\texttt{Bool}} P)^\perp &= P \text{ (if } P \text{ positive)} & P^\perp &= \neg_{\texttt{Bool}} P \text{ (if } P \text{ positive)}\\
(A \wedge B)^\perp &= A^\perp \vee B^\perp & (A \vee B)^\perp &= A^\perp \wedge B^\perp\\
(A \rightarrow B)^\perp &= A \setminus B & (A \setminus B)^\perp &= A \rightarrow B\\
(\forall x^\tau A)^\perp &= \exists x^\tau A^\perp & (\exists x^\tau A)^\perp &= \forall x^\tau A^\perp
\end{aligned}$$

As usual, one has $(F^\perp)^\perp = F$.

3.2 Truth Value of a Formula in a State

The axioms of the system $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$ give a great computational power to the system $\mathcal{T}_{\text{Class}}$: one can "compute" by a term χ_F of $\mathcal{T}_{\text{Class}}$ the truth value of any arithmetical formula F. When one effectively evaluates χ_F in a particular state s, we say that one computes *the truth value of a formula F in a state s*.

▶ **Definition 6** (Truth Value of a Formula F in a State s)**.** For every arithmetical formula $F(\vec{x})$ of $\mathcal{L}_{\text{Class}}$ we define, by induction on F, a term $\chi_F : \texttt{Bool}$ of system $\mathcal{T}_{\text{Class}}$, with the same free variables of F:

$$\chi_P = P,\ \ P \text{ atomic}$$

$$\begin{aligned}
\chi_{A\vee B} &= \chi_A \vee_{\texttt{Bool}} \chi_B & \chi_{\forall y^{\mathbb{N}} A} &= \chi_A[\Phi_{A^\perp}\langle \vec{x}\rangle/y] & \chi_{A\setminus B} &= \chi_A \wedge_{\texttt{Bool}} \chi_{B^\perp}\\
\chi_{A\wedge B} &= \chi_A \wedge_{\texttt{Bool}} \chi_B & \chi_{\exists y^{\mathbb{N}} A} &= \chi_A[\Phi_{A}\langle \vec{x}\rangle/y] & \chi_{A\rightarrow B} &= \chi_A \Rightarrow_{\texttt{Bool}} \chi_B
\end{aligned}$$

We define $F^s := \chi_F[s]$ and call it *the truth value of F in the state s.*

Intuitively, if $F(\vec{n})$ is a closed formula, our intended interpretation is:

1. $\chi_F(\vec{n})$ is a term of $\mathcal{T}_{\text{Class}}$ *denoting*, in any standard model of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$, the truth value of $F(\vec{n})$.
2. $F^s(\vec{n})$ is a term of $\mathcal{T}$ *computing what would be* the truth value of $F(\vec{n})$ in some standard model of $\mathsf{HA}^\omega + \mathsf{EM}$ *under the (possibly false) assumption* that the interpretation $\Phi_i \mapsto s_i$ satisfies the axioms of SK.

We remark that thus $F^s(\vec{n})$ is only a *conditional* truth value: if $F^s(\vec{n})$ is not the correct truth value of $F(\vec{n})$ – it may well happen – then the interpretation $\Phi_i \mapsto s_i$ does not satisfy the axioms of SK. This subtle point is what makes possible learning in Interactive realizability: whenever a contradiction follows, realizers are able to effectively find counterexamples to the assertion that the interpretation $\Phi_i \mapsto s_i$ satisfies the axioms of SK. We also observe that this way of computing the truth of a formula comes from the epsilon substitution method (see Avigad [8], Mints et al. [18]).

The notion of truth in a state behaves as expected with respect to involutive negation.

▶ **Proposition 7** (Truth in a State and Truth)**.** *For every arithmetical formula $F(\vec{x})$, state s and sequence of numerals $\vec{n}$, $F^s(\vec{n}) = \texttt{False} \iff (F^\perp)^s(\vec{n}) = \texttt{True}$*

We now prove a fundamental fact: the truth of a formula F in a state s is determined by the approximations that s gives to the Skolem functions of strictly lower level than the logical complexity of F.

▶ **Proposition 8.** *Let $F(\vec{x})$ be any arithmetical formula of logical complexity m, $\vec{n}$ be a sequence of numerals and s, s' be states such that $s \equiv s'$ $\mathsf{lev}(m)$. Then $F^s(\vec{n}) = F^{s'}(\vec{n})$.*

Every state s is considered as an *approximation* of the Skolem functions denoted by the constants of $\mathcal{SC}$: for each formula A, s_A may be a correct approximation of Φ_A on some arguments, but wrong on other ones. More precisely, if $\Phi_i = \Phi_A$, we are going to consider the set of $(i, \langle\vec{n}\rangle)$ such that $A^s(\vec{n}, s_A\langle\vec{n}\rangle) = \mathtt{True}$ as the real "domain" of s, representing the set of arguments at which s_A is surely a correct approximation of Φ_A, in the sense that s_A returns an appropriate witness (but observe that the truth of A is approximated in s).

▶ **Definition 9** (Sound Updates, Domains). We define:

1. Given an update U and a state s, we define $\mathsf{dom}_s(U)$ as the set of pairs of numerals $(i, \langle\vec{n}\rangle)$ such that $A(\vec{x}, y)$ is the formula associated to Φ_i, $(i, \langle\vec{n}\rangle, m) \in U$ and $A^s(\vec{n}, m) = \mathtt{True}$. U is said to be *sound in the state* s if $(i, \langle\vec{n}\rangle, m) \in U$ implies $(i, \langle\vec{n}\rangle) \in \mathsf{dom}_s(U)$.
2. Similarly, if s is a state, we denote with $\mathsf{dom}(s)$ the set of pairs of numerals $(i, \langle\vec{n}\rangle)$ such that $A(\vec{x}, y)$ is the formula associated to Φ_i, $s_i\langle\vec{n}\rangle = m$ and $A^s(\vec{n}, m) = \mathtt{True}$.

From now onwards, for every pair of terms t_1, t_2 of system $\mathcal{T}$, we shall write $t_1 = t_2$ if they are the same term modulo the equality rules corresponding to the reduction rules of system $\mathcal{T}$ (equivalently, if they have the same normal form).

3.3 Interactive Realizability

For every formula A of $\mathcal{L}_{\text{Class}}$, we now define what type $|A|$ a realizer of A must have.

▶ **Definition 10** (Types for realizers). For each formula A of $\mathcal{L}_{\text{Class}}$ we define a type $|A|$ of $\mathcal{T}_{\text{Class}}$ by induction on A:

$$|P| = \mathtt{U}, \text{ if } P \text{ is atomic}$$

$$|A \wedge B| = |A| \times |B| \qquad |\exists x^\tau A| = \tau \times |A| \qquad |A \setminus B| = |A| \times |B^\perp|$$

$$|A \vee B| = \mathtt{Bool} \times (|A| \times |B|) \qquad |\forall x^\tau A| = \tau \to |A| \qquad |A \to B| = |A| \to |B|$$

Let now $\mathsf{p}_0 := \pi_0 : \sigma_0 \times (\sigma_1 \times \sigma_2) \to \sigma_0$, $\mathsf{p}_1 := \pi_0\pi_1 : \sigma_0 \times (\sigma_1 \times \sigma_2) \to \sigma_1$ and $\mathsf{p}_2 := \pi_1\pi_1 : \sigma_0 \times (\sigma_1 \times \sigma_2) \to \sigma_2$ be the three canonical projections from $\sigma_0 \times (\sigma_1 \times \sigma_2)$. We define the realizability relation $t \Vdash F$, where $t \in \mathcal{T}_{\text{Class}}$, $F \in \mathcal{L}_{\text{Class}}$ and $t : |F|$.

▶ **Definition 11** (Interactive Realizability). Assume s is a state, t is a closed term of $\mathcal{T}_{\text{Class}}$, $F \in \mathcal{L}_{\text{Class}}$ is a closed formula, and $t : |F|$. We define first the relation $t \Vdash_s F$ by induction and by cases according to the form of F:

1. $t \Vdash_s Q$ for some atomic Q if and only if $\overline{U} = t[s]$ implies:
   - U is sound in s and $\mathsf{dom}_s(U) \cap \mathsf{dom}(s) = \emptyset$
   - $\overline{U} = \varnothing$ implies $Q[s] = \mathtt{True}$
2. $t \Vdash_s A \wedge B$ if and only if $\pi_0 t \Vdash_s A$ and $\pi_1 t \Vdash_s B$
3. $t \Vdash_s A \vee B$ if and only if either $\mathsf{p}_0 t[s] = \mathtt{True}$ and $\mathsf{p}_1 t \Vdash_s A$, or $\mathsf{p}_0 t[s] = \mathtt{False}$ and $\mathsf{p}_2 t \Vdash_s B$
4. $t \Vdash_s A \to B$ if and only if for all u, if $u \Vdash_s A$, then $tu \Vdash_s B$
5. $t \Vdash_s A \setminus B$ if and only if $\pi_0 t \Vdash_s A$ and $\pi_1 t \vdash_s B^\perp$
6. $t \Vdash_s \forall x^\tau A$ if and only if for all closed terms $u : \tau$ of $\mathcal{T}$, $tu \Vdash_s A[u/x]$
7. $t \Vdash_s \exists x^\tau A$ if and only for some closed term $u : \tau$ of $\mathcal{T}$, $\pi_0 t[s] = u$ and $\pi_1 t \Vdash_s A[u/x]$

We define $t \Vdash F$ if and only if for all states s of $\mathcal{T}$, $t \Vdash_s F$.

The ideas behind the definition of $\Vdash_s$ in the case of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$ are those we already explained in [1, 4, 6]. A realizer is a term t of $\mathcal{T}_{\text{Class}}$, possibly containing some non-computable Skolem function of $\mathcal{SC}$; if such a function was computable, t would be an intuitionistic realizer. Since in general t is not computable, we calculate its approximation $t[s]$ at state s. t is an intelligent, self-correcting program, representing a proof/construction depending on the state s. The realizer *interacts* with the environment, which may provide a counter-proof, a counterexample invalidating the current construction of the realizer. But the realizer is always able to turn such a negative outcome into a positive information, which consists in some new piece of knowledge learned about some Skolem function Φ_i.

There are two concepts that are useful to understand the interaction of a realizer with the environment: a realizer receives as input *tests* and produces as output *predictions*.

- *Predictions.*
  - A realizer t of $A \vee B$ uses s to predict which one between A and B is realizable: if $\pi_0 t[s] = \mathtt{True}$ then A is realizable, and if $\pi_0 t[s] = \mathtt{False}$ then B is realizable.
  - A realizer u of $\exists x^\tau A$ uses s to compute $\pi_0 u[s] = w$ and to predict that w is a witness for $\exists x^\tau A$ (i.e. that $A[w/x]$ is realizable).
- *Tests.*
  - A realizer of a universal formula $\forall x^\tau A$ takes an object w as a challenge coming from the environment to provide a construction of $A[w/x]$, whose correctness will be tested at the end of computation.
  - A realizer of $A \to B$ takes a realizer of A as a challenge coming from the environment to provide a construction of B, whose correctness will be tested at the end of the computation.
  - A realizer of $A \wedge B$ may be challenged to construct A as well as B, and again the correctness of the construction will be tested at the end of computation.
  - A realizer of an atomic formula Q comes after a series of predictions and challenges that have been provided to test the construction of a complex formula; the realizer performs a final test and computes the formula Q in the state s as an experiment. Since predictions of realizers need not be always correct, it is possible that a realized atomic formula is actually false; we may have $t \Vdash_s Q$ and $Q[s] = \mathtt{False}$ in $\mathcal{T}$. If Q, though predicted to be true, is instead false, then a counterexample has been encountered; this means that the approximation s of the Skolem constants in $\mathcal{SC}$ is still inadequate. In this case, $t[s] \neq \varnothing$ by definition of $t \Vdash_s Q$. That is to say: if the construction of a realizer is wrong in a particular state, the realizer must learn from its mistakes. The point is that after every learning, the actual state can be improved with the information in $\overline{U} = t[s]$, since $(i, \langle\vec{n}\rangle) \in U$ and A is associated to Φ_i imply $A^s(\vec{n}, m) = \mathtt{True}$ and $A^s(\vec{n}, s_i\langle\vec{n}\rangle) = \mathtt{False}$.

The next proposition tells that realizability at state s respects the notion of equality of $\mathcal{T}_{\text{Class}}$ terms, when the latter is relativized to state s. That is, if two terms are equal at the state s, then they realize the same formulas in the state s.

▶ **Proposition 12** (Saturation). *If $t_1[s] = t_2[s]$ and $u_1[s] = u_2[s]$, then $t_1 \Vdash_s B[u_1/x]$ if and only if $t_2 \Vdash_s B[u_2/x]$.*

3.4 Realizability of Classical Axioms

We are now going to show how to realize EM and SK. We first need to realize the ex-falso-quodlibet axiom, for which a dummy term is enough.

▶ **Proposition 13** (Realizer of the Ex-Falso-Quodlibet Axiom). *For every formula $F(\vec{x})$ of $\mathcal{L}_{\text{Class}}$, there exists a closed term $\bot_F$ of $\mathcal{T}$ such that $\bot_F \Vvdash \bot \to F(\vec{u})$, for every sequence of closed terms $\vec{u}$ of $\mathcal{T}_{\text{Class}}$. In particular, $\bot_F$ can be defined by induction on F as follows:*

$$\begin{aligned} \bot_P &:= \lambda x^{\mathtt{U}}.\, x & \bot_{A\to B} &:= \lambda x^{\mathtt{U}} \lambda y^{|A|}.\, \bot_B x \\ \bot_{A\wedge B} &:= \lambda x^{\mathtt{U}}.\langle \bot_A x, \bot_B x\rangle & \bot_{\exists x^\tau A} &:= \lambda x^{\mathtt{U}} \langle 0^\tau, \bot_A x\rangle \\ \bot_{A\vee B} &:= \lambda x^{\mathtt{U}}.\langle \mathtt{False}, \bot_A x, \bot_B x\rangle & \bot_{\forall x^\tau A} &:= \lambda x^{\mathtt{U}} \lambda y^\tau.\bot_A x \\ \bot_{A\smallsetminus B} &:= \lambda x^{\mathtt{U}} \langle \bot_A x, \bot_{B^\perp} x\rangle \end{aligned}$$

In Interactive realizability, as we shall see many times, realizers continually interact with each other and game theory is a very useful tool to describe they behaviour (see [5]). Interestingly, for realizing Skolem axioms and some instances of EM we are lead to consider once again Tarski games. In these kind of games, there are two players and an arithmetical formula on the board; the first player – usually called Eloise – tries to show that it is true, while the second player – usually called Abelard – tries to show that the formula is false. Thus, Eloise wins when true atomic formulas are on the board while Abelard wins with false ones. In the case of formulas of the shape $A \vee B, \exists x^{\mathtt{N}} A$, Eloise moves: in the first case by choosing a side of the disjunction and in the second case by choosing a numeral as a witness for the existential quantifier. In the case of formulas of the shape $A \wedge B, \forall x^{\mathtt{N}} A$, Abelard moves: in the first case by choosing a side of the conjunction and in the second case by choosing a numeral as a counterexample to the universal quantifier. In the case of formulas of the shape $A \to B$, Abelard gives a winning strategy for A to Eloise and they play the game for B. An Eloise strategy for A is represented by a term $\mathscr{E}$ of type $|A|$, while an Abelard strategy for A is represented by a term $\mathscr{A}$ of type $|A^\perp|$. Thus one may define the Tarski game between strategies through a game operator $\star$ (which indeed resembles the operator $\star$ of symmetric lambda calculus [9]) that puts Eloise against Abelard. The result of the game is thus $\mathscr{E} \star \mathscr{A}$ and it is a term of type $|\bot| = \mathtt{U}$. If $\mathscr{E}$ and $\mathscr{A}$ happens to be interactive realizers, they represent self-correcting constructions. $\mathscr{E}$ and $\mathscr{A}$ challenge the construction of each other, but who looses the interaction is always able to partially repair its construction, i.e. to learn some information about some Skolem functions, which he puts in some update. That is to say, $\mathscr{E} \star \mathscr{A}$ realizes $\bot$. Details follow.

▶ **Proposition 14** (A Tarski Game Between Strategies). *Let F be an arithmetical formula and $\mathscr{E} : |F|$, $\mathscr{A} : |F^\perp|$ two terms of $\mathcal{T}_{\text{Class}}$. Define by induction and according to the shape of F a term $\mathscr{E} \star \mathscr{A} : |\bot|$ as follows:*

$$\begin{aligned} &(F = P, P \text{ atomic})\ \mathscr{E} \star \mathscr{A} := \mathscr{E} \Cup \mathscr{A} \\ &(F = A \to B)\ \mathscr{E} \star \mathscr{A} := \mathscr{E}(\pi_0 \mathscr{A}) \star \pi_1 \mathscr{A} && (F = A \smallsetminus B)\ \mathscr{E} \star \mathscr{A} := \mathscr{A}(\pi_0 \mathscr{E}) \star \pi_1 \mathscr{E} \\ &(F = \exists y^{\mathtt{N}} A)\ \mathscr{E} \star \mathscr{A} := \pi_1 \mathscr{E} \star \mathscr{A}(\pi_0 \mathscr{E}) && (F = \forall y^{\mathtt{N}} A)\ \mathscr{E} \star \mathscr{A} := \mathscr{E}(\pi_0 \mathscr{A}) \star \pi_1 \mathscr{A} \end{aligned}$$

$$\begin{aligned} &(F = A \wedge B)\ \mathscr{E} \star \mathscr{A} := \mathsf{if}\ \mathsf{p}_0 \mathscr{A}\ \mathsf{then}\ \pi_0 \mathscr{E} \star \mathsf{p}_1 \mathscr{A}\ \mathsf{else}\ \pi_1 \mathscr{E} \star \mathsf{p}_2 \mathscr{A} \\ &(F = A \vee B)\ \mathscr{E} \star \mathscr{A} := \mathsf{if}\ \mathsf{p}_0 \mathscr{E}\ \mathsf{then}\ \mathsf{p}_1 \mathscr{E} \star \pi_0 \mathscr{A}\ \mathsf{else}\ \mathsf{p}_2 \mathscr{E} \star \pi_1 \mathscr{A} \end{aligned}$$

Then $\mathscr{E} \Vvdash_s F \wedge \mathscr{A} \Vvdash_s F^\perp \implies \mathscr{E} \star \mathscr{A} \Vvdash_s \bot$.

We now establish two important links between the concept of truth in a state and the concept of realizability in the same state.

The first result is that if a formula is true in a state s, then it is realizable in s. Intuitively, $F^s = \mathtt{True}$ means that the state s is both: i) powerful enough to compute witnesses for all the subformulas of F and their negations which are supposed to be true if F is true; ii) sharp enough to not provide counterexamples invalidating some of those witnesses. Thus F can be realized in the state s by a realizer $\mathfrak{T}_F$ which uses s to return the mentioned witnesses and "waits" for possible counterexamples.

One could expect the converse to hold as well, namely that if F is realizable in s, then F is true in s. This is not actually true, but "almost". Indeed, the second result is that if F is realizable in s and $F^s = \texttt{False}$, one has disastrous consequences: $\bot$ is realizable in s. In fact, one can define a term $\mathfrak{F}_F$ transforming any realizer $\mathscr{E}$ of F in the state s in a realizer of $\bot$ in the same s, whenever $F^s = \texttt{False}$. Indeed, if $F^s = \texttt{False}$, then $(F^\perp)^s = \texttt{True}$ by proposition 7; therefore $\mathscr{A} := \mathfrak{T}_{F^\perp}$ realizes $F^\perp$ in s. The state s is thus used to build a counterexample $\mathscr{A}$ to F, which can be put against the realizer $\mathscr{E}$ of F in the term $\mathscr{E} \star \mathscr{A}$. This latter term realizes $\bot$ by proposition 14. Intuitively, if s is a strong enough approximation, $\mathscr{A}$ wins and thus provides a counterexample to the fact that $\mathscr{E}$ is a construction of F; since $\mathscr{E}$, as realizer, is a self-correcting program, it is able to extend the state s with new information: this information realizes $\bot$ in s. If instead s is not a good approximation, $\mathscr{E}$ wins and the capabilities of $\mathscr{A}$ are used to improve the state s with an update realizing $\bot$. Formally:

▶ **Proposition 15** (Truth and Realizability in a State). *Let $F(\vec{x})$ be an arithmetical formula. There exist two terms $\mathfrak{T}_F(\vec{x})$ and $\mathfrak{F}_F(\vec{x})$ of $\mathcal{T}_{\text{Class}}$ such that for all numerals $\vec{n}$ and state s*

$$F^s(\vec{n}) = \texttt{True} \implies \mathfrak{T}_F(\vec{n}) \Vvdash_s F(\vec{n})$$
$$F^s(\vec{n}) = \texttt{False} \implies \mathfrak{F}_F(\vec{n}) \Vvdash_s \neg F(\vec{n})$$

In particular, $\mathfrak{T}_F$ and $\mathfrak{F}_F$ can be constructed by induction on F as follows:

$$\begin{array}{rl} \mathfrak{T}_P := \varnothing,\ P \textit{ atomic} & \mathfrak{T}_{A\to B} := \lambda z^{|A|}.\, \mathsf{if}\ \chi_A\ \mathsf{then}\ \mathfrak{T}_B\ \mathsf{else}\ \bot_B(\mathfrak{F}_A z) \\ \mathfrak{T}_{A\wedge B} := \langle \mathfrak{T}_A, \mathfrak{T}_B\rangle & \mathfrak{T}_{\forall y^{\mathtt{N}} A} := \lambda y^{\mathtt{N}}.\, \mathsf{if}\ \chi_A\ \mathsf{then}\ \mathfrak{T}_A\ \mathsf{else}\ \bot_A \mathsf{mkupd}\, A^\perp \langle\vec{x}\rangle\, y \\ \mathfrak{T}_{A\vee B} := \langle \chi_A, \mathfrak{T}_A, \mathfrak{T}_B\rangle & \mathfrak{T}_{\exists y^{\mathtt{N}} A} := \langle \Phi_A\langle\vec{x}\rangle, \mathfrak{T}_A[\Phi_A\langle\vec{x}\rangle/y]\rangle \\ \mathfrak{T}_{A\diagdown B} := \langle \mathfrak{T}_A, \mathfrak{T}_{B^\perp}\rangle & \mathfrak{F}_F := \lambda x^{|F|}.\, x \star \mathfrak{T}_{F^\perp} \end{array}$$

The most remarkable feature of our realizability semantics is the existence of a realizer for any instance of EM, even if our language contains the positive symbols $\vee, \exists$ which have to be realized according to the Kreisel-style clauses of our definition of realizability.

▶ **Proposition 16** (Realizer E_A of EM). *Let $A(\vec{x})$ be any arithmetical formula. Define*

$$\mathsf{E}_A := \lambda \vec{x}^{\mathtt{N}}.\, \langle \chi_A, \mathfrak{T}_A, \mathfrak{F}_A\rangle$$

Then $\mathsf{E}_A \Vvdash \forall \vec{x}^{\mathtt{N}}.\, A(\vec{x}) \vee \neg A(\vec{x})$.

We observe that the excluded middle can very well be defined as $\forall \vec{x}^{\mathtt{N}}.\, A(\vec{x}) \vee A^\perp(\vec{x})$. In the case of Σ^0_n-formulas, one would get a simplified realizer of the form $\lambda \vec{x}^{\mathtt{N}}.\, \langle \chi_A, \mathfrak{T}_A, \mathfrak{T}_{A^\perp}\rangle$, which does not use the game operator $\star$ contained in $\mathfrak{F}_A$. In the case of Σ^0_1-formulas, one recovers as a special case exactly the realizer of [1], where Interactive realizability was first defined for $\mathsf{HA} + \mathsf{EM}_1 + \mathsf{SK}_1$. We also observe that the realizer E_A, when evaluated in any state, uses two instructions: the *read from the state* operation for satisfying the constructive clauses of realizability and the *update of the state* operation for dealing with counterexamples to universal quantifiers. The operation of updating the state is used only to interpret classical steps of proofs. We now show how to realize the Skolem axiom scheme SK.

▶ **Proposition 17** (Realizer S_A of SK). *Let $A(\vec{x}, y)$ be any arithmetical formula and Φ the Skolem function constant associated to A. Define*

$$\mathsf{S}_A := \lambda \vec{x}^{\mathtt{N}} \lambda z^{|\exists y^{\mathtt{N}} A|}.\, \mathsf{if}\ \chi_{\exists y^{\mathtt{N}} A}\ \mathsf{then}\ \mathfrak{T}_A[\Phi\langle\vec{x}\rangle/y]\ \mathsf{else}\ \bot_{A(\vec{x}, \Phi\langle\vec{x}\rangle)}(\mathfrak{F}_{\exists y^{\mathtt{N}} A} z)$$

Then $\mathsf{S}_A \Vvdash \forall \vec{x}^{\mathtt{N}}.\, \exists y^{\mathtt{N}} A(\vec{x}, y) \to A(\vec{x}, \Phi\langle\vec{x}\rangle)$.

3.5 Curry-Howard Correspondence for $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$

In figure 2, we define a standard natural deduction system for $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$ (see [19], for example) together with a term assignment in the spirit of Curry-Howard correspondence for classical logic. We replace purely universal axioms (i.e., Π^0_1-axioms) with Post rules, which

Contexts With Γ we denote contexts of the form $x_1 : A_1, \dots, x_n : A_n$, with $x_1, \dots, x_n$ proof variables and $A_1, \dots, A_n$ formulas of $\mathcal{L}_{\text{Class}}$.

Axioms $\Gamma, x : A \vdash x^{|A|} : A$

Conjunction $\dfrac{\Gamma \vdash u : A \quad \Gamma \vdash t : B}{\Gamma \vdash \langle u, t\rangle : A \wedge B}$ $\quad \dfrac{\Gamma \vdash u : A \wedge B}{\Gamma \vdash \pi_0 u : A}$ $\quad \dfrac{\Gamma \vdash u : A \wedge B}{\Gamma \vdash \pi_1 u : B}$

Subtraction $\dfrac{\Gamma \vdash u : A \quad \Gamma \vdash t : B^\perp}{\Gamma \vdash \langle u, t\rangle : A \setminus B}$ $\quad \dfrac{\Gamma \vdash u : A \setminus B}{\Gamma \vdash \pi_0 u : A}$ $\quad \dfrac{\Gamma \vdash u : A \setminus B}{\Gamma \vdash \pi_1 u : B^\perp}$

Implication $\dfrac{\Gamma \vdash u : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash ut : B}$ $\quad \dfrac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x^{|A|} u : A \to B}$

Disjunction Intro. $\dfrac{\Gamma \vdash u : A}{\Gamma \vdash \langle \mathtt{True}, u, d^{|B|}\rangle : A \vee B}$ $\quad \dfrac{\Gamma \vdash u : A}{\Gamma \vdash \langle \mathtt{False}, d^{|A|}, u\rangle : A \vee B}$

Disjunction Elim. $\dfrac{\Gamma \vdash u : A \vee B \quad \Gamma, x : A \vdash w_1 : C \quad \Gamma, x : B \vdash w_2 : C}{\Gamma \vdash \mathsf{if}\ \mathsf{p}_0 u\ \mathsf{then}\ (\lambda x^{|A|} w_1)(\mathsf{p}_1 u)\ \mathsf{else}\ (\lambda x^{|B|} w_2)(\mathsf{p}_2 u) : C}$

Universal Quantification $\dfrac{\Gamma \vdash u : \forall \alpha^\tau A}{\Gamma \vdash ut : A[t/\alpha^\tau]}$ $\dfrac{\Gamma \vdash u : A}{\Gamma \vdash \lambda \alpha^\tau u : \forall \alpha^\tau A}$

where t is a term of $\mathcal{L}_{\text{Class}}$ and $\alpha^{\mathtt{N}}$ does not occur free in any formula B occurring in Γ.

Existential Quantification $\dfrac{\Gamma \vdash u : A[t/\alpha^\tau]}{\Gamma \vdash \langle t, u\rangle : \exists \alpha^\tau . A}$ $\quad \dfrac{\Gamma \vdash u : \exists \alpha^\tau . A \quad \Gamma, x : A \vdash t : C}{\Gamma \vdash (\lambda \alpha^\tau \lambda x^{|A|}\ t)(\pi_0 u)(\pi_1 u) : C}$

where α^τ is not free in C nor in any formula B occurring in Γ.

Induction $\dfrac{\Gamma \vdash u : A(0) \quad \Gamma \vdash v : \forall \alpha^{\mathtt{N}} . A(\alpha) \to A(\mathsf{S}(\alpha))}{\Gamma \vdash \lambda \alpha^{\mathtt{N}} \mathsf{R} u v \alpha : \forall \alpha^{\mathtt{N}} A}$

Post Rules $\dfrac{\Gamma \vdash u_1 : A_1\ \Gamma \vdash u_2 : A_2\ \cdots\ \Gamma \vdash u_n : A_n}{\Gamma \vdash u_1 \Cup u_2 \Cup \cdots \Cup u_n : A}$

where $n > 0$ and $A_1, A_2, \dots, A_n, A$ are atomic formulas of $\mathcal{L}_{\text{Class}}$, and the rule is a Post rule for equality, for a Peano axiom or for a classical propositional tautology or for booleans.

Post Rules with no Premises $\dfrac{}{\Gamma \vdash \varnothing : A}$

where A is an atomic formula of $\mathcal{L}_{\text{Class}}$ and an axiom of equality or a classical propositional tautology.

EM $\dfrac{}{\Gamma \vdash \mathsf{E}_A : \forall \vec{x}^{\mathtt{N}} .\ A(\vec{x}) \vee \neg A(\vec{x})}$

SK $\dfrac{}{\Gamma \vdash \mathsf{S}_A : \forall \vec{x}^{\mathtt{N}} .\ \exists y^{\mathtt{N}} A(\vec{x}, y) \to A(\vec{x}, \Phi\langle\vec{x}\rangle)}$

Figure 2 Term Assignement Rules for $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$.

are inferences of the form

$$\frac{\Gamma \vdash A_1\ \Gamma \vdash A_2\ \cdots\ \Gamma \vdash A_n}{\Gamma \vdash A}$$

where $A_1, \dots, A_n, A$ are atomic formulas of $\mathcal{L}_{\text{Class}}$ such that for every substitution $\sigma = [t_1/x_1, \dots, t_k/x_k]$ of closed terms $t_1, \dots, t_k$ of $\mathcal{T}$ and state s, $A_1\sigma[s] = \ldots = A_n\sigma[s] = \mathtt{True}$ implies $A\sigma[s] = \mathtt{True}$. Let now $\mathsf{eq} : \mathtt{N}^2 \to \mathtt{Bool}$ be a term of Gödel's system T representing equality between natural numbers. Among the Post rules, we have the Peano axioms and axioms of equality

$$\frac{\Gamma \vdash \mathsf{eq}\ \mathsf{S}(x)\, \mathsf{S}(y)}{\Gamma \vdash \mathsf{eq}\ x\, y} \quad \frac{\Gamma \vdash \mathsf{eq}\ 0\, \mathsf{S}(x)}{\Gamma \vdash \perp} \quad \frac{}{\Gamma \vdash \mathsf{eq}\ x\, x} \quad \frac{\Gamma \vdash \mathsf{eq}\, x\, y \quad \Gamma \vdash \mathsf{eq}\, y\, z}{\Gamma \vdash \mathsf{eq}\ x\, z} \quad \frac{\Gamma \vdash A(x)\ \Gamma \vdash \mathsf{eq}\, x\, y}{\Gamma \vdash A(y)}$$

and for every A_1, A_2 such that $A_1 = A_2$ is an equation of Gödel's system T (equivalently, A_1, A_2 have the same normal form in T), we have the rule

$$\frac{\Gamma \vdash A_1}{\Gamma \vdash A_2}$$

We also add a Post rule for every classical propositional tautology $A_1 \to \ldots \to A_n \to A$, where for $i = 1, \ldots, n$, A_i, A are atomic formulas obtained as combination of other atomic formulas by the Gödel's system T boolean connectives. Finally, we have a rule of case reasoning for booleans. For any atomic formula P and any formula $A[P]$ we have:

$$\frac{\Gamma \vdash A[\mathtt{True}] \quad \Gamma \vdash A[\mathtt{False}]}{\Gamma \vdash A[P]}$$

If T is any type of $\mathcal{T}$, we denote with d^T a dummy term of type T, defined by $d^{\mathtt{N}} = 0$, $d^{\mathtt{Bool}} = \mathtt{False}$, $d^{\mathtt{U}} = \varnothing$, $d^{A \to B} = \lambda z^A . d^B$ (with z^A any variable of type A), $d^{A \times B} = \langle d^A, d^B \rangle$.

It can now be proved that every theorem of $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$ is realizable (see [7]).

▶ **Theorem 18.** *If A is a closed formula such that $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK} \vdash t : A$, then $t \Vdash A$.*

4 Witness Extraction with Interactive Realizability

In this section, we turn our attention to the witness extraction problem for Π^0_2-formulas. Given a realizer $t \Vdash \forall x^{\mathtt{N}} \exists y^{\mathtt{N}} Pxy$, where P is an atomic recursive predicate, one is asked to extract from t a non-trivial program taking as input a numeral n and yielding as output a witness for the formula $\exists y^{\mathtt{N}} Pny$ (that is, a numeral m such that $Pnm = \mathtt{True}$). In the case of Interactive realizability, the problem of computing that witness can be reduced to finding a "zero" for a suitable term u of type $\mathtt{U}$, that is a state s such that $u[s] = \varnothing$. Indeed, given any numeral n and state s, the following implications hold:

$$t \Vdash \forall x^{\mathtt{N}} \exists y^{\mathtt{N}} Pxy \implies t \Vdash_s \forall x^{\mathtt{N}} \exists y^{\mathtt{N}} Pxy \implies tn \Vdash_s \exists y^{\mathtt{N}} Pny$$

$$\implies (\pi_0(tn)[s] = m \wedge \pi_1(tn) \Vdash_s Pnm) \implies (\pi_1(tn)[s] = \varnothing \implies Pnm = \mathtt{True})$$

Therefore, if s is a zero of $\pi_1(tn)$, then $\pi_0(tn)$ is equal in the state s to some witness m of the formula $\exists y^{\mathtt{N}} Pny$. Intuitively, a zero for $\pi_1(tn)$ represents a sufficient amount of information to compute the required witness. Indeed, a zero for $\pi_1(tn)$ always exists, because $\pi_1(tn)$ represents an update procedure (see [2, 8] for investigations and explanations of the concept).

▶ **Definition 19** (Avigad's Finite Update Procedures). A $k+1$*-ary typed update procedure* $k \in \mathbb{N}$ is a term $\mathbf{U} : (\mathtt{N} \to \mathtt{N})^k \to \mathtt{U}$ of $\mathcal{T}$ such that the following holds:

1. for all sequences $f = f_0, \ldots, f_k$ of closed type-$\mathtt{N} \to \mathtt{N}$ terms of $\mathcal{T}$

$$\mathbf{U}f \neq \varnothing \implies \mathbf{U}f = \{(i, n, m)\} \wedge 0 \leq i \leq k$$

2. for all sequences $f = f_0, \ldots, f_k$ and $g = g_0, \ldots, g_k$ of closed type-$\mathtt{N} \to \mathtt{N}$ terms of $\mathcal{T}$ and for all $0 \leq i \leq k$, if
   - $\mathbf{U}f = \{(i, n, m)\}$
   - for all $j < i$, $f_j = g_j$
   - $g_i(n) = m$

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

If $\mathbf{U}$ is a $k+1$-ary update procedure, a *zero* for $\mathbf{U}$ is a sequence $f = f_0, \ldots, f_k$ of closed type-$\mathtt{N} \to \mathtt{N}$ terms of $\mathcal{T}$ such that $\mathbf{U}f = \varnothing$.

Condition (2) describes *learning by levels.* If $\mathbf{U}$ is a k-ary update procedure and f is a sequence of terms approximating the Skolem functions $\Phi_0, \ldots, \Phi_k$ (and we assume $i \leq j$, implies $\mathsf{lev}(\Phi_i) \leq \mathsf{lev}(\Phi_j)$), there are two possibilities: either f is a fine approximation and then $\mathbf{U}f = \emptyset$; or f is not and then $\mathbf{U}f = \{(i, n, m)\}$, for some numerals n, m: $\mathbf{U}$ says the term f_i should be updated as to output m on input n. Moreover, if $\mathbf{U}f = (i, n, m)$, one in a sense has *learned* at level i that $\Phi_i(n) = m$ on grounds of the values of f_j, for $j < i$. Condition (2) indicates that the information $\Phi_i(n) = m$ should be preserved, unless some information in the lower levels changes.

For technical convenience we now add to $\mathcal{T}$ (and thus to system $\mathcal{T}_{\text{Class}}$), a constant $\mathsf{ch} : \mathtt{U} \to \mathtt{U}$ which chooses exactly one element from every non-empty update and maps the empty update to itself. Therefore, we assume to have in $\mathcal{T}$ conversion rules such that for every non-empty update U

$$\mathsf{ch}\,\varnothing \mapsto \varnothing \qquad \mathsf{ch}\,\overline{U} \mapsto \{(i, n, m)\}, \text{ for some } (i, n, m) \in U$$

For simplicity, we are going to consider only *proof-like terms* of $\mathcal{T}_{\text{Class}}$: a term t is said to be proof-like if: i) every occurrence in t of the constant mkupd is of the form mkupd_i, where i is some numeral, and Φ_i occurs in t; ii) no update constant different from $\varnothing$ occurs in t. Indeed, that is the syntactic form of every interactive realizer extracted from some proof in $\mathsf{HA}^\omega + \mathsf{EM} + \mathsf{SK}$.

▶ **Proposition 20** (The Update Procedure Associated to an Atomic Realizer). *Let Q be an atomic formula of $\mathcal{L}_{\text{Class}}$ and suppose $t \Vdash Q$, with t proof-like. Let without loss of generality $\Phi_0, \ldots, \Phi_k$ the list of all Skolem function constants of t ordered by levels: if $i \leq j$, then $\mathsf{lev}(\Phi_i) \leq \mathsf{lev}(\Phi_j)$. Define*

$$\mathbf{U} := \lambda f_0^{\mathtt{N}\to\mathtt{N}} \ldots \lambda f_k^{\mathtt{N}\to\mathtt{N}}.\, \mathsf{ch}(t[f_0/\Phi_0 \ldots f_k/\Phi_k])$$

Then $\mathbf{U}$ is an update procedure.

From now on, the term $\mathbf{U}$ of proposition 20 will be called the $(k+1)$*-ary update procedure associated to t.* There is a standard way to compute a zero for any update procedure and thus for the correspondent atomic realizer. In order to do that, if $f = f_0, \ldots, f_k$ is a sequence of terms of type $\mathtt{N} \to \mathtt{N}$ and $\overline{U}$ is an update constant, we define a term $f \oplus \overline{U}$ which changes the values of f_i according to the triples $(i, n, m) \in U$, where $i = \mathsf{min}\,\overline{U}$, leaves f_j for $j < i$ unchanged and changes every f_j, with $j > i$, to be equal to the constant function $\lambda x^{\mathtt{N}} 0$.

▶ **Definition 21** (Updates of Functions). For each numeral i, we define a term $\oplus_i : (\mathtt{N} \to \mathtt{N}) \to \mathtt{U} \to (\mathtt{N} \to \mathtt{N})$ as follows:

$$\oplus_i := \lambda f^{\mathtt{N}\to\mathtt{N}} \lambda u^{\mathtt{U}} \lambda x^{\mathtt{N}} \ \mathsf{if}\ \mathsf{min}\, u > i \ \mathsf{then}\ fx \ \mathsf{else\ if}\ \mathsf{min}\, u = i \ \mathsf{then}\ (\mathsf{get}\, u\, i\, x\, fx) \ \mathsf{else}\ 0$$

We shall write $t_1 \oplus_i t_2$ in place of $\oplus_i t_1 t_2$. If $f = f_0, \ldots, f_k$ is a sequence of terms of type $\mathtt{N} \to \mathtt{N}$ and $u : \mathtt{U}$, we define $f \oplus u := f_0 \oplus_0 u, \ldots, f_k \oplus_k u$.

▶ **Theorem 22** (Zero Theorem). *Let Q be an atomic formula of $\mathcal{L}_{\text{Class}}$, suppose t is a proof-like term such that $t \Vdash Q$ and let $\mathbf{U}$ be the $(k+1)$-ary update procedure associated to t. Let s be any state. Define, by induction on n, a sequence $\{r_n\}_{n \in \mathbb{N}}$ of $k+1$-ary sequences of type-$\mathtt{N} \to \mathtt{N}$ terms as follows:*

$$r_0 := s_0, \ldots, s_k$$
$$r_{n+1} := r_n \oplus (\mathbf{U} r_n) \text{ (see definition 21)}$$

Then, there exists a n such that $t[(r_n)_0/\Phi_0 \ldots (r_n)_k/\Phi_k] = \varnothing$.

Using the results of [3, 6], we are even able to extract a program belonging to system $\mathcal{T}$.

▶ **Theorem 23** (Program Extraction via Interactive Realizability). *Let t be a term of $\mathcal{T}_{\mathit{Class}}$ and suppose that $t \Vdash \forall x^{\mathtt{N}} \exists y^{\tau} Pxy$, with $P : \mathtt{N} \rightarrow \tau \rightarrow \mathtt{Bool}$ closed term of system $\mathcal{T}$. Then:*

1. *From t one can effectively define a recursive function f such that for every numeral n, $f(n) : \tau$ is a term of system $\mathcal{T}$ such that $Pn(f(n)) = \mathtt{True}$.*
2. *f can be represented in system $\mathcal{T}$.*

Remark. We observe that our algorithm for witness extraction is not the last word on the topic, for it is not particularly optimized for real-world execution. However, thanks to our realizability interpretation, we have now achieved a sharp understanding and control of the learning process which is implicit in every computational interpretation of classical logic. This is crucial: the inefficiency of the algorithms extracted from classical proofs is usually due to their inability to backtrack without forgetting important information that they have acquired during the computation. It is already evident that dramatically more efficient algorithm are possible, by managing in a more sophisticated way the update of the states. For example, multiple updates of the states can be allowed at one time and in the proof of the Zero theorem one does not need to “set to zero” every approximation corresponding to a Skolem function of higher level of the first level which is corrected by an update. For reasons of space, we leave these optimizations to future work.

References

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

2 F. Aschieri, *Transfinite Update Procedures for Predicative Systems of Analysis*, Proceedings of Computer Science Logic, 2011.

3 F. Aschieri, *A Constructive Analysis of Learning in Peano Arithmetic*, Annals of Pure and Applied Logic, 2011, doi:10.1016/j.apal.2011.12.004.

4 F. Aschieri, S. Berardi, *A New Use of Friedman's Translation: Interactive Realizability*, Festschrift of Helmut Schwichtenberg, Ontos-Verlag Series in Mathematical Logic, to appear.

5 F. Aschieri, *Learning Based Realizability for HA + EM1 and 1-Backtracking Games: Soundness and Completeness*, Annals of Pure and Applied Logic, to appear.

6 F. Aschieri, *Interactive Realizability for Second-Order Heyting Arithmetic with* EM_1 *and* SK_1, Technical Report, http://hal.inria.fr/hal-00657054.

7 F. Aschieri, *Interactive Realizability for Classical Peano Arithmetic with Skolem Axioms*, Technical Report, http://hal.inria.fr/hal-00685360.

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

9 F. Barbanera, S. Berardi, *A Symmetric Lambda-Calculus for Classical Program Extraction*, Information and Computation, 1996.

10 S. Berardi and U. de' Liguoro, *Interactive Realizers. A New Approach to Program Extraction from Nonconstructive Proofs*, ACM Transactions on Computational Logic, 2012.

11 T. Coquand, *A Semantic of Evidence for Classical Arithmetic*, Journal of Symbolic Logic, 1995.

12 H. Friedman, *Classically and Intuitionistically Provable Recursive Functions*, Lecture Notes in Mathematics, 1978, Volume 669/1978, 21-27.

13 K. Gödel, *Uber eine bisher noch nicht benutzte Erweiterung des finiten Standpunktes*, Dialectica 12, pp. 280-287 (1958).

14 G. Gentzen, *Die Widerspruchsfreiheit der reinen Zahlentheorie*. Mathematische Annalen,1935.

15 J.-Y. Girard, *Proofs and Types*, Cambridge University Press (1989).

16 G. Kreisel, *On Weak Completeness of Intuitionistic Predicate Logic*, Journal of Symbolic Logic, vol. 27, 1962.

17 J-L. Krivine, *Typed lambda-calculus in classical Zermelo-Fraenkel set theory*, Archive for Mathematical Logic, 40(3), 2001.

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

19 M. H. Sorensen, P. Urzyczyn, *Lectures on the Curry-Howard isomorphism*, Studies in Logic and the Foundations of Mathematics, vol. 149, Elsevier, 2006.

20 A. Troelstra, D. van Dalen, *Constructivism in Mathematics, vol. I*, North-Holland, 1988.

Relational Parametricity for Higher Kinds

Robert Atkey

University of Strathclyde, UK
Robert.Atkey@strath.ac.uk

Abstract

Reynolds' notion of relational parametricity has been extremely influential and well studied for polymorphic programming languages and type theories based on System F. The extension of relational parametricity to higher kinded polymorphism, which allows quantification over type operators as well as types, has not received as much attention. We present a model of relational parametricity for System Fω, within the impredicative Calculus of Inductive Constructions, and show how it forms an instance of a general class of models defined by Hasegawa. We investigate some of the consequences of our model and show that it supports the definition of inductive types, indexed by an arbitrary kind, and with reasoning principles provided by initiality.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, F.3.2 Semantics of Programming Languages

Keywords and phrases Relational Parametricity, Higher Kinds, Polymorphism

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.46

1 Introduction

Reynolds defined relational parametricity to formalise the intuition that, in the absence of introspection capabilities, a polymorphic program must act uniformly in the choice of type instantiation [14]. The inability of a program to rely on details of data representation that it has not been explicitly exposed to forms the backbone of reasoning based on information hiding and abstraction. The core of Reynolds' idea is that a parametrically polymorphic program should preserve all relations between any pair of types that it is instantiated with.

Since Reynolds' definition, many interesting consequences have been revealed. Wadler called some of these consequences "Theorems for free" [20] where theorems can be stated about polymorphically typed programs just by looking at their types. It is also possible to prove that the polymorphic λ-calculus allows encodings of categorical constructions such as finite products and coproducts, initial algebras and final coalgebras, as long as the model is parametric in Reynolds' sense, as demonstrated by Hasegawa [7].

Most of the previous work on parametricity has been based on languages and type theories building on System F, where type quantification is only over types. Modern programming languages now include *higher kinded* polymorphism where programs may be parameterised by type operators of kinds like $* \to *$ as well as normal types of kind $*$. Type operators are simply functions operating on types, where the kinds act as a type system at "one level up" to classify types. The purely functional language Haskell uses type operators of kind $* \to *$ to represent constructions such as monads [11], and the GHC compiler has recently added support for user defined kinds [21]. The JVM-based language Scala [12] also includes support for type operators. Languages in the ML family such as OCaml and SML do not support higher kinded polymorphism in their core languages, but do support a form of it through their module systems. Indeed, Rossberg *et al.* [16] have shown that ML-style modules can be understood by translation into a language with higher kinds.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 46–61
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we consider the prototypical calculus with type operators and higher kinded polymorphism, System Fω. We give a concrete model of this calculus within the impredicative variant of the Calculus of Inductive Constructions, building on the interpretation of kinds as reflexive graphs, due to Hasegawa [8, 7] and Robinson and Rosolini [15]. The use of reflexive graphs builds Reynolds' identity extension principle into the interpretation, ensuring that we are able to perform hypothetical reasoning with parametric functions. We make use of this ability to show that we can represent κ indexed inductive types, for arbitrary kinds κ, within System Fω, and prove their initiality. Proving initiality is essential for reasoning about members of inductive types. When $\kappa = *$, we obtain a method for encoding and reasoning about Generalised Algebraic Data Types (GADTs) [5].

1.1 Background

To our knowledge, the earliest formulation of a theory of relational parametricity for higher kinded types is due to Hasegawa [8], building on his previous work on relationally parametric System F [7]. Hasegawa gives a category theoretic approach, based on interpreting kinds and types in categories of *r-frames*: sets equipped with a notion of reflexive relations between their elements, building in Reynolds' identity extension property. We recall Hasegawa's definition of r-frame in Section 3.3, and describe how it relates to our model construction, and to the reflexive graph approach of Robinson and Rosolini [15]. Hasegawa presents a set of general requirements on a model in order for it to interpret System Fω, and demonstrates two instances: a PER-based model and a syntactic model. Hasegawa also develops a number of applications of models, using notions of enriched category theory, including the existence of initial algebras and final coalgebras for a certain class of internally defined functors.

Takeuti [17] formulated a definition of relational parametricity for dependent type theories in the λ-cube, including System Fω, and developed a small amount of enriched category theory in this setting.

Bernardy, Jansson and Paterson [3] present a slightly different formulation of parametricity for pure type systems (PTSs), including those in the λ-cube. A key feature of this work is that the relational interpretation of a type is expressed within an augmented version of the PTS that is started with. Bernardy *et al.* prove an abstraction theorem for their translation, with the neat property that the same translation that translates types to relational interpretations also translates terms to the proof of the abstraction theorem for that term. This work does not take into account the identity extension aspect of parametricity, nor do the authors construct a model with the property that all members of universal types are relationally parametric, so the constructions we present in Section 4 are not possible in their setting.

Voigtländer [18] has examined an extension of free theorems to a fragment of System Fω, where he only considers functions with types of form $\forall_\kappa \alpha.\ A[\alpha] \to B[\alpha]$, where κ may be a higher kind. Voigtländer derives several useful free theorems relating functions that operate polymorphically over monads. We expect that all his free theorems are true in our model.

Vytiniotis and Weirich [19] present a syntactic model of parametricity for System Fω extended with representation types. They define a relational interpretation of types where great care is required, in their syntactic setting, to ensure that the interpretation is coherent under type equality. This is handled for us in our extensional denotational setting. Vytiniotis and Weirich consider an extended system with type representations and prove that, due to parametricity, run-time type casting using type representations will always be equivalent to the identity function. In comparison to the present work, they do not need to consider a definition of the interpretation of kinds that forces there to be a distinguished identity relation for every type, because they need only consider the types that are syntactically definable. We have also considered constructions within parametric System Fω beyond equality types.

1.2 Contributions of this Paper

- We define a concrete type theoretic model of relationally parametric System Fω, a polymorphic λ-calculus with higher kinded types. We give our model in direct terms, not using category theoretic language, in an attempt to make the definition more accessible. We also relate our model to a standard non-parametric semantics (Theorem 4), in order to show that the relationally parametric model can be used to reason about interpretations in the non-parametric model.
- We demonstrate that our relationally parametric model of System Fω allows for the definition of indexed inductive types, with an initiality property that allows for reasoning. Hasegawa also proves the existence of initial algebras in relationally parametric models of System Fω, for functors satisfying a condition he calls the *middle-lax* property. We demonstrate initial algebras for all definable functors, and present our proof in more elementary terms. We also demonstrate how to interpret data kinds, such as type-level natural numbers.

We emphasise that even though our model is constructed on paper with the impredicative Calculus of Inductive Constructions, we have not yet completed a full verification of all our results in Coq. Also, there are evidently many more constructions that one might consider in a relationally parametric model of System Fω, including final coalgebras. We have some preliminary results in this direction, but leave development of them to future work.

2 Type Polymorphism with Higher Kinds

To fix notation we define the syntax and typing of System Fω in this section, following the standard presentations [13]. There are three levels: *kinds*, which classify *types*, which classify *terms*. The language of kinds consists of simple types over a single base kind $*$ that represents all types that can classify terms. We use meta-syntactic variables $\kappa, \kappa_1, \kappa_2, ...$ to stand for kinds. The grammar $\kappa ::= * \mid \kappa_1 \to \kappa_2$ defines the collection of kinds that we consider. We will look at extending the calculus with additional kinds in Section 5.

The language of types is an applied simply-typed λ-calculus with constants for function and universal types. The kinding rules that classify types by kind are shown in Section 1. We will use the general name "type" for both proper types of kind $*$ and for higher kinded type operators. We use Θ as a meta-variable to stand for kinding contexts $\alpha_1 : \kappa_1, ..., \alpha_n : \kappa_n$, Greek letters $\alpha, \beta, ...$ for type variables in the calculus and capital roman letters $A, B, F, ...$ for meta-syntactic variables standing for arbitrary types. Types come equipped with an equational theory ($\equiv$) built from the standard typed $\beta\eta$ rules for the simply typed λ-calculus.

The typing rules for terms are shown in Section 2. These are the standard rules for System Fω. We assume throughout that all the types in the typing context Γ are of base kind according to the kinding context Θ. We use the notation $A\{B/\alpha\}$ to represent capture avoiding substitution of B for all occurrences of α in A. The final typing rule imports the equational theory of types into the typing judgement, allowing a term to have many syntactically different types. Terms come equipped with an equational theory of their own: $\beta\eta$ equalities for λ-abstraction and Λ-abstraction.

$$\frac{\alpha : \kappa \in \Theta}{\Theta \vdash \alpha : \kappa} \qquad \frac{\Theta, \alpha : \kappa_1 \vdash A : \kappa_2}{\Theta \vdash \lambda\alpha : \kappa_1.\ A : \kappa_1 \to \kappa_2} \qquad \frac{\Theta \vdash F : \kappa_1 \to \kappa_2 \qquad \Theta \vdash A : \kappa_1}{\Theta \vdash FA : \kappa_2}$$

$$\frac{\Theta \vdash A : * \qquad \Theta \vdash B : *}{\Theta \vdash A \to B : *} \qquad \frac{\Theta, \alpha : \kappa \vdash A : *}{\Theta \vdash \forall_\kappa \alpha.\ A : *}$$

Figure 1 Types and their Kinds.

$$\frac{x : A \in \Gamma}{\Theta \mid \Gamma \vdash x : A} \qquad \frac{\Theta \mid \Gamma \vdash e : A \qquad \Theta \vdash A \equiv B : *}{\Theta \mid \Gamma \vdash e : B}$$

$$\frac{\Theta \mid \Gamma, x : A \vdash e : B}{\Theta \mid \Gamma \vdash \lambda x : A.\ e : A \to B} \qquad \frac{\Theta \mid \Gamma \vdash e_1 : A \to B \qquad \Theta \mid \Gamma \vdash e_2 : A}{\Theta \mid \Gamma \vdash e_1 e_2 : B}$$

$$\frac{\Theta, \alpha : \kappa \mid \Gamma \vdash e : A \qquad \alpha \notin fv(\Gamma)}{\Theta \mid \Gamma \vdash \Lambda\alpha : \kappa.\ e : \forall_\kappa \alpha.A} \qquad \frac{\Theta \mid \Gamma \vdash e : \forall_\kappa \alpha.A \qquad \Theta \vdash B : \kappa}{\Theta \mid \Gamma \vdash e\ [B] : A\{B/\alpha\}}$$

Figure 2 Terms and their Types.

3 A Relationally Parametric Model

We now present the construction of a relationally parametric model of System Fω in the impredicative Calculus of Inductive Constructions (CIC). CIC is a dependently typed λ-calculus with impredicative polymorphism and inductive types, and forms the basis of the Coq proof assistant. The presence of impredicative polymorphism in our meta-theory simplifies the presentation of our model, allows us to concentrate on the parametricity aspects, and permits straightforward reasoning inside the model. This technique has been used to construct models of System F with Kripke parametricity for the purposes of studying the adequacy of Higher Order Abstract Syntax (HOAS) encodings [1]. Our model is an instance of the class of models defined by Hasegawa, and also those defined by Robinson and Rosolini. We cover the connection in Section 3.3. We motivate this class of models in Section 3.2.

3.1 Setting up the Metatheory

Impredicative CIC allows us to quantify over all objects of sort Set to generate a new object of sort Set. This feature can be enabled in the Coq proof assistant by means of the `-impredicative-set` command line option. CIC already includes an impredicative sort of propositions Prop. Importantly, this is disjoint from the sort Set, where we build our denotations of types.

We require two additional axioms to be added to CIC. The first of these is extensionality for functions, which states that two functions are equal if they are equal for all inputs: $\forall A \in \mathsf{Type}, B \in A \to \mathsf{Type}, f, g \in (\forall a.Ba).\ (\forall x.\ fx = gx) \to f = g$. Extensionality for functions allows our denotational model to smoothly support the η equality rules of System Fω, without requiring complex constructions involving setoids. We also require propositional extensionality, which will allow us to treat equivalent propositions as equal:

$\forall P, Q \in \mathsf{Prop},\ (P \leftrightarrow Q) \to P = Q$. Propositional extensionality implies proof irrelevance: all proofs of a given proposition are equal: $\forall P \in \mathsf{Prop}.\ \forall p_1, p_2 \in P.\ p_1 = p_2$. These axioms allow us to define data with embedded proofs that are equal when their computational parts are equal, which allows us to prove more equalities between denotations of types.

We informally justify our use of these axioms, plus impredicativity, by the existence of models of CIC in intuitionistic set theory. In the remainder of the paper, we use informal set theoretic notation and do not explicitly highlight the uses of these axioms. Note that everywhere we refer to "set"s, we mean CIC objects of sort Set.

3.2 Types, Relations and Identity Extension

We consider the extension of Reynolds' definition of relational parametricity to a type system with higher kinds, in order to motivate the model construction in the rest of this section.

If we model types of kind $*$ as sets, then the modelling of the rest of the kind hierarchy has an obvious choice: we interpret kinds of the form $\kappa_1 \to \kappa_2$ as functions:

$$[\![*]\!] = \mathsf{Set} \qquad\qquad [\![\kappa_1 \to \kappa_2]\!] = [\![\kappa_1]\!] \to [\![\kappa_2]\!]$$

To define relational parametricity, we need to consider what relations between the interpretations of types look like. At the base kind, $*$, this will be just the collection of all relations between a pair of sets: we denote the collection of relations between sets A and B as $\mathsf{Rel}(A, B)$. At higher kinds, an obvious approach is to consider relation transformers:

$$\begin{array}{rcl} [\![\kappa]\!]^{\mathcal{R}} & \in & [\![\kappa]\!] \times [\![\kappa]\!] \to \mathsf{Set} \\ [\![*]\!]^{\mathcal{R}} & = & (A, B) \mapsto \mathsf{Rel}(A, B) \\ [\![\kappa_1 \to \kappa_2]\!]^{\mathcal{R}} & = & (F, G) \mapsto \forall A, B \in [\![\kappa_1]\!].[\![\kappa_1]\!]^{\mathcal{R}}(A, B) \to [\![\kappa_2]\!]^{\mathcal{R}}(FA, GB) \end{array}$$

Following relationally parametric models of System F (e.g., Jacobs [9], Section 8.4), for any well-kinded type $\alpha_1 : \kappa_1, \cdots, \alpha_n : \kappa_n \vdash A : \kappa$, we will have an underlying interpretation $[\![A]\!]^f \in [\![\kappa_1]\!] \times \cdots \times [\![\kappa_n]\!] \to [\![\kappa]\!]$ and a relational interpretation $[\![A]\!]^r \in \forall \overrightarrow{A}, \overrightarrow{B}.[\![\kappa_1]\!]^R(A_1, B_1) \times \cdots \times [\![\kappa_n]\!]^R(A_n, B_n) \to [\![\kappa]\!]^R(F\overrightarrow{A}, F\overrightarrow{B})$.

However, this interpretation of kinds suffers from a problem. In order to allow for hypothetical reasoning using relational parametricity, we must ensure that the interpretation of a type of the form $\forall_\kappa \alpha.A$ only contains elements that actually preserve all relations (i.e., are parametric). If the type A is open—it has free type variables—then in order to state this property we need a default relational interpretation for each of the free type variables. In Reynolds' formulation of relational parametricity, where all type variables have kind $*$, the default relational interpretation is given by equality. Moreover, given a type with free type variables, substituting the equality relation for all the free variables in the relational interpretation of the type should give the equality relation: this is Reynolds' *identity extension* property, which is needed to prove the abstraction theorem (see Theorem 3, below).

In the situation with higher kinds, some of the variables may be of functional kinds like $\kappa_1 \to \kappa_2$, so to even state identity extension we need a notion of identity relation for any type at these kinds. We cannot use equality, because we require a relation transformer. Moreover, this transformer should obey the identity extension property itself; sending the identity relations for kind κ_1 to the identity relations for kind κ_2. But what are these identity relations? There does not seem to be an obvious way to assign an identity relation transformer to an arbitrary type operator, so we need to equip each element of $[\![\kappa_1 \to \kappa_2]\!]$ with a distinguished relation transformer with the identity extension property.

3.3 Kinds as Reflexive Graphs

Hasegawa [7, 8] and Robinson and Rosolini [15] give us the solution. Instead of defining the underlying and relational semantics of kinds separately, we treat each kind as a *reflexive graph*. A reflexive graph consists of two collections: one of *objects* and one of *relations*. Every relation is assigned a source and target object, and every object a has a distinguished "identity" relation, whose source and target are a. Hasegawa calls such structures *r-frames*, and notes that they are essentially categories without composition. Robinson and Rosolini note that the category of reflexive graphs is just the category of presheaves over the category $\bullet \mathrel{\substack{\xleftarrow{\delta_0} \\ \xrightarrow{i} \\ \xleftarrow{\delta_1}}} \bullet$ such that $\delta_0 \circ i = id$ and $\delta_1 \circ i = id$. It immediately follows that there is an interpretation of higher kinds, using the cartesian closed structure of this presheaf category, although Robinson and Rosolini do not make use of this fact. This solves the problem we identified above: the assignment of a distinguished identity relation to every inhabitant of the denotation of a higher kind. Dunphy and Reddy [6] also use reflexive graphs to study relational parametricity.

It the rest of this section, we present a concrete model of System Fω within CIC that interprets kinds as reflexive graphs. We do not make explicit use of the presheaf structure underlying the model, in order to keep the presentation straightforward and accessible.

3.4 Interpretation of Kinds

A kind κ is interpreted as a triple of a carrier A; a function that takes a pair of elements of A and returns the set of "relations" between them; and a special distinguished "identity" relation for every element of A. We write this specification as a CIC type as follows:

$$\Sigma A \in \mathsf{Type}.\ \Sigma R \in A \times A \to \mathsf{Type}.\ \forall a \in A.\ R(a,a)$$

In the following, we make use of the projection functions $-^{U}$, $-^{\mathcal{R}}$ and $-^{\Delta}$ to obtain the first, second and third components of the interpretations of kinds.

We define an interpretation for all kinds by induction on their structure. At base kind, the carrier is simply the type of all sets; relations between A and B are subsets of $A \times B$; and the distinguished identity relation is exactly the equality relation:

$$[\![*]\!] = (\mathsf{Set}, \mathsf{Rel}, \equiv)$$

At higher kinds $\kappa_1 \to \kappa_2$, the interpretation is more involved. The carrier is the set of pairs of functions from the carrier of κ_1 to the carrier of κ_2, along with their associated distinguished identity-preserving identity relations; the relations between two semantic type operators (F,R) and (G,S) are relation transformers; and the distinguished identity relation for (F,R) is just R.

$$\begin{aligned}
[\![\kappa_1 \to \kappa_2]\!] = \quad & (\{(F,R) \mid F \in [\![\kappa_1]\!]^{\mathcal{U}} \to [\![\kappa_2]\!]^{\mathcal{U}}, R \in (\forall AB.\ [\![\kappa_1]\!]^{\mathcal{R}}(A,B) \to [\![\kappa_2]\!]^{\mathcal{R}}(FA,FB)), \\
& \qquad\qquad \forall A \in [\![\kappa_1]\!]^{\mathcal{U}}.\ RAA([\![\kappa_1]\!]^{\Delta}A) = [\![\kappa_2]\!]^{\Delta}(FA)\}, \\
& ((F,R),(G,S)) \mapsto \forall AB.[\![\kappa_1]\!]^{\mathcal{R}}(A,B) \to [\![\kappa_2]\!]^{\mathcal{R}}(FA,GB), \\
& (F,R) \mapsto R\ )
\end{aligned}$$

Kinding contexts $\Theta = \alpha_1 : \kappa_1, ..., \alpha_n : \kappa_n$ are interpreted as follows:

$$\begin{aligned}
[\![\alpha_1 : \kappa_1, ..., \alpha_n : \kappa_n]\!] = \quad & ([\![\kappa_1]\!]^{\mathcal{U}} \times ... \times [\![\kappa_n]\!]^{\mathcal{U}}, \\
& \ (\theta,\theta') \mapsto [\![\kappa_1]\!]^{\mathcal{R}}(\pi_1\theta, \pi_1\theta') \times ... \times [\![\kappa_n]\!]^{\mathcal{R}}(\pi_n\theta, \pi_n\theta'), \\
& \ \theta \mapsto ([\![\kappa_1]\!]^{\Delta}(\pi_1\theta), ..., [\![\kappa_n]\!]^{\Delta}(\pi_n\theta)))
\end{aligned}$$

We use π_i to denote the ith projection. The carrier component of the interpretation of a kinding context is simply the product of the underlying semantics of the context members, and the relational component is simply the product of the relational components of the context members. This naturally leads to the identity component being defined as the tuple of the identity components of the interpretations of the context members. We overload the projections $-^U$, $-^{\mathcal{R}}$ and $-^{\Delta}$ for the denotations of kinding contexts as well as kinds.

3.5 Interpretation of Types

A kinding judgement $\Theta \vdash A : \kappa$ establishing the well-formedness of a type A is interpreted as a pair of a function $\llbracket A \rrbracket^f \in \llbracket \Theta \rrbracket^{\mathcal{U}} \to \llbracket \kappa \rrbracket^{\mathcal{U}}$ and a relation transformer $\llbracket A \rrbracket^r \in (\forall \theta\theta' \in \llbracket \Theta \rrbracket^{\mathcal{U}}. \llbracket \Theta \rrbracket^{\mathcal{R}}(\theta, \theta') \to \llbracket \kappa \rrbracket^{\mathcal{R}}(\llbracket A \rrbracket^f \theta, \llbracket A \rrbracket^f \theta'))$ such that the identity extension property holds. We give the full statement below in Theorem 1.

We now give an interpretation of the kinding judgements of Section 1, by induction on the kinding derivation. Type variables $\Theta \vdash \alpha_i : \kappa_i$, where α_i is the ith variable in Θ, are interpreted using projections:

$$\llbracket \alpha_i \rrbracket^f \theta = \pi_i \theta \qquad\qquad \llbracket \alpha_i \rrbracket^r \theta\theta'\rho = \pi_i \rho$$

For λ-abstraction of types, we must return a pair of the type operator and the distinguished identity relation on the operator. This is derived from the relational interpretation of the premise and the identity relation on the context. In essence, type-level λ-abstraction in System Fω is interpreted as λ-abstraction at the meta-level:

$$\begin{aligned}
\llbracket \lambda\alpha : \kappa_1.\ A \rrbracket^f \theta &= (\lambda X \in \llbracket \kappa_1 \rrbracket^{\mathcal{U}}.\ \llbracket A \rrbracket^f (\theta, X), \\
&\qquad \lambda XY \in \llbracket \kappa_1 \rrbracket^{\mathcal{U}}, R \in \llbracket \kappa_1 \rrbracket^{\mathcal{R}}(X, Y).\ \llbracket A \rrbracket^r (\theta, X)(\theta, Y)(\llbracket \Theta \rrbracket^{\Delta} \theta, R)) \\
\llbracket \lambda\alpha : \kappa_1.\ A \rrbracket^r \theta\theta'\rho &= \lambda XY \in \llbracket \kappa_1 \rrbracket^{\mathcal{U}}, R \in \llbracket \kappa_1 \rrbracket^{\mathcal{R}}(X, Y).\ \llbracket A \rrbracket^r (\theta, X)(\theta', Y)(\rho, R)
\end{aligned}$$

Type-level application is interpreted as application at the meta-level:

$$\begin{aligned}
\llbracket F\ A \rrbracket^f \theta &= \pi_1(\llbracket F \rrbracket^f\ \theta)\ (\llbracket A \rrbracket^f\ \theta) \\
\llbracket F\ A \rrbracket^r \theta\theta'\rho &= \llbracket F \rrbracket^r \theta\theta'\rho\ (\llbracket A \rrbracket^f \theta)(\llbracket A \rrbracket^f \theta')(\llbracket A \rrbracket^r \theta\theta'\rho)
\end{aligned}$$

We now turn to the interpretation of the two constructors of actual types in the kinding system of Section 1. For function types, we use the standard logical relations interpretation of function types as preservation of relations:

$$\begin{aligned}
\llbracket A \to B \rrbracket^f \theta &= \llbracket A \rrbracket^f \theta \to \llbracket B \rrbracket^f \theta \\
\llbracket A \to B \rrbracket^r \theta\theta'\rho &= \{(f, g) \mid \forall (x, y) \in \llbracket A \rrbracket^r \theta\theta'\rho.\ (fx, gy) \in \llbracket B \rrbracket^r \theta\theta'\rho\}
\end{aligned}$$

Finally, a universal type $\forall_\kappa \alpha.A$ is interpreted as the set of all type-parameterised inhabitants of the open type A that are relationally parametric. We use the distinguished identity relation on contexts in this definition, where Θ is the current context in the kinding derivation. The interpretation of universally quantified types as just those elements that are relationally parametric is key to the applications of the model that we present in Section 4. This definition allows us to state results that range over all inhabitants of $\llbracket \forall_\kappa \alpha.A \rrbracket^f$, not just those that arise from closed programs.

$$\begin{aligned}
\llbracket \forall_\kappa \alpha.\ A \rrbracket^f \theta &= \{x \in (\forall X \in \llbracket \kappa \rrbracket^{\mathcal{U}}.\ \llbracket A \rrbracket^f (\theta, X)) \mid \\
&\qquad \forall XY, R \in \llbracket \kappa \rrbracket^{\mathcal{R}}(X, Y).\ (xX, xY) \in \llbracket A \rrbracket^r (\theta, X)(\theta, Y)(\llbracket \Theta \rrbracket^{\Delta} \theta, R)\} \\
\llbracket \forall_\kappa \alpha.\ A \rrbracket^r \theta\theta'\rho &= \{(x, y) \mid \forall XY, R \in \llbracket \kappa \rrbracket^{\mathcal{R}}(X, Y).\ (xX, yY) \in \llbracket A \rrbracket^r (\theta, X)(\theta', Y)(\rho, R)\}
\end{aligned}$$

The following theorem states that the clauses above are well-defined: all the definitions above are well-typed, and the identity extension property is preserved.

▶ **Theorem 1.** *For all well-kinded types* $\Theta \vdash A : \kappa$ *there exist functions:*

$$[\![A]\!]^f \in [\![\Theta]\!]^{\mathcal{U}} \to [\![\kappa]\!]^{\mathcal{U}} \quad \textit{and} \quad [\![A]\!]^r \in (\forall \theta\theta' \in [\![\Theta]\!]^{\mathcal{U}}.[\![\Theta]\!]^{\mathcal{R}}(\theta, \theta') \to [\![\kappa]\!]^{\mathcal{R}}([\![A]\!]^f\theta, [\![A]\!]^f\theta'))$$

such that identity extension holds: $\forall \theta \in [\![\Theta]\!]^{\mathcal{U}}.\ [\![A]\!]^r\theta\theta([\![\Theta]\!]^{\Delta}\theta) = [\![\kappa]\!]^{\Delta}([\![A]\!]^f\theta)$

▶ **Theorem 2** (Soundness of $\beta\eta$). *If* $\Theta \vdash A \equiv B : \kappa$ *then* $[\![A]\!]^f = [\![B]\!]^f$ *and* $[\![A]\!]^r = [\![B]\!]^r$*.*

We extend the interpretation of types to the interpretation of typing contexts in the obvious way: contexts $\Gamma = x_1 : A_1, ..., x_n : A_n$ (where all the A_i are of base kind) are interpreted as tuples of the the interpretations of the individual types, and the relational interpretation is the standard one for logical relations on products.

3.6 Semantics of Terms

We omit the interpretation of terms, which is similar to the interpretation of the terms of System F in a relationally parametric model (see, for example, Bainbridge *et al.* [2]). We just state the key result—the abstraction theorem for System Fω.

▶ **Theorem 3.** *For all well-typed terms* $\Theta \mid \Gamma \vdash e : A$ *there is a function (the* $-^{\mathcal{P}}$ *superscript stands for "parametric")*

$$[\![e]\!]^{\mathcal{P}} \in (\forall \theta \in [\![\Theta]\!]^{\mathcal{U}}.\ [\![\Gamma]\!]^f\theta \to [\![A]\!]^f\theta)$$

such that, for all $\theta, \theta' \in [\![\Theta]\!]^{\mathcal{U}}$*,* $\rho \in [\![\Theta]\!]^{\mathcal{R}}(\theta, \theta')$*,* $\gamma \in [\![\Gamma]\!]^f\theta$ *and* $\gamma' \in [\![\Gamma]\!]^f\theta'$*,*

$$\textit{if } (\gamma, \gamma') \in [\![\Gamma]\!]^r\theta\theta'\rho \textit{ then } ([\![e]\!]^{\mathcal{P}}\theta\gamma, [\![e]\!]^{\mathcal{P}}\theta'\gamma') \in [\![A]\!]^r\theta\theta'\rho.$$

Moreover, this interpretation is sound for the $\beta\eta$ *equational theory of the terms.*

The interpretation of terms and the proof that they satisfy the abstraction property must be carried out simultaneously in order to show that the interpretation of Λ-abstraction is well-defined. This is due to the interpretation of the $\forall_\kappa$-types as sets of type indexed values that preserve all relations.

The next theorem relates the model we have defined in this section to the standard non-parametric model in CIC, where we interpret higher kinds simply as functions, and $\forall_\kappa$-types just using the impredicative quantification of the meta-theory. The intention is that this non-parametric semantics represents the "natural" semantics of System Fω in CIC, without the artificial scaffolding of relational parametricity. To state this theorem, it is necessary to extend the calculus with a type of booleans to act as observable values.

▶ **Theorem 4.** *For any closed term* $- \mid - \vdash e : \texttt{bool}$*, the parametric semantics of Theorem 3 and the non-parametric semantics are equal.*

The proof of this theorem is carried out by the construction of a logical relation between the parametric and non-parametric semantics. The importance of this theorem is that it allows us to use equalities between terms that we prove in the parametric semantics of Theorem 3 to reason about contextual equivalence in the non-parametric semantics. If we prove an equivalence $[\![e_1]\!]^{\mathcal{P}} = [\![e_2]\!]^{\mathcal{P}}$ in the parametric semantics, where e_1 and e_2 may be open terms, then for all contexts $C[-]$ of type `bool`, we have, by the compositionality of $[\![-]\!]^{\mathcal{P}}$, $[\![C[e_1]]\!]^{\mathcal{NP}} = [\![C[e_1]]\!]^{\mathcal{P}} = [\![C[-]]\!]^{\mathcal{P}}[\![e_1]\!]^{\mathcal{P}} = [\![C[-]]\!]^{\mathcal{P}}[\![e_2]\!]^{\mathcal{P}} = [\![C[e_2]]\!]^{\mathcal{P}} = [\![C[e_2]]\!]^{\mathcal{NP}}$, where $[\![-]\!]^{\mathcal{NP}}$ is the non-parametric semantics, and $[\![-]\!]^{\mathcal{P}}$ is the parametric semantics.

4 Applications of Higher Kinded Parametricity

We now demonstrate how, in System Fω, we can define useful data type constructions for *indexed* data types, of kind $\kappa \to *$. Such indexed data types include Generalised Algebraic Data Types (GADTs) [5], by setting $\kappa = *$. We build towards the construction of indexed inductive types in stages, defining equality types, existential types at higher kinds, product and sum types and then finally indexed inductive types as initial algebras. For each construction, relational parametricity is used to justify the η equality rules.

We reason within CIC, using the model that we constructed in the previous section. When we quantify over all types of a particular kind, we mean to interpret this as all semantic inhabitants of the denotation of this kind. Similarly, when we quantify over terms of a particular type, we mean semantic inhabitants of the denotation of the type. We often omit semantic brackets to reduce clutter. When F is a semantic type of kind κ, we write Δ_F for F's distinguished identity relation $[\![\kappa]\!]^{\Delta}(F) \in [\![\kappa]\!]^{\mathcal{R}}(F, F)$. As a shorthand for the relational interpretations of certain types, we often write type expressions with free type variables replaced by relations.

4.1 Equality Types

It is possible (Jacobs [9], Section 8.1) to extend System Fω with an equality type that records when two types (of arbitrary but equal kinds) are equal. One adds an additional kind indexed family of type operators $\mathsf{Eq}_\kappa : \kappa \to \kappa \to *$ and two kind indexed families of term constants:

$$\mathsf{refl}_\kappa : \forall_\kappa \alpha.\ \mathsf{Eq}_\kappa \alpha\alpha \qquad\qquad \mathsf{elimEq}_\kappa : \forall_\kappa \alpha\beta.\ \mathsf{Eq}_\kappa \alpha\beta \to \forall_{\kappa\to\kappa\to *}\rho.(\forall_\kappa\gamma.\ \rho\gamma\gamma) \to \rho\alpha\beta$$

that obey the following β and η equality rules:

$$\frac{A : \kappa \qquad G : \kappa \to \kappa \to * \qquad f : \forall_\kappa \alpha. G\alpha\alpha}{\mathsf{elimEq}_\kappa\ [A]\ [A]\ (\mathsf{refl}\ [A])\ [G]\ f = f\ [A]}\ (\beta)$$

$$\frac{A, B : \kappa \qquad z : \mathsf{Eq}_\kappa AB \qquad G : \kappa \to \kappa \to * \qquad t : \forall_\kappa \alpha\beta.\ \mathsf{Eq}_\kappa\ \alpha\ \beta \to G\alpha\beta}{\mathsf{elimEq}_\kappa\ [A]\ [B]\ z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathsf{refl}_\kappa\ [\gamma])) = t\ [A]\ [B]\ z}\ (\eta)$$

We can define a substitution operation, using elimEq_κ, which will be used in Section 5 below.

$$\begin{aligned} \mathit{subst}_\kappa &\ :\ \forall_\kappa \alpha\beta.\mathsf{Eq}_\kappa\alpha\beta \to \forall_{\kappa\to *}\rho.\ \rho\alpha \to \rho\beta \\ \mathit{subst}_\kappa &\ =\ \Lambda\alpha\beta.\lambda e.\Lambda\rho.\lambda x.\ \mathsf{elimEq}_\kappa\ [\alpha]\ [\beta]\ e\ [\lambda\gamma_1\gamma_2.\ \rho\gamma_1 \to \rho\gamma_2]\ (\Lambda\gamma.\lambda x.x)\ x \end{aligned}$$

In System Fω, it is possible to implement the equality type. We make the following definitions, encoding the equality type as it own eliminator.

$$\begin{aligned} \mathit{Eq}_\kappa &= \lambda\alpha\beta.\ \forall_{\kappa\to\kappa\to *}\rho.\ (\forall_\kappa\gamma.\ \rho\gamma\gamma) \to \rho\alpha\beta \\ \mathit{refl}_\kappa &= \Lambda\alpha.\ \Lambda\rho.\ \lambda f.\ f\ [\alpha] \\ \mathit{elimEq}_\kappa &= \Lambda\alpha\beta.\ \lambda e.\ \Lambda\rho.\ \lambda f.\ e\ [\rho]\ f \end{aligned}$$

The β equality rule for this implementation of equality types holds in all models of System Fω, just by β reduction. In the parametric model of Section 3, we can also show that the η equality rule holds, by using higher kinded relational parametricity.

▶ **Lemma 5.** *Let A, B be types of kind κ, and G be a type of kind $\kappa \to \kappa \to *$. Then for any $z : Eq_\kappa AB$, and $t : \forall_\kappa \alpha\beta.\ Eq_\kappa\alpha\beta \to G\alpha\beta$, we have (in the model of Section 3):*

$$t\ [A]\ [B]\ (z\ [Eq_\kappa]\ \mathit{refl}_\kappa) = z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma]))$$

Proof. Let $\Delta_G = [\![\kappa \to \kappa \to *]\!]^{\Delta}(G)$, the identity-preserving identity relation transformer associated with G. Define another relation transformer $R \in [\![\kappa \to \kappa \to *]\!]^{\mathcal{R}}(Eq_\kappa, G)$ as $R_{AB}S_{A'B'}S' = \{(x,y) \mid (t\ [A]\ [A']\ x, y) \in \Delta_G SS'\}$. We know from the parametricity property derived from z's type that:

$$(z\ [Eq_\kappa], z\ [G]) \in (\forall_\kappa \gamma.\ R\gamma\gamma) \to R\Delta_A\Delta_B \tag{1}$$

where $\Delta_A = [\![\kappa]\!]^{\Delta}(A)$ and Δ_B are the identity relations appropriate to κ for the semantic types A and B. We will instantiate (1) with refl_κ and $\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma])$, so we must ensure that these are related at $\forall_\kappa\gamma.\ R\gamma\gamma$. But, for any X, Y and $S \in [\![\kappa]\!]^{\mathcal{R}}(X,Y)$ this reduces to whether $(t\ [X]\ [X]\ (\mathit{refl}_\kappa\ [X]), t\ [Y]\ [Y]\ (\mathit{refl}_\kappa\ [Y])) \in \Delta_G SS$, which follows from t's parametricity property, ensured by its membership of the denotation of a universal type.

Now, $(z\ [Eq_\kappa]\ \mathit{refl}_\kappa, z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma]))) \in R\Delta_A\Delta_B$. By the construction of identity relations at higher kinds, $\Delta_G\Delta_A\Delta_B = \Delta_{GAB} = (\equiv)$, so when we unfold the definition of R, we have shown that, in the model, $t\ [A]\ [B]\ (z\ [Eq_\kappa]\ \mathit{refl}_\kappa) = z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma]))$. ◀

▶ **Theorem 6.** *The β and η rules stated above for* refl *and* elimEq *hold for the implementations refl and elimEq when interpreted in the model of Section 3.*

Proof. The β equality rule can be seen to hold by expanding the definitions and applying β reduction. Showing that the η equality rule holds requires parametricity. Unfolding and β reducing, the equation to show becomes $z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma])) = t\ [A]\ [B]\ z$. From Lemma 5 we know that $z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma])) = t\ [A]\ [B]\ (z\ [Eq_\kappa]\ \mathit{refl}_\kappa)$. Now we use Lemma 5 again, with an arbitrary G and $f : \forall_\kappa\gamma.G\gamma\gamma$, setting $t = \Lambda\alpha\beta.\lambda x.x\ [G]\ f$. This gives us $z\ [Eq_\kappa]\ \mathit{refl}_\kappa\ [G]\ f = z\ [G]\ f$, and so by extensionality, $z\ [Eq_\kappa]\ \mathit{refl}_\kappa = z$. Thus we have shown $z\ [G]\ (\Lambda\gamma.\ t\ [\gamma]\ [\gamma]\ (\mathit{refl}_\kappa\ [\gamma])) = t\ [A]\ [B]\ z$, as required. ◀

4.2 Existential Types

As with System F, it is possible to encode existential types in System Fω, only now we have the option of doing so for higher kinds. The specification for existential types goes as follows: for every type $F : \kappa \to *$, there is a type $\exists_\kappa\alpha.\ F\alpha$, and two combinators:

$$\begin{aligned}\mathsf{pack}_\kappa &: \forall_{\kappa\to *}\rho.\ \forall_\kappa\alpha.\ \rho\alpha \to (\exists_\kappa\alpha.\ \rho\alpha)\\ \mathsf{elimEx}_\kappa &: \forall_{\kappa\to *}\rho.\ \forall_*\beta.\ (\forall_\kappa\alpha.\ \rho\alpha \to \beta) \to (\exists_\kappa\alpha.\ \rho\alpha) \to \beta\end{aligned}$$

that obey the following equational rules:

$$\frac{F : \kappa \to * \qquad A : \kappa \qquad x : FA \qquad B : * \qquad f : \forall_\kappa\alpha.\ F\alpha \to B}{\mathsf{elimEx}_\kappa\ [F]\ [B]\ f\ (\mathsf{pack}_\kappa\ [F]\ [A]\ x) = f\ [A]\ x}\ (\beta)$$

$$\frac{F : \kappa \to * \qquad e : \exists_\kappa\alpha.\ F\alpha \qquad B : * \qquad t : (\exists_\kappa\alpha.\ F\alpha) \to B}{\mathsf{elimEx}_\kappa\ [F]\ [B]\ (\Lambda\alpha.\ \lambda x.\ t\ (\mathsf{pack}_\kappa\ [F]\ [\alpha]\ x))\ e = t\ e}\ (\eta)$$

We can implement this specification in System Fω by copying the implementation of existentials in System F. Set $\exists_\kappa\alpha.F\alpha = \forall_*\beta.(\forall_\kappa\alpha.F\alpha \to \beta) \to \beta$ and define

$$\begin{aligned}\mathit{pack}_\kappa &= \Lambda\rho\alpha.\lambda x.\Lambda\beta.\lambda f.f\ [\alpha]\ x\\ \mathit{elimEx}_\kappa &= \Lambda\rho\beta.\lambda fe.\ e\ [\beta]\ f\end{aligned}$$

In the proof of the next lemma, we make use of functional relations. This is a standard concept used in relationally parametric reasoning for System F (for example, Birkedal and Møgelberg [4]). They are called *graph relations* by Hasegawa [8].

▶ **Definition 7** (Functional Relations at Base Kind)**.** For types A, B of kind $*$ and an inhabitant f of the type $A \to B$, we define the relation $\langle f \rangle \in [\![*]\!]^{\mathcal{R}}(A, B)$ as $\langle f \rangle = \{(a, b) \mid f a = b\}$.

▶ **Lemma 8.** *For all $F : \kappa \to *$ and $B : *$, $t : (\exists_\kappa \alpha.F\alpha) \to B$ and $e : \exists_\kappa \alpha.F\alpha$, we have*

$$t\ (e\ [\exists_\kappa \alpha.F\alpha]\ (pack_\kappa\ [F])) = e\ [B]\ (\Lambda\alpha.\lambda x.\ t\ (pack_\kappa\ [F]\ [\alpha]\ x))$$

Proof. By e's parametricity, we know that $(e\ [\exists_\kappa \alpha.F\alpha], e\ [B]) \in (\forall_\kappa \alpha.\Delta_F \alpha \to \langle t \rangle) \to \langle t \rangle$. We will apply this pair to $pack_\kappa\ [F]$ and $\Lambda\alpha.\lambda x.\ t\ (pack_\kappa\ [F]\ [\alpha]\ x)$, so we must show that they are related by $\forall_\kappa \alpha.\Delta_F \alpha \to \langle t \rangle$. Given $X, Y \in [\![\kappa]\!]^{\mathcal{U}}$, $S \in [\![\kappa]\!]^{\mathcal{R}}(X, Y)$ and $(x, y) \in \Delta_F S$, we want to prove $(pack_\kappa\ [F]\ [X]\ x, t\ (pack_\kappa\ [F]\ [Y]\ y)) \in \langle t \rangle$ which is equivalent to $t\ (\Lambda\beta.\lambda f.\ f\ [X]\ x) = t\ (\Lambda\beta.\lambda f.\ f\ [Y]\ y)$, by unfolding the definition of $\langle t \rangle$ and $pack_\kappa$. Now it is possible to prove this equality by using extensionality and f's parametricity.

Thus we have shown $(e\ [\exists_\kappa \alpha.F\alpha]\ (pack_\kappa\ [F]), e\ [B]\ (\Lambda\alpha.\lambda x.t\ (pack_\kappa\ [F]\ [\alpha]\ x))) \in \langle t \rangle$. Unfolding the definition of $\langle t \rangle$, this gives us what we want. ◀

▶ **Theorem 9.** *This implementation of $\exists_\kappa \alpha.F\alpha$, $pack_\kappa$ and $elimEx_\kappa$ satisfies the $\beta\eta$ equality rules for existential types when interpreted in the model of Section 3.*

Proof. The β equality rule follows simply by expanding the definitions and applying the β equality rules of System Fω. For the η rule, we use Lemma 8 to reason as follows:

$$\begin{aligned} elimEx_\kappa\ [F]\ [B]\ (\Lambda\alpha.\lambda x.t\ (pack_\kappa\ [F]\ [\alpha]\ x))\ e &= e\ [B]\ (\Lambda\alpha.\lambda x.t\ (pack_\kappa\ [F]\ [\alpha]\ x)) \\ &= t\ (e\ [\exists_\kappa \alpha.F\alpha]\ (pack_\kappa\ [F])) \end{aligned}$$

The first equality is by unfolding the definition of $elimEx_\kappa$ and the second is by Lemma 8. We now make use of Lemma 8 again, with an arbitrary B and $f : \forall_\kappa \alpha.F\alpha \to B$, setting $t = \lambda x.x\ [B]\ f$. This yields $e\ [\exists_\kappa \alpha.F\alpha]\ (pack_\kappa\ [F])\ [B]\ f = e\ [B]\ f$, and hence, by extensionality, $e\ [\exists_\kappa \alpha.F\alpha]\ (pack_\kappa\ [F]) = e$. Thus we can rewrite the final line in the sequence of equations above to get $t\ e$, as required. ◀

4.3 Categories of Indexed Types

For every kind κ, we define the category of κ indexed types as follows. The objects are types of kind $\kappa \to *$ and morphisms between F and G are inhabitants of the type $\forall_\kappa \gamma.F\gamma \to G\gamma$. We write $F \Rightarrow G$ as shorthand for this type. Identities and composition in these categories are defined as follows:

$$\begin{array}{lcl@{\qquad\qquad}lcl} id_F & : & F \Rightarrow F & \circ & : & (G \Rightarrow H) \to (F \Rightarrow G) \to (F \Rightarrow H) \\ id_F & = & \Lambda\gamma.\lambda x.\ x & \circ & = & \lambda f\ g.\Lambda\gamma.\lambda x.\ f\ [\gamma]\ (g\ [\gamma]\ x) \end{array}$$

The categories of κ indexed types have all finite products and coproducts (sum types). We only sketch the proof here, which is a straightforward extension of the proof for the existence of non-indexed finite products and coproducts in System F.

▶ **Theorem 10.** *The categories of κ indexed types have all finite products and coproducts.*

Proof. *(Sketch)* Pointwise extension of the corresponding constructions for types of base kind in System F [4, 7]. For example, $A \times B = \lambda\alpha.\ \forall_* \beta.\ (A\alpha \to B\alpha \to \beta) \to \beta$. To prove the required universal properties, parametricity is needed. ◀

4.4 Functors and Initial Algebras

For a kind κ, an endofunctor on the category of κ indexed types consists of a pair (F, fmap_F) of a type F of kind $(\kappa \to *) \to (\kappa \to *)$ and an associated function $\mathit{fmap}_F : \forall_{\kappa\to*}\alpha\beta.(\alpha \Rightarrow \beta) \to (F\alpha \Rightarrow F\beta)$ that preserves identities and composition.

We now show that any such functor has an initial algebra, and so we can define κ indexed inductive types. To do so, we need the higher kinded generalisation of the functional relations previously defined at base kind in Definition 7.

▶ **Definition 11** (Functional Relations at Higher Kind)**.** Given types F, G of kind $\kappa \to *$ (i.e., objects of the category of κ indexed types) and f of type $A \Rightarrow B$ define the *functional relation* $\langle f\rangle \in [\![\kappa \to *]\!]^{\mathcal{R}}(A, B)$ as $\langle f\rangle_{XY} S = \{(x, y) \mid (f\ [X]\ x, y) \in \Delta_B S\}$.

▶ **Lemma 12** (Graph Lemma)**.** *For a functor* (F, fmap_F) *and morphism* $f : A \Rightarrow B$*, then it is the case that* $\Delta_F\langle f\rangle \subseteq \langle \mathit{fmap}_F\ [A]\ [B]\ f\rangle$*.*

Proof. Of course, by the inclusion in the lemma statement, we mean that the inclusion holds for all $X, Y \in [\![\kappa]\!]^{\mathcal{U}}$ and $S \in [\![\kappa]\!]^{\mathcal{R}}(X, Y)$. We know by the parametricity property for fmap_F that $(\mathit{fmap}_F\ [A]\ [B], \mathit{fmap}_F\ [B]\ [B]) \in ((\langle f\rangle \Rightarrow \Delta_B) \to (\Delta_F\langle f\rangle \Rightarrow \Delta_{FB})$. We proceed by supplying the suitably related arguments $(f, id_B) \in \langle f\rangle \Rightarrow \Delta_B$ and then S, to obtain $(\mathit{fmap}_F\ [A]\ [B]\ f\ X, id_{FBY}) \in \Delta_F\langle f\rangle S \to \Delta_{FB}S$. So, if $(x, y) \in \Delta_F\langle f\rangle S$ then $(\mathit{fmap}_F\ [A]\ [B]\ f\ X\ x, y) \in \Delta_{FB}S$, which implies that $(x, y) \in \langle \mathit{fmap}_F\ [A]\ [B]\ f\rangle S$. ◀

We now show that every endofunctor on the categories of κ indexed types has an initial algebra. This result allows us to define κ indexed inductive types within System Fω and use initiality to reason about them. Following the purely category theoretic definition, we define, for a functor (F, fmap_F), an F-algebra to consist of a pair of a type A of kind $\kappa \to *$ and a morphism $k_A : FA \Rightarrow B$. Given two F-algebras A, k_A and B, k_B, an F-algebra homomorphism between them is a morphism $f : A \Rightarrow B$ such that the equation $k_B \circ \mathit{fmap}_F\ [A]\ [B]\ f = f \circ k_A$ holds. An *initial* F-algebra is an F-algebra $\mu F, in_F$ such that for any other F-algebra A, k_A there exists a unique F-algebra homomorphism $\mu F \Rightarrow A$.

We state the existence of initial F-algebras in type theoretic terms as follows. For any functor (F, fmap_F), we require a type $\mu F : \kappa \to *$, along with two combinators:

$$\begin{aligned} \mathsf{in}_F &: F(\mu F) \Rightarrow F \\ \mathsf{fold}_F &: \forall_{\kappa\to*}\rho.(F\rho \Rightarrow \rho) \to (\mu F \Rightarrow \rho) \end{aligned}$$

that satisfy the following $\beta\eta$ equality rules:

$$\frac{A : \kappa \to * \qquad k_A : FA \Rightarrow A \qquad C : \kappa \qquad e : F(\mu F)C}{\mathsf{fold}_F\ [A]\ k_A\ [C]\ (\mathsf{in}_F\ [C]\ e) = k_A\ [C]\ (\mathit{fmap}_F\ [\mu F]\ [A]\ (\mathsf{fold}_F\ [A]\ k_A)\ [C]\ e)}\ (\beta)$$

$$\frac{A : \kappa \to * \qquad k_A : FA \Rightarrow A \qquad f : \mu F \Rightarrow A \qquad f \text{ is an } F\text{-algebra homomorphism}}{f = \mathsf{fold}_F\ [A]\ k_A}\ (\eta)$$

The β equality rule states that, for any F-algebra A, k_A, $\mathsf{fold}_F\ [A]\ k_A$ is an F-algebra homomorphism. The η equality rule states that $\mathsf{fold}_F\ [A]\ k_A$ is the unique F-algebra homomorphism from $\mu F, \mathsf{in}_F$ to A, k_A.

Within System Fω we can implement the above specification for any functor (F, fmap_F) and show that the β equality rule holds. Moreover, using parametricity we can further show that the η equality rule holds. We define

$$\begin{aligned} \mu F &= \lambda\alpha.\forall_{\kappa\to*}\rho.(F\rho \Rightarrow \rho) \to \rho\alpha \\ \mathit{fold}_F &= \Lambda\rho.\lambda f.\Lambda\alpha.\lambda x.\ x\ [\rho]\ f \\ \mathit{in}_F &= \Lambda\gamma.\lambda x.\Lambda\rho.\lambda f.\ f\ [\gamma]\ (\mathit{fmap}_F\ [\mu F]\ [\rho]\ (\mathit{fold}_F\ [\rho]\ f)\ [\gamma]\ x) \end{aligned}$$

▶ **Lemma 13.** *Let (F, fmap_F) be a functor. Given two F algebras $A : \kappa \to *$, $k_A : FA \Rightarrow A$ and $B : \kappa \to *$, $k_B : FB \Rightarrow B$, and an F-algebra homomorphism $h : A \Rightarrow B$, then for any type C of kind κ and $e : (\mu F)C$, we have $h\ [C]\ (e\ [A]\ k_A) = e\ [B]\ k_B$.*

Proof. By e's parametricity, we know that $(e\ [A], e\ [B]) \in (\Delta_F\langle h\rangle \Rightarrow \langle h\rangle) \to \langle h\rangle\Delta_C$. We will apply this pair to (k_A, k_B), so we must show that $(k_A, k_B) \in \Delta_F\langle h\rangle \Rightarrow \langle h\rangle$. Given $X, Y \in [\![\kappa]\!]^{\mathcal{U}}$, $S \in [\![\kappa]\!]^{\mathcal{R}}(X, Y)$ and $(x, y) \in \Delta_F\langle h\rangle S$, we need to demonstrate that $(k_A\ [X]\ x, k_B\ [Y]\ y) \in \langle h\rangle S$. Unfolding the definition of $\langle h\rangle$, this means we need to show that $(h\ [X]\ (k_A\ [X]\ x), k_B\ [Y]\ y) \in \Delta_B S$. Since h is an F-algebra homomorphism, it suffices to show that

$$(k_B\ [X]\ (\mathit{fmap}\ [A]\ [B]\ h\ [X]\ x), k_B\ [Y]\ y) \in \Delta_B S \tag{2}$$

We know by Lemma 12 that $(x, y) \in \langle \mathit{fmap}\ [A]\ [B]\ h\rangle S$. Unfolding this use of a functional relation, this gives $(\mathit{fmap}\ [A]\ [B]\ h\ [X]\ x, y) \in \Delta_{FB}S$. Now we may use the parametricity property of k_B to show (2).

We now have $(e\ [A]\ k_A, e\ [B]\ k_B) \in \langle h\rangle\Delta_C$, and unfolding the definition of $\langle h\rangle$ yields $(h\ [C]\ (e\ [A]\ k_A), e\ [B]\ k_B) \in \Delta_B\Delta_C$. Since $\Delta_B\Delta_C = \Delta_{BC} = (\equiv)$, we have shown that $h\ [C]\ (e\ [A]\ k_A) = e\ [B]\ k_B$, as required. ◀

▶ **Theorem 14.** *The implementations of μF, in_F and fold_F satisfy the $\beta\eta$ equality rules stated above, in the model of Section 3.*

Proof. The β equality rule is seen to hold by expanding the definitions and β reducing. By itself, this ensures that our definition is a weakly initial algebra. For the η equality rule, we unfold fold_F, and η expand to see that we need to prove $f\ [C]\ e = e\ [A]\ k_A$. By Lemma 13, and that f is an F-algebra homomorphism, we have $f\ [C]\ (e\ [\mu F]\ \mathit{in}) = e\ [A]\ k_A$. We now use Lemma 13 again, with an arbitrary B and $k_B : FB \Rightarrow B$, setting $h = \mathit{fold}_F\ [B]\ k_B$ (which we know to be an F-algebra homomorphism by the β-equality rule). This gives us $e\ [\mu F]\ \mathit{in}_F\ [B]\ k_B = e\ [B]\ k_B$, and so, by extensionality, $e\ [\mu F]\ \mathit{in}_F = e$. Thus we have shown $f\ [C]\ e = e\ [A]\ k_A$, as required. ◀

4.5 Generalised Algebraic Data Types

Given that we can encode existential types, products, coproducts, equality types and initial algebras in parametric System Fω, it is now possible to encode Generalised Algebraic Data Types (GADTs) [5], using the encoding of Johann and Ghani [10] into a system with initial algebras and equality types (the same encoding was used implicitly by Cheney and Hinze [5]). For example, the following Haskell declaration:

```
data Z; data S a
data Vec :: * -> * -> * where
  VNil :: Vec a Z;  VCons :: a -> Vec a n -> Vec a (S n)
```

can be encoded, assuming a kinding context containing $Z : *$, $S : * \to *$ and $\alpha : *$, by the initial algebra of the functor $F\rho\alpha = \mathit{Eq}_*\ \alpha\ Z + (\exists_*\eta.\alpha \times \rho\eta \times \mathit{Eq}_*\ \alpha\ (S\eta))$. We have used the equality type to encode the "transmuting" effect of the constructors on the type parameters.

This encoding is not immediately useful within System Fω because we are not guaranteed anything about equalities between types. Specifically, we cannot be sure that the types Z and $S\eta$ are not equal, so it is not possible to directly translate the following Haskell function:

```
head :: Store a (S n) -> a; head (SCons a _) = a
```

where we know `head` is total because `Z` $\neq$ `S n` for all `n`, by Haskell's generative semantics for `data` declarations. We can simulate this by adding the assumption $\mathit{ZisNotS} : \forall_* \eta. \mathit{Eq}\ Z\ (S\eta) \to 0$ to the context, where $0 = \forall_* \alpha.\alpha$ is the encoding of the terminal object. This allows us to define the function within System Fω. In the next section we show how to extend the calculus and model with a data kind of natural numbers, which means that we do not have to abuse types to stand for natural numbers.

5 Extension of System Fω with Additional Kinds

In the previous section, we used abstract type constructors Z and S to simulate natural numbers. However, recent versions of the GHC Haskell compiler have been extended with *data kinds*, which lift some inductive data types up to the type level. Yorgey *et al.* [21] provide the details of this lifting. As a step towards modelling data kinds, we demonstrate how our model can be extended with a kind of natural numbers by interpreting them as a discrete "category without composition".

Syntactically, we extend the grammar of kinds with a new kind of natural numbers: $\kappa ::= \ldots \mid \mathtt{nat}$. We also extend the language of types with two new constants and a kind indexed family of constants, with the following kindings:

$$\mathsf{zero} : \mathtt{nat} \qquad \mathsf{succ} : \mathtt{nat} \to \mathtt{nat} \qquad \mathsf{rec}_\kappa : \kappa \to (\mathtt{nat} \to \kappa \to \kappa) \to \mathtt{nat} \to \kappa$$

with the β-equality rules $\mathsf{rec}_\kappa\ A\ B\ \mathsf{zero} \equiv A$ and $\mathsf{rec}_\kappa\ A\ B\ (\mathsf{succ}\ n) \equiv B\ n\ (\mathsf{rec}_\kappa\ A\ B\ n)$.

Semantically, we define $[\![\mathtt{nat}]\!] = (\mathbb{N}, \lambda n_1, n_2.\{* \mid n_1 = n_2\}, \lambda n.*)$, where $\mathbb{N}$ is the set of all natural numbers and the notation $\{* \mid n_1 = n_2\}$ denotes the set $\{*\}$ when $n_1 = n_2$ and the empty set otherwise. Following Hasegawa, and thinking of the interpretation of kinds as categories without composition, the interpretation of `nat` is "discrete": there are only relations between equal objects. It is straightforward to define the semantic interpretations of the zero, succ and rec_κ constants.

Given the extension of the calculus and model with this new kind, we can use Theorem 14 to define the inductive type of `nat` indexed vectors. We set

$$F\alpha\rho n = (\mathit{Eq}_{\mathtt{nat}}\ n\ \mathsf{zero}) + (\exists_{\mathtt{nat}} n'.\ \alpha \times \rho n' \times \mathit{Eq}_{\mathtt{nat}}\ n\ (\mathsf{succ}\ n'))$$

so that the type of `nat` indexed vectors with elements of type A is given by $\mu(FA)$. It is now possible to write the `head` function within System Fω without any further assumptions: we can write a term that demonstrates that $\mathsf{succ}\ n$ and zero are not equal, using the *subst* function defined in Section 4.1, and type-level computation using rec_κ:

$$\begin{array}{lcl} \mathit{zeroIsNotSucc} & : & \forall_{\mathtt{nat}} n.\ \mathit{Eq}\ \mathsf{zero}\ (\mathsf{succ}\ n) \to 0 \\ \mathit{zeroIsNotSucc} & = & \Lambda n.\lambda e.\ \mathit{subst}_{\mathtt{nat}}\ [\mathsf{zero}]\ [\mathsf{succ}\ n]\ e\ [\mathsf{rec}_*\ 1\ (\lambda n\ \alpha.\ 0)]\ * \end{array}$$

where $1 = \forall\alpha.\alpha \to \alpha$, the standard encoding of the unit type, and $*$ is the unique inhabitant.

6 Conclusions

We have defined a concrete type theoretic model of relationally parametric System Fω (Theorem 3), based on the idea of interpreting kinds as reflexive graphs, due to Hasegawa and Robinson and Rosolini. This model allows us to reason relationally about the standard non-parametric semantics (Theorem 4). We have investigated some of the consequences of our model, and shown that it is possible to define indexed inductive types within System Fω, with an initiality property that allows for reasoning (Theorem 14). We have also shown that it is possible to extend the model with data kinds, such as the natural numbers.

Acknowledgements Thanks to Patricia Johann, Neil Ghani, Alex Simpson, Thorsten Altenkirch and Lars Birkedal for suggestions and comments on this work. This work was funded by EPSRC grant EP/G068917/1.

References

1 R. Atkey. Syntax For Free: Representing Syntax with Binding using Parametricity. In *Typed Lambda Calculi and Applications (TLCA)*, volume 5608 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2009.

2 E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism. *Theor. Comput. Sci.*, 70(1):35–64, 1990.

3 J.-P. Bernardy, P. Jansson, and R. Paterson. Parametricity and dependent types. In *Proc. 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP 2010, pages 345–356, 2010.

4 L. Birkedal and R. E. Møgelberg. Categorical models for Abadi-Plotkin's Logic for Parametricity. *Mathematical Structures in Computer Science*, 15(4):709–772, 2005.

5 J. Cheney and R. Hinze. A lightweight implementation of generics and dynamics. In *Proc. 2002 ACM SIGPLAN Workshop on Haskell*, Haskell '02, pages 90–104, 2002.

6 B. Dunphy and U. S. Redddy. Parametric limits. In *Proc. 19th IEEE Symp. on Logic in Computer Science*, LICS 2004, pages 242–251, 2004.

7 R. Hasegawa. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science*, 4(1):71–109, 1994.

8 R. Hasegawa. Relational limits in general polymorphism. *Publications of the Research Institute for Mathematical Sciences*, 30:535–576, 1994.

9 B. Jacobs. *Categorical Logic and Type Theory.* Elsevier, 1999.

10 P. Johann and N. Ghani. Foundations for structured programming with GADTs. In *Proc. 35th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, POPL 2008, pages 297–308, 2008.

11 M. P. Jones. A System of Constructor Classes: Overloading and Implicit Higher-Order Polymorphism. *J. Funct. Program.*, 5(1):1–35, 1995.

12 A. Moors, F. Piessens, and M. Odersky. Generics of a higher kind. In *Proc. 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2008, pages 423–438, 2008.

13 B. Pierce. *Types and Programming Languages.* MIT Press, 2002.

14 J. C. Reynolds. Types, Abstraction and Parametric Polymorphism. In *Proc. IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 513–523, 1983.

15 E. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. In *Proc. 9th Annual IEEE Symp. on Logic in Computer Science*, LICS 1994, pages 364–371, 1994.

16 A. Rossberg, C. V. Russo, and D. Dreyer. F-ing modules. In *Proc. ACM SIGPLAN International Workshop on Types in Language Design and Implementation*, TLDI 2010, pages 89–102, 2010.

17 I. Takeuti. The theory of parametricity in the lambda cube. Technical Report 1217, Kyoto University, 2001.

18 J. Voigtländer. Free Theorems Involving Type Constructor Classes: Functional Pearl. In *Proc. 14th ACM SIGPLAN International Conference on Functional programming*, ICFP 2009, pages 173–184, 2009.

19 D. Vytiniotis and S. Weirich. Parametricity, type equality, and higher-order polymorphism. *J. Funct. Program.*, 20(2):175–210, 2010.

20 P. Wadler. Theorems for free! In *Proc. Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA'89, pages 347–359, 1989.

21 B. A. Yorgey, S. Weirich, J. Cretin, S. Peyton Jones, D. Vytiniotis, and J. P. Magalhães. Giving Haskell a Promotion. In *Proc. 8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, TLDI '12, pages 53–66, 2012.

Higher-Order Interpretations and Program Complexity*

Patrick Baillot[1] **and Ugo Dal Lago**[2]

1 **CNRS, ENS de Lyon, INRIA, UCBL, Université de Lyon, Laboratoire LIP**
patrick.baillot@ens-lyon.fr

2 **Università di Bologna & INRIA**
dallago@cs.unibo.it

Abstract

Polynomial interpretations and their generalizations like quasi-interpretations have been used in the setting of first-order functional languages to design criteria ensuring statically some complexity bounds on programs [8]. This fits in the area of implicit computational complexity, which aims at giving machine-free characterizations of complexity classes. In this paper, we extend this approach to the higher-order setting. For that we consider the notion of simply-typed term rewriting systems [30], we define higher-order polynomial interpretations for them and give a criterion ensuring that a program can be executed in polynomial time. In order to obtain a criterion flexible enough to validate interesting programs using higher-order primitives, we introduce a notion of polynomial quasi-interpretations, coupled with a simple termination criterion based on linear types and path-like orders.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases implicit complexity, higher-order rewriting, quasi-interpretations

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.62

1 Introduction

The problem of statically analyzing the performance of programs can be attacked in many different ways. One of them consists in verifying *complexity* properties early in the development cycle, when programs are still expressed in high-level languages, like functional or object oriented idioms. And in this scenario, results from an area known as *implicit* computational complexity (ICC in the following) can be useful: they consist in characterizations of complexity classes in terms of paradigmatic programming languages (recursion schemes [25, 6], λ-calculus [26], term rewriting systems [8], etc.) or logical systems (proof-nets, natural deduction, etc.), from which static analysis methodologies can be distilled. Examples are type systems, path-orderings and variations on the interpretation method. The challenge here is defining ICC systems which are not only simple, but also *intensionally* powerful: many natural programs among those with bounded complexity should be recognized as such by the ICC system, i.e., should actually be programs *of* the system.

One of the most fertile direction in ICC is indeed the one in which programs are term rewriting systems (TRS in the following) [8, 9], whose complexity can be kept under control by way of variations of the powerful techniques developed to check termination of TRSs, namely path orderings [16], dependency pairs [28] and the interpretation method [24]. Many

* This work was partially supported by the ARC INRIA project “ETERNAL” and by the ANR-08-BLANC-0211-01 project “COMPLICE”.

Computer Science Logic 2012 (CSL’12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 62–76
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

different complexity classes have been characterized this way, from polynomial time to polynomial space, to exponential time to logarithmic space. And remarkably, many of the introduced characterizations are intensionally very powerful, in particular when the interpretation method is relaxed and coupled with recursive path orderings, like in quasi-interpretations [9].

The cited results indeed represent the state-of-the art in resource analysis for *first-order* functional programs, i.e. when functions are *not* first-class citizens. If the class of programs of interest includes higher-order functional programs, the techniques above can only be applied if programs are either defunctionalized or somehow put in first-order form, for example by applying a translation scheme due to the second author and Simone Martini [15]. However, it seems difficult to ensure in that case that the target first-order programs satisfy termination criteria such as those used in [9]. The article [10] proposed to get around this problem by considering a notion of *hierarchical union* of TRSs, and showed that this technique allows to handle some examples of higher-order programs. This approach is interesting but it is not easy to assess its generality, besides particular examples. In the present work we want to switch to a higher-order interpretations setting, in order to provide a more abstract account of such situations.

We thus propose to generalize TRS techniques to systems of higher-order rewriting, which come in many different flavours [21, 23, 30]. The majority of the introduced higher-order generalizations of rewriting are quite powerful but also complex from a computational point of view, being conceived to model not only programs but also proofs involving quantifiers. As an example, even computing the reduct of a term according to a reduction rule can in some cases be undecidable. Higher-order generalizations of TRS techniques [22, 29], in turn, reflect the complexity of the languages on top of which they are defined. Summing up, devising ICC systems this way seems quite hard.

In this paper, we consider one of the simplest higher-order generalizations of TRSs, namely Yamada's simply-typed term rewriting systems [30] (STTRSs in the following), we define a system of higher-order polynomial interpretations [29] for them and prove that, following [8], this allows to exactly characterize, among others, the class of polynomial time computable functions. We show, however, that this way the class of (higher-order) programs which can be given a polynomial interpretation does not include interesting and natural examples, like `foldr`, and that this problem can be overcome by switching to another technique, designed along the lines of quasi-interpretations [9]. This is the subject of sections 3 and 4 below.

An extended version of this paper with all proofs is available [3].

2 Simply-Typed Term Rewriting Systems

2.1 Definitions and Notations

We recall here the definition of a STTRS, following [30, 2]. We will actually consider a subclass of STTRSs, basically the one of those STTRSs whose rules' left hand side consists in a function symbol applied to a sequence of *patterns*. For first-order rewrite systems this corresponds to the notion of *constructor rewrite system*.

We consider a denumerable set of base types, which we call *data-types*, that we denote $D, E, \ldots$. *Types* are defined by the following grammar:

$$A, B ::= D \;\big|\; A_1 \times \cdots \times A_n \to A.$$

A *functional type* is a type which contains an occurrence of →. Some examples of base types are the type W_n of n-ary words and the type NAT of tally integers.

We denote by $\mathcal{F}$ the set of *function symbols* (or just *functions*), $\mathcal{C}$ the set of *constructors* and $\mathcal{X}$ the set of *variables*. Constructors $\mathbf{c} \in \mathcal{C}$ have a type of the form $D_1 \times \cdots \times D_n \to D$, for $n \geq 0$. For instance W_n has constructors **empty** of type W_n and $\mathbf{c}_1, \ldots, \mathbf{c}_n$ of type $W_n \to W_n$. Functions $\mathtt{f} \in \mathcal{F}$, on the other hand, can have any functional type. Variables $x \in \mathcal{X}$ can have any type. *Terms* are typed and defined by the following grammar:

$$t, t_i := x^A \;\big|\; \mathbf{c}^A \;\big|\; \mathtt{f}^A \;\big|\; (t^{A_1 \times \cdots \times A_n \to A}\; t_1^{A_1} \ldots t_n^{A_n})^A$$

where $x^A \in \mathcal{X}$, $\mathbf{c}^A \in \mathcal{C}$, $\mathtt{f}^A \in \mathcal{F}$. We denote by $\mathcal{T}$ the set of all terms. Observe how application is primitive and is in general treated differently from other function symbols. This is what makes STTRSs different from ordinary TRSs. $FV(t)$ is the set of variables occurring in t. t is closed iff $FV(t) = \emptyset$.

To simplify the writing of terms we will often elide their type. We will also write $(t\ \overline{s})$ for $(t\ s_1 \ldots s_n)$. Therefore any term t is of the form $(\ldots((\alpha\ \overline{s_1})\ \overline{s_2}) \ldots \overline{s_k})$ where $k \geq 0$ and $\alpha \in \mathcal{X} \cup \mathcal{C} \cup \mathcal{F}$. We will also use the following convention: any term t of the form $(\ldots((s\ \overline{s_1})\ \overline{s_2}) \ldots \overline{s_k})$ will be written $\llparenthesis s\ \overline{s_1}\ \ldots \overline{s_k} \rrparenthesis$ or $\llparenthesis s\ s_{11}\ \ldots s_{1n_1} \ldots s_{k1} \ldots s_{kn_k} \rrparenthesis$. Observe however that, e.g., if t has type $A_1 \times A_2 \to (B_1 \times B_2 \to B)$, t_i has type A_i for $i = 1, 2$, s_i has type B_i for $i = 1, 2$, then both $(t\ t_1\ t_2)$ and $((t\ t_1\ t_2)\ s_1\ s_2)$ are well-typed (with type $B_1 \times B_2 \to B$ and B, respectively), but $(t\ t_1)$ and $(t\ t_1\ t_2\ s_1)$ are not well-typed. We define the size $|t|$ of a term t as the number of symbols (elements of $\mathcal{F} \cup \mathcal{C} \cup \mathcal{X}$) it contains. We denote $t\{x/s\}$ the substitution of term s for x in t.

A *pattern* is a term generated by the following grammar:

$$p, p_i := x^A \;\big|\; (\mathbf{c}^{D_1 \times \ldots \times D_n \to D}\; p_1^{D_1} \ldots p_n^{D_n}).$$

$\mathcal{P}$ is the set of all patterns. Observe that patterns of functional type are necessarily variables. We consider *rewriting rules* in the form $t \to s$ such that:

1. t and s are terms of the same type A, $FV(s) \subseteq FV(t)$, and any variable appears at most once in t;
2. t must have the form $\llparenthesis \mathtt{f}\ \overline{p_1} \ldots \overline{p_k} \rrparenthesis$ where each $\overline{p_i}$ for $i \in 1, \ldots, k$ consists of patterns only. The rule is said to be a rule *defining* $\mathtt{f}$, while the total number of patterns in $\overline{p_1}, \ldots, \overline{p_k}$ is the *arity* of the rule.

Now, a *simply-typed term rewriting system* (STTRS in the following) is a set R of non-overlapping rewriting rules such that for every function symbol $\mathtt{f}$, every rule for $\mathtt{f}$ has the same arity, which is said to be the arity *of* $\mathtt{f}$. A *program* $\mathtt{P} = (\mathtt{f}, R)$ is given by a STTRS R and a chosen function symbol $\mathtt{f} \in \mathcal{F}$.

In the next section, a notion of reduction will be given which crucially relies on the concept of a value. More specifically, only values will be passed as arguments to functions. Formally, we say that a term is a *value* if either:

1. it has a type D and is in the form $(\mathbf{c}\ v_1 \ldots v_n)$, where $v_1, \ldots, v_n$ are themselves values;
2. or it has functional type A and is of the form $\llparenthesis \mathtt{f}\ v_1\ \ldots\ v_n \rrparenthesis$, where the terms in $v_1, \ldots\ v_n$ are themselves values and n is *strictly* smaller than the arity of $\mathtt{f}$.

Condition 2 is reminiscent of the λ-calculus, where an abstraction is a value. We denote values as v, u and the set of all values as $\mathcal{V}$.

2.2 STTRSs: Dynamics

The evaluation of terms will be formalized by a rewriting relation. Before that we need to introduce notions of substitution and unification.

A substitution σ is a mapping from variables to values with a finite domain, and such that $\sigma(x^A)$ has type A. A substitution σ is extended in the natural way to a function from $\mathcal{T}$ to itself, that we shall also write σ. The image of a term t under the substitution σ is denoted $t\sigma$. *Contexts* are defined as terms but with the proviso that they contain exactly one occurrence of a special constant $\bullet^A$ (*hole*) having type A. They are denoted as $\mathcal{C}$, $\mathcal{D}$ ... If $\mathcal{C}$ is a context with hole $\bullet^A$, and t is a term of type A, then $\mathcal{C}\{t\}$ is the term obtained from $\mathcal{C}$ by replacing the occurrence of $\bullet^A$ by t. Consider a STTRS R. We say that s reduces to t in call-by-value, denoted as $s \to_R t$, if there exists a rule $l \to r$ of R, a context $\mathcal{C}$ and a substitution σ such that $l\sigma$ is a closed term, $s = \mathcal{C}\{l\sigma\}$ and $t = \mathcal{C}\{r\sigma\}$. When there is no ambiguity on R, we simply write $\to$ instead of $\to_R$.

2.3 Typed λ-calculi as STTRSs

Please notice that one of the advantages of STTRSs over similar formalisms (like [23]) is precisely the simplicity of the underlying unification mechanism, which does not involve any notion of binding and is thus computationally simpler than higher-order matching. There is a price to pay in terms of expressivity, obviously. The choice of the STTRS framework as higher-order calculus is *not* too restrictive, however: one can show that typed λ-calculi equipped with *weak call-by-value reduction* can be seen as STTRSs. This is achieved using ideas developed for encodings of the λ-calculus into first-order term rewrite systems [15]. In particular, abstractions become function symbols, in the spirit of λ-lifting. More about these embeddings can be found in [3], where encodings of PCF and Gödel's T are described in detail.

3 Higher-Order Polynomial Interpretations

We want to demonstrate how first-order rewriting-based techniques for ICC can be adapted to the higher-order setting. Our goal is to devise criteria ensuring complexity bounds on programs of *first-order type* possibly containing subprograms of *higher-order types*. A typical application will be to find out under which conditions a higher-order functional program such as *e.g.* `map`, `iteration` or `foldr`, fed with a (first-order) polynomial time program produces a polynomial time program.

As a first illustrative step we consider the approach based on polynomial interpretations from [8], which offers the advantage of simplicity. We thus build a theory of *higher-order polynomial interpretations* for STTRSs. It starts from a particular concrete instantiation of the methodology proposed in [30] for proving termination by interpretation, on which we prove additional properties in order to obtain polynomial time complexity bounds.

Higher-order polynomials (HOPs) take the form of terms in a typed λ-calculus whose only base type is that of natural numbers. To each of those terms can be assigned a strictly monotonic function in a category $\mathbb{FSPOS}$ with products and functions. So, the whole process can be summarized by the following diagram:

$$\begin{tikzcd}\text{STTRSs} \arrow[r, "{[\cdot]}"] & \text{HOPs} \arrow[r, "{[\![\cdot]\!]}"] & \mathbb{FSPOS}\end{tikzcd}$$

3.1 Higher-Order Polynomials

Let us consider types built from a base type $\mathbf{N}$:

$$A, B ::= \mathbf{N} \bigm| A \to A.$$

The expression $A^n \to B$ stands for the type $\underbrace{A \to \ldots \to A}_{n \text{ times}} \to B$. Let C_P be the following set of constants: $C_P = \{+ : \mathbf{N}^2 \to \mathbf{N}, \times : \mathbf{N}^2 \to \mathbf{N}\} \cup \{\overline{n} : \mathbf{N} \mid \ n \in \mathbb{N}^\star\}$. Observe that in C_P we have constants of type $\mathbf{N}$ only for *strictly* positive integers. We consider the following grammar of Church-typed terms:

$$M := x^A \ \mid \ \mathbf{c}^A \mid (M^{A\to B} N^A)^B \mid (\lambda x^A.M^B)^{A \to B},$$

where $\mathbf{c}^A \in C_P$ and *in* $(\lambda x^A.M^B)$ *we require that* x *occurs free in* M. A *higher-order polynomial* (HOP) is a term of this grammar which is in β-normal form. We use an infix notation for $+$ and $\times$. We assume given the usual set-theoretic interpretation of types and terms, denoted as $[\![A]\!]$ and $[\![M]\!]$: if M has type A and $FV(M) = \{x_1^{A_1}, \ldots, x_n^{A_n}\}$, then $[\![M]\!]$ is a map from $[\![A_1]\!] \times \ldots \times [\![A_n]\!]$ to $[\![A]\!]$. We denote by $\equiv$ the equivalence relation which identifies terms which denote the same function, e.g. we have: $\lambda x.(\overline{2} \times ((\overline{3} + x) + y)) \equiv \lambda x.(\overline{6} + (\overline{2} \times x + \overline{2} \times y))$. Noticeably, even if HOPs can be built using higher-order functions, the first order fragment only contains polynomials:

▶ **Lemma 3.1.** *If M is a HOP of type $\mathbf{N}^n \to \mathbf{N}$ and such that $FV(M) = \{y_1 : \mathbf{N}, \ldots, y_k : \mathbf{N}\}$, then the function $[\![M]\!]$ is bounded by a polynomial function.*

3.2 Semantic Interpretation

Now, we consider a subcategory $\mathbb{FSPOS}$ of the category $\mathbb{SPOS}$ of strict partial orders as objects and *strictly monotonic* total functions as morphisms. Objects of $\mathbb{FSPOS}$ are freely generated as follows:

- $\mathcal{N}$ is the domain of *strictly positive* integers, equipped with the natural strict order $\prec_{\mathcal{N}}$,
- if σ, τ are objects, then $\sigma \times \tau$ is obtained by the product ordering,
- $\sigma \to \tau$ is the set of strictly monotonic total functions from σ to τ, equipped with the following strict order: $f \prec_{\sigma\to\tau} g$ if for any a of σ we have $f(a) \prec_\tau g(a)$.

Actually we will also need to compare the semantics of terms which do not have the same free variables. For that we define: if $f \in \sigma_1 \times \ldots \times \sigma_n \to \tau$, $g \in \sigma_1 \times \ldots \times \sigma_m \to \tau$ and $n \leq m$, then: $f \prec g$ if $\forall a_1 \in \sigma_1, \ldots \forall a_m \in \sigma_m,\ f(a_1, \ldots, a_n) \prec_\tau g(a_1, \ldots, a_m)$.

$\mathbb{FSPOS}$ is a subcategory of $\mathbb{SET}$ with all the necessary structure to interpret types and terms. $[\![A]\!]_\prec$ denotes the semantics of A as an object of $\mathbb{FSPOS}$: we choose to set $[\![\mathbf{N}]\!]_\prec = \mathcal{N}$, while $[\![A_1 \times \ldots \times A_n \to A]\!]_\prec$ is $[\![A_1]\!]_\prec \times \ldots \times [\![A_n]\!]_\prec \to [\![A]\!]_\prec$. Let M be a HOP of type A with free variables $x_1^{A_1}, \ldots, x_n^{A_n}$. Then for every $e \in [\![A_1 \times \ldots \times A_n]\!]_\prec$, there is a naturally defined $f \in [\![A]\!]_\prec$. Moreover, this correspondence is strictly monotone and thus defines an element of $[\![A_1 \times \ldots \times A_n \to A]\!]_\prec$ which we denote as $[\![M]\!]_\prec$.

3.3 Assignments and Polynomial Interpretations

We consider $\mathcal{X}$, $\mathcal{C}$ and $\mathcal{F}$ as in Sect. 2. To each variable x^A we associate a variable $\underline{x}^{\underline{A}}$ where $\underline{A}$ is obtained from A by replacing each occurrence of base type by the base type $\mathbf{N}$ and by curryfication. We will sometimes write x (resp. A) instead of $\underline{x}$ (resp. $\underline{A}$) when it is clear from the context.

An *assignment* $[\,\cdot\,]$ is a map from $\mathcal{C} \cup \mathcal{F}$ to HOPs such that if $f \in \mathcal{C} \cup \mathcal{F}$ has type A, $[f]$ is a closed HOP of type $\underline{A}$. Now, for $t \in \mathcal{T}$ of type A, we define an HOP $[t]$ of type $\underline{A}$ by induction on t:

- if $t = x \in \mathcal{X}$, then $[t]$ is $\underline{x}$;
- if $t \in \mathcal{C} \cup \mathcal{F}$, $[t]$ is already defined;

- otherwise, if $t = (t_0\, t_1 \dots t_n)$ then $[t] \equiv (\dots([t_0][t_1])\dots[t_n])$.

Observe that in practice, computing $[t]$ will in general require to do some β-reduction steps.

Now, we say that an assignment $[\cdot]$ is a *higher polynomial interpretation* or simply a *polynomial interpretation* for a STTRS R iff for every $l \to r \in R$, we have that $[\![r]\!]_\prec \prec [\![l]\!]_\prec$. Note that in the particular case where the program only contains first-order functions, this notion of polynomial interpretation coincides with the classical one for first-order TRSs. In the following, we assume that $[\cdot]$ is a polynomial interpretation for R. A key property is the following, which tells us that the interpretation of terms strictly decreases along any reduction step:

▶ **Lemma 3.2.** *If $s \to t$, then $[\![t]\!]_\prec \prec [\![s]\!]_\prec$.*

As a consequence, the interpretation of terms (of base type) is itself a bound on the length of reduction sequences:

▶ **Proposition 3.3.** *Let t be a closed term of base type D. Then $[t]$ has type $\mathbf{N}$ and any reduction sequence of t has length bounded by $[\![t]\!]_\prec$.*

3.4 A Complexity Criterion

Proving a STTRS to have a polynomial interpretation is not enough to guarantee its time complexity to be polynomially bounded. To ensure that, we need to impose some constraints on the way constructors are interpreted.

We say that the assignment $[\cdot]$ is *additive* if any constructor $\mathbf{c}$ of type $D_1 \times \dots \times D_n \to D$, where $n \geq 0$, is interpreted by a HOP $M_\mathbf{c}$ whose semantic interpretation $[\![M_\mathbf{c}]\!]_\prec$ is a polynomial function of the form: $p(y_1,\dots,y_n) = \sum_{i=1}^{n} y_i + \gamma_\mathbf{c}$, with $\gamma_\mathbf{c} \geq 1$. Additivity ensures that the interpretation of first-order values is proportional to their size:

▶ **Lemma 3.4.** *Let $[\cdot]$ be an additive assignment. Then there exists $\gamma \geq 1$ such that for any value v of type D, we have $[\![v]\!]_\prec \leq \gamma \cdot |v|$.*

A function $f : (\{0,1\}^*)^m \to \{0,1\}$ is said to be *representable* by a STTRS R if there is a function symbol $\mathtt{f}$ of type $(W_2)^n \to W_2$ in R which computes f in the obvious way. We can now state the main result about polynomial interpretations:

▶ **Theorem 3.5** (Polynomial Bound). *Let R be a STTRS with an additive polynomial interpretation $[\cdot]$. Consider a function symbol $\mathtt{g}$ of type $(W_2)^n \to W_2$. Then, there exists a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that, for any $w_1,\dots,w_n \in \{0,1\}^\star$, any reduction of $(\mathtt{g}\ \underline{w_1}\dots\underline{w_n})$ has length bounded by $p(|w_1|,\dots,|w_n|)$. This holds more generally for $\mathtt{g}$ of type $D_1 \times \dots \times D_n \to D$.*

A key property we use in the proof of Theorem 3.5 is that function symbols of first-order type are interpreted by functions which are bounded by a polynomial. This is a consequence of Lemma 3.1 and is the main reason why we have chosen to define the interpretation of terms via HOPs.

▶ **Corollary 3.6.** *The functions on binary words representable by STTRSs admitting an additive polynomial interpretation are exactly the polytime functions.*

The fact that all polynomial time functions can be represented follows from [8], essentially because our setting subsumes that of this paper.

The results we have just described are quite robust: one is allowed to extend C_P with new combinators, provided their set-theoretic semantics are strictly monotone functions for which Lemma 3.1 continues to hold. However, the class of polynomial time STTRSs which can be proved such by way of higher-order polynomial interpretations is quite restricted, as we are going to argue.

3.5 Examples

Consider the STTRS defined by the following rules:

$$\begin{aligned}((\mathtt{map}\ f)\ \mathbf{nil}^D) &\to \mathbf{nil}^E;\\ ((\mathtt{map}\ f)\ (\mathbf{cons}^D\ x\ xs)) &\to (\mathbf{cons}^E\ (f\ x)\ ((\mathtt{map}\ f)\ xs));\end{aligned}$$

with the following types:

$$\begin{aligned}f &: D \to E; & \mathtt{map} &: (D \to E) \to L(D) \to L(E);\\ \mathbf{nil}^D &: L(D); & \mathbf{cons}^D &: D \times L(D) \to L(D);\\ \mathbf{nil}^E &: L(E); & \mathbf{cons}^E &: E \times L(E) \to L(E).\end{aligned}$$

Here $D, E, L(D), L(E)$ are base types. For simplicity we use just one **cons** and one **nil** notation for both types D and E. The interpretation below was given in [30] for proving termination, but here we show that it also gives a polynomial time bound. Now, we choose the following assignment of HOPs:

$$\begin{aligned}[\mathbf{nil}] &= 2 : \mathbf{N};\\ [\mathbf{cons}] &= \lambda n.\lambda m.(n+m+1) : \mathbf{N} \to \mathbf{N} \to \mathbf{N};\\ [\mathtt{map}] &= \lambda\phi.\lambda n.n \times (\phi\ n) : (\mathbf{N} \to \mathbf{N}) \to \mathbf{N} \to \mathbf{N}.\end{aligned}$$

One can check that the condition $[\![r]\!]_\prec \prec [\![l]\!]_\prec$ holds for both rules above. We thus have an additive polynomial interpretation for `map`, therefore Corollary 3.6 applies and we can conclude that for any `f` also satisfying the criterion, (`map f`) computes a polynomial time function.

Now, one might want to apply the same method to an iterator `iter`, of type $(D \to D) \times D \to NAT \to D$, which when fed with arguments f, d, n iterates f exactly n times starting from d. However there is no additive polynomial interpretation for this program. Actually, this holds for very good reasons: `iter` can produce an exponential-size function when fed with a fast-growing polynomial time function, *e.g.* `double` $: NAT \to NAT$.

One way to overcome this issue could be to show that `iter` *does* admit a valid polynomial interpretation, *provided* its domain is restricted to some particular functions, admitting a *small* polynomial interpretation, of the form $\lambda n.(n+c)$, for some constant c. This could be enforced by considering a refined type systems for HOPs. But the trouble is that there are very few programs which admit a polynomial interpretation of this form! Intuitively the problem is that polynomial interpretations need to bound simultaneously the execution time *and* the size of the intermediate values. In the sequel we will see how to overcome this issue.

4 Beyond Interpretations: Quasi-Interpretations

The previous section has illustrated our approach. However we have seen that the intensional expressivity of higher-order polynomial interpretations is too limited. In the first-order setting this problem has been overcome by decomposing into *two distinct conditions* the role

$$\frac{\mathtt{f}^{A} \in \mathcal{NF}}{\Gamma \mid \Delta \vdash \mathtt{f} : A} \qquad \frac{\mathbf{c}^{A} \in \mathcal{C}}{\Gamma \mid \Delta \vdash \mathbf{c} : A} \qquad \frac{}{\Gamma \mid x : A, \Delta \vdash x : A} \qquad \frac{}{x : D, \Gamma \mid \Delta \vdash x : D}$$

$$\frac{\begin{array}{c}\mathtt{f}^{A_1,\ldots,A_n \to B} \in \mathcal{RF}, \text{with arity } n \\ \Gamma \mid \emptyset \vdash s_i : A_i\end{array}}{\Gamma \mid \Delta \vdash (\!(\mathtt{f}\ s_1 \ldots s_n)\!) : B} \qquad \frac{\begin{array}{c}\Gamma \mid \Delta \vdash t : A_1 \times \ldots \times A_n \to B \\ \Gamma \mid \Delta_i \vdash s_i : A_i\end{array}}{\Gamma \mid \Delta, \Delta_1, \ldots, \Delta_n \vdash (t\ s_1 \ldots s_n) : B}$$

Figure 1 A Linear Type System for STTRS terms.

played by polynomial interpretations [27, 9]: (i) a termination condition, (ii) a condition enforcing a bound on the size of values occurring during the computation. In [9], this has been implemented by using: for (i) some specific recursive path orderings, and for (ii) a notion of *quasi-interpretation*. We will examine how this methodology can be extended to the higher-order setting.

The first step will take the form of a termination criterion defined by a linear type system for STTRSs together with a path-like order, to be described in Section 4.1 below. The second step consists in shifting from a semantic world of strictly monotonic functions to one of monotonic functions. This corresponds to a picture like the following, and is the subject of sections 4.2 and 4.3.

$$\begin{tikzcd} \text{STTRSs} \arrow[r, "{[\cdot]}"] & \text{HOMPs} \arrow[r, "{[\![\cdot]\!]}"] & \mathbb{FPOS} \end{tikzcd}$$

4.1 The Termination Criterion

The termination criterion has two ingredients: a typing ingredient and a syntactic ingredient, expressed using an order $\sqsubset$ on the function symbols. Is it restrictive for expressivity? The syntactic ingredient is fairly expressive, since it allows to validate all programs coming from System T (see [3] for more details). As to the full termination criterion, including the typing ingredient, it is general enough to embed Hofmann's SLR [19] and LFPL [20], which are distinct restrictions of System T capturing polytime functions.

Formally, introducing the typing ingredient requires splitting the class $\mathcal{F}$ into two disjoint classes $\mathcal{RF}$ and $\mathcal{NF}$. The intended meaning is that functions in $\mathcal{NF}$ will not be defined in a recursive way, while functions in $\mathcal{RF}$ can. We further assume given a strict order $\sqsubset$ on $\mathcal{F}$ which is well-founded. If t is a term, $t \sqsubset \mathtt{f}$ means that for any $\mathtt{g}$ occurring in t we have $\mathtt{g} \sqsubset \mathtt{f}$. The rules of a linear type system for STTRS terms are in Figure 1. In a judgement $\Gamma \mid \Delta \vdash t : A$, the sub-context Δ is meant to contain linear variables while Γ is meant to contain non-linear variables.

A STTRS *satisfies the termination criterion* if every rule $(\!(\mathtt{f}\ \overline{p_1} \ldots \overline{p_k})\!) \to s$ satisfies:

1. either $\mathtt{f} \in \mathcal{RF}$, there are a term r and sequences of patterns $\overline{q_1}, \ldots, \overline{q_k}$ such that $s = r\{x/(\!(\mathtt{f}\ \overline{q_1} \ldots \overline{q_k})\!)\}$, we have $\Gamma \mid x : B, \Delta \vdash r : B$, $r \sqsubset \mathtt{f}$, for any i, j, $q_{i,j}$ is subterm of $p_{i,j}$ and there exist i_0, j_0 s.t. $q_{i_0,j_0} \neq p_{i_0,j_0}$;
2. or we have $\Gamma \mid \Delta \vdash s : B$ and $s \sqsubset \mathtt{f}$.

Observe that because of the typability condition in 1., this termination criterion implies that there is at most one recursive call in the right-hand-side s of a rule.

Given a term t, its *definitional depth* is the maximum, over any function symbol $\mathtt{f}$ ap-

$$\begin{aligned}
\mathcal{TS}_v(X) =& 1, \quad \text{if } v \text{ is a first order value,}\\
\mathcal{TS}_{((\texttt{f}\ t_1 \ldots t_n))}(X) =& 1 + \left(\sum_{\substack{t_j\ \in \mathcal{FO} \\ j \leq \mathit{arity}(\texttt{f})}} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{\substack{t_j\ \in \mathcal{HO} \\ j \leq \mathit{arity}(\texttt{f})}} n \cdot X \cdot \mathcal{TS}_{t_j}(X) \right)\\
&+ \left(\sum_{j \geq \mathit{arity}(\texttt{f})+1} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{s \in \mathcal{R}(\texttt{f})} n \cdot X \cdot \mathcal{TS}_s(X) \right), \quad \text{if } \texttt{f} \in \mathcal{RF};\\
\mathcal{TS}_{((\texttt{f}\ t_1 \ldots t_n))}(X) =& 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right) + \left(\sum_{s \in \mathcal{R}(\texttt{f})} \mathcal{TS}_s(X) \right), \quad \text{if } \texttt{f} \in \mathcal{NF};\\
\mathcal{TS}_{((\texttt{c}\ t_1 \ldots t_n))}(X) =& 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right);\\
\mathcal{TS}_{((x\ t_1 \ldots t_n))}(X) =& 1 + \left(\sum_{1 \leq j \leq n} \mathcal{TS}_{t_j}(X) \right).
\end{aligned}$$

Figure 2 The Definition of $\mathcal{TS}_{(\cdot)}(X)$.

pearing in t, of the length of the longest descending $\sqsubset$-chain starting from $\texttt{f}$. The definitional depth of t is denoted as $\partial(t)$. By a standard reducibility argument, one can prove that every term of a STTRS satisfying the termination criterion is strongly normalizing (see again [3] for the details).

In the rest of this section, we show that all that matters for the time complexity of STTRSs satisfying the termination criterion is the size of first-order values that can possibly appear along the reduction of terms. In other words, we are going to prove that if the latter is bounded, then the complexity of the starting term is known, modulo a fixed polynomial. Showing this lemma, which will be crucial in the following, requires introducing many auxiliary definitions and results.

Given a term t and a natural number $n \in \mathbb{N}$, n is said to be *a bound of first order values for* t if for every reduct s of t, if s contains a first-order value v, then $|v| \leq n$. Suppose a function symbol $\texttt{f}$ takes n base arguments. Then $\texttt{f}$ is said to have *base values bounded by a function* $q : \mathbb{N}^n \rightarrow \mathbb{N}$ if $(\texttt{f}\ t_1 \ldots t_n)$ has $q(|t_1|, \ldots, |t_n|)$ as a bound of its first-order values whenever $t_1, \ldots, t_n$ are first-order values. Given a function symbol $\texttt{f}$, $\mathcal{R}(\texttt{f})$ denotes the set of terms appearing in the right-hand side of rules for $\texttt{f}$, not taking into account recursive calls. For every term t, define its *space-time weight* as a polynomial $\mathcal{TS}_t(X)$ on the indeterminate X, by induction on $(\partial(t), |t|)$, following the lexicographic order, as in Figure 2. We denote here by $\mathcal{FO}$ (resp. $\mathcal{HO}$) the arguments t_j of $\texttt{f}$ of base type (resp. functional type). The *collapsed size* $||t||$ of a term t is its size, where however all first-order values count for 1. We define a rewrite relation $\Rightarrow$ which is like $\rightarrow$, except that whenever a recursive function symbol is unfolded, it is unfolded *completely* in just one rewrite step.

We are now ready to explain why the main result of this section holds. First of all, $\mathcal{TS}_t(X)$ is an upper bound on the collapsed size of t, a result which can be proved by induction on t:

▶ **Lemma 4.1.** *For every $n \geq 1$ and for every t, $\mathcal{TS}_t(n) \geq ||t||$.*

Moreover, $\mathcal{TS}_t(X)$ decreases along any $\Rightarrow$ step if X is big enough:

▶ **Lemma 4.2.** *If n is a bound of first-order values for t, and $t \Rightarrow s$, then $\mathcal{TS}_t(n) > \mathcal{TS}_s(n)$.*

It is now easy to reach our goal:

▶ **Proposition 4.3.** *Suppose that R satisfies the termination criterion. Moreover, suppose that* `f` *has base values bounded by a function $q : \mathbb{N}^n \to \mathbb{N}$. Then, there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that if $t_1, \ldots, t_n$ are first-order values and $(\mathtt{f}\ t_1 \ldots t_n) \to^m s$, then $m, |s| \leq p(q(|t_1|, \ldots, |t_n|))$.*

To convince yourself that linearity is needed to get a result like Proposition 4.3, consider the following STTRS, whose terms cannot be typed in our linear type system:

$$\begin{array}{lcc}
((\mathtt{comp}\ x\ y)\ z) & \to & (x\ (y\ z)) \\
(\mathtt{autocomp}\ x) & \to & (\mathtt{comp}\ x\ x) \\
(\mathtt{id}\ x) & \to & x \\
(\mathtt{expid}\ \mathbf{0}) & \to & \mathtt{id} \\
(\mathtt{expid}\ (\mathtt{s}\ x)) & \to & (\mathtt{autocomp}\ (\mathtt{expid}\ x))
\end{array}$$

Both `id` and $(\mathtt{expid}\ t)$ (for every value t of type NAT) can be given type $NAT \to NAT$. Actually, they all are the same function, extensionally. But try to see what happens if `expid` is applied to natural numbers of growing sizes: there is an exponential blowup going on which does not find any counterpart in first-order values.

4.2 Higher-Order Max-Polynomials

We want to refine the type system for higher-order polynomials, in order to be able to use types to restrict the domain of functionals. The grammar of types is now the following one:

$$S ::= \mathbf{N} \mid S \multimap S; \qquad A ::= S \mid A \to A.$$

Types of the first (resp. second) grammar are called linear types (resp. types) and denoted as $R, S \ldots$ (resp. $A, B, C \ldots$). The linear function type $\multimap$ is a subtype of $\to$, i.e., one can define a relation $\sqsubseteq$ between types by stipulating that $S \multimap R \sqsubseteq S \to R$ and by closing the rule above in the usual way, namely by imposing that $A \to B \sqsubseteq C \to E$ whenever $C \sqsubseteq A$ and $B \sqsubseteq E$.

We now consider the following new set of constructors:

$$D_P = \{+ : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}, max : \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}, \times : \mathbf{N} \to \mathbf{N} \to \mathbf{N}\} \cup \{\overline{n} : \mathbf{N} \mid n \in \mathbb{N}^\star\},$$

and we define the following grammar of Church-typed terms

$$M := x^A \mid \mathbf{c}^A \mid (M^{A \to B} N^A)^B \mid (\lambda x^A . M^B)^{A \to B} \mid (M^{S \multimap R} N^S)^R \mid (\lambda x^S . M^R)^{S \multimap R}$$

where $\mathbf{c}^A \in D_P$. We also require that:

- in $(\lambda x^A . M^B)^{A \to B}$, the variable x^A occurs *at least once* in M^B;
- in $(\lambda x^S . M^R)^{S \multimap R}$, the variable x^S occurs *exactly once* in M^R and in linear position (i.e., it cannot occur on the right-hand side of an application $N^{A \to B} L^A$).

One can check that this class of Church-typed terms is preserved by β-reduction. A *higher-order max-polynomial* (HOMP) is a term as defined above and which is in β-normal form. We define the following objects and constructions on objects:

- $\mathcal{N}$ is the domain of *strictly positive* integers, equipped with the natural partial order, denoted here $\leq_{\mathcal{N}}$,

- if σ, τ are objects, then $\sigma \times \tau$ is obtained by the product ordering,
- $\sigma \Rightarrow \tau$ is the set of monotonic total functions from σ to τ, equipped with the extensional order: $f \leq_{\sigma \Rightarrow \tau} g$ if for any a of σ we have $f(a) \leq_{\tau} g(a)$.

This way, one obtains a subcategory $\mathbb{FPOS}$ of the category $\mathbb{POS}$ with partial orders as objects and *monotonic* total functions as morphisms. As before with $\prec$ we define $\leq$ so as to compare the semantics of terms which do not have the same free variables.

In order to interpret the $\multimap$ construction in this category we introduce a notion of *size*. A size is a (finite) multiset of elements of $\mathbb{N}$. The empty multiset will be denoted $\emptyset$. Given a multiset $\mathcal{S}$, we denote by $\max \mathcal{S}$ its maximal element and by $\sum \mathcal{S}$ the sum of its elements. By convention $\max \emptyset = \sum \emptyset = 0$. Now, given an object σ of the category $\mathbb{FPOS}$, we say that an element $e \in \sigma$ *admits a size* in the following cases:

- If σ is $\mathcal{N}$, then e is an integer n, and $\mathcal{S}$ is a size of e iff we have: $\max \mathcal{S} \leq n \leq \sum \mathcal{S}$.
- If $\sigma = \sigma_1 \times \cdots \times \sigma_n$, then $\mathcal{S}$ is a size of $e = (e_1, \ldots, e_n)$ iff there exists for any $i \in \{1, \ldots, n\}$ a multiset $\mathcal{S}_i$ which is a size of e_i, and such that $\mathcal{S} = \cup_{i=1}^{n} \mathcal{S}_i$.
- If $\sigma = \tau \Rightarrow \rho$, then $\mathcal{S}$ is a size of e iff for any f of τ which has a size $\mathcal{T}$, $\mathcal{S} \cup \mathcal{T}$ is a size of $e(f)$. $\tau \rightarrowtriangle \rho$ is the subset of all those functions in σ which admit a size.

We denote by $[\![A]\!]_{\leq}$ the semantics of A as an object of $\mathbb{FPOS}$, where $\mathbf{N}$ is mapped to $\mathcal{N}$, $\rightarrow$ is mapped to $\Rightarrow$ and $\multimap$ to $\rightarrowtriangle$. As for HOPs, any HOMP M can be naturally interpreted as a monotonic function between the appropriate partial orders, which we denote by $[\![M]\!]_{\leq}$. We will speak of the *size* of an HOMP M, by which we mean a size of its interpretation $[\![M]\!]_{\leq}$. Note that not all terms admit a size. For instance $\times : \mathbf{N} \rightarrow \mathbf{N} \rightarrow \mathbf{N}$ does not admit a size. If M reduces to N, then they have the same sizes, if any. Let us examine some examples:

- The term $\overline{n}$ of type $\mathbf{N}$ admits the following sizes: $[n]$, $\underbrace{[1, \ldots, 1]}_{k \text{ times}}$ with $k \geq n$, and more generally $[n_1, \ldots, n_k]$ such that $\forall i \in \{1, k\}$, $n_i \leq n$ and $\sum_{i=1}^{k} n_i \geq n$.
- The terms $\max$ and $+$ of type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$ admit as size $\emptyset$ or $[0]$.
- The terms $\lambda x.(x + 3)$, $\lambda x. \max(x, 3)$ of type $\mathbf{N} \multimap \mathbf{N}$ both have size 3.
- The term $\lambda f.(f\ 2\ 3)$ of type $(\mathbf{N} \multimap \mathbf{N}) \multimap \mathbf{N}$ has size $[2, 3]$.

Actually, if we consider first-order terms with types of the form $\mathbf{N} \multimap \ldots \multimap \mathbf{N}$, it is sufficient to consider singletons as sizes. If we only wanted to deal with these terms we could thus use integers for sizes, instead of multisets. Non-singleton multisets only become necessary when we move to higher-order types, as in the last example above:

▶ **Proposition 4.4.** *If M is a HOMP of type $\mathbf{N}^k \multimap \mathbf{N}$ with free variables $x_1 : \mathbf{N}, \ldots, x_n : \mathbf{N}$ which are linear in M, then it admits a size of the form $[m]$ where $m \in \mathbb{N}$.*

The following will be useful to obtain the Subterm Property:

▶ **Lemma 4.5.** *For every type A there is a closed HOMP of type A.*

4.3 Higher-Order Quasi-Interpretations

Now, a HOMP assignment $[\,\cdot\,]$ is defined by: for any $f^A \in \mathcal{X}$ (resp. $f^A \in \mathcal{C} \cup \mathcal{F}$), $[f]$ is a variable $\underline{f}$ (resp. a closed HOMP M) with a type B, where B is obtained from (the curryfication of) A by:

- replacing each occurrence of a base type D by $\mathbf{N}$,
- replacing each occurrence of $\rightarrow$ in A by either $\rightarrow$ or $\multimap$.

For instance if $A = (D_1 \rightarrow D_2) \rightarrow D_3$ we can take for B any of the types: $(\mathbf{N} \multimap \mathbf{N}) \rightarrow \mathbf{N}$, $(\mathbf{N} \rightarrow \mathbf{N}) \rightarrow \mathbf{N}$, etc. In the sequel we will write $\underline{A}$ for any of these types B. Now, if $t = (t_0\ t_1 \ldots t_n)$ then $[t]$ is defined if for any $0 \leq i \leq n$, $[t_i]$ is defined and if $[t] \equiv (\ldots([t_0][t_1])\ldots[t_n])$ is well-typed. Additive HOMP assignments are defined just as additive HOP assignments. Now, we say that an assignment $[\cdot]$ is a *quasi-interpretation for* R if for any rule $l \to r$ of R, $[l]$ and $[r]$ are defined and have the same type, and it holds that $[\![l]\!]_{\leq} \geq [\![r]\!]_{\leq}$. Observe that contrarily to the case of polynomial interpretations, these inequalities are not strict, and moreover they are stated with respect to the new domains, taking into account the distinction between the two connectives → and ⊸.

The interpretation of a term does not, like in the strict case, necessarily decrease along a reduction step. However, it cannot increase: if $t \to^* s$, then $[\![s]\!]_{\leq} \leq [\![t]\!]_{\leq}$. This, together with the possibility of forming HOMPs of arbitrary type (Lemma 4.5) implies the following, crucial, property:

▶ **Proposition 4.6** (Subterm Property). *Suppose that an STTRS R has an additive quasi-interpretation $[\cdot]$. Then, for every function symbol* $\mathtt{f}$ *of arity n with base arguments, there is a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that if $(\mathtt{f}\ t_1 \ldots t_n) \to^* s$ and if s contains an occurrence of a base term r, then $|r| \leq p(|t_1|, \ldots, |t_n|)$.*

And here is the main result of this Section:

▶ **Theorem 4.7** (Polytime Soundness). *If an STTRS R has an additive quasi-interpretation, R satisfies the termination criterion and* $\mathtt{f}$ *has arity n with base type arguments, then there is a polynomial $p : \mathbb{N}^n \to \mathbb{N}$ such that whenever $(\mathtt{f}\ t_1 \ldots t_n) \to^m s$, it holds that $m, |s| \leq p(|t_1|, \ldots, |t_n|)$. So if* $\mathtt{f}$ *has a type $D_1 \times \cdots \times D_n \to D$ then instances of* $\mathtt{f}$ *can be computed in polynomial time.*

Proof. A consequence of Proposition 4.6 and of Proposition 4.3. ◀

Notice how Theorem 4.7 is proved by first observing that terms of STTRSs having a quasi-interpretation are bounded by natural numbers which are not too big with respect to the input, thus relying on the termination criterion to translate these bounds to *complexity* bounds.

Higher-order quasi interpretations, like their strict siblings, can be extended by enlarging D_P so as to include more combinators, provided they are bounded by polynomials. One of these extensions is discussed in [3] and allows to (re)prove LFPL programs to represent polytime functions.

4.4 Examples

Consider the program `foldr` given by:

$$((\mathtt{foldr}\ f\ b)\ \mathbf{nil}) \to b; \tag{1}$$

$$((\mathtt{foldr}\ f\ b)\ (\mathbf{cons}\ x\ xs)) \to (f\ x\ ((\mathtt{foldr}\ f\ b)\ xs)); \tag{2}$$

where functions, variables and constructors have the following types:

$$\mathtt{foldr} : (D \times E \to E) \times E \to L(D) \to E; \qquad f : D \times E \to E;$$

and **nil**, **cons** typed as in Sect. 3.5. Now, we choose as assignment:

$$[\mathbf{nil}] = 1 : \mathbf{N}; \qquad [\mathbf{cons}] = \lambda n.\lambda m.n + m + 1 : \mathbf{N} \to \mathbf{N} \to \mathbf{N};$$
$$[\mathtt{foldr}] = \lambda\phi.\lambda p.\lambda n.p + n \times (\phi\ 1\ 1) : (\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}) \to \mathbf{N} \to \mathbf{N} \to \mathbf{N}.$$

Observe the $\multimap$ in the type of the first argument of $[\texttt{foldr}]$ which is the way to restrict the domain of arguments. We then obtain the following interpretations of terms:

$$\begin{aligned}
[((\texttt{foldr}\ f\ b)\ \mathbf{nil})] &= \underline{b} + 1 \times (\underline{f}\ 1\ 1);\\
[((\texttt{foldr}\ f\ b)\ (\mathbf{cons}\ x, xs))] &= \underline{b} + (\underline{x} + \underline{xs} + 1) \times (\underline{f}\ 1\ 1);\\
[(f\ x\ ((\texttt{foldr}\ f\ b)\ xs))] &= \underline{f}\ \underline{x}\ (\underline{b} + \underline{xs} \times (\underline{f}\ 1\ 1)).
\end{aligned}$$

It is easy to see that condition $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$ holds for rule (1). As to rule (2) consider $\phi \in \mathcal{N} \multimap \mathcal{N} \multimap \mathcal{N}$. We know that ϕ has a size $c \geq 0$, and thus for every $x, y \in \mathcal{N}$,

$$c \leq \phi\ x\ y \leq x + y + c. \tag{3}$$

Then we have:

$$\begin{aligned}
\underline{f}\ \underline{x}\ (\underline{b} + \underline{xs} \times \underline{f}(1,1)) &\leq \underline{x} + \underline{b} + \underline{xs} \times (\underline{f}\ 1\ 1) + c \leq \underline{x} \times (\underline{f}\ 1\ 1) + \underline{b} + \underline{xs} \times (\underline{f}\ 1\ 1) + c\\
&\leq \underline{b} + (\underline{x} + \underline{xs} + 1) \times (\underline{f}\ 1\ 1),
\end{aligned}$$

where for the two last steps we used $(\underline{f}\ 1\ 1) \geq 1$ and $(\underline{f}\ 1\ 1) \geq c$ (because of (3)). So $[\![r]\!]_{\leq} \leq [\![l]\!]_{\leq}$ also holds for (2) and we have an additive quasi-interpretation. As to the termination criterion, it is satisfied because in rule (2), xs is a strict sub-pattern of $(\mathbf{cons}\ x\ xs)$ and the term $(f\ x\ y)$ can be typed in the linear type system as required. Summing up, we can apply Theorem 4.7 and conclude that if the termination criterion is satisfied by all functions, if $t^{D \times E \to E}$, b^{E} are terms and $[t]$ is a HOMP with type $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$, then $(\texttt{foldr}\ t\ b)$ is a polynomial time program of type $L(D) \to E$.

Note that the idea of ensuring complexity bounds when the programs are fed with functional arguments admitting additional conditions had already been suggested in [10], on particular examples. The present setting using types, however, brings a more systematic account of this property.

5 Discussion and Relation with Other ICC Systems

The authors believe that the interest of the present work does not lie much in bringing yet another ready-to-use ICC system but rather in offering a new *framework* in which to design ICC systems and prove their complexity properties. Indeed, considered as an ICC system our setting presents two limitations:

1. given a program one needs to *find* an assignment and to *check* that it is a valid quasi-interpretation, which in general will be difficult to automatize;
2. the termination criterion currently does not allow to reuse higher-order arguments in full generality.

To overcome 2. we think it will be possible to design more liberal termination criteria, while attacking 1. could possibly consist in defining type systems such that if a program is well-typed, then it admits a quasi-interpretation, and for which one could devise type-inference algorithms. On the other hand, recently introduced techniques for inferring higher-order polynomial interpretations [17] could shed some light on this issue, which is however outside the scope of this paper.

Related and Further Work. Let us first compare our approach to other frameworks for proving complexity soundness results. At first-order, we have already emphasized the fact that our setting is an extension of the quasi-interpretation approach of [9] (see also [1] for the relation with non-size-increasing, at first-order). At higher-order, various approaches based on realizability have been used [14, 11]. While these approaches were developed for logics or System T-like languages, our setting is adapted to a language with recursion and pattern-matching. We think it might also be easier to use in practice.

Let us now discuss the relations with known ICC systems. Several variants of System T based on restriction of recursion and linearity conditions [19, 7, 12] have been proposed which characterize polynomial time. Another system [20] based on a linear type system for non-size-increasing computation, called LFPL, offers more intensional expressivity. Terms of the latter calculus can indeed be proved to be reducible in polynomial time by showing they admit quasi-interpretations and satisfy the termination criterion (details are omitted here, due to space constraints, but can be found in [3]). With respect to [20], the advantages we bring are a slightly more general handling of higher-order arguments, but also the possibility to capture size-increasing polytime algorithms. As an example, we are able to assign a quasi-interpretation to (STTRSs computing) functions in Bellantoni and Cook's algebra BC [6] (see again [3]).

Some other works are based on type systems built out of variants of linear logic [5, 18, 4]. They are less expressive for first-order functions but offer more liberal disciplines for handling higher-order arguments. In future work we will examine if they could suggest a more flexible termination condition for our setting, maybe itself based on quasi-interpretations, following [13].

6 Conclusions

We have advocated the usefulness of simply typed term rewriting systems to smoothly extend notions from first-order rewrite systems to the higher-order setting. Our main contribution is a new framework for studying (and distilling) ICC systems for higher-order languages. While up to now quite distinct techniques had been successful for providing expressive criteria for polynomial time complexity at first-order and at higher-order respectively, our approach brings together these techniques: interpretation methods on the one hand, and semantic domains and type systems on the other. We have illustrated the strength of this framework by designing an ICC system for polynomial time based on a termination criterion and on quasi-interpretations, which allows to give some sufficient conditions for programs built with higher-order functionals (like `foldr`) to work in bounded time. We think this setting should allow in future work to devise new, more expressive, systems for ensuring complexity bounds for higher-order languages.

References

1 Roberto Amadio. Synthesis of max-plus quasi-interpretations. *Fundam. Inform.*, 65:29–60, 2005.

2 Takahito Aoto and Toshiyuki Yamada. Termination of simply typed term rewriting by translation and labelling. In *RTA 2003*, volume 2706 of *LNCS*. Springer, 2003.

3 Patrick Baillot and Ugo Dal Lago. Higher-order interpretations and program complexity (long version). Available at `http://hal.archives-ouvertes.fr/hal-00667816`, 2012.

4 Patrick Baillot, Marco Gaboardi, and Virgile Mogbil. A polytime functional language from light linear logic. In *ESOP 2010*, volume 6012 of *LNCS*, pages 104–124, 2010.

5 Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Inf. Comput.*, 207(1):41–62, 2009.

6 Stephen J. Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the poly-time functions. *Computational Complexity*, 2:97–110, 1992.

7 Stephen J. Bellantoni, Karl-Heinz Niggl, and Helmut Schwichtenberg. Higher type recursion, ramification and polynomial time. *Ann. Pure Appl. Logic*, 104(1-3):17–30, 2000.

8 Guillaume Bonfante, Adam Cichon, Jean-Yves Marion, and Hélène Touzet. Algorithms with polynomial interpretation termination proof. *J. Funct. Program.*, 11(1):33–53, 2001.

9 Guillaume Bonfante, J.-Y. Marion, and Jean-Yves Moyen. Quasi-interpretations a way to control resources. *Theor. Comput. Sci.*, 412(25):2776–2796, 2011.

10 Guillaume Bonfante, Jean-Yves Marion, and Romain Péchoux. Quasi-interpretation synthesis by decomposition. In *ICTAC 2007*, volume 4711 of *LNCS*, pages 410–424. Springer, 2007.

11 Aloïs Brunel and Kazushige Terui. Church => Scott = Ptime: an application of resource sensitive realizability. In *DICE 2010*, volume 23 of *EPTCS*, pages 31–46, 2010.

12 Ugo Dal Lago. The geometry of linear higher-order recursion. In *LICS 2005*, pages 366–375, 2005.

13 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *LICS 2011*, pages 133–142, 2011.

14 Ugo Dal Lago and Martin Hofmann. Realizability models and implicit complexity. *Theor. Comput. Sci.*, 412(20):2029–2047, 2011.

15 Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda-calculus. In *ICALP 2009*, volume 5556 of *LNCS*, pages 163–174. Springer, 2009.

16 Nachum Dershowitz. Orderings for term-rewriting systems. *Theor. Comput. Sci.*, 17(3):279–301, 1982.

17 Carsten Fuhs and Cynthia Kop. Polynomial interpretations for higher-order rewriting. In *RTA 2012*, volume 15 of *LIPIcs*, pages 176–192. Schloss Dagstuhl, 2012.

18 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for lambda -calculus. In *CSL 2007*, volume 4646 of *LNCS*, pages 253–267. Springer, 2007.

19 Martin Hofmann. A mixed modal/linear lambda calculus with applications to Bellantoni-Cook safe recursion. In *CSL 1997*, volume 1414 of *LNCS*, pages 275–294, 1997.

20 Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003.

21 Jean-Pierre Jouannaud and Mitsuhiro Okada. A computation model for executable higher-order algebraic specification languages. In *LICS 1991*, pages 350–361, 1991.

22 Jean-Pierre Jouannaud and Albert Rubio. The higher-order recursive path ordering. In *LICS 1999*, pages 402–411, 1999.

23 Jan Willem Klop, Vincent van Oostrom, and Femke van Raamsdonk. Combinatory reduction systems: Introduction and survey. *Theor. Comput. Sci.*, 121(1&2):279–308, 1993.

24 D. Lankford. On proving term rewriting systems are noetherian. Technical Report MTP-3, Louisiana Tech. University, 1979.

25 D. Leivant. Predicative recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.

26 Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundam. Inform.*, 19(1/2), 1993.

27 Jean-Yves Marion and Jean-Yves Moyen. Efficient First Order Functional Program Interpreter with Time Bound Certifications. In *LPAR 2000*, volume 1955 of *LNAI*, pages 25–42. Springer, 2000.

28 Georg Moser and Andreas Schnabl. The derivational complexity induced by the dependency pair method. *Logical Methods in Computer Science*, 7(3), 2011.

29 Jaco van de Pol. *Termination of Higher-order Rewrite Systems.* PhD thesis, Utrecht University, 1996.

30 Toshiyuki Yamada. Confluence and termination of simply typed term rewriting systems. In *RTA 2001*, volume 2051 of *LNCS*, pages 338–352. Springer, 2001.

Knowledge Spaces and the Completeness of Learning Strategies*

Stefano Berardi[1] and Ugo de'Liguoro[2]

1 **Dipartimento di Informatica, Università di Torino**
c.so Svizzera 185 Torino, Itatly
stefano@di.unito.it

2 **Dipartimento di Informatica, Università di Torino**
c.so Svizzera 185 Torino, Itatly
deliguoro@di.unito.it

Abstract

We propose a theory of learning aimed to formalize some ideas underlying Coquand's game semantics and Krivine's realizability of classical logic. We introduce a notion of knowledge state together with a new topology, capturing finite positive and negative information that guides a learning strategy. We use a leading example to illustrate how non-constructive proofs lead to continuous and effective learning strategies over knowledge spaces, and prove that our learning semantics is sound and complete w.r.t. classical truth, as it is the case for Coquand's and Krivine's approaches.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.1.2 [Theory of Computation]: Modes of Computation; Interactive and reactive computation; I.2.6 [Artificial Intelligence]: Learning Induction.

Keywords and phrases Classical Logic, Proof Mining, Game Semantics, Learning, Realizability.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.77

1 Introduction

Several methods have been proposed to give a recursive interpretation of non-recursive constructions of mathematical objects, whose existence and properties are classically provable. A non-exhaustive list includes the continuation-based approach initiated by Griffin [10], the game theoretic semantics of classical arithmetic by Coquand [5] and Krivine's realizability of classical logic [12].

As observed by Coquand, there is a common informal idea underlying the different approaches, which is learning. With respect to the dialogic approach, learning consists into interpreting the strategy for the defender of a statement against the refuter by a strategy guiding the interaction between a learning agent and the "world", representing what can be experienced by direct computation.

Under the influence of Gold's ideas [8, 9] and of Hayashi's Limit Computable Mathematics [11] we have proposed a formal theory of "learning" and "well founded limits" in [2]. In the theory the goal of the learning process is to find an evidence, or a witness as it is usually called, of the truth of some given sentence, which is the "problem" that the learning strategy solves. Such an evidence is always tentative, since it could be attained only in the ideal

* This work was partially supported by PRIN project n. 2008H49TEH.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 77–91
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

limit. The task of the learning strategy is to tell how to react to the discovery that the current guess is actually wrong, and this is done on the basis of the knowledge collected in the learning process, which includes all the "counterexamples" that have been seen up to the time.

Here we propose the same idea, but in the different perspective of topological spaces and continuous maps. We assume having an ideal object being the result of some non-effective mental construction and satisfying some decidable property. We say that this construction may be learned w.r.t. to the property if we may find a "finite approximation" of the construction which still satisfies the property. In particular given a classical proof of an existential statement, we see the computational content of the proof as the activity of guessing more and more about the object (individual) the sentence is about, without ever obtaining a full information; in a discrete setting such as natural numbers, the approximation is actually a different object, which we think as "close" to the ideal one, the "limit", only with respect to some given property which both satisfy.

When reasoning about ideal objects, we deal with descriptions rather than with the objects themselves. While learning of an ideal object we have step by step certain amounts of knowledge, which consist of pieces of evidence (e.g. decidable statements): therefore we topologize *states of knowledge* to express the idea that a continuous strategy only depends on finite positive and negative information to yield finite approximations of the ideal limit. We call *interactive realizer* any continuous function of states of knowledge that roughly tells which are the further guesses to improve the given knowledge, and how to react to the discovery of negative evidences (e.g. counterexamples to certain assumptions). We claim that an interactive realizer corresponds to a lambda term with continuations in Griffin's work, to a classical realizer in Krivine's sense and to a winning strategy in the sense of Coquand: however, we will not support this claim here.

We call a *model* any perfect (usually infinite) knowledge state. The main result of this paper is that any object that can be ideally learned in a model can be effectively learned in a finite state of knowledge approximating the model, and this state is found by means of a realizer. Since models represent classical truth, the completeness theorem can be read as stating that the learning semantics for classical proofs is complete w.r.t. classical truth, namely Tarskian truth, as it is the case for Coquand's dialogic semantics of proofs and for Krivine's classical realizability, and that the learning process is indeed effective.

The paper is organized as follows. In §2 we introduce a motivating example, which is used throughout the paper. In §3 we define the state knowledge and topology. In §4 the concept of relative truth is introduced to define sound, complete and model knowledge states. Finally in §5 we define interactive realizability and prove the completeness theorem. Due to space restrictions, proofs of technical lemmas have been omitted.

Related works

The suggestion by Coquand that the dialogic interpretation of classical proofs could be seen as learning of some abstract entities can be found in [4], a preliminary version of [5]. The idea has been illustrated by means of a suggestive example by von Plato in [14]. Beside Krivine's [12], Miquel's work in [13] illustrates in detail the behavior of classical realizability of existential statements (also in comparison to Friedman's method), and we strongly believe that the construction is a learning process in the sense of the present paper.

Learning in the limit of undecidable properties and ideal entities comes from Gold's work [8, 9], and has been recently rediscovered by Hayashi e.g. in [11]. We have investigated the concept of learning in the limit incorporating Coquand's ideas in [2], although in a combinatory rather than topological perspective. We have further elaborated the concept of learnable in the limit in [3], where a "solution" in terms of the following §5 is called "individual", since we identify the ideal limit with the map generating its approximations from states of knowledge. The formal definition of the state topology, however, is new with the present work, as well as the treatment of the general, non-monotonic case. Also the concept of "interactive realizability" has been introduced in [3], but in the simpler case of monotonic learning. Essentially the same construction as in [3] is used in [1] to define a realizability interpretation of **HA** plus excluded middle restricted to Σ^0_1-formulas. It turns out that interactive realizability is a generalisation of Kleene's realizability, and this motivates the terminology.

2 Solving problems by learning

To illustrate the idea of learning strategies, either monotonic or non-monotonic, we propose an example suggested by Coquand and developed by Fridlender in [6]. Let f_1, f_2 be total functions over $\mathbb{N}$. Fix some integer $k > 0$ and consider the statements: "there is an increasing sequence of k integers which is weakly increasing w.r.t. f_1" and "there is an increasing sequence of k integers which is weakly increasing w.r.t. f_1 and f_2". Formally:

$$\exists x_1 \ldots \exists x_k.\, x_1 < \cdots < x_k \;\wedge\; f_1(x_1) \leq \cdots \leq f_1(x_k) \tag{1}$$

$$\exists x_1 \ldots \exists x_k.\, x_1 < \cdots < x_k \;\wedge\; f_1(x_1) \leq \cdots \leq f_1(x_k) \;\wedge\; f_2(x_1) \leq \cdots \leq f_2(x_k) \tag{2}$$

We look at these statements as the *problems* of finding a k-tuple $n_1 < \cdots < n_k$ witnessing their truth. We begin by observing that these statements can be proved classically as follows. For any $f : \mathbb{N} \to \mathbb{N}$ and $A \subseteq \mathbb{N}$ we say that "n is a local minimum of f w.r.t. A" (shortly n is an f, A-minimum) if n is the minimum of f in $A \cap [n, \infty[$. Formally:

$$f, A\text{-min}(n) \Leftrightarrow \forall y \in A.\ n < y \Rightarrow f(n) \leq f(y).$$

Observe that the predicate $f, A\text{-min}(n)$ is undecidable in general, even if f is recursive and A decidable. The statement $\neg f, A\text{-min}(n)$ is classically equivalent to $\exists y \in A.\ n < y \wedge f(n) > f(y)$. For any $f : \mathbb{N} \to \mathbb{N}$ and infinite $A \subseteq \mathbb{N}$, we denote by

$$A_f = \{a \in A \mid f, A\text{-min}(a)\} \subseteq A$$

the set of all f, A-minima. We now study the set A_f.

▶ **Lemma 1.** *For any $f : \mathbb{N} \to \mathbb{N}$ and infinite $A \subseteq \mathbb{N}$, the set $A_f = \{a \in A \mid f, A\text{-min}(a)\}$ of f, A-minima is infinite, and f is monotonic over A_f.*

Proof. Toward a contradiction suppose that A_f is finite, possibly empty. Then there exists $a_0 = \min(A \setminus A_f)$, as A is infinite. By definition of a_0 for all $a \in A$ with $a \geq a_0$ we have $\neg f, A\text{-min}(a)$, that is: there is some $a' > a$, $a' \in A$ such that $f(a) > f(a')$. If we choose $a = a_0$ we deduce that there exists some $a_1 \in A$ such that $a_0 < a_1$ and $f(a_0) > f(a_1)$, and so on. By iterating the reasoning we get an infinite sequence $a_0 < a_1 < a_2 < \cdots$ such that $f(a_i) > f(a_{i+1})$ for all $i \in \mathbb{N}$, which is on turn an infinite descending chain in $\mathbb{N}$, that is an absurdity. ■

▶ **Theorem 2.** *Both statements (1) and (2) are classically provable.*

Proof. In Lemma 1.2 take $A = \mathbb{N}$: then f_1 is monotonic over the infinite set A_{f_1}. If we take the first k elements of A_{f_1} we have an increasing sequence of k integers whose values are weakly increasing under f_1, establishing (1).

To prove (2) we use use again Lemma 1.2 taking $A = \mathbb{N}_{f_1}$, which we know to be infinite by the same lemma; then f_1, f_2 are both monotonic over the infinite set $A_{f_2} = (\mathbb{N}_{f_1})_{f_2}$. ■

The proof of Lemma 1 and its use in the proof of Theorem 2 are non-constructive, as they rely on the "computation" of the minimum of f in A for certain f and A. In order to compute the minimum of f we need, in general, to know infinitely many values of f. However, this proof may be interpreted by a computation as soon as we only require finitely many $n_1, \ldots, n_k$ that satisfy (1) or (2): $n_1, \ldots, n_k$ may be found using a finite knowledge about f.

Indeed the proof of Lemma 1 can be turned into an effective strategy to learn a solution to problem (1), assuming that f_1 is recursive. The basic remark is that we do not actually need to know the infinitely many elements of $\mathbb{N}_{f_1}$, nor we have to produce some n which belongs to $\mathbb{N}_{f_1}$ *beyond any doubt.* We can approximate the infinite set $\mathbb{N}_{f_1}$ by some finite set B, whose elements are not necessarily in $\mathbb{N}_{f_1}$, rather they are just $f_1, \mathbb{N}$-minima *as far as we know.* B is a kind of hypothesis about $\mathbb{N}_{f_1}$.

More precisely, we will find a set B such that $f_1, B\text{-min}(n)$ for all $n \in B$. This property is trivially true for any singleton set, say $\{0\}$. In the general case, if B has k elements or more, then by definition of f_1, B-min the set B is a solution to (1). Otherwise we take any $m > \max(B)$ and try adding m to B. Since we cannot decide whether any n is a local minimum of f_1, we are not allowed to increase B to $B \cup \{m\}$, because it could be the case that $f_1(n) > f_1(m)$ for some $n \in B$. Rather we update B, by removing all $n \in B$ such that $f_1(n) > f_1(m)$:

$$B' = \{n \in B \mid f_1(n) \leq f_1(m)\} \cup \{m\},$$

The new set B' includes m and satisfies the invariant property of containing only f_1, B'-minima. The cardinality of B' is not necessarily greater than that of B, so that we need an argument to conclude that, starting from the singleton $\{0\}$ and iterating the step from B to B', the learning agent will eventually reach a k-element set with the required property.

The termination argument in the case works as follows. Although the sequence of sets is not increasing w.r.t. inclusion, the knowledge that some elements are *not* local minima of f_1 grows monotonically, since more and more pairs n, m are found such that $f_1(n) > f_1(m)$. From this remark, one can prove by a fixed-point argument (over a suitable topology) that the growth of knowledge eventually ends, which implies that a set B with k elements will be found after finitely many steps. In this case we speak of *monotonic learning.*

In order to include an example of non-monotonic learning, we assume that both f_1 and f_2 are recursive, and we outline an effective computation approximating an initial segment of $(\mathbb{N}_{f_1})_{f_2}$, solving problem (2). The informal interpretation we include here will be formalized using the notion of layered valuation.

As it happens in the classical proof of Theorem 2, we iterate the same method used for problem (1), and build a $C \subseteq B$ of f_2, C-minima, where B is the current approximation of the infinite set $\mathbb{N}_{f_1}$. In doing so the learning agent assumes that B is a subset of $\mathbb{N}_{f_1}$ (though he cannot be certain of this), and that all elements of C are f_2-minima w.r.t. $\mathbb{N}_{f_1}$ (again an uncertain belief). At each step, the learner takes some $m \in B$, such that $m > \max(C)$, and manages to add m to C, possibly by removing some of its elements, by computing:

$$C' = \{p \in C \mid f_2(p) \leq f_2(m)\} \cup \{m\} \subseteq B.$$

This is only possible when such an m exists in B: if not the algorithm generating B has to be resumed to get a larger set containing an element greater than $\max(C)$. But since B does not grow monotonically, elements of C will be dropped while computing C' also because they are no longer in B. This makes the convergence proof much harder. Indeed the knowledge accumulated while building B takes the simple form of (sets of statements) $f_1(n) > f_1(m)$, and it grows monotonically; on the contrary the "knowledge" gathered while computing C consists of more complex statements of the form: $m \in B \wedge f_2(n) > f_2(m)$, with B changing non-monotonically during the computation of C. This knowledge is the conjunction of an hypothesis $m \in B$ and a fact, $f_2(n) > f_2(m)$. This second layer of of knowledge, mixing hypothesis and fact, does not grow monotonically, because any hypothesis $m \in B$ may turn out to be false: therefore it is unsafe, yet it guides the construction of C. In this case we speak *non-monotonic learning*. Non-monotonic learning is the more general form of learning.

3 States of knowledge and their topology

There are three kinds of entities in learning: questions, answers and states of knowledge. The main concern are states of knowledge, which on turn are certain sets of answers. Answers are viewed as atomic objects, since their internal structure is immaterial. Questions instead are represented indirectly by equivalence classes of answers, each to be thought of as the set of alternative, incompatible choices for an answer to the same question.

▶ **Definition 3** (Knowledge Structure and State of Knowledge)**.** A *knowledge structure* $(\mathbb{A}, \sim)$ consists of a a non-empty at most countable set $\mathbb{A}$ of *answers* and an equivalence relation $\sim \subseteq \mathbb{A} \times \mathbb{A}$. As a topological space, $\mathbb{A}$ is equipped with the discrete topology.

The set $\mathbb{Q} = \mathbb{A}/_{\sim}$ of equivalence classes $[x]$ w.r.t. $\sim$ is the set of *questions*, and it is equipped with the discrete topology.

A subset $X \subseteq \mathbb{A}$ is a *state of knowledge* if for all $x \in \mathbb{A}$ the set $X \cap [x]$ is either empty or a singleton. We denote by $\mathbb{S}$ the set of knowledge states and by $\mathbb{S}_{\mathit{fin}}$ the subset of finite elements of $\mathbb{S}$.

If $x \sim y$ then x, y are two answers to the same question. The equivalence class $[x]$ abstractly represents the question answered by x. Two answers $x, y \in \mathbb{A}$ are *compatible*, written $x\#y$, if they are not different answers to the same question:

$$x\#y \;\Leftrightarrow\; x = y \vee x \not\sim y.$$

Example 1 Let us reconsider the example in §2. A knowledge structure $(\mathbb{A}_0, \sim_0)$ for learning a solution to (1) can be defined by taking $\mathbb{A}_0 = \{(n, m) \in \mathbb{N} \times \mathbb{N} \mid n < m\}$, where we interpret a pair (n, m) as the statement: "m is a counterexample to $f_1, \mathbb{N}\text{-min}(n)$", and more precisely as the formula:

$$n < m \wedge f_1(n) > f_1(m).$$

If we think of m as the answer to the question about n, we obtain the definition of the relation $(n, m) \sim_0 (n', m')$ by $n = n'$.

A knowledge structure $(\mathbb{A}_1, \sim_1)$ for learning a solution to (2) can be defined by taking $\mathbb{A}_1 = \mathbb{A}_0$ and $\sim_1 = \sim_2$. $(\mathbb{A}_1, \sim_1)$ has not the same intended meaning as $(\mathbb{A}_0, \sim_0)$: an answer $(n, m) \in \mathbb{A}_1$ is interpreted by the statement:

$$n < m \wedge f_2(n) > f_2(m) \wedge n, m \in \mathbb{N}_{f_1}.$$

Finally we set $\mathbb{A}_2 = \mathbb{A}_0 \uplus \mathbb{A}_1 = \{(i,n,m) \mid i \in \{0,1\} \;\&\; (n,m) \in \mathbb{A}_i\}$, namely the disjoint union of $\mathbb{A}_0$ and $\mathbb{A}_1$ and we define $(i,n,m) \sim_2 (j,n',m')$ if and only if $i = j$ and $n = n'$ (that is $(n,m) \sim_i (n',m')$).

Let $\mathbb{S}_2$ be the knowledge space associated to $\mathbb{A}_2$ and $X \in \mathbb{S}_2$ be any state of knowledge. Then we interpret $(0,n,m) \in X$ by "the agent knows at X that $n < m$ and $f_1(n) > f_1(m)$", hence that n is not in $\mathbb{N}_{f_1}$. We interpret: for all $p \in \mathbb{N}$, $(0,n,p) \notin X$ by "the agent knows at X of no p such that $n < p$ and $f_1(n) > f_1(p)$", hence he believes that n is in $\mathbb{N}_{f_1}$. We write $(\mathbb{N}_{f_1})^X = \{n \in \mathbb{N} \mid \forall p \in \mathbb{N}.(0,n,p) \notin X\}$ for the set of n which the agent believes to be in $\mathbb{N}_{f_1}$ at X. In the same way we write $((\mathbb{N}_{f_1})_{f_2})^X = \{n \in (\mathbb{N}_{f_1})^X \mid \forall p \in \mathbb{N}.(1,n,p) \notin X\}$ for the set of n which the agent believes to be in $(\mathbb{N}_{f_1})_{f_2}$ at X. We will see in the following sections that $(\mathbb{A}_2, \sim_2)$ is a knowledge structure apt to learn (2).

The "state of knowledge" of a finite agent should be finite; for the sake of the theory we also consider infinite states of knowledge, which are naturally approximated by finite ones in a sense to be made precise by a topology. Let us define a query map $\mathsf{q} : \mathbb{S} \times \mathbb{Q} \to \mathcal{P}_{fin}(\mathbb{A})$ by $\mathsf{q}(X,[x]) = X \cap [x]$. Then $\mathsf{q}(X,[x])$ is either a singleton $\{y\}$, meaning that y is the answer at X to the question $[x]$, or the empty set, meaning that the agent knows at X of no answer to $[x]$, and he assumes that there is none. Take the discrete topology over $\mathcal{P}_{fin}(\mathbb{A})$; we then consider the smallest topology over $\mathbb{S}$ making q continuous.

▶ **Definition 4** (State Topology). The *state topology* $(\mathbb{S}, \Omega(\mathbb{S}))$ is generated by the sub-basics A_x, B_x, with $x \in \mathbb{A}$:

$$\begin{aligned} A_x &= \{X \in \mathbb{S}_{fin} \mid x \in X\} = \{X \in \mathbb{S}_{fin} \mid X \cap [x] = \{x\}\}, \\ B_x &= \{X \in \mathbb{S}_{fin} \mid X \cap [x] = \emptyset\}. \end{aligned}$$

A_x is the set of all states X such that $\mathsf{q}(X,[x]) = \{x\}$, which means that at state X the answer $x \in \mathbb{A}$ has been selected to the question $[x] \in \mathbb{Q}$; on the other hand if $X \in B_x$, that is $\mathsf{q}(X,[x]) = \emptyset$, then at X the learning agent knows of no answer to the question $[x]$. Let X, Y, Z range over $\mathbb{S}$, and s, t over $\mathbb{S}_{fin}$. By definition, a basic open of $\Omega(\mathbb{S})$ has the shape:

$$\mathcal{O}_{U,V} = \bigcap_{x \in U} A_x \cap \bigcap_{y \in V} B_y,$$

for finite $U, V \subseteq \mathbb{A}$. If $\neg x \# y$, that is $x \sim y$ and $x \neq y$, then $A_x \cap A_y = \emptyset$, because if $x, y \in X$ then X is inconsistent, so that $\emptyset$ is a basic open. On the other hand if $x \sim y$ then $B_x = B_y$. Therefore without loss of generality we assume U, V to be consistent, and that all basic opens of $\Omega(\mathbb{S})$ are of the form $\mathcal{O}_{s,t}$, for some $s, t \in \mathbb{S}_{fin}$. Summing up, assume that $s = \{x_1, \ldots, x_n\}$ and $t = \{y_1, \ldots, y_m\}$. Then $X \in \mathcal{O}_{s,t}$ means that the agent at X knows the answers $x_1, \ldots, x_n$ to the questions $[x_1], \ldots, [x_n]$ (a finite positive information), while he knows of no answer to the questions $[y_1], \ldots, [y_m]$, and assumes that there is none (a finite negative information).

The state topology is distinct from, yet strictly related to, several well-known topologies. $\Omega(\mathbb{S})$ is discrete if and only if $\mathbb{Q}$ is finite (there are finitely many equivalence classes). $\Omega(\mathbb{S})$ is homeomorphic to the product space $\Pi_{[x] \in \mathbb{Q}}([x] \uplus 1)$ of the discrete topologies over $[x] \uplus 1$, where 1 is any singleton (representing the "undefined" answer to the question $[x]$) and $\uplus$ is disjoint union. Every state topology is homeomorphic to some subspace of the Baire topology over $\mathbb{N}^{\mathbb{N}}$. The state topology is totally disconnected and Hausdorff: it is compact (that is, is a Stone space) if and only if all equivalence classes are finite. If all equivalence classes in $\mathbb{Q}$ are singletons (that is, if $\sim$ is the equality relation on $\mathbb{A}$) and $\mathbb{Q}$ is infinite then $\Omega(\mathbb{S})$ is homeomorphic to the Cantor space $2^{\mathbb{N}}$. If all equivalence classes in $\mathbb{Q}$ are infinite and $\mathbb{Q}$ is infinite, then $\Omega(\mathbb{S})$ is homeomorphic to the whole Baire space.

A clopen is an open and closed set; hence clopens are closed under complement, finite unions and intersections. There are significative examples of clopen sets in $\mathbb{S}$.

▶ **Lemma 5** (Sub-basic opens of State Topology are clopen)**.** *Assume that $x \in \mathbb{A}$, $f : \mathbb{S} \to I$ is continuous, I is a discrete space and $J \subseteq I$. Then:*

1. *B_x is clopen*
2. *A_x is clopen*
3. *$f^{-1}(J)$ is clopen.*

▶ Remark. As a consequence of Lemma 5.1 we have that the predicate $n \in \mathbb{N}^X_{f_1}$ is continuous in X, since $\{X \mid n \in \mathbb{N}^X_{f_1}\} = B_{(0,n,n+1)}$ which is a clopen set, and indeed $n \notin \mathbb{N}^X_{f_1}$ if and only if $X \in \bigcup_{m>n} A_{(0,n,m)}$, namely the complement of $B_{(0,n,n+1)}$. A similar remark holds for $n \in (\mathbb{N}_{f_1})^X_{f_2}$.

It is instructive to compare the state topology to Scott and Lawson topologies over $\mathbb{S}$. First observe that $\mathbb{S}$ is a poset by subset inclusion, and it is downward closed. It follows that $(\mathbb{S}, \cap, \subseteq)$ is an inf-semilattice with bottom $\emptyset$. $\mathbb{S}$ is closed under arbitrary but non-empty inf, as the empty inf, namely the whole set $\mathbb{A}$, is not consistent in general. Indeed $\mathbb{S}$ is not closed under union, unless the compatibility relation is the identity. We say that X and Y are compatible w.r.t. inclusion, if $X \subseteq Z \supseteq Y$ for some $Z \in \mathbb{S}$. Clearly the union of a family $\mathcal{U} \subseteq \mathbb{S}$ belongs to $\mathbb{S}$ if and only if all elements of $\mathcal{U}$ are pairwise compatible sets. By this $\mathbb{S}$ is closed under directed sups, so it is a coherence space in the sense of Girard and a cpo, which in fact has compacts $K(\mathbb{S}) = \mathbb{S}_{\mathit{fin}}$ and it is algebraic.

It follows that the *Scott topology* over the cpo $(\mathbb{S}, \subseteq)$ is determined by taking all the A_x with $x \in \mathbb{A}$ as sub-basics. On the other hand any B_x is a Scott-closed set, since its complement $\mathbb{S} \setminus B_x$ is equal to the union $\bigcup\{A_y \mid y \in [x]\}$ of Scott opens. However B_x is not Scott-open because it is not upward closed.

Recall that (see [7]) the *lower topology* over a poset is generated by the complements of principal filters; the *Lawson topology* is the smallest refinement of both the lower and the Scott topology. In case of the cpo $(\mathbb{S}, \subseteq)$ the Lawson topology is generated by the sub-basics:

$$\overline{X\uparrow} = \{Y \in \mathbb{S} \mid X \not\subseteq Y\} \quad \text{and} \quad s\uparrow = \{Y \in \mathbb{S} \mid s \subseteq Y\},$$

for $X \in \mathbb{S}$ and $s \in \mathbb{S}_{\mathit{fin}}$, representing the negative and positive information respectively.

The state topology includes the Lawson topology, and in general it is finer than that. The next lemma tells that all Lawson opens are open in the state topology, but if some equivalence class $[x]$ is infinite then some open of the state topology is not open in the Lawson topology. Recall that $\Omega(\mathbb{S})$ denotes the family of open sets in the state topology.

▶ **Lemma 6.**

1. *All basic opens of Lawson topology are in $\Omega(\mathbb{S})$.*
2. *For all $x \in \mathbb{A}$, B_x is Lawson-open if and only if $[x]$ is finite.*

As an immediate consequence of Lemma 6 we have the following.

▶ **Theorem 7** (State versus Lawson Topology)**.** *The state topology $\Omega(\mathbb{S})$ refines the Lawson topology over the cpo $(\mathbb{S}, \subseteq)$, and they coincide if and only if $[x]$ is finite for all $x \in \mathbb{A}$.*

4 Relative truth and layered states

Answers to a question can be either true or false. In the perspective of learning we think of truth values with respect to the actual knowledge that a learning agent can have at some stage of the process, so that we relativize the valuation of the answers to the knowledge states. Furthermore the example of learning the solution to problem (2) in §2 shows that there can be dependencies among answers in a state of knowledge. We formalize this by means of a stratification into levels of the set of answers. In the example of §1 we only need levels 0 and 1. In the definition, however, we allow any number of levels, even transfinite.

Denote with Ord the class of ordinals. Let us assume the existence of a map $\mathsf{lev} : \mathbb{A} \to \mathit{Ord}$, associating to each answer x the ordinal $\mathsf{lev}(x)$, and such that any two answers to the same question are of the same level. If $X \in \mathbb{S}$ and $\alpha \in \mathit{Ord}$ we write $X \restriction \alpha = \{x \in X \mid \mathsf{lev}(x) < \alpha\}$. We can now make precise the notions of level and of truth of an answer w.r.t. a knowledge state. Let us denote with $2 = \{\mathsf{true}, \mathsf{false}\}$ the set of truth values.

▶ **Definition 8** (Layered Valuations). A *layered knowledge structure* is any tuple $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$, with $(\mathbb{A}, \sim)$ a knowledge structure, $\mathsf{lev} : \mathbb{A} \to \mathit{Ord}$ and $\mathsf{tr} : \mathbb{A} \times \mathbb{S} \to 2$ two maps such that:

1. *Two answers to the same question have the same level*:

$$\forall x, y \in \mathbb{A}.(x \sim y \Rightarrow \mathsf{lev}(x) = \mathsf{lev}(y))$$

2. tr *is continuous*, by taking $\mathbb{A}$ and 2 with the discrete topology, $\mathbb{S}$ with the state topology $\Omega(\mathbb{S})$, and $\mathbb{A} \times \mathbb{S}$ with the product topology;
3. tr *is layered:* $\forall x \in \mathbb{A}, X \in \mathbb{S}.\mathsf{tr}(x, X) = \mathsf{tr}(x, X \restriction \mathsf{lev}(x))$

Set $\mathsf{T}_x = \mathsf{tr}(x)^{-1}(\{\mathsf{true}\}) = \{X \in \mathbb{S} \mid \mathsf{tr}(x, X) = \mathsf{true}\}$, and similarly $\mathsf{F}_x = \mathsf{tr}(x)^{-1}(\{\mathsf{false}\})$. When tr is a layered valuation, if $X \in \mathsf{T}_x$ or $X \in \mathsf{F}_x$, we say that x is true or false w.r.t. X respectively. By definition, the truth of x w.r.t. X depends only on the answers of lower level than $\mathsf{lev}(x)$; it follows that

$$\mathsf{lev}(x) = 0 \Rightarrow \mathsf{tr}(x, X) = \mathsf{tr}(x, \emptyset)$$

that is, the truth value of answers of level 0 is absolute, and it depends just on the choice of tr. Level 0 answers play the role of "facts". On the contrary answers of level greater than 0 are better seen as empirical hypothesis, that are considered true as far as they are not falsified by answers of lower level.

Example 2 Continuing example 1, let us set $\mathsf{lev}(i, n, m) = i$. Then we define the meaning of the answers in $\mathbb{A}_2$ via the mapping tr by putting: $\mathsf{tr}((0, n, m), X) = \mathsf{true}$ if and only if $f_1(n) > f_1(m)$, and: $\mathsf{tr}((1, n, m), X) = \mathsf{true}$ if and only if $n, m \in (\mathbb{N}_{f_1})^X$ and $f_2(n) > f_2(m)$. Recall that $(\mathbb{N}_{f_1})^X$ is the set of n such that $(0, n, p) \notin X$ for all p, and that if $(i, n, m) \in \mathbb{A}_2 = \mathbb{A}_0 \uplus \mathbb{A}_1$ then $n < m$.

Clearly tr is layered since $\mathsf{tr}((i, n, m), X)$ does not depend on X when $i = 0$, while when $i = 1$ it depends on $X \restriction 1 = \{(i, n, m) \in X \mid i = 0\}$, which is morally $\mathbb{A}_0 \cap X$.

To see that tr is continuous let us observe that $\mathsf{tr}((0, n, m), X)$ is the constant function w.r.t. X, and that $\mathsf{tr}((1, n, m), X) = \mathsf{true}$ iff and only if $f_2(n) > f_2(m)$ and $n, m \in (\mathbb{N}_{f_1})^X$, and the set of X for which this is true is a clopen as a consequence of Lemma 5.

From now on, we assume that some layered knowledge structure $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$ has been fixed, with some level map lev and some continuous layered truth predicate tr. We now introduce the set $\mathtt{S}$ of *sound* knowledge states (those from which nothing should be removed), the set $\mathtt{C}$ of *complete* knowledge states (those to which nothing should be added), the set $\mathtt{M}$

of *model* states (the "perfect" states, those from which nothing should be removed and to which nothing should be added).

▶ **Definition 9** (Sound and Complete States). Let $X \in \mathbb{S}$, $x \in \mathbb{A}$. Then:

1. X is *sound* if $\forall x \in \mathbb{A}.\ x \in X \Rightarrow \mathsf{tr}(x, X) = \mathsf{true}$;
2. X is *complete* if $\forall x \in \mathbb{A}.\ X \cap [x] = \emptyset \Rightarrow \mathsf{tr}(x, X) = \mathsf{false}$;
3. X is a *model* if it is sound and complete.

We call $\mathtt{S}$, $\mathtt{C}$ and $\mathtt{M}$ the sets of sound, complete and model states respectively.

A state of knowledge X is sound if all the answers it contains are true w.r.t. X itself; X is complete if no answer which is true w.r.t. X and compatible with the answers in X can be consistently added to X; hence X is a model if it is made of answers true w.r.t. X and it is maximal. We think of a model X as a perfect representation of the world. For instance with respect to the examples in §2 and §3, if X is a model then the sets $(\mathbb{N}_{f_1})^X$ and $(\mathbb{N}_{f_1})^X_{f_2}$ are equal to $(\mathbb{N}_{f_1})$ and $(\mathbb{N}_{f_1})_{f_2}$ respectively, that is the beliefs of the agent perfectly agree with absolute truth.

In spite of this interpretation, models are far from being unique even w.r.t. a fixed map tr. Two models can include two different answers to the same question, because a question can have many true answers, while w.r.t. any state of knowledge each question is associated to a memory cell having room for a single answer.

Let us define $\mathtt{S}_x = \{X \in \mathbb{S} \mid x \in X \Rightarrow \mathsf{tr}(x, X) = \mathsf{true}\}$, or equivalently $\mathtt{S}_x = (\mathbb{S} \setminus A_x) \cup \mathsf{T}_x$; $\mathtt{C}_x = \{X \in \mathbb{S} \mid X \cap [x] = \emptyset \Rightarrow \mathsf{tr}(x, X) = \mathsf{false}\}$, that is $\mathtt{C}_x = (\mathbb{S} \setminus B_x) \cup \mathsf{F}_x$, and $\mathtt{M}_x = \mathtt{S}_x \cap \mathtt{C}_x$. Clearly we have $\mathtt{S} = \bigcap_{x \in \mathbb{A}} \mathtt{S}_x$, $\mathtt{C} = \bigcap_{x \in \mathbb{A}} \mathtt{C}_x$ and $\mathtt{M} = \bigcap_{x \in \mathbb{A}} \mathtt{M}_x$.

From a topological viewpoint, it is interesting to observe that all the above subsets of $\mathbb{S}$ are closed in $\Omega(\mathbb{S})$, while some of them are clopen.

▶ **Lemma 10.** *For all $x \in \mathbb{A}$, $\mathsf{T}_x, \mathsf{F}_x, \mathtt{S}_x, \mathtt{C}_x, \mathtt{M}_x$ are clopen in $\Omega(\mathbb{S})$. $\mathtt{S}, \mathtt{C}, \mathtt{M}$ are closed in $\Omega(\mathbb{S})$.*

It is immediate that sound sets exist, as well as complete ones: trivial examples are $\emptyset$ which is vacuously sound, and any set X including one answer x for each equivalence class $[x] \in \mathbb{Q}$, which is vacuously complete but not necessarily sound. Here is a non-trivial though simple example of these concepts.

Example 3 Suppose that $f_1(0) = 2 = f_1(n)$ for all $n > 2$, and that $f_1(1) = 1$ and $f_1(2) = 0$. If we consider the states over $\mathbb{A}_0$ only, and the restriction to $\mathbb{A}_0$ of the mapping tr in Example 2, we have the models $\{(0, 0, 1), (0, 1, 2)\}$ and $\{(0, 0, 2), (0, 1, 2)\}$. Any subset of these sets is sound, while $\{(0, n, n+1) \mid n \in \mathbb{N}\}$ is complete but not sound.

It is not obvious, however, that models exist in general.

▶ **Theorem 11** (Existence of Models). *For every layered knowledge structure $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$ and space of knowledge $\mathbb{S}$ over it, there exists a model $X \in \mathbb{S}$.*

Proof. Fix a layered valuation tr, and an arbitrary indexing $x_0, x_1, \ldots$ of the countable set $\mathbb{A}$. For each $x \in \mathbb{A}$ and $Y \in \mathbb{S}$ set:

$$\gamma(x, Y) = \begin{cases} \{x_i\} & \text{if } i \text{ is the minimum index } j \text{ s.t.} \\ & x_j \in [x] \wedge \mathsf{tr}(x_j, Y) = \mathsf{true}, \text{ if it exists} \\ \emptyset & \text{otherwise} \end{cases}$$

Now define inductively for each $\alpha \in \mathit{Ord}$:

$$X_\alpha = \bigcup \{\gamma(x, X_{<\alpha}) \mid \mathsf{lev}(x) = \alpha\} \quad \text{where} \quad X_{<\alpha} = \bigcup_{\beta < \alpha} X_\beta$$

In words, X_α is obtained by choosing an answer x', if any, for each equivalence class $[x]$ with $\mathsf{lev}(x) = \alpha$, such that x' is true w.r.t. all the choices made at previous stages $\beta < \alpha$. Since $x_i \in [x]$ implies that $\mathsf{lev}(x_i) = \mathsf{lev}(x)$, X_α is made of answers of level α.

Then we prove that $X = \bigcup_{\alpha \in \mathit{Ord}} X_\alpha$ is a model. First by construction X_α is consistent for all α, because it contains at most one answer for each equivalence class; this implies that X is consistent, since two answers in the same equivalence class are in the same X_α. Second, if $x \in X \restriction \alpha$ then $x \in X_{<\alpha}$, so that:

$$\mathsf{tr}(x, X) = \mathsf{tr}(x, X \restriction \mathsf{lev}(x)) = \mathsf{tr}(x, X_{<\mathsf{lev}(x)}) = \mathsf{true}$$

and X is sound. Finally, for all $x \in \mathbb{A}$, if $X \cap [x] = \emptyset$ then

$$\begin{array}{rcl} X \cap [x] = \emptyset & \Rightarrow & X_{\mathsf{lev}(x)} \cap [x] = \emptyset \\ & \Rightarrow & \forall x' \in [x].\ \mathsf{tr}(x', X_{<\mathsf{lev}(x)}) = \mathsf{false} \\ & \Rightarrow & \mathsf{tr}(x, X_{<\mathsf{lev}(x)}) = \mathsf{false} \\ & \Rightarrow & \mathsf{tr}(x, X) = \mathsf{false} \end{array}$$

by $\mathsf{tr}(x, X) = \mathsf{tr}(x, X_{<\mathsf{lev}(x)})$. Therefore X is complete and hence a model.

■

The construction of Theorem 11 is not effective, even when the layered knowledge structure is recursive. Assume that $\gamma \in \mathit{Ord}$ is the number of levels of the knowledge structure. If we look closely to the proof, we see that we defined a model by some $\Delta^0_{1+\gamma}$-predicate. In particular, if there are infinitely many levels, then the definition is not an arithmetical predicate. We claim that the recursive complexity in our result is optimal: for any γ there is some recursive layered knowledge structure with γ levels, whose models are all (the extensions of) $\Delta^0_{1+\gamma}$-complete predicates, and therefore are never $\Delta^0_{1+\delta}$-predicate, for any $\delta < \gamma$. In general models are not recursive sets, and *a fortiori* are not finite.

5 Interactive Realizability

Given a layered knowledge structure $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$, the goal of a learning process is to reach some sound $X \in \mathbb{S}$ which is sufficiently large to compute a solution to the problem at hand, e.g. a k-tuple $n_1, \ldots, n_k$ of natural numbers witnessing the truth of (1) or of (2) in Section 2. To make this precise, we formally define what does it mean that a *problem* $P \subseteq \mathbb{N}$ has a solution α relative to a state X. Informally, we require that $\alpha(X)$ is a number continuously depending on a knowledge state X, which satisfies P whenever X is a model. In the terminology of [3] α is an "individual".

▶ **Definition 12** (Solution of a Problem w.r.t. a Knowledge Structure)**.** Let $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$ be a layered knowledge structure and $\mathbb{S}$ its space of states of knowledge. Given a continuous $\alpha : \mathbb{S} \to \mathbb{N}$ (where $\mathbb{N}$ is a discrete space) a predicate $P \subseteq \mathbb{N}$ (a *problem*), and $X \in \mathbb{S}$, we define:

1. $X \models_{\mathbb{A}} \alpha : P \Leftrightarrow \alpha(X) \in P$,
2. $\models_{\mathbb{A}} \alpha : P \Leftrightarrow \forall X \in \mathbb{S}.\ X$ is a model $\Rightarrow X \models_{\mathbb{A}} \alpha : P$.

When $\models_{\mathbb{A}} \alpha : P$ we say that α is a *solution* of P w.r.t. $(\mathbb{A}, \sim)$.

We shall omit the subscript $\mathbb{A}$ in $\models_{\mathbb{A}}$ when $\mathbb{A}$ is understood.

Example 4 Let $(\mathbb{A}_2, \sim_2)$ be the knowledge structure defined in example 1 in §3, and $\mathbb{S}_2$ its knowledge space. Fix $k \in \mathbb{N}$; writing $\langle n_1, \ldots, n_k \rangle$ for the code number of the k-tuple

$n_1, \dots, n_k$ we define the "problem" P_2:

$$P_2 = \{\langle n_1, \dots, n_k\rangle \mid \bigwedge_{i<k} (n_i < n_{i+1} \ \wedge \ f_1(n_i) \leq f_1(n_{i+1}) \ \wedge \ f_2(n_i) \leq f_2(n_{i+1}))\},$$

P_2 is the set of all (coding of) k-tuple witnessing that (1) and (2) in §2 are true. Now for any $X \in \mathbb{S}$ define:

$$\alpha_2(X) = \min\{\langle n_1, \dots, n_k\rangle \mid n_1 < \cdots < n_k \ \wedge \ n_1, \dots, n_k \in (\mathbb{N}_{f_1})^X_{f_2}\}$$

where min is understood as the lexicographic ordering of the k-tuples. By definition the mapping α_2 picks the first k elements in the set $(\mathbb{N}_{f_1})^X_{f_2}$ in increasing order. α_2 is no dummy search procedure, is a reading primitive that assumes that X has been given. α_2 is always defined because $\mathbb{N}^X_{f_1}$ and $(\mathbb{N}_{f_1})^X_{f_2}$ are infinite for every $X \in \mathbb{S}_2$. Indeed this can be proved by a relativization to X of the argument of Lemma 1: if $n \notin \mathbb{N}^X_{f_1}$ then there exists $m \in \mathbb{N}$ s.t. $n < m$ but $f_1(n) > f_1(m)$ in the knowledge state X, namely we have $(0, n, m) \in X$. Were $\mathbb{N}^X_{f_1}$ finite, we would be able to find infinitely many such m forming an infinite increasing chain, and so an infinite descending chain via f_1. Similarly one proves that $(\mathbb{N}_{f_1})^X_{f_2}$ is infinite (quantifying over $\mathbb{N}^X_{f_1}$ in place of $\mathbb{N}$ and coding the counterexamples known at X by $(1, n, m) \in X$).

We show that α_2 is continuous. Let $\alpha_2(X) = \langle n_1, \dots, n_k\rangle$: then $n_i \in (\mathbb{N}_{f_1})^X_{f_2}$ for all $i \leq k$, and and for all $m < n_k$ with $m \neq n_1, \dots, n_k$, we have $m \notin (\mathbb{N}_{f_1})^X_{f_2}$. Conversely one can check that for all $Y \in \mathbb{S}$, if $\bigwedge_{i \leq k} n_i \in (\mathbb{N}_{f_1})^Y_{f_2}$ and $\bigwedge_{m<n_k, m \neq n_1, \dots, n_k} m \notin (\mathbb{N}_{f_1})^Y_{f_2}$ then $\alpha_2(Y) = \langle n_1, \dots, n_k\rangle$. But since we know that the predicate $n \in (\mathbb{N}_{f_1})^Y_{f_2}$ is continuous in Y (see the remark after Lemma 5), the last condition defines a finite intersection of clopens, which is clopen.

We show that α_2 is a solution of P_2, that is, that $\models \alpha_2 : P_2$. Let X be a model: then we have $(\mathbb{N}_{f_1})^X_{f_2} = (\mathbb{N}_{f_1})_{f_2}$. Since $\alpha(X) = \langle n_1, \dots, n_k\rangle \in (\mathbb{N}_{f_1})^X_{f_2}$, we deduce $\alpha_2(X) \in (\mathbb{N}_{f_1})_{f_2}$, that is, that f_1 and f_2 are weakly increasing w.r.t. $n_1, \dots, n_k$. Thus, $\alpha_2(X) \in P_2$.

A solution α is some way to produce an inhabitant $\alpha(X) \in P$ out of any model X. A learning strategy for a problem P admitting a solution α w.r.t. $(\mathbb{A}, \sim, \mathsf{lev}, \mathsf{tr})$ is ideally a search procedure of some model $X \in \mathbb{S}$. But models are in general infinite and non-recursive states of knowledge: to make learning effective we rely on the continuity of α which implies that if $\alpha(X) = n \in P$ for some model X there exists a finite $s \subseteq X$ such that $\alpha(s) = n$.

We describe the search of such finite sound approximations of a model X via certain continuous functions r over $\mathbb{S}$. The function r such that for any sound X (not necessarily a model) the set $r(X)$ is a finite set of answers that are not in X but are compatible with the answers in X and true w.r.t. X, and if $r(X) = \emptyset$ then $\alpha(X) \in P$. When such a function exists for given P and α we say that it is a *realizer* and that the solution α is realized by r.

▶ **Definition 13** (Realizers and Zeros). A *realizer* is a continuous map $r : \mathbb{S} \to \mathcal{P}_{fin}(\mathbb{A})$ such that for all $X \in \mathbb{S}$ and all $x \in r(X)$:

1. $X \cap [x] = \emptyset$,
2. $\mathsf{tr}(x, X) = \mathsf{true}$.

We denote by $\mathcal{R}$ the set of realizers. Finally we say that $X \in \mathbb{S}$ is a *zero* of $r \in \mathcal{R}$ if $r(X) = \emptyset$.

We can see a realizer r as the essential part of a learning strategy, which tries to update the current state of knowledge. This is obtained by evaluating $r(X)$ to get a finite set of new answers by which X could be soundly extended. To see this let $\mathsf{new} : \mathbb{S} \times \mathcal{P}_{fin}(\mathbb{A}) \to \mathcal{P}_{fin}(\mathbb{A})$ be defined by:

$$\mathsf{new}(X, U) = \{x \in U \mid X \cap [x] = \emptyset \wedge \mathsf{tr}(x, X) = \mathsf{true}\}.$$

The function new is continuous, where $\mathcal{P}_{\mathit{fin}}(\mathbb{A})$ and $\mathbb{S} \times \mathcal{P}_{\mathit{fin}}(\mathbb{A})$ are taken with discrete and product topology respectively.

Say that an *operator* over $\mathbb{S}$ is any continuous map $r : \mathbb{S} \to \mathcal{P}_{\mathit{fin}}(\mathbb{A})$, and call $Op_{\mathbb{S}}$, or simply Op, the set of operators over $\mathbb{S}$. For $r \in Op$ we set: $\widehat{r}(X) = \mathsf{new}(X, r(X))$. Note that the set of realizers is the subset of operators such that $r = \widehat{r}$.

Example 5 We propose a realizer solving the problem (1), expressed by the predicate:

$$P_1 = \{\langle n_1, \ldots, n_k\rangle \mid \bigwedge_{i<k} (n_i < n_{i+1} \ \wedge \ f_1(n_i) \le f_1(n_{i+1}))\}.$$

We first define a function $\beta_1(X, \langle n_1, \ldots, n_h\rangle)$ extending any list $\langle n_1, \ldots, n_h\rangle$ with $h \le k$ to a solution of (1):

$$\beta_1(X, \langle n_1, \ldots, n_h\rangle) = \begin{cases} \langle n_1, \ldots, n_h\rangle & \text{if } h = k \\ \beta_1(X, \langle n_1, \ldots, n_h, m\rangle) & \text{where } m \text{ is the minimum s.t.} \\ & m \in \mathbb{N}^X_{f_1} \text{ and} \\ & m > n_h \text{ if } h > 0. \end{cases}$$

A function solving (1) may then be defined by $\alpha_1(X) = \beta_1(X, \langle\,\rangle)$. We claim that $\models \alpha_1 : P_1$. Indeed β_1 is continuous w.r.t. X since $m \in \mathbb{N}^X_{f_1}$ is equivalent to $X \notin B_{(0,m,m+1)}$ which is a clopen by 5.1, so that α_1 is continuous. Further, if X is a model, then $\alpha_1(X) \in P_1$. We define a realizer r_1 looking for some X such that $\alpha_1(X)$ solves P_1. $r_1(X)$ takes any knowledge state X, and adds to it the first counterexample to (1) we may find in the list $\langle n_1, \ldots, n_k\rangle$ generated by α_1, unless $\alpha_1(X)$ solves P_1. We define r_1 in two steps: first, we define a map g_1 finding the first counterexample to (1) in a given list:

$$g_1(\langle n_1, \ldots, n_k\rangle) = \begin{cases} \{(0, n_i, n_{i+1}) \mid 1 \le i < k \text{ min. s.t.} & \\ \qquad\qquad f_1(n_i) > f_1(n_{i+1})\} & \text{if } i \text{ exists} \\ \emptyset & \text{otherwise.} \end{cases}$$

Then we define the realizer by composing g_1 with the output of α_1: $r_1(X) = g_1(\alpha_1(X))$. r_1 is a realizer, because $r_1(X)$ always outputs atoms not in X. Indeed, if $\alpha_1(X) = \langle n_1, \ldots, n_k\rangle \in \mathbb{N}^X_{f_1}$, and if $r_1(X)$ output the atom $f_1(n_i) > f_1(n_{i+1})$, then $(f_1(n_i) > f_1(n_{i+1})) \notin X$ by definition of $\langle n_1, \ldots, n_k\rangle \in \mathbb{N}^X_{f_1}$. We may use $r_1(X)$ to extend X until we find some X such that $r_1(X) = \emptyset$. Whenever $r_1(X) = \emptyset$ we have $g_1(\alpha_1(X)) = \emptyset$, hence $\alpha_1(X)$ solves (1) by definition of g_1.

To step from P_1 to P_2, namely to problem (2), we just replace $\mathbb{N}^X_{f_1}$ by $\mathbb{N}^X_{f_1,f_2}$, namely:

$$\beta_2(X, \langle n_1, \ldots, n_h\rangle, k) = \begin{cases} \langle n_1, \ldots, n_h\rangle & \text{if } h = k \\ \beta_2(X, \langle n_1, \ldots, n_h, m\rangle, k) & \text{where } m \text{ is the minimum s.t.} \\ & m \notin (\mathbb{N}_{f_1})^X_{f_2} \\ & \text{and } m > n_h \text{ if } h > 0. \end{cases}$$

and

$$g_2(\langle n_1, \ldots, n_k\rangle) = \begin{cases} \{(0, n_i, n_{i+1}) \mid 1 \le i < k \text{ min. s.t.} & \\ \quad f_1(n_i) > f_1(n_{i+1}) \vee f_2(n_i) > f_2(n_{i+1})\} & \text{if } i \text{ exists} \\ \emptyset & \text{otherwise.} \end{cases}$$

We have that $\alpha_2(X) = \beta_2(X, \langle\,\rangle, k)$, where α_2 is from example 4. Now let us define $r_2(X) = g_2(\beta_2(X, \langle\,\rangle, k)) = g_2(\alpha_2(X))$. Then we can show that r_2 is a realizer looking for some X such that $\alpha_2(X) \in P_2$ just as in the case of r_1 above.

If X is a model and $r \in \mathcal{R}$ then $r(X) = \emptyset$ by definition; on the other hand if $\models_{\mathbb{A}} \alpha : P$ then $\alpha(s) = \alpha(X) = n \in P$ for some finite $s \subseteq X$. Since r is continuous, the condition that reveals that the approximation s of X is good enough to compute an $n \in P$ is that $r(s) = r(X) = \emptyset$. This suggests that a constructive way to meet the requirement about models in the definition of $\models_{\mathbb{A}} \alpha : P$ is to ask that sound zeros of a realizer r are enough to find inhabitants of P via α, and then look for finite sound zeros of r. We turn this into the following definition.

▶ **Definition 14** (Interactive Realizability). Let $\alpha : \mathbb{S} \to \mathbb{N}$ be continuous (w.r.t. the discrete topology over $\mathbb{N}$), and $P \subseteq \mathbb{N}$ a predicate:

1. $r \in \mathcal{R}$ *interactively realizes* P w.r.t. α, written $r \vdash \alpha : P$, if and only if:

$$\forall X \in \mathbb{S}.\ X \text{ sound zero of } r \Rightarrow \alpha(X) \in P$$

2. P is *interactively realizable* w.r.t. α, written $\vdash \alpha : P$, if and only if:

$$\exists r \in \mathcal{R}.\ r \vdash \alpha : P.$$

If P_1 and P_2 are the predicates defined in examples 4 and 5, α_1, α_2 their respective solutions and r_1, r_2 the realizers from example 5; then we claim (without proof) that $r_i \vdash \alpha_i : P_i$ for both $i = 1, 2$.

The main result of the paper is that the apparently stronger $r \vdash \alpha : P$ for some $r \in \mathcal{R}$ is equivalent to $\models \alpha : P$. That is, whenever $\alpha : P$ is valid then it is interactively learnable, and we have some strategy to find some finite X such that $\alpha(X) \in P$.

Before we establish the existence of sound finite zeros of any $r \in \mathcal{R}$. This is a non trivial fact because, whenever we add to some state Y (no matter whether finite or not) a $y \in r(Y)$, we know that $\mathsf{tr}(z, Y) = \mathsf{true}$ for all $z \in Y$, but we do not know about the value of $\mathsf{tr}(z, Y \cup \{y\})$, so that $Y \cup \{y\}$ is not necessarily sound. Moreover it is not true that if $s \subseteq X$ and X is sound then s is sound.

Example 6 Let us redefine f_1 by $f_1(0) = 10, f_1(1) = 30, f_1(2) = 20$ and define $f_2(0) = 20, f_2(1) = 10, f_2(2) = 20$. Also we let $x = (1, 0, 1)$ meaning that $0 \notin (\mathbb{N}_{f_1})_{f_2}$, and $y = (0, 1, 2)$ meaning that $1 \notin \mathbb{N}_{f_1}$. Then $\mathsf{tr}(x, \{x\}) = \mathsf{true}$ since at $\{x\}$ it is likely that $0 \notin (\mathbb{N}_{f_1})_{f_2}$ because of the counterexample in the point 1; but $\mathsf{tr}(x, \{x, y\}) = \mathsf{false}$ because the discovery that $1 \in \mathbb{N}_{f_1}$ contradicts counterexample on point 1.

We prove below that finite sound zeros exist for all $r \in \mathcal{R}$ and that these are finite approximations of sound states of knowledge which are themselves zeros of r, hence in particular of models.

▶ **Lemma 15.** *If $X \in \mathbb{S}$ is sound, $s \in \mathbb{S}_{\mathit{fin}}$ is a finite state such that $s \subseteq X$, then there exists a finite sound $t \in \mathbb{S}_{\mathit{fin}}$ such that $s \subseteq t \subseteq X$.*

We are now in place to conclude the proof that every realizer has a finite sound zero. We do not provide an effective method to find some, but we claim that we can obtain it by a suitable sequence of answers added by the realizer and of removal of answers.

▶ **Theorem 16** (Existence of Sound and Finite Zeros of Realizers). *If $r \in \mathcal{R}$, then there exists a finite sound zero $t \in \mathbb{S}_{\mathit{fin}}$ of r.*

Proof. Models exist by Theorem 11 and they are sound by definition, hence $r(X) = \emptyset$ for some sound $X \in \mathbb{S}$ since $r \in \mathcal{R}$. By continuity there is a basic open $\mathcal{O}_{s_0,t_0}$ such that $X \in \mathcal{O}_{s_0,t_0}$ and $r(\mathcal{O}_{s_0,t_0}) = \emptyset$. This implies that $s_0 \subseteq X$ and $X \cap t_0 = \emptyset$, so that a fortiori any finite $t \in \mathbb{S}_{\mathit{fin}}$ such that $s_0 \subseteq t \subseteq X$ satisfies $t \cap t_0 = \emptyset$ and therefore $t \in \mathcal{O}_{s_0,t_0}$, i.e. it is a zero of r. By Lemma 15 there exists a sound t among them, which is the desired finite sound zero of r. ■

We come now to the completeness theorem. Our thesis is that interactive realizability is complete in the sense that if $\alpha(X) \in P$ for all models X, then we may replace the model X by the finite sound zeros of a suitable realizer $r \in \mathcal{R}$.

▶ **Theorem 17.** *(Completeness of Realization) For any continuous $\alpha : \mathbb{S} \to \mathbb{N}$ and predicate $P \subseteq \mathbb{N}$: $\vdash \alpha : P \Leftrightarrow \models \alpha : P$.*

Proof.

($\Rightarrow$) If $X \in \mathbb{S}$ is a model then X is a sound zero of any realizer by Definition 13; hence if $r \vdash \alpha : P$ for some $r \in \mathcal{R}$ we immediately have $\alpha(X) \in P$, i.e. $X \models \alpha : P$ for arbitrary model X.

($\Leftarrow$) We have to show that, if $\models \alpha : P$, namely if $\alpha(X) \in P$ for $X \in \mathtt{M}$, then there exists an $r \in \mathcal{R}$ such that $r \vdash \alpha : P$. We establish the contrapositive:

$$\alpha(X) \notin P \Rightarrow X \text{ not sound } \vee \ r(X) \neq \emptyset$$

for some realizer r and arbitrary $X \in \mathbb{S}$.

If $\alpha(X) \notin P$ then, by the hypothesis, $X \notin \mathtt{M}$, hence $X \notin \mathtt{S}$ or $X \notin \mathtt{C}$. By definition of $\mathtt{S}$ and $\mathtt{C}$, this implies that $\exists x \in \mathbb{A}.\ X \notin \mathtt{S}_x \vee X \notin \mathtt{C}_x$. Fix an enumeration $x_0, x_1, \ldots$ of the countable set $\mathbb{A}$. Let us define $r : \mathbb{S} \to \mathcal{P}_{\mathit{fin}}(\mathbb{A})$ by:

$$r(X) = \begin{cases} \emptyset & \text{if } \alpha(X) \in P \\ \{x_i\} & \text{where } i = \min\{j \in \mathbb{N} \mid X \in \mathbb{S} \setminus \mathtt{M}_{x_j}\}, \text{ else.} \end{cases}$$

Then r is a total function since if $\alpha(X) \notin P$ then $X \in \mathbb{S} - \mathtt{M}$ so that $\{x_j \in \mathbb{A} \mid X \in \mathbb{S} \setminus \mathtt{M}_{x_j}\} \neq \emptyset$. If r is continuous then $\widehat{r}(X) = \mathsf{new}(X, r(X))$ because $X \in \mathbb{S} \setminus \mathtt{M}_{x_j}$ implies $X \in \mathbb{S} \setminus \mathtt{C}_{x_j}$, and consequently, r is a realizer. We have $\widehat{r} \vdash \alpha : P$. Indeed, assume for contradiction that $\alpha(X) \notin P$, $X \in \mathtt{S}$ and $\widehat{r}(X) = \emptyset$. Then $r(X) = \{x_i\}$ and $X \in \mathbb{S} \setminus \mathtt{M}_{x_i} = (\mathbb{S} \setminus \mathtt{S}_{x_i}) \cup (\mathbb{S} \setminus \mathtt{C}_{x_i})$. Since $X \in \mathtt{S} \subseteq \mathtt{S}_{x_i}$, then $X \in \mathbb{S} \setminus \mathtt{C}_{x_i}$. We conclude that $\widehat{r}(X) = \{x_i\}$, contradiction.

To see that r is continuous it suffices to check that both $r^{-1}(\emptyset)$ and $r^{-1}(\{x\})$ (for any $x \in \mathbb{A}$) are opens in $\Omega(\mathbb{S})$. Now $r(X) = \emptyset$ if and only if $\alpha(X) \in P$, that is $X \in \alpha^{-1}(P)$ which is clopen by Lemma 10. On the other hand $X \in r^{-1}(\{x\})$ if and only if:

$$\exists i.\ x_i = x \ \wedge \ X \in (\mathbb{S} \setminus \mathtt{M}_{x_i}) \wedge \ \forall j < i.\, X \in \mathtt{M}_{x_j}.$$

This is equivalent to $X \in \mathtt{M}_{x_0} \cap \ldots \cap \mathtt{M}_{x_{i-1}} \cap (\mathbb{S} \setminus \mathtt{M}_{x_i})$ which, by Lemma 10, is a finite intersection of clopens, hence a clopen itself.

■

6 Concluding remarks and further work

We have defined the notions of state of knowledge and of state topology. We have then redefined in the more general setting of non-monotonic learning, the concepts of individual (here called "solution") and of interactive realizer that we treated elsewhere, proving the completeness of learnability w.r.t. validity, which is the counterpart of classical truth in the present setting.

The definitions and results obtained are aimed at the development of a full theory of learning strategies and of their convergence properties, which is work in progress. We also observe that the solution and the realizer illustrated in the examples of §5 are crude simplifications of the learning strategy implicit in the example of §2, which is capable of using the counterexamples in a more ingenuous and efficient way. The investigation of the interpretation of classical proofs in terms of learning strategies is a natural further step, extending the work we have done in the monotonic case.

Since learning strategies working with finite approximations are effective (and indeed we have shown that finite and sound knowledge states exist and suffice), a question of efficiency of the algorithms one extracts from proofs with our method is naturally there, together with the analysis of suitable data structures representing time and logical dependancies, which are essential to complete the present approach.

References

1 Federico Aschieri and Stefano Berardi. Interactive Learning-Based Realizability for Heyting Arithmetic with $\mathtt{EM}_1$. *Logical Methods in Computer Science*, 6(3), 2010.

2 Stefano Berardi and Ugo de' Liguoro. Toward the interpretation of non-constructive reasoning as non-monotonic learning. *Information and Computation*, 207(1):63–81, 2009.

3 Stefano Berardi and Ugo de' Liguoro. Interactive realizers. A new approach to program extraction from non constructive proofs. *ACM Transactions on Computational Logic*, 13(2):11:1–11:21, 2012.

4 Thierry Coquand. A semantics of evidence for classical arithmetic. In Gordon Plotkin Gérard Huet and Claire Jones, editors, *Proceedings of the Second Workshop on Logical Frameworks*, pages 87–99, `http://www.lfcs.inf.ed.ac.uk/research/types-bra/proc/`, 1991.

5 Thierry Coquand. A semantics of evidence for classical arithmetic. *J. Symb. Log.*, 60:325–337, 1995.

6 Daniel Fridlender. Highman's lemma in theory. In Eduardo Giménez and Christine Paulin-Mohring, editors, *Types for Proofs and Programs, International Workshop TYPES'96, Aussois, France, December 15-19, 1996, Selected Papers*, volume 1512 of *LNCS*, pages 112–133, 1998.

7 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove, and Dana S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.

8 E. Mark Gold. Limiting recursion. *J. Symb. Log.*, 30:28–48, 1965.

9 E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

10 Timothy G. Griffin. The formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90, San Francisco, CA, USA, 17–19 Jan 1990*, pages 47–57. ACM Press, New York, 1990.

11 Susumu Hayashi. Mathematics based on incremental learning, excluded middle and inductive inference. *Theor. Comp. Sci.*, 350:125–139, 2006.

12 Jean-Louis Krivine. Realizability in classical logic. In *Interactive Models of Computation and Program Behavior*, Panoramas et Synthèses, 2009.

13 Alexandre Miquel. Existential witness extraction in classical realizability and via a negative translation. *Logical Methods in Computer Science*, 7(2), 2011.

14 Jan von Plato. A Constructive Approach to Sylvester's Conjecture. *J. UCS*, 11(12):2165–2178, 2005.

Bounded Satisfiability for PCTL[†]

Nathalie Bertrand[1,2], John Fearnley[2], and Sven Schewe[2]

1 **Inria Rennes Bretagne Atlantique, Rennes, France**
2 **Department of Computer Science, University of Liverpool, Liverpool, UK**

Abstract

While model checking PCTL for Markov chains is decidable in polynomial-time, the decidability of PCTL satisfiability is a long standing open problem. While general satisfiability is an intriguing challenge from a purely theoretical point of view, we argue that general solutions would not be of interest to practitioners: such solutions could be too big to be implementable or even infinite. Inspired by bounded synthesis techniques, we turn to the more applied problem of seeking models of a bounded size: we restrict our search to implementable – and therefore reasonably simple – models. We propose a procedure to decide whether or not a given PCTL formula has an implementable model by reducing it to an SMT problem. We have implemented our techniques and found that they can be applied to the practical problem of sanity checking – a procedure that allows a system designer to check whether their formula has an unexpectedly small model.

1998 ACM Subject Classification I.2.2 Automatic Programming, F.4.1 Mathematical Logic.

Keywords and phrases Satisfiability, Temporal Logic, Probabilistic Logic.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.92

1 Introduction

PCTL [9] is a popular logic for the specification of probabilistic systems. The model checking problem for PCTL formulas over Markov chains has been widely studied: it is known to be solvable in polynomial time, and mature tools, such as PRISM [13] and MRMC [10], have been developed. By contrast, satisfiability procedures for PCTL have received much less attention. Recently, it has been shown that the satisfiability problem for the qualitative fragment of PCTL is EXPTIME-complete [4]. However, the satisfiability problem for PCTL itself is not even known to be decidable.

In this paper, we take a step back, and ask the following question: even if we could find an algorithm for the satisfiability of PCTL, would it be useful to a practitioner? We believe that it would not. Even in the qualitative fragment of PCTL, there are already formulas for which there are only infinite state models. Moreover, the problem of deciding, for this fragment of PCTL, whether there is a finite model is EXPTIME-hard [4]. Obviously, the situation is at least as bad for the full PCTL logic.

When practitioners use satisfiability procedures, it is likely that they are interested in whether their formula has an *implementable* model. A constructive satisfiability procedure may return an infinite state model, or a model with bizarre transition probabilities that may be difficult to implement in practice. Neither of these two situations seems to be desirable. Hence, our goal is to solve the following problem.

“Does a PCTL specification ϕ have an implementable model?”

[†] This work was supported by the Engineering and Physical Science Research Council grant EP/H046623/1 ‘Synthesis and Verification in Markov Game Structures’ and a Leverhulme Trust Visiting Fellowship. Full version available at `http://arxiv.org/abs/1204.0469`.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 92–106
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our results are inspired by the work on bounded synthesis for LTL specifications [16, 8, 6, 11, 7], where the question of whether there is a small reactive system for an LTL formula is considered. Building on this, we define the *bounded satisfiability* problem for PCTL formulas, where the goal is to find a *simple* model of a PCTL formula. In our setting, a model is simple if it has a small number of states, and if it uses only rational transition probabilities that can be easily simulated by, for example, coin tossing. We believe that having a simple model is a prerequisite for having an implementable model. Certainly, infinite models, and models whose probabilities cannot be simulated by coin tossing do not seem to be useful in practice.

Results

In this paper, we introduce the concept of a simple model, and the bounded satisfiability problem. This is the problem of finding, for a given PCTL formula ϕ and bound b, a simple model of ϕ with at most b states. We provide a reduction from bounded satisfiability to SMT. While PCTL satisfiability is not known to be decidable, our results show that the bounded satisfiability problem can be decided. We also provide complexity results for the bounded satisfiability problem. We show that it is NP-complete in the size of the minimal model. Furthermore, we show that approximating the size of the minimal model is NP-hard.

We have constructed a simple implementation of our reduction from bounded satisfiability to SMT, and we have solved the resulting constraint systems using the Yices SMT solver [5]. We tested this implementation on an academic case study, and our results show that the bounded satisfiability problem can indeed be solved when the number of states required is small.

Practical Applications

While our simple implementation does show that small models can be found by our techniques, the size of these models is clearly well below what would be required for constructing systems in an industrial setting. On the other hand, we argue that a bounded synthesis procedure for even a small number of states is useful for the purpose of *sanity checking.*

In model checking, we attempt to verify that a potentially buggy system satisfies a specification. However, in recent years it has become increasingly clear that the specification itself may also contain bugs. See, for example, the work on vacuity checking [3, 15, 12, 1, 2]. It can also be remarkably difficult to detect these errors, since a model checking procedure will simply output “yes” in the case where a buggy system satisfies the buggy specification.

We propose that bounded satisfiability has a role to play in helping system designers find bugs in their specifications. Consider, for instance, a complicated specification of a network protocol that allows, among many other things, the network to go down and subsequently be recovered. A buggy specification may allow a model that goes down immediately after service has been restored, which would allow the model to circumvent most of the specification. This model will have far fewer states than a model that implements a correctly functioning network. In this case, while a correct system may be too large to build with our techniques, it is quite possible that the broken system could be built.

Hence, we propose that bounded satisfiability should be used as a sanity check, in order to test that a formula does not have an error that can be exploited by a small model. Suppose that a system designer has a large system that is known to satisfy some complicated specification. If the bounded satisfiability procedure produces a small model for the specification, then there is a problem that can be resolved in one of two ways. Firstly, it may be the case that the small model does precisely what the designer wants, and in this case the overly-complex large system can be replaced by the small one. The more likely outcome is that the small model does not do what the designer intended. In this case, the designer now has a small counter-example, which can be used to help correct the specification.

Our experimental results show that our procedure is particularly suitable for sanity checking. While our implementation does not seem to scale well with the number of states in the model, it does scales well with size of the input formula. This indicates that sanity checking may indeed be possible for the type formulas that are used in practice.

2 Preliminaries

2.1 Markov chains

We recall below the definition of discrete-time Markov chains, simply referred to as Markov chains in the sequel.

▶ **Definition 1** (Markov chain). A Markov chain is a tuple $\mathcal{M} = (S, \mathbf{P}, \iota, L)$, where S is a finite or countable set of states, $\mathbf{P} : S \times S \to [0, 1]$ is a probabilistic transition function, $\iota \in S$ is an initial state, and L is a labelling function mapping states to atomic propositions. $\mathbf{P}$ satisfies, for all $s \in S$:

$$\sum_{s' \in S} \mathbf{P}(s, s') = 1$$

A *path* in $\mathcal{M}$ is a sequence of states $\pi = s_0 s_1 \cdots \in S^*$ such that, for every $i \in \mathbb{N}$, $\mathbf{P}(s_i, s_{i+1}) > 0$. The set of all paths starting in a state $s \in S$ is denoted $\mathsf{Paths}(s)$. In order to define a probability measure $\Pr_{\mathcal{M}}$ over suitable sets of paths, we first explain how a measure is associated with basic sets of paths called cylinder sets, which gather all paths sharing a given finite prefix. For $s_0 s_1 \cdots s_k$ a finite sequence of states, we let $\mathsf{Cyl}(s_0 s_1 \cdots s_k) = \{\pi \in \mathsf{Paths}(s_0) \mid s_0 s_1 \cdots s_k \prec \pi\}$ where $\prec$ is the usual prefix order, and define its measure as $\Pr^{s_0}_{\mathcal{M}}(\mathsf{Cyl}(s_0 s_1 \cdots s_k)) = \prod_{0 \leq i < k} \mathbf{P}(s_i, s_{i+1})$. For $s \neq s_0$, $\Pr^{s}_{\mathcal{M}}(\mathsf{Cyl}(s_0 s_1 \cdots s_k)) = 0$.

Now $\Pr^{s}_{\mathcal{M}}$ can be extended to a set of reasonable sets of paths, namely the σ-algebra generated from cylinder sets. This σ-algebra over $\mathsf{Paths}(\mathcal{M}) = \bigcup_{s \in S}(\mathsf{Paths}(s))$ precisely consists of the smallest collection of sets of paths in $\mathcal{M}$ that contains the empty set, all $\mathsf{Cyl}(s_0 s_1 \cdots s_k)$ for any finite sequence $s_0 s_1 \cdots s_k$ of states, and is closed under complementation and countable union. The extension of $Pr^{s}_{\mathcal{M}}$ from cylinders to the σ-algebra they generate is unique, and we still denote it $\Pr^{s}_{\mathcal{M}}$. Note that not all sets of paths are measurable with respect to $\Pr^{s}_{\mathcal{M}}$, but the sets we will consider in this paper are simple enough to avoid such difficulties. We use $\Pr_{\mathcal{M}}$ as an abbreviation of $\Pr^{\iota}_{\mathcal{M}}$.

2.2 PCTL

Probabilistic computation tree logic (PCTL) [9] is a probabilistic variant of CTL, where path quantifiers are replaced by probabilistic operators. PCTL is interpreted over Markov chains, and one can for example specify that the probability measure of the set of paths satisfying a given until property exceeds some threshold. Formally, the syntax of PCTL is given by the following grammar:

▶ **Definition 2** (PCTL syntax). Let AP be a set of atomic propositions. The syntax of a PCTL formula is:

$$\phi ::= \top \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathbb{P}_{\bowtie\lambda}\tau$$
$$\tau ::= \bigcirc\, \phi \mid \phi\, \mathcal{U}\, \phi \mid \phi\, \mathcal{U}^{\leq n}\, \phi$$

where $a \in AP$ is an atomic proposition, $\bowtie$ is a comparison operator in $\{<, \leq, =, \geq, >\}$, $\lambda \in [0,1]$ is a rational threshold, and $n \in \mathbb{N}$.

Formulas produced by the production rules of ϕ and τ are called state formulas and path formulas, respectively. State formulas are also called PCTL formulas. PCTL formulas are interpreted over Markov chains. The *step-bounded until* operator has the intuitive semantics that $\mathbb{P}_{\bowtie\lambda}(\phi\,\mathcal{U}^{\leq n}\,\psi)$ is true if the probability is $\bowtie \lambda$ that "ψ holds within the next n steps, and ϕ is true until ψ is true". Hence, the step-unbounded until formula $\mathbb{P}_{\bowtie\lambda}(\phi\,\mathcal{U}\,\psi)$ can be thought of $\mathbb{P}_{\bowtie\lambda}(\phi\,\mathcal{U}^{<\infty}\,\psi)$.

The syntax and semantics of PCTL only differ from those of CTL by using probabilistic path operators $\mathbb{P}_{\bowtie\lambda}(\bigcirc \cdots)$, $\mathbb{P}_{\bowtie\lambda}(\cdots\,\mathcal{U}\,\cdots)$ and $\mathbb{P}_{\bowtie\lambda}(\cdots\,\mathcal{U}^{\leq n}\,\cdots)$ instead of universal and existential ones.

▶ **Definition 3** (PCTL semantics)**.** Let $\mathcal{M} = (S, \mathbf{P}, \iota, L)$ be a Markov chain, $s \in S$ a state of $\mathcal{M}$, and ϕ, ϕ' PCTL formulas. We have:

- $\mathcal{M}, s \models \top$ for all $s \in S$,
- $\mathcal{M}, s \models a$ iff $a \in L(s)$,
- $\mathcal{M}, s \models \phi \wedge \phi'$ iff $\mathcal{M}, s \models \phi$ and $\mathcal{M}, s \models \phi'$,
- $\mathcal{M}, s \models \neg\phi$ iff $\mathcal{M}, s \not\models \phi$,
- $\mathcal{M}, s \models \mathbb{P}_{\bowtie\lambda}\tau$ iff $\mathrm{Pr}^{s}_{\mathcal{M}}\big(\{\pi \in \mathsf{Paths}(s) \mid \pi \models \tau\}\big) \bowtie \lambda$.

Finally, for a path $\pi = s_0 s_1 \cdots \in \mathsf{Paths}(s)$ and PCTL formulas ϕ, ϕ' we have:

- $\mathcal{M}, \pi \models \bigcirc\,\phi$ iff $s_1 \models \phi$,
- $\mathcal{M}, \pi \models \phi\,\mathcal{U}\,\phi'$ iff $\exists i.\ s_i \models \phi'$ and $\forall j < i.\ s_j \models \phi$,
- $\mathcal{M}, \pi \models \phi\,\mathcal{U}^{\leq n}\,\phi'$ iff $\exists i \leq n.\ s_i \models \phi'$ and $\forall j < i.\ s_j \models \phi$.

Note that the semantics is well-defined because specified sets of paths are indeed measurable. We use the usual shorthand notations known from CTL, such as $\Diamond\,\phi \equiv \top\,\mathcal{U}\,\phi$ and $\Box\,\phi \equiv \neg\Diamond\,\neg\phi$. Using duality of eventually and always operators and the duality of lower and upper bounds of our probabilistic path operators, we can, for example, express $\mathbb{P}_{\leq\lambda}(\Box\,\phi) \equiv \mathbb{P}_{\geq 1-\lambda}(\Diamond\,\neg\phi)$.

3 Setting and problem statement

The model checking problem for PCTL over Markov chains is known to be solvable in polynomial time [9]. In contrast to this, satisfiability, that is, the decision problem that asks whether or not a formula has a model, is a long standing open problem for PCTL. Only recently, satisfiability for the restricted fragment of qualitative PCTL, where thresholds can only take value 0 or 1, has been shown to be decidable (EXPTIME-complete [4]). Already this qualitative fragment does not have the finite model property. As an example formula $\phi = \mathbb{P}_{>0}\big(\Box\,(\neg a \wedge \mathbb{P}_{>0} \bigcirc a)\big)$ is satisfiable (it has a model with infinitely many states) but has no finite model [4]. Whether or not a formula has a model surely is a challenging theoretical question, but ultimately not a question of practical interest, especially if all of its models are infinite. On the contrary, an interesting question in practice is whether or not a formula has a *simple* model, where by 'simple' one intends reasonably small and implementable, at least. The problem is then to determine whether or not a formula admits a model with a bounded number of states. This bounded satisfiability problem [16, 8, 6, 11, 7] has first been studied in [16] for LTL and distributed architectures, where it is reduced to an SMT problem.

In the context of the probabilistic branching time logic PCTL and Markov chain models, we advocate that non-implementability can not only result from a large (or even infinite) state space, but also from the transition probabilities. The representation of arbitrary

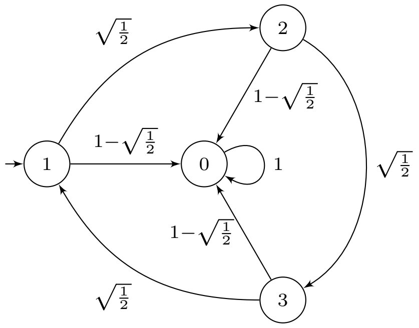


Figure 1 An irrational model for ψ_0.

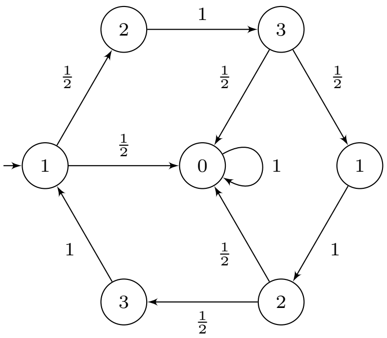


Figure 2 A simple model for ψ_0.

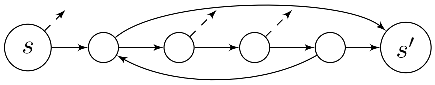
rational or irrational probabilities definitely forms a source of complexity when it comes to implementing a system described by a model. To illustrate this problem, let us consider the following PCTL formula over 2 atomic propositions $\{p, q\}$, which are used to encode $0 \equiv \neg p \wedge \neg q$, $1 \equiv \neg p \wedge q$, $2 \equiv p \wedge \neg q$, and $3 \equiv p \wedge q$.

$$\begin{aligned}\psi_0 :=& 1 \wedge \mathbb{P}_{=1}\Box\Bigg(\Big(0 \to \mathbb{P}_{=1}(\bigcirc 0)\Big)\\ &\wedge \Big(1 \to \mathbb{P}_{=1}(\bigcirc (2 \vee 0))\Big) \wedge \Big(2 \to \mathbb{P}_{=1}(\bigcirc (3 \vee 0))\Big) \wedge \Big(3 \to \mathbb{P}_{=1}(\bigcirc (1 \vee 0))\Big)\\ &\wedge \Big(1 \to \mathbb{P}_{=1/2}(1 \vee 2\,\mathcal{U}\,3)\Big) \wedge \Big(2 \to \mathbb{P}_{=1/2}(2 \vee 3\,\mathcal{U}\,1)\Big) \wedge \Big(3 \to \mathbb{P}_{=1/2}(3 \vee 1\,\mathcal{U}\,2)\Big)\Bigg)\end{aligned}$$

The formula specifies that the initial state is labelled with 1, and the 0 states form a sink. It also requires that all successors of states labelled 1 are labelled 2 or 0 (and similarly for 2 and 3); and last, with probability exactly $\frac{1}{2}$, from the state labelled 1, only 1 or 2 are visited until the state labelled 3 is reached (symmetrically for 2 and 3). A model with four states for this formula is represented in Figure 1. Although the bounds in ψ_0 are all rationals (as required by the PCTL syntax), the formula forces every four-state model to have irrational probabilities on their edges. Indeed, the model shown in Figure 1 is the only four state model of ψ_0. The first two lines of ψ_0, for example, require that the initial state is labelled with 1, and that all successors of states labelled with 0 must also be labelled with 0. Also, successors of state 1 are among 0 and 2, and so forth. Moreover, letting $\mathbf{P}(i, j)$ be the probability to transition from state i to state j, the three last conjuncts in ψ_0 imply that $\mathbf{P}(1,2) \cdot \mathbf{P}(2,3) = \mathbf{P}(2,3) \cdot \mathbf{P}(3,1) = \mathbf{P}(3,1) \cdot \mathbf{P}(1,2) = \frac{1}{2}$. This yields $\mathbf{P}(1,2) = \mathbf{P}(2,3) = \mathbf{P}(3,1) = \sqrt{\frac{1}{2}}$. We emphasise that models with irrational probabilities are not implementable, because they require infinite memory.

If we use the same example formula ψ_0 but allow for a larger number of states, then the specification becomes easier to satisfy. Seven states suffice for the rational model shown in Figure 2. This model uses only rational probabilities. Moreover, all transitions carry the probability $\frac{1}{2}$ or 1. If we allow for multi-graphs, then we can split transitions with probabilities 1 into two transitions with probability $\frac{1}{2}$ with the same source and target. This inspires our definition of simple models: this example model can be represented as a multi-graph with two (potentially equivalent) outgoing transitions per node, each of which is taken with probability $\frac{1}{2}$.

Moreover, any rational probability $\frac{p}{q}$ can be written using its binary encoding of the form $0.vw^\omega$ with $v, w \in \{0,1\}^*$. A transition from state s to state s' with probability $\frac{3}{10}$, for example, can be written $0.0(1001)^\omega$, and can be encoded by the Markov chain below, where all transition probabilities are $\frac{1}{2}$:



We now introduce *simple Markov chains* (SMCs) and discuss the semantics of PCTL when applied to simple Markov chains.

▶ **Definition 4** (Simple Markov Chains). A Markov chain $\mathcal{M} = (S, \mathbf{P}, \iota, L)$ is called a *simple Markov chain* (SMC) if it satisfies the following.

- The state space of $\mathcal{M}$ is finite ($|S| \in \mathbb{N}$).
- The domain of the probability function $\mathbf{P}$ is $\{0, \frac{1}{2}, 1\}$.
- $\mathcal{M}$ contains an atomic proposition $p_\exists$ that cannot be used in specifications. A state s is called a *real* state if $p_\exists$ is contained in its label ($p_\exists \in L(s)$), and it is called a *hidden* state otherwise.
- The initial state ι is real, and, from each state, there is a path to a real state.

Intuitively, hidden states are the states used to simulate probabilities that are not $\frac{1}{2}$, and the last constraint guarantees that the probability measure of paths that eventually always stay in hidden states is 0. For the purposes of PCTL, the hidden states should not be counted towards the truth of a path formula. In particular, the next operator should refer to the next real state, and the bounded until operator with bound n should refer to an until that should be satisfied after n real states have been seen. The SMC semantics thus differs from the standard PCTL semantics in the definition of path formulas, which we now redefine.

▶ **Definition 5** (SMC Semantics). The semantics of state formulas is as in Definition 3, while the definition of path formulas changes as follows:

- $\mathcal{M}, \pi \models_{SMC} \bigcirc \phi$ iff $\exists i.\ s_i \models_{SMC} \phi$ and s_i is real and $\forall 0 < j < i.\ \ s_j$ is hidden,
- $\mathcal{M}, \pi \models_{SMC} \phi\, \mathcal{U}\, \phi'$ iff $\exists i.\ s_i \models_{SMC} \phi'$ and s_i is real and $\forall j < i.$ either $s_j \models_{SMC} \phi$ or s_j is hidden,
- $\mathcal{M}, \pi \models_{SMC} \phi\, \mathcal{U}^{\leq n}\, \phi'$ iff $\exists i.\ s_i \models_{SMC} \phi'$ and s_i is real and $\forall j < i.$ either $s_j \models_{SMC} \phi$ or s_j is hidden and $|\{j < i \mid s_j \text{ is real }\}| \leq n$.

Given a Markov chain with rational transition probabilities, we have already shown how an equivalent simple Markov chain can be constructed, by simulating the transition probabilities with hidden states. On the other hand, every simple Markov chain has an equivalent Markov chain, which can be obtained by computing the probability of moving from one real state to the next in the simple Markov chain. Therefore, we obviously have the following equivalence.

▶ **Proposition 6.** *For a PCTL formula ϕ, there is a finite Markov chain $\mathcal{M}$ with rational transition probabilities and $\mathcal{M} \models \phi$ if, and only if, there is a simple Markov chain $\mathcal{M}'$ that satisfies $\mathcal{M}' \models_{SMC} \phi$.*

Having only transition probabilities $\frac{1}{2}$ and 1 is very convenient in practice, because then all random choices can immediately be simulated by a device as simple as tossing a fair coin. The example formula ψ_0 and the argumentation above motivate the quest for models that are simple in two respects: they have a reasonable number of states, and all transition probabilities are equal to $\frac{1}{2}$ or 1. We can now formally state our bounded satisfiability problem:

▶ **Definition 7** (Bounded satisfiability problem). **Input:** A PCTL formula ϕ and a bound $b \in \mathbb{N}$.
Question: Does there exist a simple Markov chain $\mathcal{M}$ with at most b states, such that $\mathcal{M} \models_{SMC} \phi$?

For PCTL, the synthesis problem can be solved by a satisfiability algorithm. Suppose we want a probabilistic environment that gives us proposition p with probability 0.5, and not p with probability 0.5. Then, we can add the following requirement to our formula:

$$\mathbb{P}_{=1}\Box\big(\mathbb{P}_{=0.5}(\bigcirc p) \wedge \mathbb{P}_{=0.5}(\bigcirc \neg p)\big).$$

This can obviously be generalised for multiple input propositions, and non-uniform probability distributions.

To check whether there is a simple model $\mathcal{M}$ of ϕ with b states that satisfies a formula can clearly be done in non-deterministic polynomial time in the size of the related model checking problem (that is, in the joint size of the model and specification): it can simply guess the model and then check its correctness in polynomial time. It is also NP hard. Indeed, from a SAT instance f and a bound b, one can build a formula for which the smallest model is of size b if f is satisfiable and $b+1$ is f is not satisfiable. Intuitively, using bit strings to encore numbers between 1 and $b+1$, one can force the model to have b different states, and if f is unsatisfiable, require an extra state.

▶ **Proposition 8.** *For a given PCTL formula ϕ and a bound b, the problem of deciding whether there is a simple model $\mathcal{M}$ with $\mathcal{M} \models_{SMC} \phi$ that has b states is NP-complete in the joint size of ϕ and $\mathcal{M}$.*

Moreover, we can also show that approximating the size of a smallest model is NP-hard. The reason for this is that it is simple to find a PCTL formula for which the smallest model is of size $\approx 2^n$, where n is polynomial in the specification. Hence, we can construct a PCTL formula ϕ that either requires that a boolean formula ψ is true in the first state, or requires that we build an exponential model. Therefore, if ψ has a satisfying assignment, then ϕ has a model of size 1, otherwise the smallest model of ϕ has exponentially many sates.

▶ **Proposition 9.** *It is NP-hard to approximate the size of the smallest model of a PCTL formula ϕ within a factor that is polynomial in $|\phi|$.*

4 Reduction to an SMT problem

A satisfiability modulo theories (SMT) problem is the decision problem that consists in determining whether or not a logical formula expressed in boolean logic and using additional theories is satisfiable. Let ϕ be a PCTL formula over a set of atomic propositions AP. Suppose that we wish to solve the bounded satisfiability problem for ϕ with the bound b. In this section we will construct a system of SMT constraints that are satisfiable if, and only if, the formula ϕ has a model with b states. The size of the SMT formula will be linear in the number of sub-formulas in ϕ, and the SMT formula can be constructed in linear time (assuming that the bounds for bounded until formulas are given in unary). The theories we use in our SMT constraints are linear real arithmetic and uninterpreted function symbols. SMT for for this theory is NP-complete.

4.1 The model

We begin by introducing the functions and constraints that will define our model. We define the type $\mathsf{States} = \{1, 2, \dots, b\}$, with the intention that each integer in States will represent one state of the model. We define the following functions.

- We define the *left* successor function $\mathsf{left} : \mathsf{States} \to \mathsf{States}$ and the *right* successor function $\mathsf{right} : \mathsf{States} \to \mathsf{States}$. These functions give the two outgoing transitions from each of the states. A transition from a state s to a state s' with probability 1 can be simulated by setting $\mathsf{left}(s) = \mathsf{right}(s) = s'$.
- We define the *existence* function $\mathsf{exists} : \mathsf{States} \to \mathbb{B}$, where $\mathsf{exists}(s)$ is true if s is a real state, and false if s is a hidden state.
- For each atomic proposition $a \in AP$, we define a function $\mathsf{truth}_a : \mathsf{States} \to \mathbb{B}$, where $\mathsf{truth}_a(s)$ indicates that the atomic proposition a is true in state s.

Using the functions we have defined, it is possible to define a model that visits only a finite number of real states before getting stuck in hidden states. To avoid this, we introduce a function $\mathsf{dist}_\exists : \mathsf{States} \to [0, 1]$, and the following two constraints:

$$\forall s \cdot \mathsf{exists}(s) \leftrightarrow \mathsf{dist}_\exists(s) = 0,$$

$$\forall s \cdot \neg\mathsf{exists}(s) \to \Big(\mathsf{dist}_\exists(s) > \mathsf{dist}_\exists(\mathsf{left}(s))\Big) \vee \Big(\mathsf{dist}_\exists(s) > \mathsf{dist}_\exists(\mathsf{right}(s))\Big).$$

The first constraint states that $\mathsf{dist}_\exists(s)$ may only be 0 when s is real. The second constraint states that at each hidden state, the value of $\mathsf{dist}_\exists(s)$ must be strictly larger than either $\mathsf{dist}_\exists(\mathsf{left}(s))$ or $\mathsf{dist}_\exists(\mathsf{right}(s))$.

We argue that, if the model satisfies these two constraints, then it cannot get stuck in hidden states. A hidden state s can satisfy the second constraint if, and only if, there is a finite path from s to a real state. Hence, there is some probability p, with $p > 0$, to move from s to some real state. Since this holds for all hidden states, the probability of not eventually reaching a real state must be 0.

4.2 The formula

For each sub-formula ψ of ϕ, we associate a function $\mathsf{sat}_\psi : \mathsf{States} \to \mathbb{B}$, where $\mathsf{sat}_\psi(s)$ will be true if and only if state s satisfies ψ. In this section we will describe the constraints that are placed on sat_ψ.

Non-temporal operators

We begin by giving the constraints for sat_ψ for the case where ψ is a non-temporal operator. We define the following constraints:

- If ψ is an atomic proposition $a \in AP$, then we add the constraint:

$$\forall s \cdot \mathsf{sat}_\psi(s) \leftrightarrow \mathsf{truth}_a(s). \tag{1}$$

- If ψ is $\neg\psi'$, then we add the constraint:

$$\forall s \cdot \mathsf{sat}_\psi(s) \leftrightarrow \neg\mathsf{sat}_{\psi'}(s). \tag{2}$$

- If ψ is $\psi_1 \wedge \psi_2$, then we add the constraint:

$$\forall s \cdot \mathsf{sat}_\psi(s) \leftrightarrow \mathsf{sat}_{\psi_1}(s) \wedge \mathsf{sat}_{\psi_2}(s). \tag{3}$$

Next formulas

We now define the constraints on $\mathsf{sat}_\psi(s)$ in the case where ψ is $\mathbb{P}_{\bowtie\lambda}(\bigcirc\ \psi')$. It should be noted that we are only interested in whether ψ' holds in the next real state, and that we must account for the fact that several hidden states may be visited before we arrive at a real state. It is for this reason that we introduce the function $\mathsf{value}_\psi : \mathsf{States} \to [0,1]$. Our intention is that this function should give, for each hidden state, the probability that ψ' holds in the next real state. We define the following constraints on $\mathsf{value}_\psi(s)$.

$$\begin{aligned}
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi'}(s) \to \mathsf{value}_\psi(s) = 1,\\
&\forall s \cdot \mathsf{exists}(s) \wedge \neg\mathsf{sat}_{\psi'}(s) \to \mathsf{value}_\psi(s) = 0,\\
&\forall s \cdot \neg\mathsf{exists}(s) \to \mathsf{value}_\psi(s) = \frac{1}{2}\cdot\Big(\mathsf{value}_\psi(\mathsf{left}(s)) + \mathsf{value}_\psi(\mathsf{right}(s))\Big).
\end{aligned}$$

The first two constraints set the value of a real state s to be either 1 or 0 depending on whether ψ' holds at s. The final constraint sets the value of a hidden state to be the average of the values of its successors. Recall that we have already introduced constraints to ensure that the system cannot get stuck in hidden states. Therefore, these constraints are sufficient to force, for each hidden state s, the function $\mathsf{value}_\psi(s)$ to give the probability that ψ' holds at the next real state.

We can now introduce the constraint for $\mathsf{sat}_\psi(s)$. This constraint simply checks whether the probability of ψ' occurring in the next real state satisfies the bound $\bowtie \lambda$.

$$\forall s \cdot \mathsf{sat}_\psi(s) \leftrightarrow \frac{1}{2}\cdot\Big(\mathsf{value}_\psi(\mathsf{left}(s)) + \mathsf{value}_\psi(\mathsf{right}(s))\Big) \bowtie \lambda.$$

Until formulas

We now define the constraints on $\mathsf{sat}_\psi(s)$ in the case where ψ is $\mathbb{P}_{\bowtie\lambda}(\psi_1\ \mathcal{U}\ \psi_2)$. Following our approach for the next operator, we once again define a function $\mathsf{value}_\psi : \mathsf{States} \to [0,1]$. Our intention is that $\mathsf{value}_\psi(s)$ should give the probability that the until formula ψ is satisfied at the state s. We place the following constraints on the function value_ψ:

$$\begin{aligned}
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi_2}(s) \to \mathsf{value}_\psi(s) = 1,\\
&\forall s \cdot \mathsf{exists}(s) \wedge \neg\mathsf{sat}_{\psi_1}(s) \wedge \neg\mathsf{sat}_{\psi_2}(s) \to \mathsf{value}_\psi(s) = 0,\\
&\forall s \cdot \neg\mathsf{exists}(s) \vee \big(\mathsf{sat}_{\psi_1}(s) \wedge \neg\mathsf{sat}_{\psi_2}(s)\big) \to\\
&\qquad\qquad\qquad\qquad \mathsf{value}_\psi(s) = \frac{1}{2}\cdot(\mathsf{value}_\psi(\mathsf{left}(s)) + \mathsf{value}_\psi(\mathsf{right}(s))).
\end{aligned}$$

The first constraint sets the probability to 1 for real states that satisfy the right-hand side of the until, and the second constraint sets the probability to 0 for real states that satisfy neither the left-hand side nor the right-hand side of the until. The final constraint deals with real states that only satisfy the left-hand side of the until, and with hidden states. In both of these cases, we set the probability of the state to be the average of the probability of its successors.

In contrast to the next operator, the constraints that we have introduced are not sufficient to capture the probability of an until operator. To see the problem, consider a model in which ψ_1 is satisfied at all states, and ψ_2 is not satisfied at any state. Our constraints so far would allow $\mathsf{value}_\psi(s)$ to take any value in $[0,1]$ in such a model. To solve this, we introduce a distance function $\mathsf{dist}_\psi : \mathsf{States} \to [0,1]$, which ensures that ψ_2 can be reached with non-zero probability.

$$\begin{aligned}
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi_2}(s) \leftrightarrow \mathsf{dist}_\psi(s) = 0,\\
&\forall s \cdot \mathsf{value}_\psi(s) = 0 \leftrightarrow \mathsf{dist}_\psi(s) = 1,\\
&\forall s \cdot \mathsf{value}_\psi(s) \neq 0 \wedge \big(\neg\mathsf{exists}(s) \vee \neg\mathsf{sat}_{\psi_2}(s)\big) \rightarrow\\
&\qquad\qquad \Big(\mathsf{dist}_\psi(s) > \mathsf{dist}_\psi(\mathsf{left}(s))\Big) \vee \Big(\mathsf{dist}_\psi(s) > \mathsf{dist}_\psi(\mathsf{right}(s))\Big).
\end{aligned}$$

If a state satisfying ψ_2 can be reached with non-zero probability from a state s, then it is clear that we can set $\mathsf{dist}_\psi(s) < 1$. On the other hand, if no state satisfying ψ_2 can be reached from s, then the third constraint cannot be satisfied. Therefore, the second constraint must be used, which sets $\mathsf{dist}_\psi(s) = 1$, and then correctly sets $\mathsf{value}_\psi(s) = 0$.

Having specified the value function value_ψ, the constraint for the function sat_ψ simply compares the value to the bound given by λ:

$$\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_\psi(s) \leftrightarrow \mathsf{value}_\psi(s) \bowtie \lambda.$$

Bounded until formulas

We now give the constraints for sat_ψ for the case where ψ is $\mathbb{P}_{\bowtie\lambda}(\psi_1\, \mathcal{U}^{\leq n}\, \psi_2)$. The constraints that we introduce here can be seen as a generalisation of the constraints that are used for next formulas. For each i in the range $0 \leq i \leq n$, we introduce a function $\mathsf{value}_{\psi,i} : \mathsf{States} \rightarrow [0,1]$. The function $\mathsf{value}_{\psi,i}(s)$ is intended to give the probability that $\psi_1\, \mathcal{U}^{\leq i}\, \psi_2$ holds at the state s. We can start by giving a constraint for the function $\mathsf{value}_{\psi,0}$:

$$\begin{aligned}
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi_2}(s) \rightarrow \mathsf{value}_{\psi,0}(s) = 1,\\
&\forall s \cdot \mathsf{exists}(s) \wedge \neg\mathsf{sat}_{\psi_2}(s) \rightarrow \mathsf{value}_{\psi,0}(s) = 0,\\
&\forall s \cdot \neg\mathsf{exists}(s) \rightarrow \mathsf{value}_{\psi,0}(s) = \frac{1}{2} \cdot \Big(\mathsf{value}_{\psi,0}(\mathsf{left}(s)) + \mathsf{value}_{\psi,0}(\mathsf{right}(s))\Big).
\end{aligned}$$

Having defined $\mathsf{value}_{\psi,0}$, we can now define $\mathsf{value}_{\psi,i}$ inductively. For each i in the range $1 \leq i \leq n$, we add the constraints:

$$\begin{aligned}
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi_2}(s) \rightarrow \mathsf{value}_{\psi,i}(s) = 1,\\
&\forall s \cdot \mathsf{exists}(s) \wedge \neg\mathsf{sat}_{\psi_1}(s) \wedge \neg\mathsf{sat}_{\psi_2}(s) \rightarrow \mathsf{value}_{\psi,i}(s) = 0,\\
&\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_{\psi_1}(s) \wedge \neg\mathsf{sat}_{\psi_2}(s) \rightarrow\\
&\qquad\qquad \mathsf{value}_{\psi,i}(s) = \frac{1}{2}\Big(\mathsf{value}_{\psi,i-1}(\mathsf{left}(s)) + \mathsf{value}_{\psi,i-1}(\mathsf{right}(s))\Big),\\
&\forall s \cdot \neg\mathsf{exists}(s) \rightarrow \mathsf{value}_{\psi,i}(s) = \frac{1}{2} \cdot \Big(\mathsf{value}_{\psi,i}(\mathsf{left}(s)) + \mathsf{value}_{\psi,i}(\mathsf{right}(s))\Big).
\end{aligned}$$

The first two constraints deal with the case where ψ_2 is true at a real state, and the case where ψ_1 and ψ_2 are both false at a real state, respectively. The third constraint deals with real states at which ψ_1 is true, and ψ_2 is false. In this case, the probability that $\psi_1\, \mathcal{U}^{\leq i}\, \psi_2$ holds at s is the same as the probability that $\psi_1\, \mathcal{U}^{\leq i-1}\, \psi_2$ holds in the next real state. Hence, the third constraint takes the average of $\mathsf{value}_{\psi,i-1}$ over the successors of s. Finally, the fourth constraint deals with the hidden states. Since moving through a hidden state does not count towards the step bound of the until, we take an average of $\mathsf{value}_{\psi,i}$ over the successors for the hidden states.

Having defined the function $\mathsf{value}_{\psi,i}$ for all i in the range $0 \leq i \leq n$, we can now define the function sat_ψ by comparing the value given by $\mathsf{value}_{\psi,n}$ with the bound λ.

$$\forall s \cdot \mathsf{exists}(s) \wedge \mathsf{sat}_\psi(s) \leftrightarrow \mathsf{value}_{\psi,n}(s) \bowtie \lambda.$$

The formula ϕ

At this point, we have introduced constraints over sat_ψ for every sub-formula of the input formula ϕ. Our final task is to ensure that ϕ itself holds at some real state in the model. To do this, we arbitrarily pick State 1, and we require that State 1 is real, and that ϕ holds at State 1. Hence, to complete our reduction, we add the constraint:

$$\mathsf{exists}(1) \wedge \mathsf{sat}_\phi(1).$$

5 Implementation and results

5.1 Implementation

In this section we describe an implementation of the reduction given in Section 4. In fact, we implement a slightly simpler version of the reduction. In particular, it would obviously be inefficient to produce the function sat_ψ for the case where ψ is of the form $\neg\psi'$, $\psi_1 \wedge \psi_2$, or a for some atomic proposition $a \in AP$. Instead, we carry out the reduction as normal, and then we iteratively apply the identities given in (1), (2), and (3). For example, we replace all instances of $\mathsf{sat}_{\psi_1 \wedge \psi_2}$ with $\mathsf{sat}_{\psi_1}(s) \wedge \mathsf{sat}_{\psi_2}(s)$. We iterate this procedure until none of the three identities can be applied.

Our implementation consists of a parser that reads PCTL formulas, performs the reduction to SMT, and then outputs the system of SMT constraints. To solve the system of constraints, we experimented with several prominent SMT solvers, and we found that the Yices [17] solver was by far the fastest for our inputs. Hence, all the results described in this section were obtained using Yices-1.0.32 on a machine with a 2.66 GHz Core i7 processor and 4 GB of RAM. The implementation as well as the examples we report on are available for download at `http://www.csc.liv.ac.uk/~john/static/pctl-smt.tar.bz2`.

Our initial intention was to test our techniques against PCTL formulas from the literature. However, after attempting to find such formulas, we ran into a problem: the PCTL formulas that we found in the literature are not interesting from the perspective of satisfiability. Formulas that are given as examples in model checking papers are often very simple, because the authors are usually interested in the performance when measured in the size of the system. This meant that most of the formulas that we found had extremely simple satisfying models. For example, all of the formulas that appear in [14] have 1 state satisfying models, and as we shall see, these instances are not challenging for our implementation. The same problem occurs for all other examples that we found in the literature. Therefore, in the following subsections, we construct two scalable PCTL formulas that can be used to measure the performance of our implementation.

5.2 The lossy channel example

In this section, we test how well our implementation can construct systems. We define a formula $\mathsf{channel}_u$ that represents a lossy channel with u users. For each i in the range $1 \leq i \leq u$, there is an atomic proposition send_i, which indicates that user i wishes to send a message, and an atomic proposition $\mathsf{deliver}_i$, which indicates that the message belonging to

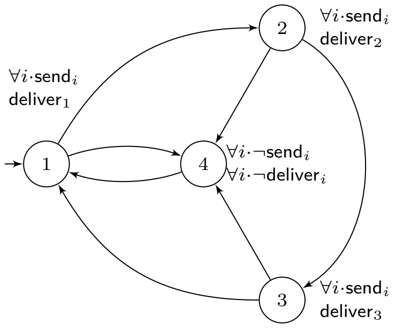


Figure 3 A model for $\mathsf{channel}_3$.

Users	Time (s)
2	0.063
3	0.765
4	12.879
5	18.172
6	8784.490

Figure 4 Experimental results for $\mathsf{channel}_u$.

user i has been delivered. The formula $\mathsf{channel}_u$ is defined to be:

$$\mathsf{channel}_{1,u} := \mathbb{P}_{\geq 0.1}(\bigcirc \bigwedge_{1\leq i\leq u} \neg\mathsf{deliver}_i)$$

$$\mathsf{channel}_{2,u} := \bigwedge_{1\leq i\leq u} \mathbb{P}_{=0.5}(\bigcirc\, \mathsf{send}_i)$$

$$\mathsf{channel}_{3,u} := \bigwedge_{1\leq i\leq u} \Big(\mathsf{send}_i \rightarrow \mathbb{P}_{=1}(\top\, \mathcal{U}\, \mathsf{deliver}_i)\Big)$$

$$\mathsf{channel}_{4,u} := \bigwedge_{1\leq i\leq u} (\mathsf{deliver}_i \rightarrow \bigwedge_{1\leq k\leq u, k\neq i} \neg\mathsf{deliver}_k)$$

$$\mathsf{channel}_u := \mathbb{P}_{=1}\Big(\Box\, (\mathsf{channel}_{1,u} \wedge \mathsf{channel}_{2,u} \wedge \mathsf{channel}_{3,u} \wedge \mathsf{channel}_{4,u})\Big).$$

The formula $\mathsf{channel}_{1,u}$ specifies that the channel is lossy: specifically that at least ten percent of the time the channel should not deliver a message at all. The formula $\mathsf{channel}_{2,u}$ specifies that the users should actually use the channel. It states that, in each step, there is a fifty percent chance that each user attempts to send a message. The formula $\mathsf{channel}_{3,u}$ states that, once a message has been sent, it should eventually be delivered. Finally, the formula $\mathsf{channel}_{4,u}$ states that the channel may only deliver one message in each step.

The formula $\mathsf{channel}_u$ has a model with $u + 1$ states. For example, in Figure 3, we show the structure of a model for $\mathsf{channel}_3$. The atomic propositions send_1, send_2, and send_3, are true in all states but state 4. We also have that $\mathsf{deliver}_1$ is true in state 1, that $\mathsf{deliver}_2$ is true in state 2, and that $\mathsf{deliver}_3$ is true in state 3. Other than these exceptions, the $\mathsf{deliver}_i$ atomic propositions are false at all other states.

It is also not difficult to see that the formula $\mathsf{channel}_u$ cannot have a model with fewer than $u + 1$ states. This is because, with probability 1, each of the send_i atomic propositions must become true. Once this has occurred, we require at least $u + 1$ distinct states: since two messages cannot be delivered at the same time, we require at least one state for each $\mathsf{deliver}_i$, and since the channel is lossy, we require at least one state in which no $\mathsf{deliver}_i$ is true.

In the table in Figure 4, we show experimental results for the formula $\mathsf{channel}_u$. In each case, we asked our implementation to find a model with $u+1$ states for the formula $\mathsf{channel}_u$. These results show that our implementation can construct systems with a small number of states very quickly. However, the exponential nature of the problem catches up to us quite quickly, and checking for the existence of a model with 7 states for the formula $\mathsf{channel}_6$ already takes more than two hours. The SMT solver was not able to solve the constraint system for $\mathsf{channel}_7$ within a reasonable amount of time. (We stopped the experiment after a day.)

5.3 A lossy channel with bugs

In this section we study the ability of our implementation to find bugs in specifications. We define an extension to our lossy channel that allows the system to go down, and we will require that, if the system does go down, then a number of recovery steps will be taken in order to restore service. Unfortunately, our formula will have a bug, and our goal is to find out how well our implementation can find this bug.

Our formula will be called $\mathsf{broken}_{u,r}$, which represents a lossy channel with u users, and which can recover from an error in r steps. In addition to the atomic propositions used by $\mathsf{channel}_u$, we add several new atomic propositions. The atomic proposition up indicates whether the network is up or down. For each j in the range $1 \leq j < r$, there is an atomic proposition $\mathsf{recover}_j$, that indicates that the network is in the jth step of the recovery procedure. We will require that each of the recovery propositions must be true before the system can come back up.

Our formula will reuse the formulas $\mathsf{channel}_{1,u}$ and $\mathsf{channel}_{4,u}$ from the previous section. However, we replace the other two formulas with the following:

$$\begin{aligned}
\mathsf{broken}_{2,u,r} &:= \bigwedge_{1\leq i\leq u} (\mathsf{up} \wedge \bigwedge_{1\leq j\leq r} \neg\mathsf{recover}_j) \rightarrow \mathbb{P}_{=0.5}(\bigcirc\, \mathsf{send}_i) \\
\mathsf{broken}_{3,u,r} &:= \bigwedge_{1\leq i\leq u} \Big(\mathsf{up} \wedge \mathsf{send}_i \rightarrow \mathbb{P}_{=1}(\top\, \mathcal{U}\, \mathsf{deliver}_i)\Big) \\
\mathsf{broken}_{5,u,r} &:= \bigwedge_{1\leq i\leq u} (\neg\mathsf{up} \rightarrow \bigwedge_{1\leq k\leq u, k\neq i} \neg\mathsf{deliver}_k)
\end{aligned}$$

The formula $\mathsf{broken}_{2,u,r}$ specifies that, if the channel is up, and not in a recovery state, then users should be able to send messages. The formula $\mathsf{broken}_{3,u,r}$ specifies that, if a user sends a message while the network is up, then that message should be delivered. Finally, the formula $\mathsf{broken}_{5,u,r}$ specifies that, if the network is not up, then no messages can be delivered.

In addition to these formulas, we also specify the recovery procedure. We define:

$$\mathsf{broken}^0_r := \begin{cases} \neg\mathsf{up} \rightarrow \mathbb{P}_{\geq 0.99}(\bigcirc\, \mathsf{up}) & \text{if } r = 1, \\ \neg\mathsf{up} \rightarrow \mathbb{P}_{\geq 0.99}(\bigcirc\, \mathsf{recover}_1) & \text{otherwise,} \end{cases}$$

and, for all j in the range $1 \leq j \leq r$, we define:

$$\mathsf{broken}^j_r := \begin{cases} \mathsf{recover}_j \rightarrow \mathbb{P}_{=1}(\bigcirc\, \mathsf{recover}_{j+1} \wedge \bigwedge_{1\leq k\leq j} \neg\mathsf{recover}_k) & \text{if } j < r, \\ \mathsf{recover}_j \rightarrow \mathbb{P}_{=1}(\bigcirc\, \mathsf{up} \wedge \bigwedge_{1\leq k\leq r} \neg\mathsf{recover}_k) & \text{if } j = r. \end{cases}$$

If $r = 1$, then these formulas specify that the system should recover in one step after it has gone down. For other values of r, these formulas specify that the system should pass through each of the recovery states before the channel comes back up. We can now specify the full formula:

$$\begin{aligned}
\mathsf{broken}_{u,r} := \mathbb{P}_{=1}\Big(\Box\, (&\mathsf{channel}_{1,u} \wedge \mathsf{broken}_{2,u,r} \\
&\wedge \mathsf{broken}_{3,u,r} \wedge \mathsf{channel}_{4,u} \wedge \mathsf{broken}_{5,u,r} \wedge \bigwedge_{1\leq j\leq r} \mathsf{broken}^j_r)\Big).
\end{aligned}$$

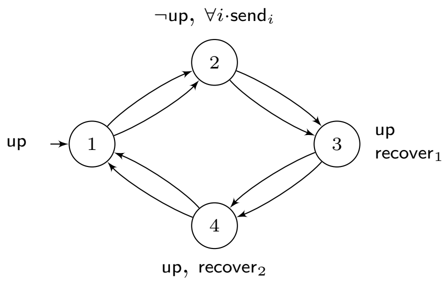


Figure 5 A model for broken$_{u,3}$.

Users	Time (s)			
	$r = 1$	$r = 2$	$r = 3$	$r = 4$
10	0.192	0.376	13.852	89.908
20	0.609	1.643	2.666	72.341
30	0.869	2.723	58.825	278.150
40	1.139	2.153	23.023	167.234
50	1.886	10.695	55.375	336.153
60	4.510	9.624	64.603	502.790
70	6.912	11.622	54.162	216.548
80	7.672	40.855	55.568	370.268
90	4.457	46.538	151.892	565.718
100	12.111	47.440	231.619	1927.258

Figure 6 Experimental results for broken$_{u,r}$.

Unfortunately, this formula contains a bug: if the system immediately goes down after being up, then no messages are ever delivered. Figure 5 shows the structure of a model that satisfies broken$_{u,3}$ for all u. We have that up is satisfied in all states except 2, that recover$_1$ is satisfied in state 3, and that recover$_2$ is satisfied in state 4. The propositions deliver$_i$ are not satisfied in any state, and the propositions send$_i$ are only satisfied in state 2.

We asked our implementation to solve broken$_{u,r}$ for varying values of u and r. The results are displayed in Figure 6. These results show a similar scaling with respect to r as was found for the previous example: as the size of the smallest model increases, our performance gets progressively worse, and we were unable to obtain results for $r = 5$ within a reasonable amount of time. However, these results show that our implementation scales well with respect to the number of users. This indicates that, while the running time of the procedure depends strongly on the size of the minimal model, there is a much smaller dependence on the size of the PCTL formula. Indeed, the formula broken$_{100,r}$ contains over 300 temporal operators. It is for this reason that we claim that our techniques are particularly suited for sanity checking, because this application requires us to construct a small model for a complex formula. These results show that our procedure can handle such situations.

6 Conclusion

In this paper, we have introduced the *bounded satisfiability* problem, which is a simplification of the satisfiability problem for PCTL, that restricts consideration to models that can be implemented. As was expected, bounded satisfiability is NP-complete in the minimal output. To offset this negative result, we provided a reduction from bounded satisfiability to an SMT problem, because in practice SMT solvers can often answer large queries.

Our experimental results allow for two interpretations. Our first set of experimental results shows that the difficulty of finding a solution to the system of SMT constraints depends strongly on the size of the minimal model. Hence, we consider it unlikely that these techniques will be able to construct the large systems that would be useful in practice. On the other hand, our second set of benchmarks showed that the running time of the SMT solver does not depend strongly on the size of the PCTL formula. Indeed, we were able to construct systems that satisfy formulas with hundreds of temporal operators. It would seem that, while our techniques are not able to construct models that are large enough be useful in practice, they are able to handle the large specifications that may appear in this setting.

This motivates the idea of sanity checking, where a system designer wishes to ensure that there are no errors in a specification that could lead to a small satisfying model. Our results indicate that our techniques are capable of providing a sanity checking procedure.

References

1 R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced vacuity detection in linear temporal logic. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, LNCS 2725, pages 368–380. Springer, 2003.

2 T. Ball and O. Kupferman. Vacuity in testing. In *Proceedings of the Second International Conference on Tests and Proofs (TAP 2008)*, LNCS 4966, pages 4–17. Springer, 2008.

3 I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–163, 2001.

4 T. Brázdil, V. Forejt, J. Kretínský, and A. Kucera. The satisfiability problem for probabilistic CTL. In *Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008)*, pages 391–402. IEEE Computer Society, 2008.

5 L. M. de Moura, B. Dutertre, and N. Shankar. A tutorial on satisfiability modulo theories. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV 2007)*, LNCS 4590, pages 20–36. Springer, 2007.

6 R. Ehlers. Symbolic bounded synthesis. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010)*, LNCS 6174, pages 365–379. Springer, 2010.

7 R. Ehlers. Unbeast: Symbolic bounded synthesis. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011)*, LNCS 6605, pages 272–275. 2011.

8 B. Finkbeiner and S. Schewe. SMT-based synthesis of distributed systems. In *Proceedings of the 2nd Workshop on Automated Formal Methods (AFM 2007)*, pages 69–76. 2007

9 H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

10 J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.

11 O. Kupferman, Y. Lustig, M. Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proceedings of the 28th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, LIPIcs 9, pages 615–626.

12 O. Kupferman and M. Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, 2003.

13 M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, LNCS 6806, pages 585–591. Springer, 2011.

14 M. Z. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. In *Proceedings of the Joint International Conferences on Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS/FTRTFT 2004)*, LNCS 3253, pages 293–308. Springer, 2004.

15 M. Purandare and F. Somenzi. Vacuum cleaning CTL formulae. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, LNCS 2404, pages 485–499. Springer, 2002.

16 S. Schewe and B. Finkbeiner. Bounded synthesis. In *Proceedings of the 5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*, LNCS 4762, pages 474–488. Springer, 2007.

17 Yices website: `http://yices.csl.sri.com/`.

A Concurrent Logical Relation

Lars Birkedal, Filip Sieczkowski, and Jacob Thamsborg

IT University of Copenhagen, Denmark
`{birkedal, fisi, thamsborg}@itu.dk`

Abstract

We present a logical relation for showing the correctness of program transformations based on a new type-and-effect system for a concurrent extension of an ML-like language with higher-order functions, higher-order store and dynamic memory allocation.

We show how to use our model to verify a number of interesting program transformations that rely on effect annotations. In particular, we prove a Parallelization Theorem, which expresses when it is sound to run two expressions in parallel instead of sequentially. The conditions are expressed solely in terms of the types and effects of the expressions. To the best of our knowledge, this is the first such result for a concurrent higher-order language with higher-order store and dynamic memory allocation.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases verification, logical relation, concurrency, type and effect system

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.107

1 Introduction

Relational reasoning about program equivalence is useful for reasoning about the correctness of program transformations, data abstraction (representation independence), compiler correctness, etc. The standard notion of program equivalence is contextual equivalence and in recent years, there have been many improvements in reasoning methods for higher-order ML-like languages with general references, based on bisimulations, e.g., [17, 23, 25], traces [18], game semantics [21], and Kripke logical relations, e.g., [1, 2, 8, 12].

In this paper we present the first Kripke logical relation for reasoning about equivalence of a *concurrent* higher-order ML-like language with higher-order store and dynamic memory allocation.

To state and prove useful equivalences about concurrent programs, it is necessary to have some way of restricting the contexts under which one proves equivalences. This point was made convincingly in the recent paper by Liang et. al. [19], who presented a rely-guarantee-based simulation for verifying concurrent program transformations for a first-order imperative language (with first-order store). Here is a very simple example illustrating the point. Consider two expressions

$$e_1 \equiv x := 1; y := 1 \quad \text{and} \quad e_2 \equiv y := 1; x := 1.$$

Here x and y are variables of type `refint`. The expressions e_1 and e_2 are not contextually equivalent. (To see why, consider expression $e_3 \equiv x := 0;\ y := 0$, and note that running e_1 in parallel with e_3 may result in a state with $!x = 0$ and $!y = 1$, but that cannot be the case when we run e_2 in parallel with e_3.) The issue is, of course, that the context may also modify the references x and y. On the other hand, if we know that no other threads have access to x or y, then it should be the case that e_1 and e_2 are equivalent. We can express this restriction on the contexts using a refined region-based type-and-effect system.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 107–121
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We first recall that a type-and-effect system is a type system that classifies programs according to which side effects the programs may have. A variety of effect systems have been proposed for higher-order programming languages, e.g., [15, 20, 27], see [16] for a recent overview. Effect systems can often be understood as specifying the results of a static analysis, in the sense that it is possible to automatically infer types and effects. Effect systems can be used for different purposes: they were originally proposed by Lucassen and Gifford [20] for parallelization purposes but they have also, e.g., been used as the basis for implementing ML using a stack of regions for memory management [27, 9]. In a recent series of papers, Benton et. al. have argued that another important point of effect systems is that they can be used as the basis for effect-based program transformations, e.g., compiler optimizations, [6, 5, 3, 4], see also [26]. The idea is that certain program transformations are only sound under additional assumptions about which effects program phrases may, or rather may not, have.

Now, returning to our example, we refine the types of x and y to be $\mathsf{ref}_\rho\mathsf{int}$ and $\mathsf{ref}_\sigma\mathsf{int}$, respectively. Intuitively, this expresses that x and y are references in different regions, but it does not put any restrictions on whether other threads may access x or y. Thus, when we type e_1 and e_2 we will use two contexts of region variables, one for public regions that can be used by other concurrently running threads, and one for private regions that are under the control of the present thread. This idea is inspired by recent work on concurrent separation logic, e.g., [22, 11, 29, 13]. We use a vertical bar to separate public and private regions: the typing context

$$\rho, \sigma \mid \emptyset \mid x : \mathsf{ref}_\rho\mathsf{int}, y : \mathsf{ref}_\sigma\mathsf{int}$$

expresses that ρ and σ are public regions, whereas the typing context

$$\emptyset \mid \rho, \sigma \mid x : \mathsf{ref}_\rho\mathsf{int}, y : \mathsf{ref}_\sigma\mathsf{int}$$

expresses that ρ and σ are private regions. The expressions e_1 and e_2 are well-typed in the latter context and, with this refined typing, they are indeed contextually equivalent, because our type-and-effect system guarantees that no well-typed context can access regions ρ or σ. (The expressions are also well-typed in the former context, but not contextually equivalent with that refined typing.)

In this paper we present a step-indexed Kripke logical relations model of a type-and-effect system with public and private regions for a concurrent higher-order language with general references. Our model is constructed over the operational semantics of the programming language, and builds on recent work by Thamsborg and Birkedal on logical relations for the sequential sub-language [26]. Note that the type-and-effect annotations are just annotations; the operational semantics of the language is standard and regions only exist in our semantic model, not in the operational semantics.

As an important application of our model we prove a Parallelization Theorem, which expresses when it is sound to run two expressions in parallel instead of sequentially. To the best of our knowledge, this is the first such result for a higher-order language with higher-order store and dynamic memory allocation. Here is a very simple instance of the theorem. Consider two expressions

$$e_1 \equiv y := !x + !y \quad \text{and} \quad e_2 \equiv z := !x + !z,$$

each well-typed in a context

$$\emptyset \mid \rho_x, \rho_y, \rho_z \mid x : \mathsf{ref}_{\rho_x}\mathsf{int}, y : \mathsf{ref}_{\rho_y}\mathsf{int}, z : \mathsf{ref}_{\rho_z}\mathsf{int},$$

i.e., where x, y, and z are references in distinct private regions. In this context, running e_1 and e_2 sequentially is contextually equivalent to running e_1 and e_2 in parallel. Intuitively, this also makes sense: e_1 and e_2 update references in distinct regions, and it is unproblematic that they both read (but not write) from the same region.

As mentioned, this was a simple instance of the Parallelization Theorem. We stress that the theorem is expressed solely in terms of the type and efffects of the expressions e_1 and e_2, so a compiler may automatically infer that it is safe to parallelize two expressions by looking at the inferred effect types, and without reasoning about all interleavings. Moreover, the theorem applies to contexts and expressions with general higher types (not just with references to integers and unit types). Note that the distinction between private and public regions is also crucial here (parallelization would not be sound if the effects of the expressions were on public regions).

Our type-and-effect system crucially also includes a region-masking rule. Traditionally, this rule has been used to hide local effects on regions, which makes it possible to view a computation as pure even if it uses effects locally and makes the effect system stronger, in the sense that it can justify more program transformations. Here we also observe that the masking rule can be used for introducing private regions, since the masking rule intuitively guarantees that effects on a region are not leaked to the context. It is well-known that region-masking makes the model construction for a sequential language technically challenging, see the extensive discussion in [26]. Here it is yet more challenging because of concurrency; we explain how our model ensures soundness of the masking rule in Section 3.

The extension with concurrency also means that when we define the logical relation for contextual approximation and relate two computations e_1 and e_2, we cannot simply require relatedness after e_1 has completed evaluation (as in the sequential case), since other threads should be allowed to execute as well. We explain our approach to relating concurrent computations in Section 3; it is informed by recent soundness proofs of unary models of concurrent separation logic [30, 10].

Another challenge arises from the fact that since our language includes dynamically allocated general references, the existence of the logical relation is non-trivial; in particular, the set of Kripke worlds must be recursively defined. Here we build on our earlier work [7] and define the worlds as a solution to a recursive metric-space equation. Indeed, to focus on the essential new aspects due to the extension with concurrency, we deliberately choose to use the exact same notion of worlds as we used for the sequential sub-language in [26]. In the same vein, we here consider a monomorphically typed higher-order programming language with general references, but leave out universal and existential types as well as recursive types. However, we want to stress that since our semantic techniques (step-indexed Kripke logical relations over recursively defined worlds) do indeed scale well to universal, existential, and recursive types, e.g. [7, 12], it is possible to extend our model to a language with such types. We conjecture that it is also possible to extend our model to richer effect systems involving region and effect polymorphism, but we have not done so yet.

All proofs are deferred to the long version of the paper; it can be found online at the following address: `www.itu.dk/people/birkedal/papers/longsamba.pdf`.

2 Language and Typing

We consider a standard call-by-value lambda calculus with general references, and extended with parallel composition and an atomic construct. We assume countably infinite, pairwise disjoint sets of *region variables* $\mathcal{RV}$ (ranged over by ρ), *locations* $\mathcal{L}$ (ranged over by l) and

$$
\begin{array}{rcl}
\pi & ::= & rd_\rho \mid wr_\rho \mid al_\rho \\
\varepsilon & ::= & \pi_1, \ldots, \pi_n \\
\tau & ::= & \mathsf{1} \mid \mathsf{int} \mid \tau_1 \times \tau_2 \mid \mathsf{ref}_\rho\, \tau \\
& & \mid \tau_1 \to_{\varepsilon}^{\Pi,\Lambda} \tau_2 \\
v & ::= & x \mid \langle\rangle \mid \langle v_1, v_2 \rangle \\
& & \mid \mathsf{fun}\, f(x).e \mid l \\
e & ::= & v \mid \mathsf{proj}_i\, v \mid v\, e \mid \mathsf{ref}\, v \mid !\, v \\
& & \mid v_1 := v_2 \mid \mathsf{par}\ e_1\ \mathsf{and}\ e_2 \\
& & \mid \mathsf{cas}\ (v_1, v_2, v_3) \mid \mathsf{atomic}\ e \\
E & ::= & [] \mid v\, E \mid \mathsf{par}\ E\ \mathsf{and}\ e_2 \\
& & \mid \mathsf{par}\ e_1\ \mathsf{and}\ E
\end{array}
$$

Figure 1 Syntax.

$$
\begin{array}{rcl}
(E[\mathsf{proj}_i\, \langle v_1, v_2 \rangle] \mid h) & \longmapsto & (E[v_i] \mid h) \\
(E[(\mathsf{fun}\, f(x).e)\, v] \mid h) & \longmapsto & (E[e[\mathsf{fun}\, f(x).e/f, v/x]] \mid h) \\
(E[\mathsf{ref}\, v] \mid h) & \longmapsto & (E[l] \mid h[l \mapsto v]) \text{ if } l \notin dom(h) \\
(E[l := v] \mid h) & \longmapsto & (E[\langle\rangle] \mid h[l{:=}v]) \text{ if } l \in dom(h) \\
(E[!\, l] \mid h) & \longmapsto & (E[h(l)] \mid h) \quad \text{if } l \in dom(h) \\
(E[\mathsf{par}\ v_1\ \mathsf{and}\ v_2] \mid h) & \longmapsto & (E[\langle v_1, v_2 \rangle] \mid h) \\
(E[\mathsf{cas}\ (l, n_1, n_2)] \mid h) & \longmapsto & (E[1] \mid h[l{:=}n_2]) \\
& & \text{if } l \in dom(h) \text{ and } h(l) = n_1 \\
(E[\mathsf{cas}\ (l, n_1, n_2)] \mid h) & \longmapsto & (E[0] \mid h) \\
& & \text{if } l \in dom(h) \text{ and } h(l) \neq n_1 \\
(E[\mathsf{atomic}\ e] \mid h) & \longmapsto & (E[v] \mid h') \\
& & \text{if } (e \mid h) \longmapsto^{*} (v \mid h') \\
(E[\mathsf{atomic}\ e] \mid h) & \longmapsto & (E[\mathsf{atomic}\ e] \mid h)
\end{array}
$$

Figure 2 Operational semantics.

program variables (ranged over by x, y, f). As usual, the reduction relation is between configurations, $(e \mid h) \longmapsto (e' \mid h')$ where heaps $h, h' \in \mathcal{H}$ are finite maps from locations to values. Figures 1 and 2 give the syntax and operational semantics; we denote the set of expressions $\mathcal{E}$ and the set of values $\mathcal{V}$. The evaluation contexts allow parallel evaluation inside par expressions, and there is a new primitive reduction covering the case when the two subcomputations have terminated. For technical simplicity, we allow an $\mathsf{atomic}\ e$ expression to reduce to itself, possibly introducing more divergence than the diverging behaviours of e. The syntax is kept minimal; in examples we may use additional syntactic sugar, e.g., writing $\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$ for $(\mathsf{fun}\, f(x).e_2)\, e_1$ for some fresh f. For $e \in \mathcal{E}$, we write $\mathrm{FV}(e)$ and $\mathrm{FRV}(e)$ for the sets of free program variables and region variables, respectively; also we define $\mathrm{rds}\, \varepsilon = \{\rho \in \mathcal{RV} \mid rd_\rho \in \varepsilon\}$ and similarly for writes and allocation.

The form of the judgments of our type-and-effect system is standard with one important refinement: regions are partitioned into *public* and *private* regions, with the purpose of restricting interference from the environment. In greater detail, a typing judgement looks like this:

$$\Pi \mid \Lambda \mid \Gamma \vdash e : \tau,\ \varepsilon.$$

The Γ, e and τ are the usual: the *variable context* Γ assigns types to program variables in the expression e, with the resulting type of τ. To get an idea of — or rather an upper bound of — the side-effects of e, we split the heap into *regions*; these are listed in Π and Λ. We track memory accesses by adding a set ε of *effects* of the form rd_ρ, wr_ρ and al_ρ, where ρ is a region. Roughly, a computation with effect rd_ρ may read one or more locations in region ρ, and similarly for writes and allocation. This setup goes back to Lucassen and Gifford [20].

The novelty, as mentioned in the Introduction, is our partition of regions into the *public* ones Π and the *private* ones Λ. As opposed to the rest of the judgment, this public-private division does not make promises about the behavior of e. Instead, it states the expectations that e has of the environment: threads running in parallel with e may — in a well-typed manner — read, write and allocate in the public regions but must leave the private regions

$$\frac{}{\Pi \mid \Lambda \mid \Gamma, x : \tau \vdash x : \tau,\ \emptyset} \qquad \frac{}{\Pi \mid \Lambda \mid \Gamma \vdash \langle\rangle : \mathsf{1},\ \emptyset} \qquad \frac{\Pi \mid \Lambda \mid \Gamma \vdash v : \tau_1 \times \tau_2,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{proj}_i\, v : \tau_i,\ \varepsilon}$$

$$\frac{\Pi \mid \Lambda \mid \Gamma \vdash v_1 : \tau_1,\ \varepsilon_1 \qquad \Pi \mid \Lambda \mid \Gamma \vdash v_2 : \tau_2,\ \varepsilon_2}{\Pi \mid \Lambda \mid \Gamma \vdash \langle v_1, v_2\rangle : \tau_1 \times \tau_2,\ \varepsilon_1 \cup \varepsilon_2} \qquad \frac{\Pi \mid \Lambda \mid \Gamma, f : \tau_1 \to_{\varepsilon}^{\Pi,\Lambda} \tau_2, x : \tau_1 \vdash e : \tau_2,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{fun}\, f(x).e : \tau_1 \to_{\varepsilon}^{\Pi,\Lambda} \tau_2,\ \emptyset}$$

$$\frac{\Pi \mid \Lambda \mid \Gamma \vdash v : \tau_1 \to_{\varepsilon}^{\Pi,\Lambda} \tau_2,\ \varepsilon_1 \qquad \Pi \mid \Lambda \mid \Gamma \vdash e : \tau_1,\ \varepsilon_2}{\Pi \mid \Lambda \mid \Gamma \vdash v\, e : \tau_2,\ \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon} \qquad \frac{\Pi \mid \Lambda \mid \Gamma \vdash v : \tau,\ \varepsilon \qquad \rho \in \Pi, \Lambda}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{ref}\, v : \mathsf{ref}_\rho \tau,\ \varepsilon \cup \{al_\rho\}}$$

$$\frac{\Pi \mid \Lambda \mid \Gamma \vdash v_1 : \mathsf{ref}_\rho \tau,\ \varepsilon_1 \qquad \Pi \mid \Lambda \mid \Gamma \vdash v_2 : \tau,\ \varepsilon_2}{\Pi \mid \Lambda \mid \Gamma \vdash v_1 := v_2 : \mathsf{1},\ \varepsilon_1 \cup \varepsilon_2 \cup \{wr_\rho\}} \qquad \frac{\Pi \mid \Lambda \mid \Gamma \vdash v : \mathsf{ref}_\rho \tau,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash {!}v : \tau,\ \varepsilon \cup \{rd_\rho\}}$$

$$\frac{\Pi \mid \Lambda, \rho \mid \Gamma \vdash e : \tau,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash e : \tau,\ \varepsilon - \rho}\ (\rho \notin \mathrm{FRV}(\Gamma, \tau)) \qquad \frac{\cdot \mid \Pi, \Lambda \mid \Gamma \vdash e : \tau,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{atomic}\ e : \tau,\ \varepsilon}\ (\mathrm{als}\,\varepsilon \subseteq \mathrm{rds}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon)$$

$$\frac{\Pi, \Lambda \mid \cdot \mid \Gamma \vdash e_1 : \tau_1,\ \varepsilon_1 \qquad \Pi, \Lambda \mid \cdot \mid \Gamma \vdash e_2 : \tau_2,\ \varepsilon_2}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{par}\ e_1\ \mathsf{and}\ e_2 : \tau_1 \times \tau_2,\ \varepsilon_1 \cup \varepsilon_2}$$

$$\frac{\Pi \mid \Lambda \mid \Gamma \vdash v_1 : \mathsf{ref}_\rho \mathsf{int},\ \varepsilon_1 \qquad \Pi \mid \Lambda \mid \Gamma \vdash v_2 : \mathsf{int},\ \varepsilon_2 \qquad \Pi \mid \Lambda \mid \Gamma \vdash v_3 : \mathsf{int},\ \varepsilon_3}{\Pi \mid \Lambda \mid \Gamma \vdash \mathsf{cas}\ (v_1, v_2, v_3) : \mathsf{int},\ \{wr_\rho, rd_\rho\} \cup \varepsilon_1 \cup \varepsilon_2 \cup \varepsilon_3}$$

$$\frac{\Pi \mid \Lambda \mid \Gamma \vdash e : \tau_1,\ \varepsilon_1 \qquad \Pi, \Lambda \vdash \tau_1 \leq \tau_2 \qquad \varepsilon_1 \subseteq \varepsilon_2}{\Pi \mid \Lambda \mid \Gamma \vdash e : \tau_2,\ \varepsilon_2}\ (\mathrm{FRV}(\varepsilon_2) \subseteq \Pi, \Lambda)$$

$$\frac{}{\Theta \vdash \tau \leq \tau}\ (\mathrm{FRV}(\tau) \subseteq \Theta) \qquad \frac{\Theta \vdash \tau_1 \leq \tau_1' \qquad \Theta \vdash \tau_2 \leq \tau_2'}{\Theta \vdash \tau_1 \times \tau_2 \leq \tau_1' \times \tau_2'}$$

$$\frac{\Theta \vdash \tau_1' \leq \tau_1 \qquad \Theta \vdash \tau_2 \leq \tau_2' \qquad \varepsilon_1 \subseteq \varepsilon_2 \qquad \Pi_1 \subseteq \Pi_2 \qquad \Lambda_1 \subseteq \Lambda_2}{\Theta \vdash \tau_1 \to_{\varepsilon_1}^{\Pi_1,\Lambda_1} \tau_2 \leq \tau_1' \to_{\varepsilon_2}^{\Pi_2,\Lambda_2} \tau_2'}\ (\mathrm{FRV}(\varepsilon_2), \Pi_2, \Lambda_2 \subseteq \Theta)$$

Figure 3 Typing and subtyping relations. Notice that for a typing judgement $\Pi \mid \Lambda \mid \Gamma \vdash e : \tau,\ \varepsilon$ we always have $\mathrm{FRV}(\Gamma, \tau, \varepsilon) \subseteq \Pi \cup \Gamma$.

untouched.

When running parallel threads, the private regions of the parent are shared between the children, and so are public from their point of view; this is reflected in the typing rule for parallel composition, c.f. Figure 3. Note that the parent thread only continues once both children have terminated; as a consequence, the parent regains ownership of its private regions before it goes on. Running an expression atomically temporarily makes all regions private. The side condition is a technical necessity. Finally, new, private regions are introduced by the so-called masking rule:

$$\frac{\Pi \mid \Lambda, \rho \mid \Gamma \vdash e : \tau,\ \varepsilon}{\Pi \mid \Lambda \mid \Gamma \vdash e : \tau,\ \varepsilon - \rho}\ (\rho \notin \mathrm{FRV}(\Gamma, \tau))$$

The subtraction of ρ in the conclusion removes any read, write or allocation effects tagged with ρ. The reading of the masking rule is that we make a brand new, empty region ρ for e to use, but once e has terminated we forget about ρ again; this works out since the side condition prevents e from leaking locations from ρ. Traditionally, the masking rule has been used to do memory-management [27] as well as a means of hiding local effects to facilitate effect-based program transformations [5, 26]. Here we make another use of the rule: we observe that, moreover, e cannot leak locations from ρ while running and so ρ is a *private* region for the duration of e. After all, the only means of inter-thread communication is shared memory. Note that from the perspective of the context, this rule allows to *remove* a private region, and prepare a setup for application of the parallel composition.

All the typing rules are in Figure 3. We just remark here, that reference types are tagged with the region where the location resides and that function arrows are tagged with the latent effects as well as with the public and private regions that the function expects; the latter is natural once we remember that a function is basically just a suspended, well-typed expression.

Because of the nondeterminism arising from par and shared references, the definition of contextual equivalence could take into account both may- and must-convergence. In this paper we only consider may-equivalence and formally we define (may-) contextual approximation by:

▶ **Definition 1.** $\Pi \,|\, \Lambda \,|\, \Gamma \vdash e \lesssim_{\downarrow} e' : \tau, \varepsilon$ if and only if for all h and C typed such that $\cdot \,|\, \cdot \,|\, \cdot \vdash C[e], C[e'] : \mathsf{int}, \emptyset$, whenever $(C[e] \,|\, h) \downarrow$ then $(C[e'] \,|\, h) \downarrow$.

Here, as usual, $(e \,|\, h) \downarrow$ means that $(e \,|\, h) \longmapsto {}^{*}(v \,|\, h')$ for some value v and some h'.

Contextual equivalence, $\Pi \,|\, \Lambda \,|\, \Gamma \vdash e \approx e' : \tau, \varepsilon$, is then defined as $\Pi \,|\, \Lambda \,|\, \Gamma \vdash e \lesssim_{\downarrow} e' : \tau, \varepsilon$ and $\Pi \,|\, \Lambda \,|\, \Gamma \vdash e' \lesssim_{\downarrow} e : \tau, \varepsilon$. Note that the diverging behaviours introduced by our operational semantics of atomic e do not influence may-contextual equivalence.

3 Definition of the logical relation

Semantic Types and Worlds We give a Kripke or world-indexed logical relation. This is a fairly standard approach to modeling dynamic allocation; in combination with higher-order store, however, it comes with a fairly standard problem: the *type-world circularity*. Roughly, semantic types are indexed over worlds and worlds contain semantic types, so both need to be defined before the other. A specific instance of this circularity was solved recently by Thamsborg and Birkedal [26] based on metric-space theory developed by Birkedal et. al. [7]; we re-use that solution here. Semantic types (and worlds) are constructed as a fixed-point of a endo-functor on a certain category of metric-spaces. We do not care about that, though; we just give the result of the construction. In addition, we largely ignore the fact that we actually deal in metric spaces and not just plain sets; the little metric machinery we need is deferred to the appendix of the long version of the paper.

There is a set $\mathbf{T}$ of *semantic types* and a set $\mathbf{W}$ of *worlds*; types are world-indexed relations on values and worlds describe the regions and type-layouts of heaps, roughly speaking. Take a type $\mu \in \mathbf{T}$ and apply it to a world $w \in \mathbf{W}$ and you get an indexed relation on values, i.e., $\mu(w) \subseteq \mathbb{N} \times \mathcal{V} \times \mathcal{V}$. These relations are downwards closed in the first coordinate; we read $(k, v_1, v_2) \in \mu(w)$ as saying that v_1 and v_2 are related at type μ up to approximation k assuming world w.

We assume a countably infinite set of region names $\mathcal{RN}$; a world $w \in \mathbf{W}$ contains finitely many such $|w| \subseteq_{\mathit{fin}} \mathcal{RN}$. Some of these $\mathrm{dom}(w) \subseteq |w|$ are *live* and the rest are *dead*. To each live region $r \in \mathrm{dom}(w)$ we associate a finite partial bijection $w(r)$ on locations decorated with types, i.e., $w(r) \subseteq_{\mathit{fin}} \mathcal{L} \times \mathcal{L} \times \widehat{\mathbf{T}}$ such that for $(l_1, l_2, \mu), (m_1, m_2, \nu) \in w(r)$ we have that both $l_1 = m_1$ and $l_2 = m_2$ imply $l_1 = m_1$, $l_2 = m_2$ and $\mu = \nu$. We write $\mathrm{dom}_1(w(r))$ for the set of left hand side locations in the bijection and $\mathrm{dom}_2(w(r))$ for the right hand side ones; different regions must have disjoint left and right hand side locations. For convenience, we set $\mathrm{dom}_1^{\mathrm{A}}(w) = \bigcup_{r \in A \cap \mathrm{dom}(w)} \mathrm{dom}_1(w(r))$ whenever $A \subseteq |w|$, and we write $\mathrm{dom}_1(w)$ for $\mathrm{dom}_1^{|\mathrm{w}|}(w)$, i.e., the set of all left hand side locations. Similarly for the right hand side.

Worlds evolve and types adapt. Triples of two locations and a type can be added to a live region, as long as different regions remain disjoint. Orthogonal to this, one can add a fresh, i.e., neither live nor dead, region name with an associated empty partial bijection. And one can kill any live region, rendering it dead and losing the associated the partial bijection in the process. The reflexive, transitive closure of all three combined is a preorder $\sqsubseteq$ on worlds; it is a crucial property of types that they respect this, i.e., that $w \sqsubseteq w' \implies \mu(w) \subseteq \mu(w')$ for any two $w, w' \in \mathbf{W}$ and any $\mu \in \mathbf{T}$. This is *type monotonicity* and it prevents values from fleeing types over time.

Finally, to tie the knot, there is an isomorphism $\iota : \widehat{\mathbf{T}} \to \mathbf{T}$ from the odd types stored in worlds to proper types. Whenever a type is extracted from a world it needs to be coerced by this isomorphism before it can be applied to some world.

The Logical Relation and Interpretation of Types Often, a logical relation goes like this: two computations are related if they (from related heaps) reduce to related values (and heaps); this is the *extensional* view: we do not care about the intermediate states. As we consider concurrency, however, a computation can be interrupted and so we need to start caring. In our setup, public regions are accessible from the environment. To address this, we assume that before each reduction step, the public regions hold related values; in return, we promise related values after the step. In other words, the *granularity of extensionality* is just one step for the public regions. For private regions, however, there is no interference and the granularity is an entire computation as usual. This is the fundamental idea; it is how we propose to stay extensional in the face of concurrency.

Without further ado, let us look into the cornerstone of our model: the safety relation defined in Figure 6; auxiliary relations are defined in Figure 8. What does it mean to have

$$(k, h_1^\circ, h_2^\circ, e_1, e_2, h_1, h_2) \in \mathbf{safe}_{\tau,\varepsilon}^{\Pi,\Lambda,A,R}\, w^\circ, w?$$

Overall, it says that after environment interference, we can match the behavior of e_1, i.e., termination or any one-step reduction, by zero or more steps of e_2; match in the sense of (re-)establishing certain relations, including safety itself. Safety is a *local* property of a pair of computations, this is crucial: it has no knowledge of computations running concurrently and h_1 and h_2 are the *local heaps*, i.e., the parts of the global heaps that the e_1 respectively e_2 control exclusively. The computations consider $R(\Pi)$ to be their public, $R(\Lambda)$ to be their private and A to be their *anonymous* regions. The latter intuitively are private regions that have been masked out: they exist only for the duration of these computations, but we have to track them to deny the environment access; this is another difficulty imposed by concurrency. Safety is indexed by a world w as well; note that worlds are *global* things: all concurrent threads share one world, i.e., they agree about the division of the heap into regions and the types associated to locations. Finally k is intuitively the number of steps we are safe for, h_1° and h_2° are the (private parts of) the initial local heaps, τ is the expected return type, ε the effects and w° the initial world.

We unroll the definition in writing. The first pair of big square brackets — the *prerequisites* — translates to 'the environment interferes'. This yields a new world w' subject to the constraints of the environment transition relation: no public, private or anonymous regions are killed, and the latter two see no allocation either. The actual contents of the public regions are unknown, but we are free to assume that they hold related values of the proper type, at least where we have read effects; this is the *public heaps* g_1 and g_2 in the precondition relation. In addition we have *frames* f_1 and f_2 that cover the remainder of the world and a triple-split relation that ensures coherence between the domains of corresponding parts of the world and the heaps, see Figures 4 and 8.

The left hand side is irreducible in the *termination branch* and takes one step in the *progress branch*. In either case, we must match this in zero or more steps on the right hand

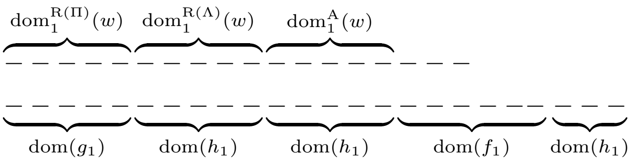

$\mathrm{dom}_1^{\mathrm{R}(\Pi)}(w)$
$\mathrm{dom}_1^{\mathrm{R}(\Lambda)}(w)$
$\mathrm{dom}_1^{\mathrm{A}}(w)$
$\mathrm{dom}(g_1)$
$\mathrm{dom}(h_1)$
$\mathrm{dom}(h_1)$
$\mathrm{dom}(f_1)$
$\mathrm{dom}(h_1)$

Figure 4 The left hand side of the triple-split relation. The top dashed line is $\mathrm{dom}_1(w)$, the bottom dashed line $\mathrm{dom}(g_1 \cdot h_1 \cdot f_1)$. The local heap h_1 has a *private* part matching the private regions, an *anonymous* part matching the anonymous regions and an *off-world* part outside the domain of the world. The frame f_1 must cover regions that are neither public, private nor anonymous.

side, not touching the frame; this means finding a future world w'' and relating a number of things. The choice of future world is restricted by the self transition relation: we must not kill private or public regions, but we can allocate in them, and regions that we know nothing about must be left untouched; this is our promise to the environment. In the termination branch, we are furthermore required to kill off all anonymous regions as the computation is done; any new regions added in the progress branch go to the set of anonymous regions. In both branches, the changes made to the public heap must be well-typed and permitted by the effects and, if we are done, we check the changes made to (the private part of) the local heaps as well; the fact that the public heaps are compared across a single stage and the (private parts of) the local heaps are compared across the entire computations is the crux of the idea of having different granularities of extensionality.

In addition to performing actual allocation, we have the possibility of moving existing locations from, say, the off-world part of the local heap into the public heap or the private part of the local heap; this is a subtle point that permits the actual allocation of new locations and the corresponding extension of the world to be temporarily out of sync.

We have glossed over one aspect of safety: the right hand side takes steps in the ordinary operational semantics, but the left hand side works in the *instrumented operational semantics.* A reduction $(e \mid h) \rightarrow^n_\mu (e' \mid h')$ in the latter implies a similar reduction in former; in addition it counts the steps of a reduction with all atomic commands 'unfolded' (with unfolding itself counting one step) and it records all heap accesses; the formal definition is deferred to the appendix of the long version of the paper. We need the former for compatibility of the atomic typing rule below: atomic commands really unfold as they execute, hence we must count the number of 'unfolded' steps. It is less immediate that we must test the actual reads, writes and allocations, recorded by μ, against the effects described by ε, as done in the progress branch of safety. But if omitted, our present proof of the Parallelization Theorem falls short, since it relies on the following simple, but crucial commutation property:

▶ **Lemma 2.** *If we have $l \notin \mu$ and $(e \mid h) \rightarrow^n_\mu (e' \mid h')$, then $(e \mid h[l \mapsto v]) \rightarrow^n_\mu (e' \mid h'[l \mapsto v])$.*

The actual logical relation is given in Figure 7. The existentially quantified $a \in \mathbb{N}$ is the minimal number of anonymous regions required to run; apart from that it uses safety in a straightforward way. There is some asymmetry to these definitions: the anonymous regions A are required to exist (and be empty) in the world beforehand, but are killed off in the termination branch; also the precondition on the (private parts of) the initial local heaps is in the logical relation whereas the postcondition lives in the termination branch. The interpretation of types is in Figure 5. Interpreting the function type looks daunting, but a function is just a suspended expression with a single free variable, hence we have to restate most of the logical relation in the definition. Apart from that, we just

$$\llbracket 1 \rrbracket^R w = \{(k, (), ()) \mid k \in \mathbb{N}\} \qquad \llbracket \mathsf{int} \rrbracket^R w = \{(k, n, n) \mid k \in \mathbb{N} \wedge n \in \mathbb{Z}\}$$

$$\llbracket \tau_1 \times \tau_2 \rrbracket^R w = \Big\{(k, (v_{11}, v_{21}), (v_{12}, v_{22})) \mid (k, v_{11}, v_{12}) \in \llbracket \tau_1 \rrbracket^R w \wedge (k, v_{21}, v_{22}) \in \llbracket \tau_2 \rrbracket^R w\Big\}$$

$$\llbracket \mathsf{ref}_\rho \tau \rrbracket^R w = \begin{cases} \left\{ \begin{array}{c} (k, l_1, l_2) \mid \exists \mu \in \widehat{\mathbf{T}}.\, (l_1, l_2, \mu) \in w(R(\rho)) \wedge \\ \forall w' \sqsupseteq w.\ \llbracket \tau \rrbracket^R w' \overset{k}{=} (\iota\, \mu)(w') \end{array} \right\} & R(\rho) \in \mathrm{dom}(w) \\ \{(k, v_1, v_2) \mid k \in \mathbb{N} \wedge v_1, v_2 \in \mathcal{V}\} & R(\rho) \notin \mathrm{dom}(w) \end{cases}$$

$$\llbracket \tau_1 \to_\varepsilon^{\Pi,\Lambda} \tau_2 \rrbracket^R w =$$

$$\begin{cases} \left\{ \begin{array}{l} (k, \mathsf{fun}\, f(x).e_1, \mathsf{fun}\, f(x).e_2) \mid \exists a \in \mathbb{N}.\, \forall j < k.\, \forall w' \sqsupseteq w. \\ \quad \forall A \subseteq \mathrm{dom}(w').\, \forall v_1, v_2 \in \mathcal{V}.\, \forall h_1, h_2, h_1', h_2' \in \mathcal{H}. \\ \left[\begin{array}{l} R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w') \wedge A \# R(\Pi \cup \Lambda) \wedge |A| \geq a \wedge w'(A) = \emptyset \wedge \\ (j, v_1, v_2) \in \llbracket \tau_1 \rrbracket^R w' \wedge h_1' \subseteq h_1 \wedge h_2' \subseteq h_2 \wedge (j, h_1', h_2') \in \mathbf{P}_\varepsilon^{\Lambda,R}\ w' \end{array} \right] \Rightarrow \\ \qquad (j, h_1', h_2', (\mathsf{fun}\, f(x).e_1)\, v_1, (\mathsf{fun}\, f(x).e_2)\, v_2, h_1, h_2) \in \mathbf{safe}_{\tau_2,\varepsilon}^{\Pi,\Lambda,A,R}\ w', w' \end{array} \right\} & \\ & R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w) \\ \{(k, v_1, v_2) \mid k \in \mathbb{N} \wedge v_1, v_2 \in \mathcal{V}\} & R(\mathrm{FRV}(\varepsilon)) \not\subseteq \mathrm{dom}(w) \end{cases}$$

Figure 5 Interpretation of types. We require $R : \mathcal{RV} \rightharpoonup_{\mathit{fin}} \mathcal{RN}$ injective with $\mathrm{FRV}(\tau) \subseteq \mathrm{dom}(R)$. We assume $R(\mathrm{FRV}(\tau)) \subseteq |w|$ above, otherwise we define $\llbracket \tau \rrbracket^R w$ to be the empty set. In the interpretation of functions, and also below, we write $\#$ to denote disjoint sets. We get that $\llbracket \tau \rrbracket^R \in \mathbf{T}$.

remark that the $R(\rho) \notin \mathrm{dom}(w)$ case of reference interpretation is part of an approach to handling dangling pointers (due to region masking) proposed recently in [26]; similarly for the $R(\mathrm{FRV}(\varepsilon)) \not\subseteq \mathrm{dom}(w)$ case.

To conclude this subsection we give a theorem that, combined with the upcoming compatibility, means that logical relatedness implies contextual may-approximation. The proof is in the appendix of the long version of the paper and it is not hard, but it is worth noting that we need a proof at all: with sequential languages, this is a result one reads off the definition of the logical relation.

▶ **Theorem 3** (May-Equivalence). *Assume that* $\cdot \,|\, \cdot \,|\, \cdot \models e_1 \preceq e_2 : \mathsf{int}, \emptyset$ *holds. Take any* $h_1, h_2 \in \mathcal{H}$. *If there are* e_1', h_1' *with* $(e_1 \,|\, h_1) \overset{*}{\longmapsto} (e_1' \,|\, h_1')$ *such that* $\mathrm{irr}(e_1'|h_1')$ *holds, then there is* $n \in \mathbb{Z}$ *such that* $e_1' = \underline{n}$ *and* h_2' *such that* $(e_2 \,|\, h_2) \overset{*}{\longmapsto} (\underline{n}, h_2')$.

Compatibility of the Logical Relation The logical relation is compatible, i.e., respects all typing rules. This is a *sine qua non* of logical relations; it implies the fundamental lemma stating that every well-typed expression is related to itself. And, as discussed just above, it makes the logical relation approximate contextual may-approximation:

▶ **Theorem 4.** $\Pi \,|\, \Lambda \,|\, \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon$ *implies* $\Pi \,|\, \Lambda \,|\, \Gamma \vdash e_1 \lesssim_{\downarrow} e_2 : \tau, \varepsilon$.

Compatibility means that each typing rule induces a lemma by reading the (unary) typing judgments as the corresponding (binary) logical relations.The three most interesting of these have to do with concurrency and the divide between public and private regions; they are listed here and proofs are given in the appendix of the long version of the paper:

▶ **Lemma 5.** $\Pi \,|\, \Lambda, \rho \,|\, \Gamma \models e_1 \preceq e_2 : \tau,\, \varepsilon$ *implies* $\Pi \,|\, \Lambda \,|\, \Gamma \models e_1 \preceq e_2 : \tau,\, \varepsilon - \rho$ *provided that* $\rho \notin \mathrm{FRV}(\Gamma, \tau)$.

$$
\begin{aligned}
&(k, h_1^\circ, h_2^\circ, e_1, e_2, h_1, h_2) \in \mathbf{safe}_{\tau,\varepsilon}^{\Pi,\Lambda,A,R}\, w^\circ, w \\
&\quad\iff \\
&\forall j \leq k.\, \forall w', g_1, g_2, f_1, f_2. \\
&\quad\Big[\mathbf{envtran}^{\Pi,\Lambda,A,R}\, w, w' \wedge (j, g_1, g_2) \in \mathbf{P}_\varepsilon^{\Pi,R}\; w' \wedge \\
&\quad(g_1, h_1, f_1, g_2, h_2, f_2) \in \mathbf{splits}^{\Pi,\Lambda,A,R}\, w'\Big] \Rightarrow \\
&\quad\Big[\mathrm{irr}(e_1 | g_1 \cdot h_1 \cdot f_1) \Rightarrow \\
&\qquad\exists e_2', w'', h_1', h_2', g_1', g_2'. \\
&\qquad\quad(e_2 \,|\, g_2 \cdot h_2 \cdot f_2) \xmapsto{*} (e_2' \,|\, g_2' \cdot h_2' \cdot f_2) \wedge \mathbf{selftran}^{\Pi,\Lambda,A,R}\, w', w'' \wedge \\
&\qquad\quad\emptyset = (A \cap \mathrm{dom}(w'')) \cup (\mathrm{dom}(w'') \setminus \mathrm{dom}(w')) \wedge g_1 \cdot h_1 = g_1' \cdot h_1' \wedge \\
&\qquad\quad(g_1', h_1', f_1, g_2', h_2', f_2) \in \mathbf{splits}^{\Pi,\Lambda,\emptyset,R}\, w'' \wedge (j, g_1, g_2, g_1', g_2') \in \mathbf{Q}_\varepsilon^{\Pi,R}\; w', w'' \wedge \\
&\qquad\quad(j, e_1, e_2') \in [\![\tau]\!]^R\, (w'') \wedge \exists h_1'' \subseteq h_1', h_2'' \subseteq h_2'.\, (j, h_1^\circ, h_2^\circ, h_1'', h_2'') \in \mathbf{Q}_\varepsilon^{\Lambda,R}\; w^\circ, w''\Big] \wedge \\
&\quad\Big[\forall e_1', h_1^\dagger, \mu, n \leq j.\, (e_1 \,|\, g_1 \cdot h_1 \cdot f_1) \rightarrow_\mu^n (e_1' \,|\, h_1^\dagger) \Rightarrow \\
&\qquad\exists e_2', w'', A', h_1', h_2', g_1', g_2'. \\
&\qquad\quad(e_2 \,|\, g_2 \cdot h_2 \cdot f_2) \xmapsto{*} (e_2' \,|\, g_2' \cdot h_2' \cdot f_2) \wedge \mathbf{selftran}^{\Pi,\Lambda,A,R}\, w', w'' \wedge \\
&\qquad\quad A' = (A \cap \mathrm{dom}(w'')) \cup (\mathrm{dom}(w'') \setminus \mathrm{dom}(w')) \wedge h_1^\dagger = g_1' \cdot h_1' \cdot f_1 \wedge \\
&\qquad\quad(g_1', h_1', f_1, g_2', h_2', f_2) \in \mathbf{splits}^{\Pi,\Lambda,A',R}\, w'' \wedge (j - n, g_1, g_2, g_1', g_2') \in \mathbf{Q}_\varepsilon^{\Pi,R}\; w', w'' \wedge \\
&\qquad\quad\mu \in \mathbf{effs}_{\varepsilon,h_1'}^{A',R}\, w'' \wedge (j - n, h_1^\circ, h_2^\circ, e_1', e_2', h_1', h_2') \in \mathbf{safe}_{\tau,\varepsilon}^{\Pi,\Lambda,A',R}\, w^\circ, w''\Big]
\end{aligned}
$$

Figure 6 Safety. The predicate is defined by well-founded induction. Nontrivial requirements are: $\Pi \# \Lambda$, $\mathrm{FRV}(\tau, \varepsilon) \subseteq \Pi \cup \Lambda$, $\mathrm{FV}(e_1, e_2) = \emptyset$, $R : \Pi \cup \Lambda \hookrightarrow |w^\circ|$, $R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w^\circ)$ and $w \sqsupseteq w^\circ$ with $\mathrm{dom}(w^\circ) \cap R(\Pi \cup \Lambda) \subseteq \mathrm{dom}(w)$, $A \subseteq \mathrm{dom}(w)$ and $A \# R(\Pi \cup \Lambda)$. See Figure 8 for auxiliary definitions. We refer to the contents of the big square brackets as the *prerequisites*, the *termination branch* and the *progress branch*, respectively.

$$
\begin{aligned}
&\Pi \mid \Lambda \mid \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon \\
&\quad\iff \\
&\exists a \in \mathbb{N}.\, \forall k \in \mathbb{N}.\, \forall w \in \mathbf{W}.\, \forall R : \Pi \cup \Lambda \hookrightarrow |w|.\, \forall A \subseteq \mathrm{dom}(w). \\
&\;\forall \gamma_1, \gamma_2 \in \mathcal{V}^{|\Gamma|}.\, \forall h_1, h_2, h_1', h_2' \in \mathcal{H}. \\
&\quad\Big[R(\mathrm{FRV}(\varepsilon)) \subseteq \mathrm{dom}(w) \wedge A \# R(\Pi \cup \Lambda) \wedge |A| \geq a \wedge \forall r \in A.\, w(r) = \emptyset \wedge \\
&\quad(k, \gamma_1, \gamma_2) \in [\![\Gamma]\!]^R w \wedge h_1' \subseteq h_1 \wedge h_2' \subseteq h_2 \wedge (k, h_1', h_2') \in \mathbf{P}_\varepsilon^{\Lambda,R} w\Big] \Rightarrow \\
&\qquad(k, h_1', h_2', e_1[\gamma_1/\Gamma], e_2[\gamma_2/\Gamma], h_1, h_2) \in \mathbf{safe}_{\tau,\varepsilon}^{\Pi,\Lambda,A,R} w, w.
\end{aligned}
$$

Figure 7 The logical relation with anonymous regions. We require that $\Pi \# \Lambda$, $\mathrm{FRV}(\Gamma, \tau, \varepsilon) \subseteq \Pi \cup \Lambda$ and, as always, that $\mathrm{FV}(e_1, e_2) \in |\Gamma|$.

$$\begin{aligned}&\mathbf{envtran}^{\Pi,\Lambda,A,R}\, w, w' \iff w \sqsubseteq w' \wedge \forall r \in \mathrm{dom}(w) \cap (R(\Pi \cup \Lambda) \cup A).\, r \in \mathrm{dom}(w')\\ &\quad \wedge \forall r \in \mathrm{dom}(w) \cap (R(\Lambda) \cup A).\, w(r) = w'(r).\end{aligned}$$

$$\begin{aligned}&\mathbf{selftran}^{\Pi,\Lambda,A,R}\, w, w' \iff w \sqsubseteq w' \wedge \forall r \in \mathrm{dom}(w) \setminus A.\, r \in \mathrm{dom}(w')\\ &\quad \wedge \forall r \in \mathrm{dom}(w) \setminus (R(\Pi \cup \Lambda) \cup A).\, w(r) = w'(r).\end{aligned}$$

$$\begin{aligned}&(g_1, h_1, f_1, g_2, h_2, f_2) \in \mathbf{splits}^{\Pi,\Lambda,A,R}\, w \iff\\ &\quad \mathrm{dom}(h_1) \mathbin{\#} \mathrm{dom}(g_1) \mathbin{\#} \mathrm{dom}(f_1) \wedge \mathrm{dom}(h_2) \mathbin{\#} \mathrm{dom}(g_2) \mathbin{\#} \mathrm{dom}(f_2) \wedge\\ &\quad \mathrm{dom}_1^{\mathrm{R(\Pi)}}(w) = \mathrm{dom}(g_1) \wedge \mathrm{dom}_1^{\mathrm{R(\Lambda)\cup A}}(w) \subseteq \mathrm{dom}(h_1) \wedge\\ &\quad \mathrm{dom}_1^{\mathrm{dom(w)\setminus(R(\Pi\cup\Lambda)\cup A)}}(w) \subseteq \mathrm{dom}(f_1) \wedge\\ &\quad \mathrm{dom}_2^{\mathrm{R(\Pi)}}(w) = \mathrm{dom}(g_2) \wedge \mathrm{dom}_2^{\mathrm{R(\Lambda)\cup A}}(w) \subseteq \mathrm{dom}(h_2) \wedge\\ &\quad \mathrm{dom}_2^{\mathrm{dom(w)\setminus(R(\Pi\cup\Lambda)\cup A))}}(w) \subseteq \mathrm{dom}(f_2).\end{aligned}$$

$$\begin{aligned}&(k, h_1, h_2) \in \mathbf{P}_\varepsilon^{\Theta,R}\, w \iff \mathrm{dom}(h_1) = \mathrm{dom}_1^{\mathrm{R(\Theta)}}(w) \wedge \mathrm{dom}(h_2) = \mathrm{dom}_2^{\mathrm{R(\Theta)}}(w) \wedge\\ &\quad \forall r \in R(\Theta) \cap \mathrm{dom}(w).\, \forall (l_1, l_2, \mu) \in w(r).\\ &\quad r \in R(\mathrm{rds}\,\varepsilon) \Rightarrow k > 0 \Rightarrow (k-1, h_1(l_1), h_2(l_2)) \in (\iota\,\mu)(w).\end{aligned}$$

$$\begin{aligned}&(k, h_1, h_2, h_1', h_2') \in \mathbf{Q}_\varepsilon^{\Theta,R}\, w, w' \iff\\ &\quad \mathrm{dom}(h_1) = \mathrm{dom}_1^{\mathrm{R(\Theta)}}(w) \wedge \mathrm{dom}(h_2) = \mathrm{dom}_2^{\mathrm{R(\Theta)}}(w) \wedge\\ &\quad \mathrm{dom}(h_1') = \mathrm{dom}_1^{\mathrm{R(\Theta)}}(w') \wedge \mathrm{dom}(h_2') = \mathrm{dom}_2^{\mathrm{R(\Theta)}}(w') \wedge\\ &\quad \big(\forall r \in R(\Theta) \cap \mathrm{dom}(w).\, \forall (l_1, l_2, \mu) \in w(r).\\ &\qquad [h_1(l_1) = h_1'(l_1) \wedge h_2(l_2) = h_2'(l_2)] \vee [r \in R(\mathrm{wrs}\,\varepsilon) \wedge\\ &\qquad\ k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\,\mu)(w')]\big) \wedge\\ &\quad \big(\forall r \in R(\Theta) \cap \mathrm{dom}(w).\\ &\qquad \forall (l_1, l_2, \mu) \in w'(r) \setminus w(r).\, r \in R(\mathrm{als}\,\varepsilon) \wedge\\ &\qquad k > 0 \Rightarrow (k-1, h_1'(l_1), h_2'(l_2)) \in (\iota\,\mu)(w')\big).\end{aligned}$$

$$\begin{aligned}&\mu \in \mathbf{effs}_{\varepsilon,h}^{A,R}\, w \iff \{l \mid rd_l \in \mu\} \cap \mathrm{dom}_1(w) \subseteq \mathrm{dom}_1^{\mathrm{R(rds\,\varepsilon)\cup A}}(w) \wedge\\ &\quad \{l \mid wr_l \in \mu\} \cap \mathrm{dom}_1(w) \subseteq \mathrm{dom}_1^{\mathrm{R(wrs\,\varepsilon)\cup A}}(w) \wedge\\ &\quad \{l \mid al_l \in \mu\} \cap \mathrm{dom}_1(w) \subseteq \mathrm{dom}_1^{\mathrm{R(als\,\varepsilon)\cup A}}(w) \wedge\\ &\quad \{l \mid rd_l \in \mu \vee wr_l \in \mu \vee al_l \in \mu\} \setminus \mathrm{dom}_1(w) \subseteq \mathrm{dom}(h).\end{aligned}$$

Figure 8 Six auxiliary definitions. The *environment transition* and *self transition* relations are defined for $\Pi \mathbin{\#} \Lambda$, $R : \Pi \cup \Lambda \hookrightarrow |w|$, $A \subseteq \mathrm{dom}(w)$ and $R(\Pi \cup \Lambda) \mathbin{\#} A$. The *triple-split* relation has the same prerequisites. The *precondition* relation is defined for $R : \mathcal{RV} \rightharpoonup_{fin} |w|$ injective with $\Theta \cup \mathrm{FRV}(\varepsilon) \subseteq \mathrm{dom}(R)$. The *postcondition* relation additionally requires $w' \sqsupseteq w$ such that $\mathrm{dom}(w) \cap R(\Theta) \subseteq \mathrm{dom}(w')$. Finally the *actual-effects* relation expects $R : \mathcal{RV} \rightharpoonup_{fin} |w|$ injective with $\mathrm{FRV}(\varepsilon) \subseteq \mathrm{dom}(R)$ and $A \subseteq \mathrm{dom}(w)$.

▶ **Lemma 6.** $\cdot \,|\, \Pi, \Lambda \,|\, \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon$ *implies* $\Pi \,|\, \Lambda \,|\, \Gamma \models \mathsf{atomic}\ e_1 \preceq \mathsf{atomic}\ e_2 : \tau, \varepsilon$ *if* $\mathrm{als}\,\varepsilon \subseteq \mathrm{rds}\,\varepsilon \cap \mathrm{wrs}\,\varepsilon$.

▶ **Lemma 7.** $\Pi, \Lambda \,|\, \cdot \,|\, \Gamma \models e_1 \preceq e_2 : \tau, \varepsilon$ *and* $\Pi, \Lambda \,|\, \cdot \,|\, \Gamma \models e_1^\dagger \preceq e_2^\dagger : \tau^\dagger, \varepsilon^\dagger$ *together imply* $\Pi \,|\, \Lambda \,|\, \Gamma \models \mathsf{par}\ e_1\ \mathsf{and}\ e_1^\dagger \preceq \mathsf{par}\ e_2\ \mathsf{and}\ e_2^\dagger : \tau \times \tau^\dagger, \varepsilon \cup \varepsilon^\dagger$.

4 Applications

4.1 Parallelization Theorem: Disjoint Concurrency

We now explain our Parallelization Theorem, which gives us an easy way to prove properties about the common case of disjoint concurrency, where disjointness is captured using private regions and effect annotations.

▶ **Theorem 8** (Parallelization). *Assuming that*

1. $\Pi, \Lambda \mid \cdot \mid \Gamma \vdash e_1 : \tau_1, \varepsilon_1$,
2. $\Pi, \Lambda \mid \cdot \mid \Gamma \vdash e_2 : \tau_2, \varepsilon_2$,
3. $\mathrm{rds}\,\varepsilon_1 \cup \mathrm{wrs}\,\varepsilon_1 \cup \mathrm{rds}\,\varepsilon_2 \cup \mathrm{wrs}\,\varepsilon_2 \subseteq \Lambda$,
4. $\mathrm{rds}\,\varepsilon_1 \cap \mathrm{wrs}\,\varepsilon_2 = \mathrm{rds}\,\varepsilon_2 \cap (\mathrm{wrs}\,\varepsilon_1 \cup \mathrm{als}\,\varepsilon_1) = \mathrm{wrs}\,\varepsilon_1 \cap \mathrm{wrs}\,\varepsilon_2 = \emptyset$,

the following property holds:

$$\Pi \mid \Lambda \mid \Gamma \models \langle e_1, e_2 \rangle \cong \ \mathsf{par}\ e_1\ \mathsf{and}\ e_2 : \tau_1 \times \tau_2, \varepsilon_1 \cup \varepsilon_2.$$

Intuitively, item 3 keeps the environment from detecting anything, and item 4 prevents the two computations from talking among themselves, thereby making them independent; the $\mathrm{als}\,\varepsilon_1$ in item 4 is a technicality that we cannot do without. We showed a concrete simple application of this theorem in the Introduction. More generally, example usage includes situations where we operate on two imperative data structures (say linked lists or graphs); if we only mutate parts of the data structures that are in different regions, then we may safely parallelize operations on the data structures.

The masking rule makes it possible to do more optimizations via the Parallelization Theorem: Consider, for simplicity, the familiar example of an efficient implementation *fib* of the Fibonacci function using two local references. We can use the masking rule to give it type and effect $\mathsf{int} \to_{\emptyset}^{\cdot,\cdot} \mathsf{int}, \emptyset$. This allows us to view the imperative implementation as pure, and thus by Theorem 8 we find that it is sound to optimize two sequential calls to *fib* to two parallel calls. This may sound like a simple optimization, but the point is that a compiler can perform it automatically, just based on the effect types. It also underlines how we are able to reason about more involved behaviors of concurrent threads, even though the type system provides only rough bounds on interference through the private-public distinction.

The proof of the Parallelization Theorem is quite tricky. Please see the the appendix of the long version of the paper for an informal overview of the proof and the technical details.

4.2 Non-disjoint Concurrency

We now exemplify how our logical relations model can also be used to reason compositionally about equivalences of fine-grained concurrent programs operating on public regions.

Consider the following type

$$\tau \equiv \mathsf{ref}_\rho \mathsf{int} \to_{\{rd_\rho, wr_\rho\}}^{\rho,\emptyset} 1$$

of functions that take an integer reference in a public region, possibly read and write from the reference, and return unit. The following two functions

$$\begin{array}{l} \mathsf{fun}\,\mathsf{inc}_1(x).\ \mathsf{let}\ y = !\,x\ \mathsf{in}\ \mathsf{let}\ z = y+1\ \mathsf{in} \\ \qquad\qquad\quad \mathsf{if}\ \mathsf{cas}\ (x,y,z)\ \mathsf{then}\ \langle\rangle\ \mathsf{else}\ \mathsf{inc}_1\ (x) \end{array} \quad \text{and} \quad \mathsf{fun}\,\mathsf{inc}_2(x).\ \mathsf{atomic}\ (x := !\,x+1)$$

both have type τ. (We have allowed ourselves to use a standard conditional expression; 1 corresponds to true and 0 to false.) Both functions increment the integer given in their reference arguments; inc_1 uses the fine-grained compare-and-swap to do it atomically, whereas inc_2 uses the brute-force atomic operation. Using our logical relations model, we can prove that inc_1 and inc_2 are contextually equivalent:

$$\rho \,|\, \cdot \,|\, \cdot \vdash \mathsf{inc}_1 \approx \mathsf{inc}_2 : \tau, \emptyset. \tag{1}$$

Hence, replacing inc_2 with inc_1 in any well-typed client gives two contextually equivalent expressions. Thus our logical relation models a form of *data abstraction for concurrency* (where we abstract over the granularity of concurrency in the module).

We now show how to use the equivalence of inc_1 and inc_2 to derive equivalences of two different clients using the fine-grained concurrency implementation inc_1.

To this end, consider the following two client programs of type

$$\sigma \equiv \tau \to^{\rho,\emptyset}_{\emptyset} \mathsf{ref}_\rho \mathsf{int} \to^{\rho,\emptyset}_{\{rd_\rho, wr_\rho\}} \mathsf{int},$$

$$\mathsf{fun}\,\mathsf{c}_1(\mathsf{inc}).\lambda\, n.\mathsf{inc}\, n;\, \mathsf{inc}\, n;\, !\, n \quad \text{and} \quad \mathsf{fun}\,\mathsf{c}_2(\mathsf{inc}).\lambda\, n.(\mathsf{par}\ \mathsf{inc}\, n\ \mathsf{and}\ \mathsf{inc}\, n);\, !\, n$$

Note that c_1 makes two sequential calls to inc, whereas c_2 runs the two calls in parallel. Because of the use of compare-and-swap in inc_1, we would hope that the $\mathsf{c}_1\,\mathsf{inc}_1$ and $\mathsf{c}_2\,\mathsf{inc}_1$ are contextually equivalent (in typing context $\rho \,|\, \emptyset \,|\, \emptyset$). We can prove that this is indeed the case using compositional reasoning as follows. Using our logical relation, we prove that $\mathsf{c}_1\,\mathsf{inc}_2$ is contextually equivalent to $\mathsf{c}_2\,\mathsf{inc}_2$, i.e.,

$$\rho \,|\, \cdot \,|\, \cdot \vdash \mathsf{c}_1\,\mathsf{inc}_2 \approx \mathsf{c}_2\,\mathsf{inc}_2 : \mathsf{ref}_\rho \mathsf{int} \to^{\rho,\emptyset}_{\{rd_\rho, wr_\rho\}} \mathsf{int}, \emptyset. \tag{2}$$

Finally, we conclude that $\mathsf{c}_1\,\mathsf{inc}_1$ is contextually equivalent to $\mathsf{c}_2\,\mathsf{inc}_1$ by transitivity of contextual equivalence (using (1), (2) and (1) again for the respective steps):

$$\mathsf{c}_1\,\mathsf{inc}_1 \approx \mathsf{c}_1\,\mathsf{inc}_2 \approx \mathsf{c}_2\,\mathsf{inc}_2 \approx \mathsf{c}_2\,\mathsf{inc}_1$$

This proof illustrates an important point: to show equivalence of two clients of a module implemented using fine-grained concurrency, it suffices to show that the clients are equivalent wrt. a coarse-grained implementation, and that the coarse-grained implementation is equivalent to the fine-grained implementation. This is often a lot simpler than trying to show the equivalence of the clients wrt. the fine-grained implementation directly. We can think of the coarse-grained implementation of the module (here inc_2) as the *specification* of the module and the fine-grained implementation (here inc_1) as its *implementation.*

The formal proofs of (1) and (2) follow by straightforward induction.

5 Discussion

Gifford and Lucassen [15, 20] originally proposed type-and-effect systems as a static analysis for determining which parts of a higher-order imperative program could be implemented using parallelism. Here we are able to express the formal correctness of these ideas in a succinct way by having a parallel construct in our programming language and establishing the Parallelization Theorem.

In Section 4.2 we showed how contextual equivalence can be used to *state* that compare-and-swap can be used to implement a simple form of locking, and how our logical relations model could be used to *prove* this statement. We believe that it should be possible to give similar succinct statements and proofs of other implementations of synchronization; confer work by Turon and Wand [28].

As mentioned earlier, we have deliberately used the same definition of worlds here as in [26]. As discussed there [26, Section 8.2], this notion of world has somewhat limited expressiveness: the only heap invariants we can state are those that relate values at two locations by a semantic type. To increase expressiveness, it would thus be interesting to extend our model using ideas from [12], and then investigate more examples of equivalences.

Recently, Liang et. al. [19] have proposed RGSim, a simulation based on rely-guarantee, to verify program transformations in a concurrent setting. Their actual definition [19, Definition 4] bears some resemblance to our safety relation; indeed, an early draft of *loc.cit.* was a source of inspiration. They have no division of the heap into public and private parts, instead they give a pair of rely and guarantee that, respectively, constrain the interference of the environment and the actions of the computation. Their approach is essentially untyped; one point of view is that we 'auto-instantiate' the many parameters of their simulation based on our typing information. They consider first-order languages with ground store; this obviously keeps life simple, but the example equivalences they give are not.

Our simple example of data abstraction for concurrency in Section 4.2 suggests that there could be a relationship to linearizability. We intend to explore whether a formal relationship can be established in our higher-order setting; confer work by Filipović et. al. [14].

6 Conclusion and Future Work

We have presented a logical relations model of a new type-and-effect system for a concurrent higher-order ML-like language with general references. We have shown how to use the model for reasoning about both disjoint and non-disjoint concurrency. In particular, we have proved the first automatic Parallelization Theorem for such a rich language.

In this paper, we have focused on may contextual equivalence. Future work includes investigating models for must contextual equivalence. Since our language allows the encoding of countable nondeterminism, must equivalence is non-trivial, and will probably involve indexing over ω_1 rather than ω [24]. Future work also includes extending the model to region and effect polymorphism, as well as the extension to more expressive worlds, and to other concurrency constructs such as fork-join.

The authors would like to thank Jan Schwinghammer and Xinyu Feng for discussions of aspects of this work.

References

1 Amal Ahmed. *Semantics of Types for Mutable State.* PhD thesis, Princeton University, 2004.

2 Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *POPL*, 2009.

3 N. Benton, L. Beringer, M. Hofmann, and A. Kennedy. Relational semantics for effect-based program transformations with dynamic allocation. In *PPDP*. ACM, 2007.

4 N. Benton, L. Beringer, M. Hofmann, and A. Kennedy. Relational semantics for effect-based program transformations: Higher-order store. In *PPDP*. ACM, 2009.

5 N. Benton and P. Buchlovsky. Semantics of an effect analysis for exceptions. In *TLDI*, 2007.

6 N. Benton, A. Kenney, M. Hofmann, and L. Beringer. Reading, writing and relations: Towards extensional semantics for effect analyses. In *APLAS*, 2006.

7 L. Birkedal, B. Reus, J. Schwinghammer, K. Støvring, J. Thamsborg, and H. Yang. Step-indexed Kripke models over recursive worlds. In *POPL*, 2011.

8 L. Birkedal, J. Thamsborg, and K. Støvring. Realizability semantics of parametric polymorphism, general references, and recursive types. In *FOSSACS*, 2009.

9 L. Birkedal, M. Tofte, and M. Vejlstrup. From region inference to von Neumann machines via region representation inference. In *POPL*, 1996.

10 A. Buisse, L. Birkedal, and K. Støvring. A step-indexed Kripke model of separation logic for storable locks. In *MFPS*, 2011.

11 M. Dodds, X. Feng, M.J. Parkinson, and V. Vafeiadis. Deny-guarantee reasoning. In *ESOP*, 2009.

12 D. Dreyer, G. Neis, and L. Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *ICFP 2010*, pages 143–156. ACM, 2010.

13 X. Feng, R. Ferreira, and Z. Shao. On the relationship between concurrent separation logic and assume-guarantee reasoning. In *ESOP*, 2007.

14 Ivana Filipovic, Peter W. O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *TCS*, 2010.

15 D.K. Gifford and J.M. Lucassen. Integrating functional and imperative programming. In *LISP and Functional Programming*, 1986.

16 F. Henglein, H. Makholm, and H. Niss. Effect types and region-based memory management. In B.C. Pierce, editor, *Advanced Topics in Types and Programming Languages*. 2005.

17 Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *POPL*, 2006.

18 James Laird. A fully abstract trace semantics for general references. In *ICALP*, 2007.

19 H. Liang, X. Feng, and M. Fu. A rely-guarantee-based simulation for verifying concurrent program transformations. In *POPL*, 2012.

20 J.M. Lucassen and D.K. Gifford. Polymorphic effect systems. In *POPL*, 1988.

21 A. Murawski and N. Tzevelekos. Game semantics for good general references. In *LICS*, 2011.

22 Peter W. O'Hearn. Resources, concurrency, and local reasoning. *TCS*, 2007.

23 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *TOPLAS*, 2011.

24 Jan Schwinghammer and Lars Birkedal. Step-indexed relational reasoning for countable nondeterminism. In *CSL*, 2011.

25 Eijiro Sumii. A complete characterization of observational equivalence in polymorphic λ-calculus with general references. In *CSL*, 2009.

26 Jacob Thamsborg and Lars Birkedal. A Kripke logical relation for effect-based program transformations. In *ICFP*, 2011.

27 M. Tofte and J.-P. Talpin. Implementation of the typed call-by-value λ-calculus using a stack of regions. In *Proceedings of POPL*, 1994.

28 Aaron Joseph Turon and Mitchell Wand. A separation logic for refining concurrent objects. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 247–258. ACM, 2011.

29 V. Vafeiadis and M.J. Parkinson. A marriage of rely/guarantee and separation logic. In *CONCUR*, 2007.

30 Viktor Vafeiadis. Concurrent separation logic and operational semantics. In *MFPS*, 2011.

Equivalence Constraint Satisfaction Problems*

Manuel Bodirsky[1] and Michał Wrona[2]

1 CNRS / LIX (UMR 7161)
École Polytechnique
91128 Palaiseau, France
bodirsky@lix.polytechnique.fr

2 Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden
michal.wrona@liu.se

Abstract

The following result for finite structures Γ has been conjectured to hold for all countably infinite *ω-categorical* structures Γ: either the model-complete core Δ of Γ has an expansion by finitely many constants such that the pseudovariety generated by its polymorphism algebra contains a two-element algebra all of whose operations are projections, or there is a homomorphism f from Δ^k to Δ, for some finite k, and an automorphism α of Δ satisfying $\forall x_1, \dots, x_k.\, f(x_1, \dots, x_k) = \alpha(f(x_2, \dots, x_k, x_1))$. This conjecture has been confirmed for all infinite structures Γ that have a first-order definition over $(\mathbb{Q}; <)$, and for all structures that are definable over the random graph. In this paper, we verify the conjecture for all structures that are definable over an equivalence relation with a countably infinite number of countably infinite classes.

Our result implies a complexity dichotomy (into NP-complete and P) for a family of constraint satisfaction problems (CSPs) which we call *equivalence constraint satisfaction problems*. The classification for equivalence CSPs can also be seen as a first step towards a classification of the CSPs for all relational structures that are first-order definable over Allen's interval algebra, a well-known constraint calculus in temporal reasoning.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Constraint satisfaction problems, universal algebra, model theory, Ramsey theory, temporal reasoning, computational complexity

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.122

1 Introduction

The *constraint satisfaction problem* for a fixed structure Γ with finite relational signature is the following computational problem, denoted by CSP(Γ): given a finite structure I with the same signature as Γ, decide whether there is a homomorphism from I to Γ. By selecting an appropriate structure Γ, many computational problems in various areas of theoretical computer science can be formulated as CSP(Γ), for example problems from artificial intelligence, combinatorics, finite model theory, scheduling, and database theory.

* This work has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 257039), and the National Graduate School in Computer Science (CUGS), Sweden.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 122–136
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In 1993, Feder and Vardi conjectured that CSP(Γ) is for all finite structures Γ in P or NP-complete. There has been a considerable research activity around this *dichotomy conjecture* in constraint satisfaction, producing many results of independent interest.

One of the results that came out of the attempts to prove the dichotomy conjecture is the following *universal-algebraic dichotomy*, which essentially follows from [17, 2]; also see [4]. All concepts that appear in the statement will be defined in Section 2.

▶ **Theorem 1.1** (follows from [17, 2]). *Let Γ be a finite relational structure. Then either*

- *the pseudovariety generated by the polymorphism algebra of the expansion of the core of Γ by constants contains a two-element algebra all of whose operations are projections, or*
- *there is a homomorphism f from Γ^k to Γ, for some finite $k \geq 2$, that satisfies*

$$\forall x_1, \dots, x_k .\, f(x_1, \dots, x_k) = f(x_2, \dots, x_k, x_1)\ .$$

It is known that when Γ satisfies the first item in Theorem 1.1, then CSP(Γ) is NP-hard. Bulatov, Jeavons, and Krokhin [13] made the conjecture that for finite structures Γ with finite relational signature that do not satisfy the first item in Theorem 1.1, the problem CSP(Γ) can be solved in polynomial time. This conjecture has been called the *tractability conjecture*, and obviously the tractability conjecture implies the dichotomy conjecture. The tractability conjecture has been verified for 2-element structures [22], 3-element structures [12], undirected graphs [11], and many other classes of finite structures.

While the tractability conjecture is open for general finite structures, it turns out that a generalized version of the tractability conjecture is true for several large classes of infinite relational structures Γ. To define those classes, we need the following concepts. In this paper we say that a relational structure Γ is *first-order definable* in Δ if Γ has the same domain as Δ, and for every relation R of Γ there is a first-order formula ϕ in the signature of Δ such that ϕ holds exactly on those tuples that are contained in R. The class of all structures with a first-order definition in $(\mathbb{Q}; <)$ has been studied in [6]; the CSPs for those structures are called *temporal constraint satisfaction problems* and they can be used to model many computational problems in temporal reasoning and scheduling. The class of all structures with a first-order definition over the countable universal homogeneous graph, aka the *random graph*, has been studied in [9]. All those structures are *ω-categorical*, that is, all countable models of their first-order theory are isomorphic.

The following has been conjectured for *all* ω-categorical structures (see Conjecture 5.3 from [4]; the formulation there is different, but equivalent by Theorem 5.5.18 in [4]).

▶ Conjecture 1.2. Let Γ be a countable ω-categorical relational structure. Then either

1. the model-complete core of Γ has an expansion Δ by finitely many constants such that the pseudovariety generated by the polymorphism algebra of Δ contains a two-element algebra all of whose operations are projections, or
2. the model-complete core of Γ has a polymorphism f and an automorphism α satisfying

$$\forall x_1, \dots, x_n .\, f(x_1, \dots, x_n) = \alpha(f(x_2, \dots, x_n, x_1))\ .$$

This conjecture generalizes the universal-algebraic dichotomy that holds for finite structures Γ. Conjecture 1.2 has been shown for all structures Γ definable over $(\mathbb{Q}; <)$ [6], or over the random graph [9]. Moreover, the two cases of Conjecture 1.2 correspond precisely to the cases that CSP(Γ) is NP-hard, or polynomial, respectively.

In this article, we show that Conjecture 1.2 holds for all structures that are first-order definable over $(D; Eq)$, where D is a countable infinite set, and Eq is an equivalence relation on D with infinitely many infinite classes. We show that also in this case the dichotomy described in the conjecture coincides with a complexity dichotomy for the corresponding CSPs. We call them *equivalence CSPs*, since solutions to an instance I of CSP(Γ) where Γ is first-order definable over $(D; Eq)$ can be represented by exhibiting an equivalence relation on the image of a mapping from I to Γ (and thus CSP(Γ) is always in NP).

Apart from the fact that $(D; Eq)$ is, besides $(\mathbb{Q}; <)$ and the random graph, one of the fundamental ω-categorical structures, there is additional motivation to specifically study the class of structures definable over $(D; Eq)$, and we describe this motivation in the following.

1.1 Motivation and Applications

1.1.1 Composing Classification Results

Suppose Δ_1 and Δ_2 are such that we have shown Conjecture 1.2 for all structures Γ that are definable over Δ_1 or definable over Δ_2. To better understand Conjecture 1.2 in general, we would like to prove that the conjecture also holds for all structures Γ that are definable over a structure Δ that is built from Δ_1 and Δ_2 in a simple way. One of the basic ways to construct a new ω-categorical structure Δ from ω-categorical structures Δ_1 and Δ_2 is to take infinitely many copies of Δ_2, to identify each element of Δ_1 with one of those copies, and to join the copies according to the relations in Δ_1. Formally, for $i \in \{1, 2\}$, write D_i for the domain and τ_i for the signature of Δ_i. Suppose that τ_1 and τ_2 are disjoint (otherwise rename the symbols). Then Δ is a $\tau_1 \cup \tau_2$ structure with domain $D_1 \times D_2$. A k-ary relation $R \in \tau_2$ denotes $\{((a, b_1), \ldots, (a, b_k)) \mid (b_1, \ldots, b_k) \in R^{\Delta_2}, a \in D_1\}$ in Δ; a k-ary relation $R \in \tau_1$ denotes $\{((a_1, b_1), \ldots, (a_k, b_k)) \mid (a_1, \ldots, a_k) \in R^{\Delta_1}, b_1, \ldots, b_k \in D_2\}$ in Δ.

The simplest situation for this is when $\Delta_1 = \Delta_2 = (\mathbb{N}; =)$. Note that the structure $(D; Eq)$ is isomorphic to $\big(\mathbb{N}; \{(x, y), (u, v) \mid x = u\}\big)$; that is, the relation Eq relates exactly those elements that come from the same copy of Δ_2. So the task outlined above for $\Delta_1 = \Delta_2 = (\mathbb{N}; =)$ amounts precisely to studying the class of all structures Γ definable over $(D; Eq)$.

1.1.2 Fragments of Allen's Interval Algebra

Allen's interval algebra is a formalism introduced for temporal reasoning in Artificial Intelligence [1], and plays a central role in qualitative reasoning in general. The most fundamental computational problem for Allen's interval algebra is the so-called *network satisfaction problem*, which can be viewed as the CSP for the following structure Δ: the domain $\mathbb{I}$ of Δ are the pairs $(u, v) \in \mathbb{Q}^2$ with $u < v$, and the relations of Δ are *all* binary relations R such that the 4-ary relation $\{(x, y, u, v) \mid ((x, y), (u, v)) \in R\}$ has a first-order definition in $(\mathbb{Q}; <)$. An important achievement in temporal reasoning is the complete complexity classification of the *fragments* of Allen's interval algebra in [20, 21], that is, of the constraint satisfaction problems for structures Γ obtained from Δ by removing some of the relations.

This result has been obtained *without* the universal-algebraic approach as it is used in [13, 2, 6, 9], but by a clever case distinction and heavy use of primitive positive definitions to show hardness in cases where the known algorithms do not apply. A proof based on the universal-algebraic approach would have the advantage that it would automatically yield the much stronger classification result for all structures Γ that are first-order definable in Δ. In contrast to the classification in [20], this includes structures that have relations of arity larger than two. Such a result would be a considerable extension of the result from [20], and is currently out of reach. However, for structures Γ with a first-order definition in Δ that contain the binary relation $\big\{((x, y), (u, v)) \mid y = u\big\}$ (this relation is typically denoted by m in the literature on Allen's interval algebra), a classification of the complexity of CSP(Γ) can be derived from the classification for the structures definable in $(\mathbb{Q}; <)$ (see Section 5.5.4 in [4]). Note that every structure with a definition in $(D; Eq)$ is isomorphic to a structure definable over Allen's interval algebra, by the observation that $(D; Eq)$ is isomorphic to

$$\big(\mathbb{I}; \{((x,y),(u,v)) \mid x = u\}\big) \; .$$

Hence, the classification presented here is a part of the more ambitious project to classify the CSP for all structures that are first-order definable over Allen's interval algebra.

1.2 Techniques and Outline

We give a description of our proof strategy; in this description, we freely use concepts that will be introduced in Section 2. Let Γ be a structure with a first-order definition in $(D; Eq)$. If the binary relation $E(x,y)$ defined by $Eq(x,y) \wedge x \neq y$, or the binary relation $N(x,y)$ defined by $\neg Eq(x,y)$ is not primitive positive definable in Γ, then Γ must have an endomorphism that does not preserve E or that does not preserve N. It turns out that in this case Γ is degenerate, and we use a Ramsey-theoretic analysis of the endomorphisms to reduce the classification to known results (Theorem 3.3). If E and N *are* primitive positive definable, then so is Eq, and we are in the situation that the polymorphism algebra $\mathbf{A}$ of Γ has a non-trivial congruence, namely Eq. The quotient of $\mathbf{A}$ by Eq is an algebra that contains all permutations of its domain, and for such algebras Conjecture 1.2 has already been established (Theorem 3.3). Moreover, we will consider certain algebras of $\mathbf{A}$ obtained from the congruence classes of Eq, and again they contain all permutations of their domain. The central part of the paper is a universal-algebraic argument how to combine the classification results for the quotient and the congruence classes to obtain the general classification result.

2 Tools...

2.1 ...from Model Theory

In this paper we consider two kinds of first-order structures: *relational structures* (typically ω-categorical or finite, sometimes expanded with constants) and *algebras*, that is, structures with a functional signature (see Section 2.2).

Let σ and τ be signatures with $\sigma \subseteq \tau$. When Δ is a σ-structure and Γ is a τ-structure with the same domain such that $R^\Delta = R^\Gamma$ for all $R \in \sigma$, and $f^\Delta = f^\Gamma$ for all $f \in \sigma$, then Δ is called a *reduct* of Γ, and Γ is called an *expansion* of Δ. We say that Γ is a *first-order expansion* of Δ if Γ is an expansion of Δ and all relations in Γ are first-order definable over Δ. A structure Δ is called a *finite reduct* of Γ if Δ is a reduct of Γ with a finite signature. We also write (Γ, R) for the expansion of Γ by a new relation R. Given two σ-structures Γ over the domain A and Δ over the domain B, Δ is said to be an *(induced) substructure* of Γ iff (i) $B \subseteq A$, (ii) for every n-ary function symbol f in σ the function f^Δ is a restriction of f^Γ to B^n, and (iii) for every n-ary relation symbol R in σ we have $R^\Delta = R^\Gamma \cap B^n$. For two τ-structures Γ_1 and Γ_2 the *direct product* $\Delta = \Gamma_1 \times \Gamma_2$ is the τ structure on the domain $A_1 \times A_2$, where A_1 is the domain of Γ_1 and A_2 is the domain of Γ_2 such that: (i) for every n-ary relation symbol R in τ we have $((a^1_1, a^1_2), \dots, (a^n_1, a^n_2)) \in R^\Delta$ iff $(a^1_1, \dots, a^n_1) \in R^{\Gamma_1}$ and $(a^1_2, \dots, a^n_2) \in R^{\Gamma_2}$, and (ii) for every n-ary function symbol f in τ we have that $f^\Delta\big((a^1_1, a^1_2), \dots, (a^n_1, a^n_2)\big) = \big(f^{\Gamma_1}(a^1_1, \dots, a^n_1), f^{\Gamma_2}(a^1_2, \dots, a^n_2)\big)$. The direct product $\Gamma \times \Gamma$ is also denoted by Γ^2, and the k-fold product $\Gamma \times \cdots \times \Gamma$, defined analogously, by Γ^k.

We say that a map h from the domain of a τ-structures Γ to the domain of a τ-structure Δ *preserves* a first-order τ-formula ϕ with free variables $x_1, \dots, x_n$ if for all elements $a_1, \dots, a_n$ of Γ such that Γ satisfies $\phi(a_1, \dots, a_n)$, Δ satisfies $\phi(h(a_1), \dots, h(a_n))$. A map $h \colon \Gamma \to \Delta$ is a *homomorphism* if it preserves all atomic τ-formulas. An *embedding* is an injective homomorphism satisfying the stronger condition that $(t_1, \dots, t_n) \in R^\Gamma$ if and only if $(h(t_1), \dots, h(t_n)) \in R^\Delta$, for all relation symbols $R \in \tau$. An *isomorphism* is a surjective embedding, and an *automorphism* of Γ is an isomorphism between Γ and itself. The set of all automorphisms of Γ is denoted by $\mathrm{Aut}(\Gamma)$. An *orbital* of $\mathrm{Aut}(\Gamma)$ is a binary relation of the form $\big\{(\alpha(t_1), \alpha(t_2)) \mid \alpha \in \mathrm{Aut}(\Gamma)\big\}$ for elements t_1, t_2 of Γ.

A first-order theory T is *model-complete* if every embedding between models of T preserves all first-order formulas. We say that a structure is model-complete if its theory is model-complete. A homomorphism of a structure Γ into itself is called an *endomorphism*. A structure Γ is called a *core* if all endomorphisms of Γ are embeddings. A structure Δ is called a *core of* Γ if Δ is a core as well as Γ and Δ are *homomorphically equivalent*, that is, there is a homomorphism from Γ to Δ and a homomorphism from Δ to Γ.

▶ **Theorem 2.1** ([3]). *Every ω-categorical structure Γ is homomorphically equivalent to an ω-categorical model-complete core Δ. All model-complete cores of Γ are isomorphic.*

When Δ is a model-complete core with finite relational signature, c is an element of the domain of Δ, and $(\Delta, \{c\})$ is the expansion of Δ by the unary relation $\{c\}$, then there is a polynomial-time reduction from $\mathrm{CSP}((\Delta, \{c\}))$ to $\mathrm{CSP}(\Delta)$.

A structure Γ is *homogeneous* if every isomorphism between finite substructures of Γ can be extended to an automorphism of Γ. Homogeneous structures with a finite signature are ω-categorical. Good introductions to ω-categoricity can be found in [14, 18].

2.2 . . . from Universal Algebra

Let Γ be a structure. Homomorphisms from Γ^k to Γ are called *polymorphisms* of Γ. When R is a relation over the set D, we say that $f \colon D^k \to D$ *preserves* R if f is a polymorphism of $(D; R)$, and that f *violates* R otherwise. The set of all polymorphisms of a relational structure Γ, denoted by $\mathrm{Pol}(\Gamma)$, forms an algebraic object called a *clone*. A clone on some fixed domain D is a set of operations on D containing all projections and closed under composition. A clone $\mathscr{C}$ is *locally closed* iff for all natural numbers n, for all n-ary operations g on D, if for all finite $B \subseteq D^n$ there exists an n-ary $f \in \mathscr{C}$ which agrees with g on B, then $g \in \mathscr{C}$. A set of operations $\mathscr{F}$ *locally generates* an operation f if f is in the smallest locally closed clone containing $\mathscr{F}$, denoted by $\langle \mathscr{F} \rangle$.

▶ Proposition 2.2 (see e.g. Propositions 5.1.1 and 5.2.1 in [4]). Let $\mathscr{F}$ be a set of operations on some domain D. Then the following are equivalent: (i) $\mathscr{F}$ is the polymorphism clone of a relational structure; and (ii) $\mathscr{F}$ is a locally closed clone. Moreover, $\mathscr{F}$ locally generates g if and only if g preserves all relations preserved by $\mathscr{F}$.

Primitive positive formulas over a signature τ are first-order formulas built exclusively from conjunction, existential quantifiers, equality and relation symbols from τ. The first part of the following theorem is from [7], the second part is a straightforward consequence of Theorem 5.2.3 and Lemma 5.3.5 in [4].

▶ **Theorem 2.3.** *A relation R has a primitive positive definition in an ω-categorical structure Γ if and only if R is preserved by all polymorphisms of Γ. An orbital O of $\mathrm{Aut}(\Gamma)$ has a primitive positive definition in Γ if and only if O is preserved by all endomorphisms of Γ.*

An algebra $\mathbf{A}$ whose set of operations equals $\mathrm{Pol}(\Gamma)$ is called a *polymorphism algebra* of Γ; note that polymorphism algebras are not unique, since we can freely rename the operations in $\mathbf{A}$ and still obtain a polymorphism algebra; however, since such a renaming is in our context always irrelevant, we also call $\mathbf{A}$ *the* polymorphism algebra of Γ, and denote it by $\text{Alg}(\Gamma)$, as if it were unique.

A *congruence* of an algebra $\mathbf{A}$ with domain A is an equivalence relation on A that is preserved by all operations in $\mathbf{A}$. Let $\mathbf{A}, \mathbf{B}$ be algebras with the same signature τ. If there is a surjective homomorphism h from $\mathbf{A}$ to $\mathbf{B}$, then $\mathbf{B}$ is called a *homomorphic image* of $\mathbf{A}$. The kernel $\{(a, b) \in A^2 \mid h(a) = h(b)\}$ of h is a congruence on $\mathbf{A}$. Any congruence of $\mathbf{A}$ gives rise to a *quotient algebra* of $\mathbf{A}$, denoted by $\mathbf{A}/\theta$, whose domain A/θ consists of the equivalence classes of θ, and which has the same signature as $\mathbf{A}$ so that $f^{\mathbf{A}/\theta}(a_1/\theta, \ldots, a_n/\theta) = f^{\mathbf{A}}(a_1, \ldots, a_n)/\theta$, for every $f \in \tau$ and all $a_1, \ldots, a_n \in A$; here, a_i/θ is the equivalence class of θ containing a_i.

A class $\mathcal{V}$ of algebras with the same signature is called a *pseudovariety* if $\mathcal{V}$ contains all homomorphic images, subalgebras, and finite direct products of algebras in $\mathcal{V}$. The smallest pseudovariety containing $\mathbf{A}$ is called the pseudovariety *generated* by $\mathbf{A}$, and denoted by $\mathcal{V}(\mathbf{A})$. The relevance of pseudovarieties in constraint satisfaction comes from the following fact, which is a consequence of Theorems 5.5.6 and 5.5.15 in [4].

▶ Proposition 2.4. Let Γ and Δ be ω-categorical structures. If there exists an algebra $\mathbf{A}$ in $\mathcal{V}(\text{Alg}(\Gamma))$ with the same domain as Δ and whose operations are polymorphisms of Δ, then there is for every finite reduct Δ' of Δ a finite reduct Γ' of Γ such that $\text{CSP}(\Delta')$ reduces to $\text{CSP}(\Gamma')$ in polynomial time.

The following lemma is a consequence that will be used several times.

▶ **Lemma 2.5.** *Let Γ be an ω-categorical structure such that $\mathcal{V}(\text{Alg}(\Gamma))$ contains a two-element algebra all of whose operations are projections. Then Γ has a finite reduct Γ' such that $\text{CSP}(\Gamma')$ is NP-hard.*

Proof. Suppose $\mathcal{V}(\text{Alg}(\Gamma))$ contains an algebra $\mathbf{A}$ with domain $\{0, 1\}$ all of whose operations are projections. The operations in $\mathbf{A}$ preserve the relation $R = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$. The problem $\text{CSP}((\{0, 1\}; R))$ is known under the name positive 1-IN-3-3SAT, and NP-hard [16]. Now the statement follows from Proposition 2.4. □

We remark that the two cases in Conjecture 1.2 are always disjoint; this follows along the lines of the proof of Proposition 5.6.9 and 5.6.10 in [4]; we state it here for easy reference.

▶ Proposition 2.6. Let Γ be an ω-categorical model-complete core with a polymorphism f and an automorphism α satisfying $\forall x_1, \ldots, x_n.\, f(x_1, \ldots, x_n) = \alpha(f(x_2, \ldots, x_n, x_1))$. Then for every expansion Δ of Γ by constants, every algebra in the pseudovariety generated by the polymorphism algebra of Δ contains an operation that is not a projection.

2.3 . . . from Ramsey Theory

We use Ramsey theory to show that polymorphisms must *behave canonically* on large parts of the domain; canonical behavior will be introduced below. A wider introduction to canonical operations can be found in [8] and [4]; the definitions we present here are tailored towards applications for equivalence constraint satisfaction problems.

▶ **Definition 2.7.** Let Γ and Δ be structures over the same domain D. A *behavior* of a binary operation $f \colon D^2 \to D$ on $S \subseteq D$ is a partial function that sends a pair of orbitals (O_1, O_2) of $\text{Aut}(\Gamma)$ to an orbital O_3 of $\text{Aut}(\Delta)$ such that for all $(a_1, a_2) \in S^2 \cap O_1$ and $(b_1, b_2) \in S^2 \cap O_2$ we have $(f(a_1, a_2), f(b_1, b_2)) \in O_3$. A behavior is *canonical* if it is a total function. An operation $f \colon D^2 \to D$ is canonical on S as a function from Γ^2 to Δ if it has a canonical behavior on S. If a behavior of f on S sends (O_1, O_2) to O_3, then we write

$f(O_1, O_2) =_S O_3$. *Canonical unary operations* and their behavior are defined analogously. An operation f *behaves as* an operation g on S if they share the same behavior on S. In these definitions, we might omit to specify S in case that $S = D$.

Let Γ, Δ be finite τ-structures. We write $\binom{\Delta}{\Gamma}$ for the set of all substructures of Δ that are isomorphic to Γ. When Γ, Δ, Θ are τ-structures, then we write $\Theta \to (\Delta)^{\Gamma}_{r}$ if for all colorings $\chi \colon \binom{\Theta}{\Gamma} \to \{1, \ldots, r\}$ there exists $\Delta' \in \binom{\Theta}{\Delta}$ such that χ is constant on $\binom{\Delta'}{\Gamma}$.

▶ **Definition 2.8.** A class of finite relational structures $\mathcal{C}$ that is closed under isomorphisms and substructures is called *Ramsey* if for all $\Gamma, \Delta \in \mathcal{C}$ and for every finite $k \geq 1$ there exists a $\Theta \in \mathcal{C}$ such that $\Theta \to (\Delta)^{\Gamma}_{k}$.

A structure Γ is called *Ramsey* if the class of all finite structures that embed into Γ is Ramsey. A structure is called *ordered* if it carries a binary relation that denotes a linear order on its domain. When Γ is Ramsey and ordered, then the following theorem allows us to work with canonical polymorphisms of the expansion of Γ by constants.

▶ **Theorem 2.9** ([10]). *Let Γ be a homogeneous ordered Ramsey structure with finite relational signature and domain D. Let $c_1, \ldots, c_m \in D$, and let $f \colon D^2 \to D$ be any operation. Then $\{f\} \cup \mathrm{Aut}((\Gamma, c_1, \ldots, c_m))$ locally generates an operation that is canonical as a function from $(\Gamma, c_1, \ldots, c_m)^2$ to Γ, and which is identical with f on all tuples containing only values from $c_1, \ldots, c_m$.*

3 Equivalence Constraint Satisfaction Problems

We consider structures Γ with a first-order definition in $(D; Eq)$, where D is a countably infinite domain and Eq is an equivalence relation on D with infinitely many infinite equivalence classes. In the following, such structures Γ are called *equivalence constraint languages*.

We define $E(x, y) := Eq(x, y) \wedge x \neq y$ and $N(x, y) := \neg Eq(x, y)$. Note that $Eq(x, y)$ has the primitive positive definition $\exists z (E(x, z) \wedge E(z, y))$ over $(D; E)$, and it follows in particular that every operation that preserves E also preserves Eq.

▶ **Example 3.1.** An example of an equivalence constraint language is $\Gamma := (D; \{(x, y, z) \mid E(x, y) \vee N(y, z)\})$; it follows from our classification result (Corollary 7.5) that CSP(Γ) is in P. On the other hand, consider $\Delta := (D; R)$ where $R = \big\{(x, y, z) \mid (Eq(x, y) \vee Eq(y, z)) \wedge (N(x, y) \vee N(y, z))\big\}$. It follows from Corollary 7.5 that CSP(Δ) is NP-complete.

When R_1, R_2 are binary relations over D and $a = (a_1, a_2) \in D^2$ and $b = (b_1, b_2) \in D^2$, we write $a\binom{R_1}{R_2}b$ to denote that $R_1(a_1, b_1)$ and $R_2(a_2, b_2)$.

▶ Observation 3.2. Let $f \colon D^2 \to D$ be a binary function that preserves E, and let $a, b, c \in D^2$ such that $a\binom{E}{=}b$ and $b\binom{=}{E}c$. Then $E(f(a), f(b))$ or $E(f(b), f(c))$.

Proof. Since f preserves Eq, we have $Eq(f(a), f(b))$ and $Eq(f(b), f(c))$. Since $a\binom{E}{E}c$ and f preserves E, we have $E(f(a), f(c))$. Thus, $f(a) \neq f(b)$ or $f(b) \neq f(c)$, which proves the statement. ◻

It is easy to see that $(D; Eq)$ is a homogeneous structure and therefore ω-categorical. Every structure with a first-order definition in an ω-categorical structure is again ω-categorical (see e.g. [18]); thus, all equivalence constraint languages are ω-categorical. All equivalence constraint languages are preserved by the automorphisms of $(D; Eq)$, and we make the following convention: a set of operations F *generates* an operation g if $F \cup \mathrm{Aut}((D; Eq))$ locally generates g (see Section 2.2). Moreover, we say that f generates g if $\{f\}$ generates g.

	id	const	e_{EE}	e_{NN}	$e_{N=}$
=	=	=	=	=	=
N	N	=	E	N	N
E	E	=	E	N	=

Figure 1 Canonical unary behaviors.

A linear order $<$ on $(D; Eq)$ is *convex* if for all $a < b < c$ in D, if $(a, c) \in Eq$, then $(a, b) \in Eq$. Expansions $(D; Eq, <)$ of $(D; Eq)$ by a convex linear order $<$ are Ramsey (see [19], Corollary 6.8).

An important subclass of equivalence constraint languages is the class of *equality constraint languages*, i.e., structures with a first-order definition over $(D; =)$. We will use the following theorem, which is due to [5], in a formulation from [4] (a combination of Theorem 6.3.3 and 5.5.18 in [4]). Note that a countably infinite relational structure is isomorphic to an equality constraint language if and only if it is preserved by all permutations of its domain [5].

▶ **Theorem 3.3** (of [5]). *Let Γ be an equality constraint language. Then exactly one of the following two cases applies.*

- *Γ is a model-complete core and $\mathcal{V}(\mathrm{Alg}(\Gamma))$ contains a two-element algebra whose operations are projections. In this case, Γ has a finite reduct Γ' such that $\mathrm{CSP}(\Gamma')$ is NP-complete.*
- *Γ has a binary polymorphism f and an automorphism α satisfying $\forall x, y. f(x, y) = \alpha(f(y, x))$; in fact, f can be chosen to be either constant or injective. Moreover, for all finite reducts Γ' of Γ the problem $\mathrm{CSP}(\Gamma')$ is in P.*

4 Endomorphisms

In this section we show that if an equivalence constraint language Γ has an endomorphism that violates E or N, then Γ also has one out of five canonical endomorphisms described in the following. This result will be an important first step in our complexity classification, as we will see in Section 5.

The five mentioned behaviors of canonical unary operations are denoted by *id*, *const*, e_{EE}, e_{NN}, and $e_{N=}$ and presented in Figure 1. For example, we require that $e_{EE}(N) = E$ and $e_{EE}(E) = E$. It is clear that for each of those five behaviors there exists a function from $D \to D$ with this behavior. We also use the symbols *id*, *const*, e_{EE}, e_{NN}, and $e_{N=}$ to denote a function with the respective behavior; since any two functions who have the same of these behaviors generate each other, the precise choice of those functions will not be important.

To prove the main result of this section, Theorem 4.5, we use Ramsey theory via Theorem 2.9 as follows. When e violates E or N, then there are $c_1, c_2 \in D$ such that $E(c_1, c_2)$ and $\neg E(e(c_1), e(c_2))$, or $N(c_1, c_2)$ and $\neg N(e(c_1), e(c_2))$. Let $<$ be a convex linear order on D such that $c_1 < c_2$; as mentioned before, $(D; Eq, <)$ is Ramsey. By Theorem 2.9, the operation e generates an operation f that is canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq, <)$ (and hence also canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$) and still violates E or N. We say that f has *behavior B between two points* $x, y \in D$ if f has behavior B on $\{x, y\}$.

▶ **Lemma 4.1.** *Let $f : D \to D$ be canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$. If f behaves as the identity on all infinite orbits, and if it behaves as the identity between the constants c_1, c_2 and all other points, then it preserves N.*

Proof. Since f violates N we have that $Eq(f(c_1), f(c_2))$. Let c_3 be such that $E(c_1, c_3)$ and $N(c_2, c_3)$. Then $E(f(c_1), f(c_3))$ and $N(f(c_2), f(c_3))$, contradicting transitivity of *Eq*. □

▶ **Lemma 4.2.** *Let $f: D \to D$ be canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$. If f violates E and behaves as the identity on all infinite orbits, and if it behaves as the identity between the constants c_1, c_2 and all other points, then it generates $e_{N=}$.*

Proof. Since f violates E we have $E(c_1, c_2)$ and either $f(c_1) = f(c_2)$ or $N(c_1, c_2)$. We first show that the second case is impossible. There is c_3 such that $E(c_1, c_3)$ and $E(c_2, c_3)$. Hence, $E(f(c_1), f(c_3))$, $E(f(c_2), f(c_3))$, and $N(f(c_1), f(c_2))$, contradicting transitivity of *Eq*.

So $f(c_1) = f(c_2)$, and in particular f preserves *Eq*. We show by local closure that f generates $e_{N=}$. Let F be a finite subset of D. Let e be an operation generated by f such that the cardinality k of the set $\{(x, y) \in E \cap F^2 \mid e(x) = e(y)\}$ is maximal. If $k = |E \cap F^2|$, then e behaves on F as $e_{N=}$ and we are done. Otherwise, suppose there is $(x, y) \in E \cap F^2$ such that $e(x) \neq e(y)$. Since f and therefore e preserve *Eq*, we must have $E(e(x), e(y))$. Let α be an automorphism of $(D; Eq)$ that maps $(e(x), e(y))$ to (c_1, c_2). Then the mapping $e' := f \circ \alpha \circ e$ maps x and y to the same element, and $\{(x, y) \in E \cap F^2 \mid e'(x) = e'(y)\} > k$, contradicting the choice of e. □

We now analyze canonical behavior of injective functions in the case without constants.

▶ **Lemma 4.3.** *Let $f: D \to D$ be canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$. Let S be an infinite orbit of $(D; Eq, <, c_1, c_2)$ that induces a copy of $(D; Eq)$. If f does not behave as the identity on S, then it generates e_{EE}, $e_{N=}$, e_{NN}, or a constant operation.*

Proof. Since all orbitals of $(D; Eq)$ are symmetric, the unary operation f is canonical as a function from $(D; Eq, <)$ to $(D; Eq)$ if and only if it is canonical as a function from $(D; Eq)$ to $(D; Eq)$. If f does not behave as the identity on S, then f violates E or N on S. If f violates N, then either $f(N) =_S (=)$ or $f(N) =_S E$. In the first case we must have $f(E) =_S (=)$, and f is constant on S. Since S induces a copy of $(D; Eq)$, it follows by local closure that f generates a constant operation. So suppose that $f(N) =_S E$. If $f(E) =_S E$ then f behaves as e_{EE} on S, and therefore generates e_{EE}. The case that $f(E) =_S N$ is impossible, since for u, v, w with $N(u, v)$, $N(u, w)$, $E(v, w)$ this would imply $E(f(u), f(v))$, $E(f(u), f(w))$, $N(f(v), f(w))$, contradicting transitivity of *Eq*.

So suppose that f preserves N but violates E on S. If $f(E) =_S (=)$ then f behaves as $e_{N=}$ on S and therefore generates $e_{N=}$. Otherwise, $f(E) =_S N$; in this case f behaves as e_{NN} on S, and therefore generates e_{NN}. □

Next, we analyze canonical behavior of operations in the presence of two constants.

▶ **Lemma 4.4.** *Let $c_1, c_2 \in D$ be constants and let $f: D \to D$ be canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$. Let O be an infinite orbit of $(D; Eq, <, c_1, c_2)$. If f does not behave as the identity on O, or if it does not behave as the identity between one of c_1, c_2 and a point from O, then it generates e_{EE}, $e_{N=}$, e_{NN}, or a constant operation.*

Proof. Let P be an orbit of $(D; Eq, <, c_1, c_2)$ that induces a copy of $(D; Eq)$ in $(D; Eq)$. We assume that f behaves as the identity on P; otherwise, we are done by Lemma 4.3. Between any $u \in D \setminus P$ and any $v \in P$, f must behave as the identity. To see this, observe that necessarily $N(u, v)$. Suppose that $u < v$; the case that $v < u$ is analogous. Suppose for contradiction that $Eq(f(u), f(v))$. Pick a $v' \in P \setminus \{v\}$ such that $N(u, v')$ and $N(v, v')$. Then $u < v'$ because $<$ is convex. Since f is canonical we have $Eq(f(u), f(v'))$. Since f behaves as the identity on P, we have $N(f(v), f(v'))$. This contradicts transitivity of *Eq*. We conclude that $N(f(u), f(v))$ and hence f behaves as the identity between any $u \in D \setminus P$ and $v \in P$.

First suppose that f does not behave as the identity on O. As we have observed above, we are done if O induces in $(D; Eq)$ a structure that is isomorphic to $(D; Eq)$. Otherwise, there exists a $c \in \{c_1, c_2\}$ such that $E(u, c)$ for all $u \in O$. Since f does not behave as the identity on O we have either $f(E) =_O (=)$ or $f(E) =_O N$. In the first case, by local closure f generates $e_{N=}$. In the second case, f generates e_{NN}, again by local closure.

Now suppose that f does not behave as the identity between one of the constants $c \in \{c_1, c_2\}$ and a point p from O. We have already shown in the first paragraph that we are done when O induces in $(D; Eq)$ a structure isomorphic to $(D; Eq)$. Therefore, $E(p, c)$ for all $p \in O$. If $f(E) =_O (=)$ then f generates $e_{N=}$, and if $f(E) =_O N$ then f generates e_{NN}. □

▶ **Theorem 4.5.** *Any $e\colon D \to D$ violating E or N generates e_{EE}, e_{NN}, $e_{N=}$, or a constant operation.*

Proof. Since e violates E or N, there are $c_1, c_2 \in D$ such that $E(c_1, c_2)$ and not $E(e(c_1), e(c_2))$, or $N(c_1, c_2)$ and $Eq(e(c_1), e(c_2))$. By Theorem 2.9, the operation e generates an operation f that is canonical as a function from $(D; Eq, <, c_1, c_2)$ to $(D; Eq)$ and still violates E or N. Then by Lemma 4.1 and by Lemma 4.2, either

- f generates $e_{N=}$, and we are done, or
- there is an infinite orbit O such that f does not behave as the identity on O, or
- there is an infinite orbit O such that f does not behave as the identity between one of the constants $c \in \{c_1, c_2\}$ and a point from O.

In the last two cases f generates e_{EE}, e_{NN}, $e_{N=}$, or a constant operation by Lemma 4.4. □

5 Hardness

This section has two parts: we first use the results from the previous section to show that we can focus on equivalence constraint languages where E and N are primitive positive definable. In the second part, we use Theorem 3.3 in two different ways to isolate two groups of first-order expansions of $(D; E, N)$ that have NP-hard CSPs, and correspond to Item 1 of Conjecture 1.2. This will be complemented in the next sections by the proof that the remaining first-order expansions of $(D; E, N)$ are preserved by a binary polymorphism f satisfying Item 2 of Conjecture 1.2, and correspond to polynomial-time tractable equivalence constraint satisfaction problems.

▶ **Lemma 5.1.** *Let Γ be first-order definable in $(D; Eq)$, and let Δ be the model-complete core of Γ. Then one of the following holds:*

- *the pseudovariety generated by the polymorphism algebra of Δ contains a two-element algebra all of whose operations are projections. In this case, there exists a finite reduct Γ' of Γ such that $\mathrm{CSP}(\Gamma')$ is NP-complete;*
- *Δ has a polymorphism f and an automorphism α satisfying $\forall x, y.\, f(x, y) = \alpha(f(y, x))$. In this case, for every finite reduct Γ' of Γ we have that $\mathrm{CSP}(\Gamma')$ is in P;*
- *both E and N have a primitive positive definition in Γ.*

Proof. Consider first the case that Γ has an endomorphism f that violates E or N. By Theorem 4.5 we obtain that f generates an operation e which is from $\{e_{EE}, e_{NN}, e_{N=}\}$ or a constant operation. By Theorem 2.2, e is an endomorphism of Γ. If e is constant, then we are in Case 2 and done. Otherwise, the structure Δ induced by the image of e in Γ is infinite and preserved by all permutations, and hence an equality constraint language. Moreover, Δ is a model-complete core of Γ and the statement follows directly from Theorem 3.3.

$\min_{(E,=)}$	$=$	E	N
$=$	$=$	E	?
E	E	E	?
N	?	?	N

$\min_{(N,Eq)}$	$=$	E	N
$=$	$=$	?	N
E	?	E	N
N	N	N	N

$\min_{(N,E,=)}$	$=$	E	N
$=$	$=$	E	N
E	E	E	N
N	N	N	N

Figure 2 Important behaviors: $\min_{(E,=)}$ (left), $\min_{(N,Eq)}$ (middle), and $\min_{(N,E,=)}$ (right).

Now suppose that the orbitals E or N of $(D; Eq)$ are preserved by all endomorphisms; in particular, they are preserved by all automorphisms, and hence form orbitals of $\mathrm{Aut}(\Gamma)$. By Theorem 2.3, E and N must be primitive positive definable. □

To classify first-order expansions of $(D; E)$ we use Theorem 3.3 in two different ways. The first way is via the following observation, whose proof we leave to the reader.

▶ Proposition 5.2. Let Γ be a first-order expansion of $(D; E)$. Then Eq is a congruence of $\mathbf{A} := \mathrm{Alg}(\Gamma)$ and the algebra $\mathbf{B} := \mathbf{A}/Eq$ contains all permutations of its domain.

Another way how Theorem 3.3 comes into play is as follows; again, the proof is straightforward and left to the reader.

▶ Proposition 5.3. Let Γ be a first-order expansion of $(D; E)$, and let $c \in D$ be arbitrary. Then for any $c \in D$, the set $\{d \in D \mid E(c, d)\}$ induces a subalgebra $\mathbf{B}$ of $\mathbf{A} := \mathrm{Alg}((\Gamma, c))$ that contains all permutations of its domain.

By combining those results we prove that either Γ satisfies Item 1 of Conjecture 1.2, or it has certain binary polymorphisms. Three important behaviors of binary operations, $\min_{(E,=)}$, $\min_{(N,Eq)}$, and $\min_{(N,E,=)}$ are depicted in Figure 2, which should be read analogously to Figure 1. For example, we require that $\min_{(N,Eq)}(N, E) = \min_{(N,Eq)}(E, N) = N$. The name of $\min_{(N,E,=)}$ comes from the observation that it equals the minimum operation with respect to the order $N < E < (=)$. The existence of operations with these behaviors follows from Proposition 6.1.

▶ Proposition 5.4. Let Γ be a first-order expansion of $(D; E, N)$, and let $c \in D$ be arbitrary. Then Γ is a model-complete core, and either $\mathcal{V}(\mathrm{Alg}((\Gamma, c)))$ contains a two-element algebra all of whose operations are projections, and Γ has a finite reduct Γ' such that $\mathrm{CSP}(\Gamma')$ is NP-hard, or Γ is preserved by

1. an operation f with the behavior $\min_{(E,=)}$ on $\{d \in D \mid E(c, d)\}$, and
2. an operation with the behavior $\min_{(N,Eq)}$.

Proof. Since Γ contains E and N, every endomorphism of Γ behaves as the identity. Hence, every endomorphism of Γ is locally generated by $\mathrm{Aut}(\Gamma)$. By Theorem 3.6.11 in [4], the structure Γ is a model-complete core.

The subalgebra $\mathbf{A}$ induced by $\{d \,|\, E(c, d)\}$ in $\mathbf{B} := \mathrm{Alg}((\Gamma, c))$ contains a unary function for each permutation of its domain (Proposition 5.3). Let Δ be the structure with the same domain as $\mathbf{A}$ that contains all the relations that are preserved by the operations in $\mathbf{A}$, and let $\mathbf{A}'$ be the polymorphism algebra of Δ. If $\mathcal{V}(\mathbf{A}')$ contains a two-element algebra all of whose operations are projections, then so does $\mathcal{V}(\mathbf{A})$ since every operation of $\mathbf{A}$ is also an operation of $\mathbf{A}'$. In this case, also $\mathcal{V}(\mathbf{B})$ contains this two-element algebra, and by Lemma 2.5 there is a finite reduct Γ' of Γ such that $\mathrm{CSP}((\Gamma', \{c\}))$ is NP-hard. Since Γ is a model-complete core, Theorem 2.1 shows that $\mathrm{CSP}(\Gamma')$ is NP-hard.

If $\mathcal{V}(\mathbf{A}')$ does not contain a two-element algebra all of whose operations are projections, then Theorem 3.3 implies that Δ has either a binary injective or constant polymorphism f. Since Γ contains the relations E and N, all its endomorphisms are injective, and therefore also the unary operations in $\mathbf{B}$, in $\mathbf{A}$, and in $\mathbf{A}'$ are injective. This implies that f is binary injective. Let τ be the signature of $\mathbf{A}$. Since the operations in $\mathbf{A}$ locally generate the operations in $\mathbf{A}'$ (Proposition 2.2), it follows that for every finite subset S of the domain of $\mathbf{A}$ there exists a $g \in \tau$ such that $g^{\mathbf{A}}$ behaves as f on S; since f is binary injective, $g^{\mathbf{B}}$ therefore has the behavior $\min_{(E,=)}$ on S as a function over Γ. By an easy compactness argument (see Lemma 3.1.8 in [4]), Γ has a polymorphism with the behavior $\min_{(E,=)}$ on all of $\{d \mid E(c,d)\}$, satisfying the condition of Item 1 in the statement.

The proof that Γ also has a polymorphism with behavior $\min_{(N,Eq)}$ is similar, based on the fact that Eq is a congruence of $\mathbf{C} := \mathrm{Alg}(\Gamma)$, and that the algebra $\mathbf{D} := \mathbf{C}/Eq$ contains all permutations of its domain. Similarly as above we argue that for every finite subset S of the domain of $\mathbf{D}$ there is an operation in $\mathbf{D}$ that behaves as $\min_{(N,Eq)}$ on the union of the classes of $\mathbf{D} = \mathbf{C}/Eq$ that correspond to elements in S (unless $\mathcal{V}(\mathbf{D})$ contains a two-element algebra all of whose operations are projections). As above, a compactness argument gives the existence of an operation in $\mathbf{D}$ that behaves as $\min_{(N,Eq)}$ on the entire domain of Γ. □

6 Tractability

In this section we show that equivalence CSPs that have a polymorphism with the behavior $\min_{(N,E,=)}$ can be solved in polynomial time.

▶ Proposition 6.1. There is a binary function f that is canonical as a function from $(D; Eq)^2$ to $(D; Eq)$ with the behavior $\min_{(N,E,=)}$. This function can be chosen such that there is an automorphism α of $(D; Eq)$ satisfying $\forall x, y.\, f(x,y) = \alpha(f(y,x))$.

Proof. Observe that the structure $(D; Eq)^2$ is again an equivalence relation on a countable set with infinitely many infinite classes, and by ω-categoricity there is an isomorphism i between $(D; Eq)^2$ and $(D; Eq)$. This isomorphism has the behavior $\min_{(N,E,=)}$. Let $\beta \colon D^2 \to D^2$ be defined by $(x,y) \mapsto (y,x)$. Then β is an automorphism of $(D; Eq)^2$, and $\alpha := i \circ \beta \circ i^{-1}$ is an automorphism of $(D; Eq)$ such that $i(x,y) = \alpha(i(y,x))$. □

The operation f whose existence is shown in Proposition 6.1 will also be denoted by $\min_{(N,E,=)}$, i.e., we use the same symbol for this operation and its behavior.

▶ Proposition 6.2. Let Δ be a structure that is first-order definable in $(D; Eq)$, with finite relational signature, and polymorphism $\min_{(N,E,=)}$. Then $\mathrm{CSP}(\Delta)$ can be solved in polynomial time.

Proof. We use Proposition 2.4, and Theorem 3.3. An operation f is called *essentially injective* if it can be obtained from an injective operation by adding dummy variables (that is, the function value of f does not depend on those additional arguments of f). Let $\mathbf{A}$ be an algebra with domain $\mathbb{N}$ whose operations are precisely the essentially injective operation over $\mathbb{N}$. It is easy to verify that the operations of $\mathbf{A}$ form a locally closed clone, and hence, by Proposition 2.2, there exists a relational structure Γ with polymorphism algebra $\mathbf{A}$. We will show there is an algebra $\mathbf{B}$ with domain D in the pseudovariety generated by $\mathbf{A}$ such that all operations of $\mathbf{B}$ preserve Δ. It then follows from Proposition 2.4 that there exists a finite signature reduct Γ' of Γ such that $\mathrm{CSP}(\Delta)$ reduces to $\mathrm{CSP}(\Gamma')$. Polynomial-time tractability of $\mathrm{CSP}(\Gamma')$ follows from Theorem 3.3.

The relation $C = \{((u_1,u_2),(v_1,v_2)) \in \mathbb{N} \mid u_1 = v_1\}$ is a congruence of $\mathbf{A}^2$, with infinitely many infinite congruence classes. Let b be any bijection between the congruence classes of C and the domain D of Δ, and let $\mathbf{B}$ be the homomorphic image of $\mathbf{A}^2$ with respect to the map b. We claim that every operation $f^{\mathbf{B}}$ of $\mathbf{B}$ preserves Δ. If there are $i_1, \dots, i_k \in \{1, \dots, n\}$ such that $f^{\mathbf{A}} \colon \mathbb{N}^n \to \mathbb{N}$ satisfies $f(x_1, \dots, x_n) = g(x_{i_1}, \dots, x_{i_k})$ for all $x_1, \ldots, x_n \in \mathbb{N}$, then it clearly suffices to verify the claim for g instead of f. Since $f^{\mathbf{A}}$ is essentially injective, we can therefore assume that $f^{\mathbf{A}}$ is injective. Then $f^{\mathbf{B}}(x^1, \ldots, x^n)$ behaves as $\min_{(N,E,=)}(x_1, \min_{(N,E,=)}(x_2, \ldots, \min_{(N,E,=)}(x_{n-1}, x_n) \ldots))$: indeed,

$$\begin{aligned} Eq(f^{\mathbf{B}}(x^1, \ldots, x^n), f^{\mathbf{B}}(y^1, \ldots, y^n)) &\Leftrightarrow f^{\mathbf{A}}(x_1^1, \ldots, x_1^n) = f^{\mathbf{A}}(y_1^1, \ldots, y_1^n) \\ &\Leftrightarrow x_1^i = y_1^i \text{ for all } i \leq n \\ &\Leftrightarrow Eq(x^i, y^i) \text{ for all } i \leq n \,. \end{aligned}$$

Since Δ is preserved by $\min_{(N,E,=)}$, it is also preserved by $f^{\mathbf{B}}$. □

7 Generating $\min_{(N,E,=)}$

In this section we show that when f has the behavior $\min_{(E,=)}$ on $\{d \in D \mid E(c,d)\}$ for some $c \in D$ and g has the behavior $\min_{(N,Eq)}$, then $\{f, g\}$ generates $\min_{(N,E,=)}$. Some of the proofs in this section have been omitted and can be found in the full version of the paper.

▶ **Lemma 7.1.** *Let Γ be a first-order expansion of $(D; E)$, and $c \in D$. Suppose that Γ has a polymorphism that behaves as $\min_{(E,=)}$ on $\{d \in D \,|\, E(c,d)\}$. Then Γ is also preserved by $\min_{(E,=)}$.*

In the proof of Lemma 7.1 we use the following lemma, which is inspired by similar statements in [9]. We remark that Item 2 in the statement below is formally unrelated to the notion of *independence* as studied in [15], but similar in spirit.

▶ **Lemma 7.2.** *Let Γ be a first-order expansion of $(D; E)$. Then the following are equivalent.*

1. *Γ has a polymorphism with the behavior $\min_{(E,=)}$.*
2. *For every primitive positive formula $\phi(x_1, \ldots, x_n)$ and $y_1, \ldots, y_4 \in \{x_1, \ldots, x_n\}$, when*
   - *$\phi(x_1, \ldots, x_n) \wedge E(y_1, y_2) \wedge y_3 = y_4$ and*
   - *$\phi(x_1, \ldots, x_n) \wedge y_1 = y_2 \wedge E(y_3, y_4)$*

   are satisfiable over Γ, then also $\phi(x_1, \ldots, x_n) \wedge E(y_1, y_2) \wedge E(y_3, y_4)$ is satisfiable over Γ.
3. *For every finite subset S of D, Γ has a polymorphism with the behavior $\min_{(E,=)}$ on S.*

▶ **Lemma 7.3.** *Let f be an operation with the behavior $\min_{(E,=)}$. Then f generates an operation with the behavior $\min_{(E,=)}$ that is canonical as a function from $(D; Eq)^2$ to $(D; Eq)$.*

▶ **Lemma 7.4.** *Let f and g be operations with the behavior $\min_{(N,Eq)}$ and $\min_{(E,=)}$, respectively. Then $\{f, g\}$ generates $\min_{(N,E,=)}$.*

Proof. By Lemma 7.3, we can assume that f is canonical as a function from $(D; Eq)^2$ to $(D; Eq)$. We will show that $h(x,y) := g(f(x,y), f(y,x))$ has the behavior $\min_{(N,E,=)}$. Consider arbitrary points $a = (a_1, a_2)$, $b = (b_1, b_2)$ in D^2. Because f and g preserve E and N, h also does, and hence $h(E,E) = E$ and $h(N,N) = N$. If $a\binom{E}{=}b$ or $a\binom{=}{E}b$, then because f has the behavior $\min_{(E,=)}$, we have both $E(f(a_1,a_2), f(b_1,b_2))$ and $E(f(a_2,a_1), f(b_2,b_1))$. Since g preserves E, we obtain that $E(h(a), h(b))$ and we are done in this case.

We now turn to the case where $a\binom{N}{Q}b$ and $Q \in \{E, =\}$, and show that $N(f(a_1,a_2), f(b_1,b_2))$ or $N(f(a_2,a_1), f(b_2,b_1))$. Assume the contrary. Let α be an automorphism of $(D; Eq)$ such that $\alpha(a_2) = a_1$. Then $(a_1,a_1)\binom{N}{Q}(b_1, \alpha(b_2))$ and $(a_1,a_1)\binom{Q}{N}(\alpha(b_2), b_1)$. By transitivity of Eq, we have that $(b_1, \alpha(b_2))\binom{N}{N}(\alpha(b_2), b_1)$. Since f is canonical as a function from $(D; Eq)^2$ to $(D; Eq)$, we have that $Eq(f(a_1,a_1), f(b_1,\alpha(b_2)))$ and $Eq(f(a_1,a_1), f(\alpha(b_2), b_1))$. Therefore, $Eq(f(b_1, \alpha(b_2)), f(\alpha(b_2), b_1))$ by transitivity of Eq. This contradicts the fact that f preserves

N. Thus we have proved that $N(f(a_1,a_2),f(b_1,b_2))$ or $N(f(a_2,a_1),f(b_2,b_1))$. Further, because g has the behavior $\min_{(N,Eq)}$, we obtain that $N(h(a),h(b))$.

The case where $a\binom{Q}{N}b$ for $Q \in \{E,=\}$ is symmetric. We have considered all the cases, and conclude that indeed h has the behavior $\min_{(N,E,=)}$. □

By combining Proposition 5.4, Proposition 6.2, and Lemma 7.4, we obtain the following.

▶ **Corollary 7.5.** *Let Γ be a first-order expansion of $(D;E,N)$. Then Γ is preserved by $\min_{(N,E,=)}$, and for every finite reduct Γ' of Γ the problem $\mathrm{CSP}(\Gamma')$ is in P, or Γ has a finite reduct Γ' such that $\mathrm{CSP}(\Gamma')$ is NP-hard.*

8 Conclusions and Future Work

We have shown that Conjecture 1.2 holds for all structures with a first-order definition over an equivalence relation with infinitely many infinite classes; moreover, the universal-algebraic dichotomy from Conjecture 1.2 corresponds in this case precisely to a complexity dichotomy of the corresponding constraint satisfaction problems. We obtain the following.

▶ **Theorem 8.1.** *For equivalence constraint languages Γ exactly one of the following holds:*

1. *There is an expansion Δ' of the model-complete core Δ of Γ by a constant such that $\mathrm{Alg}(\Delta')$ contains a two-element algebra whose operations are projections. In this case, for some finite reduct Γ' of Γ we have that $\mathrm{CSP}(\Gamma')$ is NP-complete.*
2. *The model-complete core Δ of Γ has a polymorphism f and an automorphism α satisfying $\forall x,y.\, f(x,y) = \alpha(f(y,x))$. In this case, $\mathrm{CSP}(\Gamma')$ is in P for every finite reduct Γ' of Γ.*

Proof. By Proposition 2.6, the two cases are mutually exclusive. By Lemma 5.1, we have that either Case 1 or Case 2 holds, or E and N are primitively positively definable over Γ. By Proposition 5.4 and Lemma 7.1 every first-order expansion Γ of $(D;E,N)$ is a model-complete core, and either satisfies Case 1 or is preserved by an operation f with the behavior $\min_{(E,=)}$ and an operation g with the behavior $\min_{(N,Eq)}$. Further, Lemma 7.4 implies that Γ, and in consequence every finite reduct Γ' of Γ, is preserved by $\min_{(N,E,=)}$. The tractability of each such Γ' follows from Proposition 6.2. By Proposition 6.1 there exists an automorphism α of $(D;Eq)$ satisfying $\forall x,y.\ \min_{(N,E,=)}(x,y) = \alpha(\min_{(N,E,=)}(y,x))$; hence, we are in Case 2. □

Theorem 8.1 classifies also a non-trivial class of structures that are first-order definable over Allen's Interval Algebra (see Section 1.1): recall that the structure $(\mathbb{I};R_E)$, where $R_E := \{((x,y),(u,v)) \mid x = u\}$, is isomorphic to $(D;Eq)$. In fact, we believe that the techniques of this paper can be applied to eventually classify the complexity of the CSP for all structures Γ with a first-order definition over Allen's Interval Algebra. A next step towards this goal might be to classify all such structures Γ that contain the relation R_E. In this case R_E is a congruence of $\mathrm{Alg}(\Gamma)$. Note that the quotient of $\mathrm{Alg}(\Gamma)$ by R_E, and all subalgebras corresponding to equivalence classes of R_E, contain all automorphisms of $(\mathbb{Q};<)$. Hence, the difference to the scenario of the present paper is that one might then have to use the results from [6] about first-order expansions of $(\mathbb{Q};<)$ instead of the dichotomy for equality constraint languages.

References

1 James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 Libor Barto and Marcin Kozik. New conditions for Taylor varieties and CSP. In *Proceedings of LICS*, pages 100–109, 2010.

3 Manuel Bodirsky. Cores of countably categorical structures. *Logical Methods in Computer Science*, 3(1):1–16, 2007.

4 Manuel Bodirsky. Complexity classification in infinite-domain constraint satisfaction. Memoire d'habilitation à diriger des recherches, Université Diderot – Paris 7. Available at arXiv:1201.0856, 2012.

5 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of CSR'06.

6 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):41 pp, 2009. An extended abstract appeared in the proceedings of STOC'08.

7 Manuel Bodirsky and Jaroslav Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.

8 Manuel Bodirsky and Michael Pinsker. Reducts of Ramsey structures. *AMS Contemporary Mathematics, vol. 558 (Model Theoretic Methods in Finite Combinatorics)*, pages 489–519, 2011.

9 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. In *Proceedings of STOC*, pages 655–664, 2011. Preprint of the long version available at arxiv.org/abs/1011.2894.

10 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. In *Proceedings of LICS*, pages 321–328, 2011. Preprint of full journal version available from `arxiv.org/abs/1012.2381`.

11 Andrei A. Bulatov. H-coloring dichotomy revisited. *Theoretical Computer Science*, 349(1):31–39, 2005.

12 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *Journal of the ACM*, 53(1):66–120, 2006.

13 Andrei A. Bulatov, Andrei A. Krokhin, and Peter G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.

14 Peter J. Cameron. *Oligomorphic Permutation Groups*. Cambridge University Press, Cambridge, 1990.

15 David Cohen, Peter Jeavons, Peter Jonsson, and Manolis Koubarakis. Building tractable disjunctive constraints. *Journal of the ACM*, 47(5):826–853, 2000.

16 Michael Garey and David Johnson. *A guide to NP-completeness*. CSLI Press, Stanford, 1978.

17 David Hobby and Ralph McKenzie. *The Structure of Finite Algebras*. AMS Memoir, 1988.

18 Wilfrid Hodges. *Model theory*. Cambridge University Press, 1993.

19 Alexander Kechris, Vladimir Pestov, and Stevo Todorcevic. Fraissé limits, Ramsey theory, and topological dynamics of automorphism groups. *Geometric and Functional Analysis*, 15(1):106–189, 2005.

20 Andrei A. Krokhin, Peter Jeavons, and Peter Jonsson. Reasoning about temporal relations: The tractable subalgebras of Allen's interval algebra. *Journal of the ACM*, 50(5):591–640, 2003.

21 Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

22 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC*, pages 216–226, 1978.

A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic

Johann Brault-Baron

Université de Caen/ENSICaen/CNRS - GREYC (UMR6072)
Bd Maréchal Juin 14050 Caen Cedex, France
Johann.Brault-Baron@unicaen.fr

Abstract

It is known that the data complexity of a Conjunctive Query (CQ) is determined *only* by the way its variables are shared between atoms, reflected by its hypergraph. In particular, Yannakakis [18, 3] proved that a CQ is decidable in linear time when it is α-acyclic, i.e. its hypergraph is α-acyclic; Bagan et al. [2] even state:

- *Any CQ is decidable in linear time iff it is α-acyclic. (under certain hypotheses)*

(By linear time, we mean a query on a structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\mathcal{S}|)$)

A natural question is: since the complexity of a *Negative* Conjunctive Query (NCQ), a conjunctive query where *all* atoms are negated, also only depends on its hypergraph, can we find a similar dichotomy in this case?

To answer this question, we revisit a result of Ordyniak et al. [17] — that states that satisfiability of a β-acyclic CNF formula is decidable in polynomial time — by proving that some part of their procedure can be done in linear time. This implies, under an algorithmic hypothesis (precisely: one cannot decide whether a graph is triangle-free in time $\mathcal{O}(n^2 \log n)$ where n is the number of vertices.) that is likely true:

- *Any NCQ is decidable in quasi-linear time iff it is β-acyclic.*

(By quasi-linear time, we mean a query on a structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$)

We extend the easiness result to *Signed* Conjunctive Query (SCQ) where *some* atoms are negated. This has great interest since using some negated atoms is natural in the frameworks of databases and CSP. Furthermore, it implies straightforwardly the following:

- *Any β-acyclic existential first-order query is decidable in quasi-linear time.*

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, H.2.4 Query processing

Keywords and phrases conjunctive query, hypergraph, β-acyclicity, data complexity, Davis-Putnam procedure.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.137

1 Introduction

This paper gives descriptive complexity results in a finite model theory framework. According to [16], "Finite model theory studies the expressive power of logic on finite relational structures." Here we emphasize the complexity aspect of expressive power, in relation with the considered (first-order) logic fragment.

A fragment of great interest is the primitive positive fragment, i.e. the set of sentences one can build using only atoms — relations between variables —, the conjunction $\wedge$ and the existential quantifier $\exists$. This fragment, also known as Conjunctive Queries (CQ), is fundamental in database theory (see [5] for example) and in Constraint Satisfaction Problems (CSP) (see [7]). While general queries (first-order sentences) are PSpace-complete in terms

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 137–151
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of combined complexity, conjunctive queries are “only” NP-complete in terms of combined complexity, or W[1]-Complete ([13]) in terms of parametrized complexity.

An important tractable class of conjunctive queries are the α-acyclic (or *acyclic*, for short) conjunctive queries. They are tractable in a very strong sense: Yannakakis [18] proved that an α-acyclic conjunctive query ϕ on a relational structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\phi|.|\mathcal{S}|)$; in particular, it is fixed-parameter linear, i.e. it has linear data complexity. Bagan et al.([2]) even states (partially) that α-acyclicity is a necessary condition of linear data complexity. Another result giving relevance to this class is that Gottlob et al. ([10]) proved α-acyclic CQ to be LogCFL-complete for combined complexity.

Acyclic CQ have met a great interest essentially because they are the basis of important tractable classes of queries: they have been extended to several notions of queries not-too-far from α-acyclicity. In particular, this notion was extended to bounded treewidth CQ — a notion of bounded distance from being a tree — leading to classification results, see [13]; and to the notion of bounded hyper-treewidth CQ[11], for example. These classes allow polynomial reduction to acyclic CQ.

Nevertheless, in a database context, polynomiality is not a sufficient notion of tractability: a quadratic dependency on the database size is often considered intractable. A more reasonable notion would be quasi-linear time $n(\log n)^{\mathcal{O}(1)}$ (see [14]), which we take as a definition of tractability. Notice that, according to this definition of tractability, tractable CQ are still exactly CQ that have an α-acyclic hypergraph.

One can naturally ask oneself which queries, besides α-acyclic CQ, are tractable. Notice that there are some obviously tractable queries that are not in CQ, in particular some sets operations; e.g., the query $\exists x\, R(x) \wedge \neg S(x)$ that can be interpreted as “is $R \setminus S$ non-empty?”. This example leads to a quite natural question: which extensions of CQ where *some* atoms are negated, which we call *Signed Conjunctive Queries* (SCQ), are tractable?

In order to answer this question, we investigate a simpler question: Which queries that are conjunctions of only negated atoms, that we call *Negative Conjunctive Queries* (NCQ), are tractable? The advantage of this simpler case is that we can use the same tool as the known CQ case: the notion of hypergraph. The hypergraph of a query is quite a simple object reflecting the way variables are shared between atoms. It is a widespread intuitive idea that the complexity of a CQ only depends on its hypergraph, and in fact it is easy to prove this also holds in the case of NCQ.

One might think that a NCQ can be immediately reduced to a CQ by complementing the relations. However, computing the complement is clearly not in FPT, and *a fortiori* not in quasi-linear time.

We will prove for this case a dichotomy similar to the one known for CQ: a NCQ is tractable iff it is β-acyclic, which means that its hypergraph is β-acyclic. We finally extend the easiness result by proving that β-acyclic SCQ are tractable.

Structure of this Paper

This paper is organized as follows:

- Section 1 introduces the main definitions and states the results;
- Section 2 refines a result from Ordyniak et al. [17] by proving Davis-Putnam resolution w.r.t. a variable that is a nest point is done in linear time;
- Section 3 proves the easiness result;
- Section 4, after introducing some technical points, establishes the hardness result.

2 Preliminaries and Results

We introduce here the respective statements of the two results with all necessary definitions.

2.1 Preliminaries

▶ **Definition 1** (sentence, structure, query, CQ, NCQ, SCQ). A *signature* σ is:

- a set of relation symbols,
- and an arity Ar that associates a number to each symbol R, denoted $\mathrm{Ar}\,(R)$.

A *σ-structure* $\mathcal{S}$ consists in associating a set of $\mathrm{Ar}(R)$-tuples to each of the relation symbols R of σ which is called the *interpretation of R in $\mathcal{S}$*, and denoted $R^{\mathcal{S}}$. Some relation symbols of arity 1 may be used in a particular way, and are then called *domain symbols*.

An *existential first-order sentence*, or *sentence* for short, is a usual existential first-order sentence where each variable has a *distinct* associated domain symbol D_i. More formally, a σ-sentence has the form $\exists x_1 \in D_1 \ldots \exists x_n \in D_n\ \psi$ where D_i is a domain symbol belonging to σ and ψ is a usual quantifier-free σ'-formula where σ' is σ without the $D_1, \ldots, D_n$ previously mentioned. In the whole document, ψ will refer to the quantifier-free part of ϕ.

We call query of a sentence ϕ, denoted $\mathrm{DecideQ}(\phi)$, the problem of, given a structure $\mathcal{S}$, deciding whether ϕ holds in $\mathcal{S}$. Depending on the quantifier-free formula ψ, we define classes of queries of interest:

- when $\psi = \bigwedge_i R_i\big(x_{i_0}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\big)$, $\mathrm{DecideQ}(\phi) \in \mathrm{CQ}$ the *existential Conjunctive Queries*;
- when $\psi = \bigwedge_i \neg R_i\big(x_{i_0}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\big)$, $\mathrm{DecideQ}(\phi) \in \mathrm{NCQ}$ the *existential Negative Conjunctive Queries*;
- when $\psi = \bigwedge_i \sigma_i R_i\big(x_{i_0}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\big)$, where σ_i is either $\neg$ or ϵ (nothing), $\mathrm{DecideQ}(\phi) \in \mathrm{SCQ}$ the *existential Signed Conjunctive Queries*;
- when ψ is unrestricted, i.e. written using any connectives ($\wedge$, $\vee$, $\rightarrow$, $\neg$, etc.) $\mathrm{DecideQ}(\phi) \in \mathrm{EQ}$ the *existential Queries.*

Considering the particular form of the first three classes, we may consider these formulas as conjunctions of so-called *conjuncts*, i.e. when ψ is in the form: $\psi = \bigwedge_i \sigma_i R_i\big(x_{i_0}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\big)$ each $C_i = \sigma_i R_i\big(x_{i_0}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\big)$ is a conjunct. When $\sigma(i)$ is ϵ (resp. $\neg$), we say C_i is a *positive* (resp. *negative*) *conjunct.*

▶ Remark. Defining sentences with distinguished domains symbols attached to variables is a bit unusual. Let us justify it briefly through a simple example. Consider:

$$\phi_1 = \exists x_1 \exists x_2 \exists x_3 \exists x_4\ R(x_1, x_2) \wedge R(x_3, x_2) \wedge R(x_1, x_4) \wedge R(x_3, x_4)$$
$$\phi_2 = \exists x_1 \exists x_2 \exists x_3 \exists x_4\ R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_4, x_1) \wedge R(x_3, x_4)$$

The sentences ϕ_1 and ϕ_2 obviously have the same hypergraph (see below) but $\mathrm{DecideQ}(\phi_1)$ is obviously easy (it consists in deciding whether a directed graph has at least one edge), while $\mathrm{DecideQ}(\phi_2)$ is presumably not as easy (it consists in deciding whether a directed graph has a circuit of size 4). Using our formalism guarantees that:

- the complexity of a CQ (or a NCQ) depends only on its hypergraph, and
- the easiness results (of α-acyclic CQ and β-acyclic SCQ) still hold with the usual definition, i.e. when the variables all have the same domain.

▶ **Definition 2** (complexity classes). These definitions are based on [12], see it for more details. We call QLIN (resp. LIN) the set of problems decidable in time $\mathcal{O}(n \log n)$ (resp. $\mathcal{O}(n)$) on a RAM machine where n is the size of the input. In particular, sorting is in LIN by a result of [1], and [12].

▶ Remark. This definition of qLin looks quite restrictive: $n(\log n)^{\mathcal{O}(1)}$ would be much more reasonable, as suggested by [14]. In fact, all our results hold in both cases. We chose the restrictive one in order to put the emphasis on the easiness result.

The main object of our discourse will be how the restriction of the way variables are shared in a formula allows easy decision. The way variables are shared can be described by a very simple yet powerful notion: the hypergraph.

▶ **Definition 3** (hypergraph, hypergraph of a query)**.** We call *hypergraph* $\mathcal{H}$ a set of non-empty sets, called the *edges* of $\mathcal{H}$. We call $\mathcal{V}(\mathcal{H})$ the union of its edges; the elements of $\mathcal{V}(\mathcal{H})$ are called the *vertices* of $\mathcal{H}$.

We write $\mathcal{H}(\phi)$ the hypergraph of the sentence ϕ defined as the set of variables sets appearing in atoms of ϕ: $\mathcal{H}(\phi) = \{\mathrm{Vars}\,(A) \mid A \text{ is an atom of } \phi\}$ where

$$\mathrm{Vars}\left(R_i(x_{i_1}, \ldots, x_{i_{\mathrm{Ar}(R_i)}})\right) = \{x_{i_1}, \ldots, x_{i_{\mathrm{Ar}(R_i)}}\}$$

▶ **Example 4.** If we consider the SCQ DecideQ(ϕ) where

$$\begin{cases} \phi = \exists x_1 \in D_1\, \exists x_2 \in D_2\, \exists x_3 \in D_3\, \exists x_4 \in D_4\; \psi \\ \psi = R_1(x_1,x_2) \wedge R_2(x_2,x_3) \wedge R_3(x_1,x_2,x_3) \wedge \neg R_4(x_4,x_3) \wedge \neg R_5(x_4,x_4) \end{cases}$$

then we have $\mathcal{H}(\phi) = \{\{x_1,x_2\},\{x_2,x_3\},\{x_1,x_2,x_3\},\{x_3,x_4\},\{x_4\}\}$.

In this case, $R_1(x_1,x_2)$ is a positive conjunct. By contrast, $R_5(x_4,x_4)$ is an atom but not a conjunct in this formula, while $\neg R_5(x_4,x_4)$ is a negative conjunct.

We will see that the complexity of DecideQ(ϕ) depends only on $\mathcal{H}(\phi)$. We now define the criterion that discerns easy queries from hard queries.

2.2 Acyclicity Notions

We will see some interesting properties of hypergraphs are several notions of acyclicity, which are different extensions of the (classical) graph acyclicity property.

▶ **Definition 5** (induced hypergraph, nest point)**.** Let $\mathcal{H}$ be a hypergraph. We define its induced hypergraph w.r.t. $S \subseteq \mathcal{V}(\mathcal{H})$, denoted $\mathcal{H}[S]$, as follows:

$$\mathcal{H}[S] = \{e \cap S \mid e \in \mathcal{H}\} \setminus \{\emptyset\}$$

We write $\mathcal{H}[\setminus S]$ as a shorthand for $\mathcal{H}\left[\mathcal{V}(\mathcal{H}) \setminus S\right]$. In particular, $\mathcal{H}[\setminus\{x\}]$ is the hypergraph obtained by weak vertex removal of x.

We say x is a *nest point* of $\mathcal{H}$ when for any two distinct edges e_1 and e_2 containing x, either $e_1 \subset e_2$ or $e_2 \subset e_1$. In other words, the set $\{e \in \mathcal{H} \mid x \in e\}$ is linearly ordered w.r.t. set inclusion.

Fagin's originally defined ([9]) α-acyclicity of a hypergraph $\mathcal{H}$, here denoted $\mathcal{A}_\alpha(\mathcal{H})$. We assume the reader is familiar with the notion of α-acyclicity, that is the most classical notion of acyclicity for hypergraphs. He also defined β-acyclicity as follows:

▶ **Definition 6.** We say that a hypergraph $\mathcal{H}$ is β-acyclic, denoted $\mathcal{A}_\beta(\mathcal{H})$, if each of its subsets, i.e. every $\mathcal{H}' \subseteq \mathcal{H}$, is α-acyclic.

This characterisation says that β-acyclicity is the “hereditary” closure of α-acyclicity.

We give a second characterisation of β-acyclicity. This inductive characterisation from [8], based on a result of [4], is useful for the algorithm we give for β-acyclic queries (easiness result).

▶ **Lemma 7** (β-acyclicity — inductive characterisation). *A hypergraph $\mathcal{H}$ is β-acyclic iff* either *it is empty,* or *such that:*

- *we can find $x \in \mathcal{V}(\mathcal{H})$ such that x is a nest point of $\mathcal{H}$* and
- *$\mathcal{H}[\setminus\{x\}]$ is also β-acyclic.*

We say the ordered list $(x_1, \dots, x_n)$ of the vertices of an hypergraph $\mathcal{H}$ is a Reverse Elimination Order *(*REO*) of $\mathcal{H}$ when, for all i in $\{1, \dots, n\}$, x_i is a nest point of $\mathcal{H}[\{x_1, \dots, x_{i-1}\}]$. By this characterisation, a hypergraph is β-acyclic iff it has a REO.*

Here is the third characterisation of β-acyclicity. It will be used to obtain our hardness result. This characterisation uses the notion of β-cycle defined here.

▶ **Definition 8.** A *chordless cycle* is a graph isomorphic to $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq k\} \cup \{\{x_k, x_1\}\}$ for some k. A β-cycle of a hypergraph $\mathcal{H}$ is a subset of some induced sub-hypergraph of $\mathcal{H}$ that is a chordless cycle:

$$C \text{ is a } \beta\text{-cycle of } \mathcal{H} \quad \Leftrightarrow \quad \exists S \subseteq \mathcal{V}(\mathcal{H})\ C \subseteq \mathcal{H}[S] \text{ and } C \text{ is a chordless cycle}$$

▶ **Lemma 9** (β-acyclicity as absence of β-cycles). *An hypergraph $\mathcal{H}$ is β-acyclic iff it does not have a β-cycle.*

2.3 Statement of the Results

With all these notations, we are now able to state our results:

▶ **Theorem 10** (dichotomy). *Under hypothesis that the presence of a triangle in a graph of n vertices cannot be decided in time $\mathcal{O}(n^2 \log n)$, we have:*

$$\forall \phi \in \text{NCQ} \quad \text{DecideQ}(\phi) \in \text{qLin} \quad \Leftrightarrow \quad \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. Lemma 21 proves the implication $\Leftarrow$ (easiness result) and Lemma 30 proves the implication $\Rightarrow$ (hardness result). ◀

▶ Remark. This is to be compared to the positive conjunctive queries dichotomy contained in [2]: for any $\phi \in \text{CQ}$ such that $\mathcal{H}(\phi)$ is 4-conformal,[1] we have:

$$\forall \phi \in \text{CQ} \qquad \text{DecideQ}(\phi) \in \text{Lin} \quad \Leftrightarrow \quad \mathcal{A}_\alpha(\mathcal{H}(\phi))$$

under hypothesis we cannot decide the presence of a triangle in a graph G in time $\mathcal{O}(|G|)$.

Notice that this result also holds if we replace Lin by qLin; in that case the hypothesis becomes "deciding the presence of a triangle in a graph is not in qLin."

▶ **Theorem 11** (easiness).

$$\forall \phi \in \text{EQ} \qquad \text{DecideQ}(\phi) \in \text{qLin} \quad \Leftarrow \quad \mathcal{A}_\beta(\mathcal{H}(\phi))$$

That is to say any β-acyclic existential first-order sentence is decidable in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$.

Let P be a property on hypergraphs. Under the previously mentioned complexity hypothesis (about the triangle problem), we have:

$$(\forall \phi \in \text{EQ}\ P(\mathcal{H}(\phi)) \Rightarrow \text{DecideQ}(\phi) \in \text{qLin}) \quad \Leftrightarrow \quad (\forall \mathcal{H}\ P(\mathcal{H}) \Rightarrow \mathcal{A}_\beta(\mathcal{H}))$$

This means: any property of the hypergraph of a query grants it quasi-linear decision time if and only if this property implies β-acyclicity.

[1] A hypergraph is said to be k-conformal when every clique of cardinality $\geq k$ is contained in an edge.

Proof. Put ϕ in Disjunctive Normal Form, distribute (existential) quantification over disjunctive clauses which is correct since $\exists x(A(x) \vee B(x)) \Leftrightarrow (\exists x A(x)) \vee (\exists x B(x))$. Each clause is a SCQ whose hypergraph is a subset of $\mathcal{H}(\phi)$, each clause is therefore a β-acyclic SCQ; by Lemma 21, it has a qLin decision time, therefore so has their disjunction ϕ.

Second point is a direct corollary of this easiness result (part $\Leftarrow$) and the hardness part of the NCQ dichotomy (part $\Rightarrow$). ◀

3 Davis-Putnam Resolution with respect to a Nest Point

This section gives an algorithmic result needed for the easiness result; it can be independently read for its own, or may be skipped by admitting it. This part consists in exploiting a particular property of a variable in a CNF formula to perform efficient Davis-Putnam resolution with respect to this variable.

3.1 Definition and Properties

▶ **Definition 12** (CNF, nest point of a CNF formula, Davis-Putnam resolution)**.** A *CNF formula* $F(x_1, \ldots, x_n)$ is a classic propositional formula on the variables $x_1, \ldots, x_n$ that is in Conjunctive Normal Form, i.e. $F(x_1, \ldots, x_n) = \bigwedge_i C_i$ where C_i are *clauses*, i.e. sub-formulae in the form $C_i = \bigvee_{j=1}^{n(i)} l_i^j$ where l_i^j are *literals*. Literals are either *positive* literals — variables taken in $\{x_1, \ldots, x_n\}$ — or *negative* literals — negated variables $\neg x_i$ where $x_i \in \{x_1, \ldots, x_n\}$. A formula in CNF can be thought of as a set of clauses, which are sets of literals.

We say a clause C *holds* a variable x when either $x \in C$ or $\neg x \in C$ (or both, but the clause is tautological in this case). The set of variables held by a clause C is denoted $\mathrm{Vars}\,(C) = \{x_i \mid x_i \in C \text{ or } \neg x_i \in C\}$. We write F_x the set of clauses of F holding a variable x. We say a variable x_i is a *nest point* of $F(x_1, \ldots, x_n)$ (see [17]) when for all C_1 and C_2 in F_{x_i}, either $\mathrm{Vars}\,(C_1) \subseteq \mathrm{Vars}\,(C_2)$ or $\mathrm{Vars}\,(C_2) \subseteq \mathrm{Vars}\,(C_1)$.

We define the resolvent of $F(x_1, \ldots, x_n)$ w.r.t. the variable x_1

$$\mathrm{Res}\,(F, x_1)\,(x_1, \ldots, x_n) = \left\{ C_1 \vee C_2 \;\middle|\; \begin{array}{c} (C_1 \vee x_1) \in F(x_1, \ldots, x_n) \text{ and} \\ (C_2 \vee \neg x_1) \in F(x_1, \ldots, x_n) \end{array} \right\}$$

The following results are from [17].

▶ **Lemma 13** (correctness of resolution, size of resolvent)**.** *The following propositions hold:*

- *the Davis-Putnam resolution is correct:*

$$\forall x_1, \ldots, x_n \,\mathrm{Res}\,(F, x_1)\,(x_1, \ldots, x_n) \;\Leftrightarrow\; \exists x_1\, F_{x_1}(x_1, \ldots, x_n)$$

- *if x_1 is a nest point of a formula F, then*

$$\mathrm{Res}\,(F, x_1) \;\; \textit{is a subset of} \; \{C \setminus \{x_1, \neg x_1\} \,|\, C \in F_{x_1}\}$$

Proof. Correctness of the resolution is both classical [6] and easy to see.

Let x_1 be a nest point of F. We just have to consider F_{x_1}. Since x_1 is a nest point of F_{x_1}, in F_{x_1} we can rename variables as $x_1, \ldots, x_n$ by growing size of the smallest clause containing them. Obviously, $x = x_1$, and all clauses C are such that $\mathrm{Vars}\,(C) = \{x_1, \ldots, x_k\}$ with $k \leq n$. When the computation is over, we just have to give their original names back to the variables.

Take any two clauses $C_1 \vee x$ and $C_2 \vee \neg x$ involved in resolvent computation. We know $\mathrm{Vars}\,(C_1) \subseteq \mathrm{Vars}\,(C_2)$ or the converse. Assume, w.l.o.g, $\mathrm{Vars}\,(C_1) \subseteq \mathrm{Vars}\,(C_2)$. If some variable is present with different signs in C_1 and C_2, then the resolvent is tautological. Assume this is not the case. Therefore $C_1 \vee C_2 = C_2$. ◀

3.2 Algorithm and Complexity

Here we prove the resolution can be done in linear time, which is our addition to the main result of Ordyniak et al.([17]).

▶ **Lemma 14** (resolvent computation). *Let $F(x_1, \dots, x_n)$ be a CNF sentence whose x_1 is a nest point. We can compute the resolvent of $F(x_1, \dots, x_n)$ w.r.t. x_1 in time $\mathcal{O}(|F|)$.*

Proof. The main algorithmic point consists in adopting a handy representation of the formula. Like in proof of Lemma 13, since x_1 is a nest point of F, we can rename variables in F_{x_1} by growing size of the smallest clause containing it, therefore all clauses C are such that $\text{Vars}\,(C) = \{x_1, \dots, x_k\}$ with $k \leq n$. Clauses can therefore be encoded as *words* over the alphabet $\{-, +\}$. For example, $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$ is encoded as $+--+$.

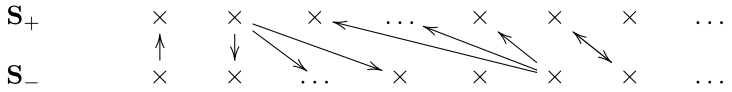
Let us consider subsumed clauses, i.e., clauses C such that we can find C' such that $C' \subseteq C$. With this encoding, C_1, encoded be w_1, subsumes C_2, encoded by w_2, if and only if w_1 is a prefix of w_2. Getting rid of subsumed clauses is done by applying the following simple algorithm, where L is the list of words encoding the set of clauses.

```
RemovePrefixed($L$):
    Sort lexicographically $L$
    $n \leftarrow 1$
    for $i$ from 2 to Card$(L)$ :
        if $L[n]$ not prefix of $L[i]$ :
            $L[n+1] \leftarrow L[i]$
            $n \leftarrow n+1$
    $L[n+1] \leftarrow$ End-Of-List
```

It is easy to see that this algorithm is correct, i.e. eliminates (in-place) any word that is prefixed by another. Sorting in lexicographical order, here denoted $<$, can be done in linear time using algorithm 3.2 page 80 in [1], see also definition 2. Therefore previous algorithm is linear.

In order to compute the resolvent of F with respect to x_1, we just need to consider pairs of clauses such that one is in the form $+\dots$ and the other in the form $-\dots$. Let us construct the lexicographically ordered sub-lists S_- and S_+ of words beginning with $-$, resp. $+$, without their leading $-$, resp. $+$. Two clauses C_1 and C_2 respectively encoded by words w_1 and w_2 have a non-tautological resolvent on x_1 if and only if $w_1 \in S_-$ and $w_2 \in S_+$ (or the converse), and w_1 is a prefix of w_2: in this case, their "resolvent" is w_2 itself, i.e. $C_2 \smallsetminus \{x_1\}$.

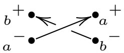
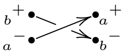
Algorithm 1, where S_- (resp. S_+) is represented by a sorted list L_1 (resp. L_2) of n_1 (resp. n_2) elements, is a variant of the fusion algorithm of two sorted lists. It is obviously linear. In order to prove Algorithm 1 is correct, we consider the bipartite directed graph where $a \rightarrow b$ means a is a prefix of b, and where words are represented in growing lexicographical order (from left to right).



The constraints of this graph are given by the assertions 1-4 given below. As a preliminary, we let the reader convince himself the following fact holds: if a word a is a prefix of a word b, then for all words w and W such that $w < a < W$ and none is a prefix of another, $w < b < W$.

```
ComputeResolvent($S_-$, $S_+$):
    $Res \leftarrow \emptyset$
    $i_- \leftarrow 1$; $i_+ \leftarrow 1$
    while $i_- < n_-$ and $i_+ < n_+$ :
        if $S_-[i_-]$ is a prefix of $S_+[i_+]$ :
            $Res \leftarrow Res \cup \{S_+[i_+]\}$
            Increment $i_+$
        elsif $S_+[i_+]$ is a prefix of $S_-[i_-]$ :
            $Res \leftarrow Res \cup \{S_-[i_-]\}$
            Increment $i_-$
        elsif $S_+[i_+] < S_-[i_-]$ :
            Increment $i_+$
        elsif $S_+[i_+] > S_-[i_-]$ :
            Increment $i_-$
    return $Res$
```

Algorithm 1 Resolvent computation.

1. The in-degree of the graph is at most one: assuming the contrary leads to $\exists u, v \in S_-$ (resp. S_+) with u prefix of v.
2. The graph has no path of length 2: exactly the same proof.
3. Edges do not cross each other. By symmetry, only two cases have to be considered:
   - Edges do not cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^- prefix of b^+ with $a^- < b^-$ and $a^+ > b^+$. Assume this is the case. a^- is prefix of a^+ and inferior and not prefix of b^-. Therefore $a^+ < b^-$. Since b^- is a prefix of b^+, $b^- < b^+$, therefore $a^+ < b^+$ (by the preliminary fact), contradictory.
   - Edges don't cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^+ prefix of b^- with $a^- < b^-$ and $a^+ > b^+$: essentially the same proof, but using the preliminary fact twice.
4. The neighbourhood of a vertex is contiguous, i.e. if a^- is a prefix of both b^+ and c^+, then for all $b^+ < d^+ < c^+$, a^- is a prefix of d^+.

All these facts together prove that no pair (a, b) where a is a prefix of b is "forgotten" by the given algorithm, which is therefore correct. ◀

4 Easiness Result

In this section, we prove that one can decide a β-acyclic SCQ with n variables on a structure $\mathcal{S}$ in time $\mathcal{O}(n|\mathcal{S}|\log|\mathcal{S}|)$. We prove it a bottom-up fashion: each step either reduces to or generalizes the previous one:

- first step uses results of the first section (Davis-Putnam resolvent computation w.r.t. a nest point in linear time) to decide β-acyclic NCQ-BoolD fast,
- second step generalizes it to β-acyclic SCQ-BoolD, and
- third step reduces β-acyclic SCQ to β-acyclic SCQ-BoolD.

For the sake of simplicity, we always assume a sentence is simple in the following sense: relation symbols all appear once, and in an atom, each variable appears at most once, and in any order of our convenience. This is justified by Corollary 27, page 149.

4.1 NCQ on the Boolean Domain

Here we make use of the results of previous section in order to get the announced complexity.

▶ **Definition 15** (NCQ-BoolD, SCQ-BoolD)**.** We say a NCQ (resp. SCQ) is *over the boolean domain* when all its domains are fixed to $\{0,1\}$ instead of being defined with domain symbols; we write it NCQ-BoolD (resp. SCQ-BoolD).

As an example, if $\phi = \exists x_1 \in D_1 \ldots x_n \in D_n\ \psi(x_1,\ldots,x_n)$ is a NCQ, then $\phi' = \exists x_1 \ldots x_n \in \{0,1\}^n \psi(x_1,\ldots,x_n)$ is a NCQ-BoolD.

The main result of this subsection, Lemma 17, gives an easiness result for NCQ-BoolD having a β-acyclic hypergraph. This result is obtained using inductively the following lemma jointly with the inductive characterisation of β-acyclicity by nest points.

▶ **Lemma 16** (NCQ-BoolD nest point)**.** *Let ϕ be a NCQ-BoolD, $\mathcal{S}$ a structure, and x a nest point of $\mathcal{H}(\phi)$. We can build another NCQ-BoolD ϕ' and another structure $\mathcal{S}'$ in time $\mathcal{O}(|\mathcal{S}|)$ (*independent *of $|\phi|$) such that* DecideQ$(\phi)(\mathcal{S}) \Leftrightarrow$ DecideQ$(\phi')(\mathcal{S}')$ *and* $\mathcal{H}(\phi') = \mathcal{H}(\phi)[\setminus\{x\}]$.

Sketch of proof. Take $\phi = \exists x_1 \ldots x_n \in \{0,1\}^n \psi(x_1,\ldots,x_n)$ with

$$\psi(x_1,\ldots,x_n) = \bigwedge_i \neg R_i\left(x_{f(i,1)},\ldots,x_{f(i,n(i))}\right)$$

The main point of the proof is transforming the assertion $\mathcal{S} \vDash \psi$ into an equivalent CNF formula, and then applying CNF nest point results. For all $(x_1,\ldots,x_n) \in \{0,1\}^n$, we have:

$$\begin{aligned}
\mathcal{S} \vDash \psi(x_1,\ldots,x_n) \quad &\Leftrightarrow \quad \bigwedge_i \neg R_i^{\mathcal{S}}\left(x_{f(i,1)},\ldots,x_{f(i,n(i))}\right)\\
&\Leftrightarrow \quad \bigwedge_i \bigwedge_{(a_1,\ldots,a_{n(i)})\in R_i^{\mathcal{S}}} \underbrace{\left(x_{f(i,1)},\ldots,x_{f(i,n(i))}\right) \neq \left(a_1,\ldots,a_{n(i)}\right)}_{C_i(a_1,\ldots,a_{n(i)})}\\
C_i(a_1,\ldots,a_{n(i)}) \quad &\Leftrightarrow \quad (x_{f(i,1)} \neq a_1) \vee \ldots \vee (x_{f(i,n(i))} \neq a_{n(i)})\\
&\Leftrightarrow \quad \sigma(a_1)x_{f(i,1)} \vee \ldots \vee \sigma(a_{n(i)})x_{f(i,n(i))}
\end{aligned}$$

where σ maps 1 to $\neg$ and 0 to ϵ. Finally, for all x_1, …, x_n we have the equivalence:

$$\mathcal{S} \vDash \psi(x_1,\ldots,x_n) \quad \Leftrightarrow \quad \underbrace{\bigwedge_i \bigwedge_{(a_1,\ldots,a_{n(i)})\in R_i^{\mathcal{S}}} \sigma(a_1)x_{f(i,1)} \vee \ldots \vee \sigma(a_{n(i)})x_{f(i,n(i))}}_{F(x_1,\ldots,x_n)}$$

Clearly, x is a nest point of $F(x_1,\ldots,x_n)$ (see the note at the end of previous definition). The transformation $(\phi,\mathcal{S}) \mapsto F$ is done in time $\mathcal{O}(|\mathcal{S}|)$ — i.e. linear time.[2] It is rather easy to see that the reverse transformation (from the propositional formula back to the corresponding NCQ-BoolD) can be done in linear time.

Lemma 13 states that $\mathrm{Res}\,(F,x)$ is a subset of F_x where x was removed, and that Davis-Putnam resolution is correct: $\forall x_1,\ldots,x_n\ \mathrm{Res}\,(F,x)\,(x_1,\ldots,x_n) \ \Leftrightarrow\ \exists x\ F_x(x_1,\ldots,x_n)$. Notice $x \in \{x_1,\ldots,x_n\}$. Furthermore, Lemma 14 states this can be done in linear time.

[2] Not exactly: the variables of the CNF are "bigger" than the 0/1 present in the structure. Nevertheless, when applying resolution, they will be encoded as signs — i.e. either + or −. In fact, the CNF form *explains* correctness but is not an actual step.

The following algorithm uses previous notations and is justified by the arguments above.

```
RemoveNestPoint($\phi$,$x$,$\mathcal{S}$):
    // $\psi$ is in the following form:  $\bigwedge_i \neg R_i \left(x_{f(i,1)}, \ldots, x_{f(i,n(i))}\right)$
    // Recall that $\sigma(1) = \neg$, $\sigma(0) = \epsilon$
    $F \leftarrow \bigwedge_i \bigwedge_{(a_1,\ldots,a_{n(i)}) \in R_i^{\mathcal{S}}} \sigma(a_1)x_{f(i,1)} \vee \ldots \vee \sigma(a_{n(i)})x_{f(i,n(i))}$
    Replace $\neg R(x_1, \ldots, x_n, x)$ by $\neg R(x_1, \ldots, x_n)$ in $\phi$
    $F \leftarrow Res(F, x)$
    Rebuild $\mathcal{S}$ from $F$
    return $(\phi,\mathcal{S})$
```

This algorithm clearly uses linear time, i.e. $\mathcal{O}(|\mathcal{S}|)$. ◀

We now state the NCQ easiness result. This lemma is stated and proved in order to give a simplified view of Lemma 19 below.

▶ **Lemma 17** (NCQ-BoolD easiness result). *A NCQ-BoolD ϕ with n variables such that $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}\left(n|\mathcal{S}|\right)$.*

Proof. The following does it:

```
DecideNCQ − BoolD($\phi$)($\mathcal{S}$):
    We know $(x_1, \ldots, x_n)$ is a REO of $\phi$.
    for $i$ from $n$ to 1 :
        $(\phi, \mathcal{S}) \leftarrow$ RemoveNestPoint($\phi$,$x_i$,$\mathcal{S}$)
        while $\phi$ contains some negative conjunct $\neg R()$ :
            if $R^{\mathcal{S}} \neq \emptyset$ :  return False
            Remove $\neg R()$ from $\phi$
    return True
```

Let us justify its correctness. By previous β-acyclicity characterisation, we know we can apply nest point removing until there is no variable left.

While nest points are inductively removed, some relations become of arity 0. The interpretation of these relations R is either empty — in this case, the assertion $\neg R()$ is a tautology and can be removed from the conjunction — or non-empty, i.e. contains only the empty tuple. In this case, the assertion $\neg R()$ is a contradiction, and the query should return "false" to mean this conjunction is not satisfiable (i.e. the sentence is not satisfied).

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

4.2 Easiness Result for SCQ on the Boolean Domain

We now refine previous result, instead of using it. It is barely more complicated.

Instead of proving easiness of general β-acyclic NCQ, we directly treat the case of SCQ. A first reason is that domains will be positive relations, therefore a NCQ is already a kind of SCQ; treating directly SCQ avoids treating domains specifically. The other reason is discussed after the following lemma.

This trick is a (weak) variant of the one presented in [19], and was inspired by it. It looks simple but is the key point allowing extension of the NCQ easiness result to wider classes.

▶ **Lemma 18.** *Let R_a^+ be a boolean relation of arity a. We can build in linear time the boolean relations R_{a-1}^+ of arity $a-1$ and R_a^- of arity a such that, for all $x_1, \ldots, x_a \in \{0,1\}^a$:*

- $R_a^+(x_1, \ldots, x_a) \Leftrightarrow R_{a-1}^+(x_1, \ldots, x_{a-1}) \wedge \neg R_a^-(x_1, \ldots, x_a)$
- $(x_1, \ldots, x_a) \in R_a^- \Rightarrow (x_1, \ldots, x_{a-1}) \in R_{a-1}^+$

```
DecideSCQ − BoolD($\phi$)($\mathcal{S}$):
    We know $(x_1, \dots, x_n)$ is a REO of $\phi$.
    for $i$ from $n$ to $1$ :
        while there is more than one positive conjunct containing $x_i$ in $\phi$ :
            Take $R(\bar{y}, \bar{z}, x_i)$ and $S(\bar{z}, x_i)$ among them
            $R^{\mathcal{S}} \leftarrow \left\{(\bar{a}, \bar{b}, c) \in R^{\mathcal{S}} \mid (\bar{b}, c) \in S^{\mathcal{S}}\right\}$
            Remove $S(\bar{z}, x_i)$ from $\phi$
        if $x_i$ in contained in a positive conjunct $R_a^+(\bar{y}, x)$ :
            Build $(R_{a-1}^+)^{\mathcal{S}}$, $(R_a^-)^{\mathcal{S}}$ from $(R_a^+)^{\mathcal{S}}$
            Replace $R_a^+(\bar{y}, x_i)$ by $R_{a-1}^+(\bar{y}) \wedge \neg R_a^-(\bar{y}, x_i)$ in $\phi$
            $(\phi, \mathcal{S}) \leftarrow$ RemoveNestPoint($\phi$,$x_i$,$\mathcal{S}$)
            $(R_{a-1}^+)^{\mathcal{S}} \leftarrow (R_{a-1}^+)^{\mathcal{S}} \smallsetminus (R_a^-)^{\mathcal{S}}$
        else
            $(\phi, \mathcal{S}) \leftarrow$ RemoveNestPoint($\phi$,$x_i$,$\mathcal{S}$)
        while $\phi$ contains a negative (resp. positive) conjunct $\neg R()$ (resp. $R()$) :
            if $R^{\mathcal{S}} \neq \emptyset$ (resp. $R^{\mathcal{S}} = \emptyset$) :
                return False
            Remove $\neg R()$ (resp. $R()$) from $\phi$
    return True
```

Algorithm 2 SCQ-BoolD decision algorithm.

Proof. Let $R_{a-1}^+ = \{(e_1, \dots, e_{a-1}) \mid \exists e_a R_a^+(e_1, \dots, e_a)\}$ and $R_a^- = \left(R_{a-1}^+ \times \{0,1\}\right) \smallsetminus R_a^+$. ◀

We could transform a SCQ-BoolD into a NCQ-BoolD, at the cost of an additional n factor, that would affect the complexity of general SCQ. However, by making a somewhat *ad hoc* algorithm, we can treat the SCQ-BoolD case *directly* and get the expected complexity $\mathcal{O}(n|\mathcal{S}|)$.

▶ **Lemma 19** (SCQ-BoolD easiness result)**.** *A SCQ-BoolD ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|)$.*

Proof. Notice the statements of the proof of Lemma 17 still hold. Apply Algorithm 2. If several positive conjuncts hold a given nest point, then we can proceed to filtering as follows: take any two positive conjuncts. The set of variables hold by one includes the set of variables held by the other, we can sort them in a lexicographical order such that the set of shared variables have the strong weight of the lexicographical order; we can finally proceed to filtering in linear time. β-acyclicity is preserved, and the REO is maintained by edge removal. In the end, there is only one positive conjunct holding the nest point.

Now the positive conjunct can be managed with Lemma 18, summed up in the following, where R/a means the relation R has arity a.

$$\begin{tikzcd}
 & (R_{a-1}^+)^{\mathcal{S}}/a-1 \arrow[drr] & & \\
(R_a^+)^{\mathcal{S}}/a \arrow[ur, "\text{Lemma 18}"] \arrow[dr] & & & (R_{a-1}^+)^{\mathcal{S}}/a-1 \\
 & (R_a^-)^{\mathcal{S}}/a \arrow[r, squiggly, "\text{Rem.NestP.}"] & (R_a^-)^{\mathcal{S}}/a-1 \arrow[ur, "\text{obvious}"] &
\end{tikzcd}$$

We build $(R_{a-1}^+)^{\mathcal{S}}$, $(R_a^-)^{\mathcal{S}}$ from $(R_a^+)^{\mathcal{S}}$ in linear time. After linear time resolvent computation, $(R_a^-)^{\mathcal{S}}$ has arity $a-1$ (the nest point has been removed) and we can proceed to a simplification in linear time justified by:

$$(R_{a-1}^+)^{\mathcal{S}}(x_1, \dots, x_{a-1}) \wedge \neg (R_a^-)^{\mathcal{S}}(x_1, \dots, x_{a-1}) \Leftrightarrow ((R_{a-1}^+)^{\mathcal{S}} \smallsetminus (R_a^-)^{\mathcal{S}})(x_1, \dots, x_{a-1})$$

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

4.3 Final Easiness Result

Here is the last (technical) result on hypergraphs we need.

▶ **Lemma 20.** *Let $\mathcal{H}_1$, $x \in \mathcal{V}(\mathcal{H}_1)$, $y \notin \mathcal{V}(\mathcal{H}_1)$ and $\mathcal{H}_2 = \{e \cup \{x, y\} \,|\, e \cup \{x\} \in \mathcal{H}_1\} \cup \{e \in \mathcal{H}_1 \,|\, x \notin e\}$. This means $\mathcal{H}_2$ is the same hypergraph as $\mathcal{H}_1$ except every edge holding x also holds y.*

We have: $(a_1, \dots, a_n, x, b_1, \dots, b_m)$ is a REO of $\mathcal{H}_1$ iff $(a_1, \dots, a_n, x, y, b_1, \dots, b_m)$ and $(a_1, \dots, a_n, y, x, b_1, \dots, b_m)$ are REOs of $\mathcal{H}_2$. In particular, $\mathcal{A}_\beta(\mathcal{H}_1) \Leftrightarrow \mathcal{A}_\beta(\mathcal{H}_2)$.

Proof. Easy considering the inductive definition of β-acyclicity (definition 7). ◀

▶ **Lemma 21** (SCQ easiness result)**.** *A SCQ ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}| \log |\mathcal{S}|)$.*

Proof. We transform a β-acyclic SCQ with n variables into a β-acyclic SCQ-BoolD having $n \log |\mathcal{S}|$ variables. Let ϕ be a β-acyclic SCQ admitting $(x_1, \dots, x_n)$ as a REO. Finding it takes a time *depending only on ϕ*.

We assume domains are reasonably encoded: we suppose there is a constant k such that, for any domain D_i, $\lceil \log_2 \max_i (D_i^{\mathcal{S}}) \rceil < k \log |\mathcal{S}|$. This assumption is reasonable: we always can sort (in linear time) the union of the domains and associate each element with its number in this order. Then we can re-encode (in-place) the whole structure in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$ by, for each occurring element, looking for its number, and replacing it by its number. Each element has therefore a size $\mathcal{O}(\log |\mathcal{S}|)$.

Let $s_i = \lceil \log_2 \max (D_i^{\mathcal{S}}) \rceil$. By the previous point, $s_i = \mathcal{O}(\log |\mathcal{S}|)$. We define

$$\phi' = \exists x_1 \in \{1, \dots, 2^{s_1}\} \dots \exists x_n \in \{1, \dots, 2^{s_n}\} \quad D_1(x_1) \wedge \dots \wedge D_n(x_n) \wedge \psi$$

We have $\mathcal{H}(\phi') = \mathcal{H}(\phi) \cup \{\{x_i\} \,|\, x_i \in \mathcal{V}(\mathcal{H}(\phi))\}$ therefore $\mathcal{H}(\phi')$ is β-acyclic. Now we can transform ϕ' into a SCQ-BoolD:

$$\phi'' = \exists (x_1^1, \dots, x_1^{s_1}, \dots, x_n^{s_n}) \in \{0,1\}^{\Sigma_i^n s_i} \quad D_1(x_1^1, \dots, x_1^{s_1}) \wedge \dots \wedge D_n(x_n^1, \dots, x_n^{s_n}) \wedge \psi'$$

with ψ' the same as ψ with every $R(x_a, x_b, \dots)$ replaced by $R(x_a^1, \dots, x_a^{s_a}, x_b^1, \dots, x_b^{s_b}, \dots)$. ϕ'' is a SCQ-BoolD, also β-acyclic due to Lemma 20. Moreover, $(x_1^1, \dots, x_1^{s_1}, \dots, x_n^1, \dots, x_n^{s_n})$ is a REO of $\mathcal{H}(\phi'')$. Noting $\Sigma_{i=1}^n s_i = \mathcal{O}(n \log |\mathcal{S}|)$ and applying Lemma 19 concludes. ◀

5 Hardness Result

We introduce a basic notion of reduction together with a trivial lemma that will be useful to prove Corollary 27, and makes simple the proof of the hardness result.

▶ **Definition 22** (linear reduction $\prec$)**.** We say a problem P_1 *reduces linearly* to a problem P_2, denoted $P_1 \prec P_2$, when we can find $f \in$ Lin such that, given an algorithm A_2 deciding P_2, $A_2 \circ f$ decides P_1 that is to say the following algorithm decides P_1:

```
Decide$P_1(I_1)$:
    $I_2 \leftarrow f(I_1)$
    return Decide$P_2(I_2)$
```

We also say P_2 *expresses* P_1. When P_1 both reduces linearly to and expresses linearly P_2, we say P_1 and P_2 are *equivalent*, denoted $P_1 \sim P_2$.

The following lemma is obvious:

▶ **Lemma 23** (linear reduction properties). *Linear reduction is transitive, i.e. if $P_1 \prec P_2$ and $P_2 \prec P_3$, then $P_1 \prec P_3$; and* $\textsc{qLin}$ *is closed by linear reduction, i.e. if $P_1 \prec P_2$ and $P_2 \in \textsc{qLin}$ then $P_1 \in \textsc{qLin}$.*

5.1 A Technical Point

In order to prove our hardness result on NCQ, we need to introduce a normalized form of NCQ; it is used to show that the complexity of a NCQ is exactly determined by its hypergraph.

▶ **Definition 24** (simple, sorted, normalized NCQ). One says that a NCQ $\phi = \exists x_1 \in D_1 \ldots \exists x_n \in D_n\ \psi$ is *simple* if each relation symbol occurs only once in ψ. ϕ is *sorted* if the tuple of variables of each atom of ψ occurs in increasing order of subscripts with no repetition, i.e. every atom of ψ is in the form $R(x_{i_1}, \ldots, x_{i_k})$ where $i_1 < \ldots < i_k$.

A NCQ is *normalized* if it is both simple and sorted.

The following lemma is obvious:

▶ **Lemma 25.** *Let ϕ_1, ϕ_2 be two normalized NCQ with the same signature and $\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2)$. Then ϕ_1 and ϕ_2 are identical up to relation symbols permutation. In particular* $\mathrm{DecideQ}(\phi_1) \sim \mathrm{DecideQ}(\phi_2)$.

▶ **Lemma 26.** *Any NCQ ϕ is equivalent to some normalized NCQ ϕ'' with $\mathcal{H}(\phi) = \mathcal{H}(\phi'')$, i.e.* $\mathrm{DecideQ}(\phi) \sim \mathrm{DecideQ}(\phi'')$.

Sketch of proof. Here we give the only non-trivial point of the proof, in an easy to generalize example. Let $\phi_1 = \exists x_1 \in D_1 \exists x_2 \in D_2 \exists x_3 \in D_3\ \neg R_1(x_1, x_2) \wedge \neg R_2(x_2, x_3)$ and ϕ_2 built by replacing *both* R_1 and R_2 by the same symbol R in ϕ_1. We prove $\mathrm{DecideQ}(\phi_1) \prec \mathrm{DecideQ}(\phi_2)$. We define, for $i \in \{1, 2, 3\}$, ${D_i}^{\mathcal{S}_2} = {D_i}^{\mathcal{S}_1} \times \{i\}$ and $R^{\mathcal{S}_2} = \left\{ ((a_1, 1), (a_2, 2)) \mid (a_1, a_2) \in {R_1}^{\mathcal{S}_1} \right\} \cup \left\{ ((a_2, 2), (a_3, 3)) \mid (a_2, a_3) \in {R_2}^{\mathcal{S}_1} \right\}$. The key point is that $R^{\mathcal{S}_2}$ is a *disjoint* union of relations corresponding to relations ${R_1}^{\mathcal{S}_1}$ and ${R_2}^{\mathcal{S}_1}$; further, the tuples originating from ${R_1}^{\mathcal{S}_1}$ (resp. ${R_2}^{\mathcal{S}_1}$) are *identified* by their specific form $((a_1, 1), (a_2, 2))$ (resp. $((a_2, 2), (a_3, 3))$). ◀

▶ **Corollary 27.** *For all NCQ ϕ_1 and ϕ_2, we have*

$$\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2) \quad \Rightarrow \quad \mathrm{DecideQ}(\phi_1) \sim \mathrm{DecideQ}(\phi_2)$$

Proof. Obvious corollary of Lemma 25 and Lemma 26. ◀

5.2 Hardness Result

▶ **Lemma 28.** *Let ϕ be a NCQ, and x a variable appearing in ϕ. Let ϕ' be the query obtained by removing every occurrence of x in ϕ, that is to say removing the $\exists x \in D_x$, and removing x in atoms where it occurs. Then* $\mathrm{DecideQ}(\phi)$ *expresses linearly* $\mathrm{DecideQ}(\phi')$.

Proof. By virtue of Corollary 27, we can assume that every relation appears once in ϕ, with associated variables in order. For simplicity, x is therefore assumed the minimal element for the order. To define the reduction, we will just define how a given relation R is transposed between $\mathcal{S}_1$ and $\mathcal{S}_2$. If x does not appear in the corresponding atom, there is no difference i.e. $R^{\mathcal{S}_2} = R^{\mathcal{S}_1}$.

In the other case, R appears as $R(x, x_{a(1)}, \ldots, x_{a(k)})$ in ϕ, with $a : \{1, \ldots, k\} \to \{1, \ldots n\}$. R appears as $R(x_{a(1)}, \ldots, x_{a(k)})$ in ϕ'. From now, completing the reduction is easy: define $R^{\mathcal{S}_2} = \big\{(1, s_1, \ldots, s_k) \,|\, (s_1, \ldots, s_k) \in R^{\mathcal{S}_1}\big\}$ and ${D_x}^{\mathcal{S}_2} = \{1\}$. ◀

▶ **Corollary 29.** *Let $\phi \in \mathrm{NCQ}$. For every ϕ' such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$ for some S, we have* $\mathrm{DecideQ}(\phi) \succ \mathrm{DecideQ}(\phi')$.

Proof. For every $x \in \mathcal{V}(\mathcal{H}(\phi)) \smallsetminus S$, apply Lemma 28, together with transitivity (Lemma 23). We have proved that $\mathrm{DecideQ}(\phi)$ expresses *some* $\mathrm{DecideQ}(\phi')$, such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$; apply Corollary 27 to prove equivalence of this last query with *all* queries having the same hypergraph; transitivity concludes. ◀

▶ **Lemma 30** (hardness result). *Under hypothesis that the problem of deciding the presence of a triangle in a graph on n vertices cannot be decided in $\mathcal{O}(n^2 \log n)$:*

$$\forall \phi \in \mathrm{NCQ} \quad \mathrm{DecideQ}(\phi) \in \textsc{qLin} \;\Rightarrow\; \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. For the sake of contradiction, assume $\mathrm{DecideQ}(\phi) \in \textsc{qLin}$ and $\neg\mathcal{A}_\beta(\mathcal{H}(\phi))$. This implies we can find $S \subseteq \mathcal{V}(\mathcal{H})$ and $h \subseteq \mathcal{H}(\phi)$ such that $h[S]$ is a chordless cycle. Then, by Corollary 29 and Corollary 27, $\mathrm{DecideQ}(\phi)$ expresses the following query of signature σ:

$$P = \mathrm{DecideQ}\left(\exists x_1 \in D_1 \ldots \exists x_k \in D_k \; \neg R_k(x_k, x_1) \wedge \bigwedge_{i=1}^{k-1} \neg R_i(x_i, x_{i+1})\right)$$

that, by Lemma 23, is also in $\textsc{qLin}$.

Now, let us associate to any graph $G = (V, E)$ with $\mathrm{Card}\,(V) = n$ a σ-structure defined as follows. For each R_i with $i > 3$, set ${R_i}^{\mathcal{S}} = \{(i, j) \in V^2 \,|\, i \neq j\}$. For each R_i with $i \leq 3$, set ${R_i}^{\mathcal{S}} = \{(i, j) \in V^2 \,|\, (i, j) \notin E\}$. Set ${D_i}^{\mathcal{S}} = V$. We have $|\mathcal{S}| = \mathcal{O}(|V|^2) = \mathcal{O}(n^2)$. If this query is in $\textsc{qLin}$, we can decide the presence of a triangle in G in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|) = \mathcal{O}(n^2 \log n^2) = \mathcal{O}(n^2 \log n)$. ◀

Concluding Remark

β-acyclic existential first-order queries have many qualities, they only lack one thing: to include α-acyclic CQ. This is to be addressed in a future paper.

Acknowledgments

The author expresses his gratitude to Étienne Grandjean for careful, multiple readings of the successive versions of this paper, despite his busy schedule, until any doubtful or unclear point had disappeared. Without his assistance, this paper would still be a confused set of sketchy notes.

The author would also like to thank anonymous referees for their quite detailed reviews that greatly contributed to improve the quality of the paper.

References

1 Aho, Hopcroft, and Ullman. *The design and analysis of computer algorithms.* 1974.

2 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. *Computer Science Logic*, 4646:208–222, 2007.

3 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

4 Andries E. Brouwer and Antoon W. J. Kolen. A super-balanced hypergraph has a nest point. *Technical report, Math. centr. report ZW146, Amsterdam*, 1980.

5 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *ACM Symp. on Theory of Computing*, pages 77–90, 1977.

6 Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

7 Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.

8 David Duris. Some characterizations of gamma and beta-acyclicity of hypergraphs. November 2008.

9 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.

10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

11 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

12 Etienne Grandjean. Sorting, Linear Time and the Satisfiability Problem. *Ann. Math. Artif. Intell.*, 16:183–236, 1996.

13 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 657–666, 2001.

14 Yuri Gurevich and Saharon Shelah. Nearly linear time. In Albert Meyer and Michael Taitslin, editors, *Logic at Botik '89*, volume 363 of *Lecture Notes in Computer Science*, pages 108–118. Springer Berlin / Heidelberg, 1989.

15 Phokion G. Kolaitis. Constraint satisfaction, databases, and logic. In *IJCAI*, pages 1587–1595, 2003.

16 Leonid Libkin. *Elements of Finite Model Theory*. Springer, 2004.

17 Sebastian Ordyniak, Daniel Paulusma, and Stefan Szeider. Satisfiability of Acyclic and Almost Acyclic CNF Formulas. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84–95, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

18 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings Conf. on Very Large Databases*, pages 82–94, 1981.

19 Bruno Zanuttini and Jean-Jacques Hébrard. A unified framework for structure identification. *Information Processing Letters*, 81(6):335 – 339, 2002.

On the equational consistency of order-theoretic models of the λ-calculus*

Alberto Carraro and Antonino Salibra

DAIS, Università Ca'Foscari Venezia
Via Torino 155, Venezia, Italy
{acarraro, salibra}@dsi.unive.it

Abstract

Answering a question by Honsell and Plotkin, we show that there are two equations between λ-terms, the so-called *subtractive equations*, consistent with λ-calculus but not satisfied in any partially ordered model with bottom element. We also relate the subtractive equations to the open problem of the order-incompleteness of λ-calculus.

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems

Keywords and phrases Lambda calculus, order-incompleteness, partially ordered models

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.152

1 Introduction

Lambda theories are congruences on the set of λ-terms, which contain β-conversion. They arise by syntactical or semantic considerations. Indeed, a λ-theory may correspond to a possible operational semantics of the lambda calculus, as well as it may be induced by a model of lambda calculus through the congruence relation associated with the interpretation function. The set of λ-theories is naturally equipped with a structure of complete lattice, whose bottom element is the least λ-theory $\lambda\beta$, and whose top element is the inconsistent λ-theory. The lattice of λ-theories is a very rich and complex structure of cardinality $2^{\aleph_0}$ (see, for example, [1, 9, 10]). Syntactical techniques are usually difficult to apply in the study of λ-theories. Therefore, semantic methods have been extensively investigated.

One of the most important contributions in the area of mathematical programming semantics was the discovery by D. Scott in the late 1960s, that complete partial orders, having their own function space as a retract, are models for the untyped lambda calculus. On the other hand, there are results that indicate that Scott's methods, based on a combination of order-theory and topology, may not in general be exhaustive: Honsell and Ronchi Della Rocca [8] have shown that there exists a λ-theory that does not arise as the theory of a Scott model. A natural completeness problem then arises for Scott semantics: whether any two λ-terms equal in all Scott models are β-convertible. This equational completeness problem is one of most outstanding open problems of λ-calculus and it seems to have appeared first in the literature in [6]. There is also an analogous consistency problem, raised by Honsell and Plotkin in [5]: whether every finite number of equations between λ-terms, consistent with the λ-calculus, has a Scott model. In this paper we answer negatively to this second question. We provide two equations (called the *subtractive equations*) consistent with λ-calculus, which have no partially ordered model with bottom element.

* The second author was partially supported by Fondation de Mathématique de Paris

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 152–166
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Although many familiar models are constructed by order-theoretic methods, it is also known that there are some models of the lambda calculus that cannot be non-trivially ordered (see [11, 12, 13]). In general, we define a combinatory algebra $\mathbf{A}$ to be *unorderable* if there does not exist a non-trivial partial order on A for which the application operation is monotone. Of course, an unorderable model can still arise from an order-theoretic construction, for instance as a subalgebra of some orderable model. The most interesting result has been obtained by Selinger [13], who, enough surprising, has shown that the standard open and closed term models of $\lambda\beta$ and $\lambda\beta\eta$ are unorderable. As a consequence of this result, it follows that if $\lambda\beta$ or $\lambda\beta\eta$ is the theory of a partially ordered model, then the denotations of closed terms in that model are pairwise incomparable, i.e. the term denotations form an anti-chain. This led Selinger [13] to study the related question of absolute unorderability: a model is absolutely unorderable if it cannot be embedded in an orderable one. Plotkin conjectures in [11] that an absolutely unorderable combinatory algebra exists, but the question is still open whether this is so. Selinger has given in [13] a syntactic characterisation of the absolutely unorderable algebras in any algebraic variety (equational class) in terms of the existence of a family of Mal'cev operators. Plotkin's conjecture is thus reduced to the question whether Mal'cev operators are consistent with the lambda calculus or combinatory logic. The question of absolute unorderability can also be formulated in terms of theories, rather than models. In this form, Selinger [13] refers to it as the *order-incompleteness* question: does there exist a λ-theory which does not arise as the theory of a non-trivial partially ordered model? Such a problem can be also characterised in terms of connected components of a partial ordering (minimal subsets which are both upward and downward closed): a λ-theory $\mathcal{T}$ is order-incomplete if, and only if, every partially ordered model, having $\mathcal{T}$ as equational theory, is partitioned in an infinite number of connected components, each one containing exactly one element. In other words, the partial order is the equality.

Toward an answer to the order-incompleteness problem, we find a strengthening $\mathcal{T}$ of the subtractive equations having the following property: every partially ordered model $\mathcal{M}$ satisfying $\mathcal{T}$ has an infinite number of connected components among which that of the looping term Ω is a singleton set. Moreover, each connected component of $\mathcal{M}$ contains the denotation of at most one $\beta\eta$-normal form. Compared to absolute unorderability, the above situation still has some missing bits. For example we are not in the position to tell where the denotations of all unsolvable λ-terms other than Ω are placed in the model. Same thing for all the solvable λ-terms which do not have a $\beta\eta$-normal form.

The inspiration for the subtractive equations comes from the notion of *subtractive variety* of algebras introduced by Ursini in [14]. A variety $\mathcal{V}$ of algebras is subtractive if it satisfies the following identities:

$$s(x,x) = 0; \qquad s(x,0) = x$$

for some binary term s and constant 0. Subtractive algebras abound in classical algebras. If we interpret the binary operator "s" as subtraction, and we use the infix notation, then we can rewrite the above identities as $x - x = 0$ and $x - 0 = x$. In the context of λ-calculus, the subtractive equations make a certain λ-term behave like a binary subtraction operator (in curryfied form) whose "zero" is the looping λ-term Ω.

In the last section of this paper we relativize to an element the notion of absolute unorderability. We say that an algebra $\mathbf{A}$ is 0-unorderable if, for every compatible partial order on $\mathbf{A}$, 0 is not comparable with any other element of the algebra. An algebra $\mathbf{A}$ in a variety $\mathcal{V}$ is absolutely 0-unorderable if, for any $\mathbf{B} \in \mathcal{V}$ and embedding $f : \mathbf{A} \to \mathbf{B}$, $\mathbf{B}$ is 0-unorderable. Generalising subtractivity to n-subtractivity ($n \geq 2$), we give a syntactic characterisation of the absolutely 0-unorderable algebras with Mal'cev-type conditions. The consistency of the two subtractive equations with λ-calculus implies the existence of absolutely Ω-unorderable combinatory algebras.

2 Preliminaries

2.1 Partial Orderings

Let $(A, \leq)$ be a partially ordered set (poset). Two elements a, b of A are: (1) *comparable* if either $a \leq b$ or $b \leq a$. A set $B \subseteq A$ is an *upward* (*downward*) closed set if $b \in B$, $a \in A$ and $b \leq a$ ($a \leq b$) imply $a \in B$.

We denote by $\approx_{\leq}$ the least equivalence relation on A containing $\leq$. A *connected component* of $(A, \leq)$ is an equivalence class of $\approx_{\leq}$. A connected component can be also characterised as a minimal subset of A which is both upward closed and downward closed. The poset $(A, \leq)$ is called *connected* if $\approx_{\leq}$ determines a unique equivalence class.

2.2 Lambda calculus

With regard to the λ-calculus we follow the notation and terminology of [1]. By Λ and Λ^o, respectively, we indicate the set of λ-terms and of closed λ-terms. By convention application associates to the left. The symbol $\equiv$ denotes syntactical equality. The following are some notable λ-terms: $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$; $\mathbf{I} \equiv \lambda x.x$; $\mathbf{K} \equiv \lambda xy.x$; $\mathbf{S} \equiv \lambda xyz.xz(yz)$.

If M is a λ-term and $\vec{P} \equiv P_1 \dots P_n$ is a sequence of λ-terms, we write $M\vec{P}$ for the application $MP_1 \cdots P_n$.

The β-reduction will be denoted by $\to_\beta$, while the η-reduction by $\to_\eta$. One step of either β-reduction or η-reduction will be denoted by $\to_{\beta\eta}$.

The letters $\xi_1, \xi_2, \dots$ denote algebraic variables (holes in Barendregt's terminology [1]). Contexts are built up as λ-terms but also allowing occurrences of algebraic variables. Substitution for algebraic variables is made without α-conversion. For example, $(\lambda x.x\xi)[xy/\xi] = \lambda x.x(xy)$.

A λ-term M is *solvable* if it has a *head normal form*, i.e., M is β-convertible to a term of the form $\lambda\vec{x}.y\vec{N}$. A λ-term M is *unsolvable* if it is not solvable. Among unsolvables we distinguish the *zero terms*, which never reduce to an abstraction.

A *λ-theory* is a congruence on Λ (with respect to the operators of abstraction and application) which contains $\alpha\beta$-conversion. We denote by $\lambda\beta$ the least λ-theory. The least extensional λ-theory $\lambda\beta\eta$ is axiomatised over $\lambda\beta$ by the equation $\lambda x.Mx = M$, where $M \in \Lambda$ and x is not free in M.

A λ-theory is *consistent* if it does not equate all λ-terms, *inconsistent* otherwise. A λ-theory is *semisensible* if it does not equate solvable and unsolvable λ-terms, so that semisensible λ-theories are consistent by definition. The set of λ-theories constitutes a complete lattice w.r.t. inclusion, whose top is the inconsistent λ-theory and whose bottom is the theory $\lambda\beta$. The λ-theory generated by a set X of identities is the intersection of all λ-theories containing X.

Although all unsolvable terms have the same Böhm tree, they do not have the same *infinite* normal form. In [2] Berarducci isolates a particular subset of the unsolvable terms, which turn out to have completely undefined behavior even in the context of infinite λ-calculus. Those terms, called *mute* terms, are defined as those unsolvables which are zero-terms and furthermore never reduce to an application whose left side is a zero-term. For example, $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ is mute. Another mute term that will be used in the rest of the paper, is defined as follows. Let $A \equiv \lambda x.x(\lambda y.yx)$, $B \equiv \lambda y.yA$ and $\Theta \equiv AB$. By a direct computation we see that the only possible reduction path starting with Θ is the following:

$$\Theta \to_\beta B(\lambda y.yB) \to_\beta (\lambda y.yB)A \to_\beta AB \equiv \Theta$$

Then Θ is a closed mute term. Throughout the paper we consider different reductions. If $\to_\gamma$ is a reduction, then we denote by $\twoheadrightarrow_\gamma$ the reflexive transitive closure of $\to_\gamma$, and we write $=_\gamma$ to denote the reflexive, symmetric and transitive closure of $\to_\gamma$. Finally we define, as usual, the γ-*reduction graph* of a term M as the set $\mathcal{G}_\gamma(M) = \{N \in \Lambda : M \twoheadrightarrow_\gamma N\}$.

2.3 Models of λ-calculus

It took some time, after Scott gave his model construction, for consensus to arise on the general notion of a model of the λ-calculus. There are mainly two descriptions that one can give: the category-theoretical and the algebraic one. Besides the different languages in which they are formulated, the two approaches are intimately connected (see [1]). The categorical notion of model is well-suited for constructing concrete models, while the algebraic one is rather used to understand global properties of models (constructions of new models out of existing ones, closure properties, etc.) and to obtain results about the structure of the lattice of λ-theories.

The algebraic description of models of λ-calculus proposes two kinds of structures, viz. the λ-*algebras* and the λ-*models*, both based on the notion of *combinatory algebra*. We will focus on λ-models. A *combinatory algebra* $\mathbf{A} = (A, \cdot, K, S)$ is a structure with a binary operation called *application* and two distinguished elements K and S called *basic combinators*. The symbol "$\cdot$" is usually omitted from expressions and by convention application associates to the left, allowing to leave out superfluous parentheses. The class of combinatory algebras is axiomatized by the equations $Kxy = x$ and $Sxyz = xz(yz)$. Intuitively elements on the left-hand side of an application are to be seen as functions operating on arguments, placed on the right-hand side. Hence it is natural to say that a function $f : A^n \to A$ is *representable* (*in* $\mathbf{A}$) if there exists an element $a \in A$ such that $f(b_1, \dots, b_n) = ab_1 \dots b_n$ for all $b_1, \dots, b_n \in A$. For example the identity function is represented by the combinator $I \equiv SKK$ and the projection on the first argument by the combinator K.

The axioms of an elementary subclass of combinatory algebras, called λ-*models*, were expressly chosen to make coherent the definition of interpretation of λ-terms. In addition to the axioms of combinatory algebra, we have:

$$\forall xy.(\forall z.\ xz = yz) \Rightarrow \mathbf{1}x = \mathbf{1}y$$
$$\mathbf{1}_2 K = K$$
$$\mathbf{1}_3 S = S,$$

where $\mathbf{1}_1 \equiv \mathbf{1} \equiv S(KI)$ and $\mathbf{1}_{n+1} \equiv S(K\mathbf{1})(S(K\mathbf{1}_n))$. The combinators $\mathbf{1}_n$ are made into inner choice operators. Indeed, given any $a \in A$, the element $\mathbf{1}_n a$ represents the same n-ary function as a and $\mathbf{1}_n c = \mathbf{1}_n d$ for all c, d representing the same n-ary function.

Let Env_A be the set of A-environments, i.e. , the functions from the set Var of λ-calculus variables to A. For every $x \in$ Var and $a \in A$ we denote by $\rho[x := a]$ the environment ρ' which coincides with ρ everywhere except on x, where ρ' takes the value a.

When $\mathbf{A}$ is a λ-model it is possible to define the following interpretation:

$|x|^{\mathbf{A}}_\rho = \rho(x);$
$|MN|^{\mathbf{A}}_\rho = |M|^{\mathbf{A}}_\rho |N|^{\mathbf{A}}_\rho;$
$|\lambda x.M|^{\mathbf{A}}_\rho = \mathbf{1}a$, where $a \in A$ is any element representing the function $b \in A \mapsto |M|^{\mathbf{A}}_{\rho[x:=b]}$.

Note that $|\lambda x.M|^{\mathbf{A}}_{\rho}$ is well-defined, since each function $b \in A \mapsto |M|^{\mathbf{A}}_{\rho[x:=b]}$ is representable under the hypothesis that $\mathbf{A}$ is a λ-model. This is the kind of interpretation we will refer to.

By the way when M is a closed λ-term and there is no worry of confusion about the model being considered, we write $|M|$ for $|M|^{\mathbf{A}}_{\rho}$.

Each λ-model $\mathbf{A}$ induces a λ-theory, denoted here by $Th(\mathbf{A})$, and called *the equational theory of* $\mathbf{A}$. Thus, $M = N \in Th(\mathbf{A})$ if, and only if, M and N have the same interpretation in $\mathbf{A}$. A *partially ordered* λ*-model*, a *po-model* for short, is a pair $(\mathbf{A}, \leq)$, where $\mathbf{A}$ is a λ-model and $\leq$ is a partial order on A which makes the application operator of $\mathbf{A}$ monotone in both arguments. A po-model $(\mathbf{A}, \leq)$ is *non-trivial* if the partial order is not discrete, i.e., $a < b$ for some $a, b \in A$ (thus A is not a singleton).

2.4 The Jacopini–Kuper technique

The Jacopini–Kuper technique, introduced by Jacopini in [7] and generalized by Kuper in [8], can be used to tackle questions of consistency of equational extensions of lambda calculus. In this section we review this technique.

Let $\mathcal{T}$ be an arbitrary consistent λ-theory, $\vec{P} = \vec{Q}$ be a set of identities $P_i = Q_i$ $(i = 1, \dots, n)$ between closed λ-terms, and $\mathcal{T}'$ be the λ-theory generated by $\mathcal{T} \cup (\vec{P} = \vec{Q})$. The idea is to reduce inconsistency of $\mathcal{T}'$ to that of $\mathcal{T}$. If $\mathcal{T}'$ is inconsistent, then there exists a finite equational proof of $\mathbf{K} =_{\mathcal{T}'} \mathbf{S}$, (where $\mathbf{K} \equiv \lambda xy.x$ and $\mathbf{S} \equiv \lambda xyz.xz(yz)$) and such a proof contains a finite number of applications of equations which are in $\vec{P} = \vec{Q}$. The Jacopini–Kuper technique, when applicable, consists in checking two conditions on the sequences $\vec{P}$ and $\vec{Q}$, namely that $\vec{P}$ is $\mathcal{T}$*-operationally less defined* than $\vec{Q}$ (see Definition 2) and that $\vec{P}$ is $\mathcal{T}$*-proof-substitutable* by $\vec{Q}$ (see Definition 3). Under these two conditions, it is possible to remove from the proof of $\mathbf{K} =_{\mathcal{T}'} \mathbf{S}$ all occurrences of equations in $\vec{P} = \vec{Q}$, thus yielding a proof of $\mathbf{K} =_{\mathcal{T}} \mathbf{S}$. This is the end of the method, because $\mathcal{T}$ is supposed to be a consistent λ-theory.

A very useful property for the application of Jacopini-Kuper method, and in particular for proving $\mathcal{T}$-proof-substitutability, is the existence of a Church-Rosser reduction, whose induced conversion coincides with the equality induced by $\mathcal{T}$ on λ-terms. This is not evident from the abstract formulation given in this section, but will be clear in the next one, when we will apply the technique.

▶ **Lemma 1.** *We have that* $\mathcal{T} \cup (\vec{P} = \vec{Q}) \vdash M = N$ *if, and only if, there exist closed terms* $F_1, \dots, F_n$ *such that*

$$\begin{array}{lll} M & =_{\mathcal{T}} & F_1\vec{P}\vec{Q} \\ F_j\vec{Q}\vec{P} & =_{\mathcal{T}} & F_{j+1}\vec{P}\vec{Q} \quad \textit{for } 1 \leq j \leq n-1 \\ F_n\vec{Q}\vec{P} & =_{\mathcal{T}} & N. \end{array}$$

Proof. By [4, Theorem 1] there exist binary contexts $C_1(\xi_1, \xi_2), \dots, C_n(\xi_1, \xi_2)$ and identities $P_{i_j} = Q_{i_j}$ in $\vec{P} = \vec{Q}$ such that

$$\begin{array}{lll} M & =_{\mathcal{T}} & C_1(P_{i_1}, Q_{i_1}) \\ C_j(Q_{i_j}, P_{i_j}) & =_{\mathcal{T}} & C_{j+1}(P_{i_{j+1}}, Q_{i_{j+1}}) \quad \text{for } 1 \leq j \leq n-1 \\ C_n(Q_{i_n}, P_{i_n}) & =_{\mathcal{T}} & N. \end{array}$$

It is sufficient to define $F_j \equiv \lambda\vec{x}\vec{y}.C_j(x_{i_j}, y_{i_j})$, where $\vec{x}$ and $\vec{y}$ are sequences of length k of fresh variables. ◀

▶ **Definition 2** (Operational definiteness)**.** We say that $\vec{P}$ is $\mathcal{T}$-operationally less defined than $\vec{Q}$ if, for every $\beta\eta$-normal form N and every term F, we have that

$$F\vec{P} =_{\mathcal{T}} N \Rightarrow F\vec{Q} =_{\mathcal{T}} N.$$

▶ **Definition 3** (Proof-substitutability)**.** We say that $\vec{P}$ is $\mathcal{T}$-proof-substitutable by $\vec{Q}$ if

$$\forall F, F' \in \Lambda^o(F\vec{P} =_{\mathcal{T}} F'\vec{P} \Rightarrow \exists G \in \Lambda^o(G\vec{P}\vec{Q} =_{\mathcal{T}} F\vec{Q} \text{ and } G\vec{Q}\vec{P} =_{\mathcal{T}} F'\vec{Q})).$$

▶ **Theorem 4.** *If $\vec{P}$ is $\mathcal{T}$-operationally less defined than $\vec{Q}$ and $\vec{P}$ is $\mathcal{T}$-proof-substitutable by $\vec{Q}$, then the λ-theory $\mathcal{T}'$ generated by $\mathcal{T} \cup (\vec{P} = \vec{Q})$ is consistent.*

Proof. Assume $\mathcal{T}'$ is inconsistent, so that $\mathbf{K} =_{\mathcal{T}'} \mathbf{S}$. Then by Lemma 1 there exists an equational proof of this identity of the form

$$\begin{array}{lll} \mathbf{K} & =_{\mathcal{T}} & F_1\vec{P}\vec{Q} \\ F_j\vec{Q}\vec{P} & =_{\mathcal{T}} & F_{j+1}\vec{P}\vec{Q} \quad \text{for } 1 \leq j \leq n-1 \\ F_n\vec{Q}\vec{P} & =_{\mathcal{T}} & \mathbf{S}. \end{array}$$

Now we show how to iteratively transform the above proof of $\mathcal{T}' \vdash \mathbf{K} = \mathbf{S}$ in a proof of $\mathcal{T} \vdash \mathbf{K} = \mathbf{S}$.

Suppose $n = 1$, i.e., we have $\mathbf{K} =_{\mathcal{T}} F_1\vec{P}\vec{Q}$ and $F_1\vec{Q}\vec{P} =_{\mathcal{T}} \mathbf{S}$. Since $\vec{P}$ is $\mathcal{T}$-operationally less defined than $\vec{Q}$, from $\mathbf{K} =_{\mathcal{T}} F_1\vec{P}\vec{Q} =_{\mathcal{T}} (\lambda\vec{y}.F_1\vec{y}\vec{Q})\vec{P}$ and $F_1\vec{Q}\vec{P} =_{\mathcal{T}} \mathbf{S}$, we get $\mathbf{K} =_{\mathcal{T}} F_1\vec{Q}\vec{Q} =_{\mathcal{T}} \mathbf{S}$.

Suppose $n > 1$. As before, by the hypothesis we get $\mathbf{K} =_{\mathcal{T}} F_1\vec{Q}\vec{Q}$ and $F_n\vec{Q}\vec{Q} =_{\mathcal{T}} \mathbf{S}$. Let $\vec{y}$ be a sequence of fresh variables. For each $j = 1, \dots, n-1$, define

$$H_j \equiv \lambda\vec{y}.F_j\vec{Q}\vec{y}; \qquad H'_{j+1} \equiv \lambda\vec{y}.F_{j+1}\vec{y}\vec{Q}.$$

By $F_j\vec{Q}\vec{P} =_{\mathcal{T}} F_{j+1}\vec{P}\vec{Q}$ $(j = 1, \dots, n-1)$ we have that

$$H_j\vec{P} =_{\mathcal{T}} H'_{j+1}\vec{P}.$$

Since $\vec{P}$ is $\mathcal{T}$-proof-substitutable by $\vec{Q}$, then there exist terms G_j $(j = 1, \dots, n-1)$ such that $G_j\vec{P}\vec{Q} =_{\mathcal{T}} H_j\vec{Q} =_{\mathcal{T}} F_j\vec{Q}\vec{Q}$ and $G_j\vec{Q}\vec{P} =_{\mathcal{T}} H'_{j+1}\vec{Q} =_{\mathcal{T}} F_{j+1}\vec{Q}\vec{Q}$. Therefore we obtain that

$$\begin{array}{rcccl} \mathbf{K} & =_{\mathcal{T}} & F_1\vec{Q}\vec{Q} & =_{\mathcal{T}} & G_1\vec{P}\vec{Q} \\ G_1\vec{Q}\vec{P} & =_{\mathcal{T}} & F_2\vec{Q}\vec{Q} & =_{\mathcal{T}} & G_2\vec{P}\vec{Q} \\ & \vdots & & \vdots & \\ G_{n-1}\vec{Q}\vec{P} & =_{\mathcal{T}} & F_n\vec{Q}\vec{Q} & =_{\mathcal{T}} & \mathbf{S} \end{array}$$

Therefore one can iterate the argument and get a proof of $\mathcal{T} \vdash \mathbf{K} = \mathbf{S}$. ◀

3 On a question by Honsell and Plotkin

In this section we turn to a question posed in [5] by Honsell and Plotkin. The problem is whether or not there exists a formula φ of first-order logic written as a possibly empty list of universal quantifiers followed by a conjunction of equalities between λ-terms such that φ does not admit po-models with bottom element. According to Honsell and Plotkin, this problem falls under the name of Π_1-consistency of the class of po-models with bottom element. We observe that in the context of λ-calculus the Π_1-consistency is equivalent to the equational consistency, which is the particular case of Π_1-consistency in which the formula φ is quantifier-free. In this section we find a counterexample to the equational consistency of the class of po-models with bottom element.

3.1 The λ-theory $\lambda\pi\phi$

We introduce two equations between λ-terms, whose models will be shown to have strong properties with respect to the possible partial orderings they can be endowed with. Of course we have to prove that the λ-theory $\lambda\pi\phi$ generated by these equations is consistent and this will be done in Section 3.2.

The two equations we are going to introduce represent within λ-calculus the notion of subtractivity, which has been introduced in Universal Algebra by Ursini [14].

▶ **Definition 5.** An algebra $\mathbf{A}$ is *subtractive* if there exist a binary term $s(x, y)$ and a constant 0 in the algebraic similarity type of $\mathbf{A}$ such that

$$s(x, x) = 0; \qquad s(x, 0) = x.$$

Subtractive algebras abound in classical algebras and in algebraic logic since term s simulates part of subtraction. If we interpret the binary operator "s" as subtraction "-" and we use the infix notation, then we can rewrite the above identities as $x - x = 0$ and $x - 0 = x$.

Let Θ be the closed mute term defined in Section 2.2. We define $s(x, y) \equiv \Theta xy$ and $0 \equiv \Omega$. Then the λ-theory $\lambda\pi\phi$ is defined as the least extensional λ-theory generated by the following two equations, called the *subtractive equations*:

$$(\pi)\ \ \Theta xx = \Omega; \qquad (\phi)\ \ \Theta x\Omega = x.$$

The intuitive meaning of the equations (π) and (ϕ) is that they make the term Θ behave like a binary subtraction operator (in curried form) whose "zero" is the term Ω. This intuition will be treated precisely in Section 5. The following theorem illustrates a curious aspect of the equations (π) and (ϕ): the choice of Ω is the right one.

▶ **Theorem 6.** *Let O be a λ-term such that $x \notin FV(O)$, and let $\mathcal{T}$ be any λ-theory including the identities $\Theta xO = x$ and $\Theta xx = O$. Then $\mathcal{T} \vdash O = \Omega$.*

Proof. We apply a technique introduced by Gordon Plotkin and Alex Simpson (see [13]). Let $Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ be the Curry fixpoint combinator. Then, for any λ-term M, define $\mu x.M \equiv Y(\lambda x.M)$. Now let $D \equiv \mu y.\mu x.\Theta xy$. Then we have $D =_\beta \Theta DD =_\mathcal{T} O$ and $D =_\beta \mu x.\Theta xD =_\mathcal{T} \mu x.\Theta xO =_\mathcal{T} \mu x.x =_\beta \Omega$, therefore $\mathcal{T} \vdash O = \Omega$. ◀

3.1.1 The λ-theory $\lambda\pi$

The extensional λ-theory $\lambda\pi$ is axiomatised over $\lambda\beta\eta$ by the equation (π). It is consistent because semisensible. We will show the consistency of $\lambda\pi\phi$ relying on the consistency of $\lambda\pi$.

We remark that the λ-theory axiomatised by $\Omega xx = \Omega$ was introduced in [12]. Here we use $\Theta xx = \Omega$ for technical reason.

The following notion of reduction will be useful in the next sections (recall from Section 2.2 the definition of reduction graph $\mathcal{G}_\beta(\Theta)$ of Θ).

▶ **Definition 7** ($\lambda\pi$-reduction)**.** We formally introduce here $\lambda\pi$-reduction, notation $\to_{\lambda\pi}$, as the contextual closure of $\to_{\beta\eta} \cup \to_\pi$, where

$$\Psi MN \to_\pi \Omega \quad \text{if } \Psi \in \mathcal{G}_\beta(\Theta) \text{ and } \lambda\pi \vdash M = N.$$

Of course the conversion $=_{\lambda\pi}$ coincides with the equality induced by $\lambda\pi$.

▶ **Theorem 8.**
- *The reduction $\to_{\lambda\pi}$ is Church-Rosser;*
- *For all terms M and N, we have $\Theta MN =_{\lambda\pi} \Omega$ iff $M =_{\lambda\pi} N$.*

Proof. As in the proof of [12, Lemma 3.1], it is sufficient to verify that $\twoheadrightarrow_\pi$ satisfies the diamond property (see [1, Lemma 3.2.2]) and that the relations $\twoheadrightarrow_{\beta\eta}$ and $\twoheadrightarrow_\pi$ commute (see [1, Def. 3.3.4]). The conclusion follows from the Hindley–Rosen Lemma (see [1, Prop. 3.3.5]). ◀

Another useful result is the forthcoming lemma, which says that all $\lambda\pi$-reduction paths may be "simulated" by a reduction path which allows π-steps only at the end.

▶ **Lemma 9** (Factorization). *If $M \twoheadrightarrow_{\lambda\pi} N$, then there exists P such that $M \twoheadrightarrow_{\beta\eta} P \twoheadrightarrow_\pi N$.*

Proof. Use iteratively the fact that whenever $M \rightarrow_\pi N \rightarrow_{\beta\eta} Q$, then there exists N' such that $M \rightarrow_{\beta\eta}^{=} N' \twoheadrightarrow_\pi Q$. ◀

▶ **Lemma 10.** *The terms Θ and Ω are not $\lambda\pi$-convertible.*

Proof. By the reduction graph of Θ and the confluence of $\rightarrow_{\lambda\pi}$. ◀

We remark that $\rightarrow_{\lambda\pi}$-residuals do not create new $\rightarrow_\pi$-redexes. For example, if M is a $\rightarrow_\pi$-redex, then the reduction $MNZ \rightarrow_\pi \Omega NZ$ only contains $\rightarrow_\pi$-redexes already present in N or Z. Any further reduction can only duplicate, erase or contract those redexes. This would not have been true if $\Omega \in \mathcal{G}_\beta(\Theta)$.

3.2 Jacopini–Kuper technique for $\lambda\pi\phi$

In this section we apply the Jacopini–Kuper technique explained in Section 2.4 to prove the consistency of the theory $\lambda\pi\phi$. More precisely, the results presented here show that the closure $\lambda x.\Theta x\Omega = \mathbf{I}$ of the equation (ϕ), that axiomatizes $\lambda\pi\phi$ over $\lambda\pi$, satisfies the hypotheses of Theorem 4.

▶ **Lemma 11.** *The term $\lambda x.\Theta x\Omega$ is $\lambda\pi$-operationally less defined than $\mathbf{I}$.*

Proof. Let F be a λ-term and N be a $\beta\eta$-normal form, and suppose $F(\lambda x.\Theta x\Omega) =_{\lambda\pi} N$. Since $\lambda\pi$-reduction is confluent and N is $\beta\eta$-normal, we have that $F(\lambda x.\Theta x\Omega) \twoheadrightarrow_{\lambda\pi} N$. By Lemma 9 there exists a term M such that

$$F(\lambda x.\Theta x\Omega) \twoheadrightarrow_{\beta\eta} M \twoheadrightarrow_\pi N.$$

Since N is a $\beta\eta$-normal form, we must have that $M \equiv N$. Therefore we have $\lambda\beta\eta \vdash F(\lambda x.\Theta x\Omega) = N$ and, since $\lambda x.\Theta x\Omega$ is unsolvable, we can apply the Genericity Lemma of lambda calculus to obtain $\lambda\beta\eta \vdash F\mathbf{I} = N$, and hence obviously $\lambda\pi \vdash F\mathbf{I} = N$ which is the desired conclusion. ◀

In Lemma 12 below we keep track of the residuals of the λ-term $\lambda x.\Theta x\Omega$ during the reduction of the term $F(\lambda x.\Theta x\Omega)$. We have three kinds of residuals: $\lambda x.\Psi x\Omega$, $\Psi M\Omega$ and Ω (with $\Psi \in \mathcal{G}_\beta(\Theta)$) as the following informal example shows:

$$\begin{array}{llll}
F(\lambda x.\Theta x\Omega) & \twoheadrightarrow_{\lambda\pi} & \cdots(\lambda x.\Theta x\Omega)\cdots(\lambda x.\Theta x\Omega)\cdots(\lambda x.\Theta x\Omega)\cdots & \\
 & \twoheadrightarrow_{\beta} & \cdots(\lambda x.\Theta x\Omega)\cdots(\lambda x.\Psi x\Omega)\cdots\cdots(\lambda x.\Theta x\Omega)\cdots & (\Theta \twoheadrightarrow_\beta \Psi) \\
 & \twoheadrightarrow_{\lambda\pi} & \cdots(\lambda x.\Theta x\Omega)\cdots(\lambda x.\Psi x\Omega)M\cdots(\lambda x.\Theta x\Omega)\cdots & \\
 & \rightarrow_{\beta} & \cdots(\lambda x.\Theta x\Omega)\cdots(\Psi M\Omega)\cdots(\lambda x.\Theta x\Omega)\cdots & (\beta\text{-reduction}) \\
 & \twoheadrightarrow_{\lambda\pi} & \cdots(\lambda x.\Theta x\Omega)\cdots(\Psi N\Omega)\cdots(\lambda x.\Theta x\Omega)\cdots & (M \twoheadrightarrow_{\lambda\pi} N) \\
 & \rightarrow_{\pi} & \cdots(\lambda x.\Theta x\Omega)\cdots(\Omega)\cdots(\lambda x.\Theta x\Omega)\cdots & (N =_{\lambda\pi} \Omega) \\
 & \twoheadrightarrow_{\lambda\pi} & \cdots\cdots\cdots &
\end{array}$$

In order to trace the residuals it is useful to enrich the syntax of λ-terms with labels as follows:

$$\begin{array}{lll} M, N & ::= & x \mid \lambda x.M \mid MN \\ & & \mid (\lambda x.\Psi x\Omega)^n \mid (\Psi M\Omega)^n \mid (\Omega)^n \quad (n \geq 1 \text{ and } \Psi \in \mathcal{G}_\beta(\Theta)) \end{array}$$

We denote by $\Lambda^{\mathbb{N}}$ the set of labelled terms and we write $\underline{M}$ for the λ-term, called *erasure* of M, obtained by erasing the labels from M.

Since $(\Omega)^n$ and $(\lambda x.\Psi x\Omega)^n$ are closed terms, then it is sufficient to extend substitution to labelled terms by setting $(\Psi M\Omega)^n[N/x] = (\Psi M[N/x]\Omega)^n$, where $\Psi \in \mathcal{G}_\beta(\Theta)$. Then we define a reduction on labelled terms as the smallest contextual reduction $\to_{\text{lab}}$ satisfying the following clauses, for all labelled terms M, N:

$$\begin{array}{ll} (\lambda x.M)N \to_{\text{lab}} M[N/x] & \\ \lambda x.Mx \to_{\text{lab}} M & \text{if } x \notin \mathrm{FV}(M) \\ (\lambda x.\Psi x\Omega)^n M \to_{\text{lab}} (\Psi M\Omega)^n & \text{if } \Psi \in \mathcal{G}_\beta(\Theta) \\ \Psi MN \to_{\text{lab}} \Omega & \text{if } \Psi \in \mathcal{G}_\beta(\Theta) \text{ and } \underline{M} =_{\lambda\pi} \underline{N}. \end{array}$$

Note that (i) $\to_{\lambda\pi} \subseteq \to_{\text{lab}}$; (ii) if $M \to_{\text{lab}} N$ then $(\Psi M\Omega)^n \to_{\text{lab}} (\Psi N\Omega)^n$. If σ is a reduction path of labelled terms, then we denote by $\underline{\sigma}$ the corresponding reduction path, where all labels are erased.

We will make use of an additional operation on labelled terms. Given terms $M, N \in \Lambda^{\mathbb{N}}$ such that $\underline{M} \equiv \underline{N}$, we define their *superposition* as the labelled term obtained from the syntax tree of $\underline{M}$ by adding a possible label k to each subtree T of $\underline{M}$ according to the following schema:

- Put $k = m + n$ if T has label m in M and n in N;
- Put $k = m$ if T has label m in M and no label in N;
- Put $k = n$ if T has label n in N and no label in M;
- Put no label otherwise.

▶ **Lemma 12.** *The term $\lambda x.\Theta x\Omega$ is $\lambda\pi$-proof-substitutable by* $\mathbf{I}$.

Proof. In this proof Ψ ranges over $\mathcal{G}_\beta(\Theta)$. Let F_1, F_2 be closed λ-terms and suppose $F_1(\lambda x.\Theta x\Omega) =_{\lambda\pi} F_2(\lambda x.\Theta x\Omega)$. Since the reduction $\to_{\lambda\pi}$ is confluent, then the two sides of the equality are the beginning of two reduction paths σ_1 and σ_2 that end in a common term R. Consider now the labelled terms $F_i(\lambda x.\Theta x\Omega)^i$ for $i = 1, 2$. Then there exists a labelled reduction path σ'_i starting with $F_i(\lambda x.\Theta x\Omega)^i$ such that $\underline{\sigma'_i} \equiv \sigma_i$. We denote by R_i the last labelled term in the reduction path σ'_i. Then we have that $R \equiv \underline{R}_1 \equiv \underline{R}_2$.

Let S be the term obtained by superposition of R_1 and R_2. Then the labels of S range over the set $\mathcal{L} = \{1, 2, 3\}$. We now describe how to extract a witness of $\lambda\pi$-proof-substitutability by suitably modifying S. All residuals with label 3 in S are common to the reduction paths σ'_1 and σ'_2. Then, if we mimic the reduction path σ_i starting from $F_i\mathbf{I}$ $(i = 1, 2)$, we will find in place of the residuals with label 3 the term $\mathbf{I}$ for $(\lambda x.\Psi x\Omega)^3$; M for $(\Psi M\Omega)^3$ and a term N ($\lambda\pi$-convertible with Ω) for $(\Omega)^3$:

$$\begin{array}{lll@{\qquad\qquad}lll} F_i(\lambda x.\Theta x\Omega) & \twoheadrightarrow_{\lambda\pi} & (i = 1, 2) & F_i\mathbf{I} & \twoheadrightarrow_{\lambda\pi} & (i = 1, 2) \\ \cdots(\lambda x.\Theta x\Omega)\cdots & \twoheadrightarrow_{\beta} & & \cdots\mathbf{I}\cdots & \equiv & \\ \cdots(\lambda x.\Psi x\Omega)\cdots & \twoheadrightarrow_{\lambda\pi} & (\Theta \twoheadrightarrow_\beta \Psi) & \cdots\mathbf{I}\cdots & \twoheadrightarrow_{\lambda\pi} & \\ \cdots(\lambda x.\Psi x\Omega)M\cdots & \to_{\beta} & & \cdots\mathbf{I}M\cdots & \to_{\beta} & \\ \cdots\Psi M\Omega\cdots & \to_{\lambda\pi} & (M \twoheadrightarrow_{\lambda\pi} N) & \cdots M\cdots & \to_{\lambda\pi} & (M \twoheadrightarrow_{\lambda\pi} N) \\ \cdots\Psi N\Omega\cdots & \to_{\lambda\pi} & (N =_{\lambda\pi} \Omega) & \cdots N\cdots & \equiv & (N =_{\lambda\pi} \Omega) \\ \cdots\Omega\cdots & & & \cdots N\cdots & & \end{array}$$

Then we let $S' \equiv S[\mathbf{I}/(\lambda x.\Psi x\Omega)^3; M/(\Psi M\Omega)^3; \Omega/(\Omega)^3]$. The last substitution Ω for $(\Omega)^3$ is possible because the term N in the above reduction path (right column) is $\lambda\pi$-convertible with Ω. We see that, by mimicking the steps in the paths σ_1, σ_2, we have that

$$(*) \qquad F_i\mathbf{I} =_{\lambda\pi} L_1, \text{ where } L_1 \text{ is the erasure of } S'[\mathbf{I}/(\lambda x.\Psi x\Omega)^i; M/(\Psi M\Omega)^i] \quad (i = 1, 2)$$

Let x_1, x_2 be fresh variables and let H be the term obtained from S' by replacing bottom-up the subterms (labeled by $i \in \mathcal{L}$)

$$\text{for } i = 1,2 \begin{cases} (\lambda x.\Psi x\Omega)^i & \text{with } x_i; \\ (\Psi M\Omega)^i & \text{with } x_i M; \\ (\Omega)^i & \text{with } x_i\Omega. \end{cases}$$

Then the following equivalences hold:

(a) $L_1 =_{\lambda\pi} H[\mathbf{I}/x_1; (\lambda x.\Psi x\Omega)/x_2]$;

(b) $L_2 =_{\lambda\pi} H[(\lambda x.\Psi x\Omega)/x_1; \mathbf{I}/x_2]$.

Therefore by setting $G \equiv \lambda x_2 x_1.H$, we obtain that

- $G(\lambda x.\Theta x\Omega)\mathbf{I} \twoheadrightarrow_\beta H[\mathbf{I}/x_1; (\lambda x.\Psi x\Omega)/x_2] =_{\lambda\pi} L_1 =_{\lambda\pi} F_1\mathbf{I}$, by $(*)$ and (a)
- $G\mathbf{I}(\lambda x.\Theta x\Omega) \twoheadrightarrow_\beta H[(\lambda x.\Psi x\Omega)/x_1; \mathbf{I}/x_2] =_{\lambda\pi} L_2 =_{\lambda\pi} F_2\mathbf{I}$, by $(*)$ and (b)

This shows that G is the witness term we were looking for. ◀

Now we are ready to give the main theorem of the section.

▶ **Theorem 13.** *The λ-theory $\lambda\pi\phi$ is consistent.*

Proof. Lemma 12 and Lemma 11 show that the hypotheses of Theorem 4 are satisfied by the equation that axiomatizes $\lambda\pi\phi$ over $\lambda\pi$, and therefore $\lambda\pi\phi$ must be consistent. ◀

3.3 The main theorem

The following results prove that there is no po-model with bottom element satisfying the equations (π) and (ϕ) that define the λ-theory $\lambda\pi\phi$.

▶ **Lemma 14.** *Let $\mathcal{M}$ be a po-model such that $\mathcal{M} \models \Theta xx = \Omega \wedge \Theta x\Omega = x$ (i.e., $Th(\mathcal{M}) \supseteq \lambda\pi\phi$). Then for all closed λ-terms P and Q we have:*

(i) If $\mathcal{M} \not\models \Theta PQ = \Omega$, then the interpretations of P and Q are in distinct connected components of $\mathcal{M}$.

(ii) The connected component of the interpretation of Ω is a singleton set.

Proof. (i) Following [12, Section 4] we define the subtraction sequence of the pair (P, Q):

$$s_1 \equiv \Theta PQ; \qquad s_{n+1} \equiv \Theta s_n\Omega.$$

By hypothesis $\mathcal{M} \models s_1 \neq \Omega$ and by subtractivity $\mathcal{M} \models s_n = s_1$ for all n. Then the conclusion follows from [12, Corollary 4.6].

(ii) Let $a \in \mathcal{M}$ and $a \neq |\Omega|$. Consider the subtraction sequence of the pair (a, Ω): $s_1 = \Theta a\Omega$ and $s_{n+1} = \Theta s_n\Omega$. Since $\mathcal{M} \models s_n = a$ for all n, again an application of [12, Corollary 4.6] implies that a and the interpretation of Ω are in distinct connected components. ◀

The situation described by Lemma 14 can be regarded to as a relativized version of absolute unorderability to one fixed element. In particular the interpretation of Ω is isolated in every model. This property will be studied in Section 5 in the framework of Universal Algebra.

We recall from [5, Theorem 7] that consistency fails for quantifier-free sentences and po-models with bottom element. The sentence $\lambda x.\Omega xx = \lambda x.\Omega \wedge \Omega \neq \Omega\Omega(\Omega\mathbf{KI})$ is consistent with the extensional λ-calculus but no po-model with bottom element satisfies it. The following theorem improves this result by Honsell and Plotkin.

▶ **Theorem 15.** *For every non-trivial po-model $\mathcal{M}$ with bottom element, we have $\mathcal{M} \not\models \Theta xx = \Omega \ \wedge\ \Theta x\Omega = x$.*

Proof. Suppose, by contradiction, that $\mathcal{M} \models \Theta xx = \Omega \wedge \Theta x\Omega = x$. Since $|\Omega|$ is comparable with $\bot$, then by Lemma 14(ii) Ω is interpreted as the bottom element $\bot$. The bottom element is comparable with all other elements of the model. This contradicts Lemma 14(ii). ◀

Therefore the equations $\lambda x.\Theta xx = \lambda x.\Omega$ and $\lambda x.\Theta x\Omega = \lambda x.x$ are indeed a counterexample to the equational consistency for the class of po-models with bottom element. We can also get a stronger result.

▶ **Theorem 16.** *If $\mathcal{M}$ is a po-model such that $Th(\mathcal{M}) \supseteq \lambda\pi\phi$, then the partial ordering of $\mathcal{M}$ is not connected.*

4 On the order-incompleteness of λ-calculus

The open problem of the *order-incompleteness* of λ-calculus was raised by Selinger in [13]: does there exist a λ-theory which does not arise as the theory of a non-trivial po-model? Such a problem can be also characterised in terms of connected components of a partial ordering (minimal subsets which are both upward and downward closed): a λ-theory $\mathcal{T}$ is order-incomplete if, and only if, every po-model, having $\mathcal{T}$ as equational theory, is partitioned in an infinite number of connected components, each one containing exactly one element. In other words, the partial order is the equality.

So far we have shown that the subtractive equations force their models not to be connected as partial orders. However, the order-incompleteness is far more distant to connectedness. Toward order-incompleteness, we propose a strengthening $\mathcal{T}$ of the λ-theory $\lambda\pi\phi$ having the following property: every po-model $\mathcal{M}$ such that $Th(\mathcal{M}) \supseteq \mathcal{T}$ has an infinite number of connected components among which that of $|\Omega|$ is a singleton set. Moreover each connected component contains the denotation of at most one $\beta\eta$-normal form.

We are now going to introduce the above-mentioned strenghtening of $\lambda\pi\phi$. It will make use of another mute term, that we will call Θ_2, obtained as follows:

- Define inductively $A_0 \equiv x$ and $A_{n+1} \equiv \lambda y.yA_n$, where $y \not\equiv x$. Note that $\mathrm{FV}(A_n) = \{x\}$, for each $n \in \mathbb{N}$.
- Now set $B_2 \equiv \lambda x.xA_2$, $C_2 \equiv (\lambda z.zB_2)$ and $\Theta_2 \equiv B_2C_2$.

It is not difficult to check that Θ_2 is a mute closed term. Moreover Ω, Θ and Θ_2 are pairwise non-$\lambda\pi$-convertible: this is an immediate consequence of the confluence of $\to_{\lambda\pi}$ and of the form of the reduction graphs of the terms in question.

Let $\mathcal{T}$ be the theory axiomatized over $\lambda\pi\phi$ by the following equations:

$\Theta_2\Omega = \mathbf{K}$;
$\Theta_2(\Theta MN) = \mathbf{S}$, $\quad$ M and N distinct closed $\beta\eta$-normal forms.

Next we show that $\mathcal{T}$ is consistent. In order to do that it suffices, by compactness reasons, to prove that any finite subset of the above equations is eliminable from a proof of $\mathcal{T} \vdash \mathbf{K} = \mathbf{S}$ via the Jacopini–Kuper technique. The proof of this fact closely resembles the consistency proof given for $\lambda\pi\phi$ (see Section 3.2), so we will just sketch it, only considering the extension of $\lambda\pi$ by three equations $\Theta_2(\Theta MN) = \mathbf{S}$, $\Theta_2\Omega = \mathbf{K}$ and $\lambda x.\Theta x\Omega = \mathbf{I}$, where (M, N) is an arbitrary but fixed pair of closed distinct $\beta\eta$-normal forms.

Define the two sequences $\vec{P} = \Theta_2(\Theta MN), \Theta_2\Omega, \lambda x.\Theta x\Omega$ and $\vec{Q} = \mathbf{S}, \mathbf{K}, \mathbf{I}$.

We observe that it follows directly from Lemma 11 that $\vec{P}$ is $\lambda\pi$-operationally less defined than $\vec{Q}$.

▶ **Lemma 17.** *$\vec{P}$ is $\lambda\pi$-proof-substitutable by $\vec{Q}$.*

Proof. In this proof Ψ and Ψ_2 range, respectively, over $\mathcal{G}_\beta(\Theta)$ and $\mathcal{G}_\beta(\Theta_2)$. Let F_1, F_2 be closed λ-terms and suppose $F_1\vec{P} =_{\lambda\pi} F_2\vec{P}$. Since the reduction $\rightarrow_{\lambda\pi}$ is confluent, then the two sides of the equality are the beginning of two reduction paths σ_1 and σ_2 that end in a common term R.

Consider now the labelled term

- $A_1 \equiv F_1(\Theta_2(\Theta MN))^1(\Theta_2\Omega)^4(\lambda x.\Theta x\Omega)^{10}$
- $A_2 \equiv F_2(\Theta_2(\Theta MN))^2(\Theta_2\Omega)^5(\lambda x.\Theta x\Omega)^{11}$

Then there exist labelled reduction paths σ'_i starting with A_i $(i = 1, 2)$ such that $\underline{\sigma'_i} \equiv \sigma_i$. We denote by R_i the last labelled term in the reduction path σ'_i. Then we have $R \equiv \underline{R_i}$ $(i = 1, 2)$. Let S be the term obtained by superposition of R_1 and R_2. Then the labels of S range over the set $\mathcal{L} = \{1, 2, 3, 4, 5, 9, 10, 11\}$. Note that if S has a labelled subterm of the shape $(\Theta_2\Omega)^l$, then $l \in \{4, 5, 9\}$ because the contrary would require $\Theta MN \rightarrow_{\lambda\pi} \Omega$ (by Theorem 8(ii)), which is impossible because it would imply $\lambda\pi \vdash M = N$, contradicting the consistency of $\lambda\pi$ (as a consequence of Böhm's Theorem [1, Thm. 10.4.2]).

We now describe how to extract a witness of $\lambda\pi$-proof-substitutability by suitably modifying S. All residuals with label 3 ,9, or 21 in S are common to the reduction paths σ'_1 and σ'_2. Then, if we mimic the reduction path σ_i starting from $F_i\mathbf{I}$ $(i = 1, 2)$, we will find in place of the residuals with label 21 the term $\mathbf{I}$ for $(\lambda x.\Psi x\Omega)^{21}$; M for $(\Psi M\Omega)^{21}$ and a term N ($\lambda\pi$-convertible with Ω) for $(\Omega)^{21}$. Similarly those residuals with labels 3 and 9 are replaced by $\mathbf{S}$ and $\mathbf{K}$, respectively. Then we let

$$S' \equiv S[\mathbf{I}/(\lambda x.\Psi x\Omega)^{21}; M/(\Psi M\Omega)^{21}; \Omega/(\Omega)^{21}; \mathbf{S}/(\Theta_2(\Theta MN))^3; \mathbf{K}/(\Theta_2\Omega)^9]$$

and we define a term H out of S' by replacing bottom-up some subterms (labeled by $i \in \mathcal{L}$), using fresh variables $x_1, x_2, x_3, x_4, x_5, x_{10}, x_{11}$ as follows

$$\text{for } i = 10, 11 \begin{cases} (\lambda x.\Psi x\Omega)^i & \text{with } x_i; \\ (\Psi M\Omega)^i & \text{with } x_i M; \\ (\Omega)^i & \text{with } x_i\Omega. \end{cases} \quad \text{for } i = 4, 5 \text{ and } j = 1, 2 \begin{cases} (\Psi_2\Omega)^i & \text{with } x_i; \\ (\Psi_2(\Psi MN))^j & \text{with } x_j. \end{cases}$$

Finally, as in the proof of Lemma 12, it is possible to find a term G such that:

$$G\vec{P}\vec{Q} \twoheadrightarrow_\beta H[(\Psi_2(\Psi MN))/x_1; \mathbf{S}/x_2; (\Psi_2\Omega)/x_4; \mathbf{K}/x_5; \mathbf{I}/x_{10}; (\lambda x.\Psi x\Omega)/x_{11}] =_{\lambda\pi} F_1\vec{Q}$$

$$G\vec{Q}\vec{P} \twoheadrightarrow_\beta H[\mathbf{S}/x_1; (\Psi_2(\Psi MN))/x_2; \mathbf{K}/x_4; (\Psi_2\Omega)/x_5; (\lambda x.\Psi x\Omega)/x_{10}; \mathbf{I}/x_{11}] =_{\lambda\pi} F_2\vec{Q}$$

◀

The following theorem, which relies on Lemma 17, it is analogous to Theorem 13.

▶ **Theorem 18.** *The λ-theory $\mathcal{T}$ is consistent.*

We conclude the section with a theorem that improves a result in [12], where it is shown that every po-model $\mathcal{M}$ such that $Th(\mathcal{M}) = \lambda\pi$ has an infinite number of connected components.

▶ **Theorem 19.** *Let $\mathcal{M}$ be a po-model such that $Th(\mathcal{M}) \supseteq \mathcal{T}$. Then $\mathcal{M}$ has an infinite number of connected components and it has the following properties:*

1. *The interpretation in $\mathcal{M}$ of two distinct $\beta\eta$-normal forms belongs to different connected components;*
2. *The connected component of $|\Omega|$ is a singleton set.*

Proof. Let M, N be two distinct $\beta\eta$-normal forms and suppose, by way of contradiction, that $|M|$ and $|N|$ lie in the same connected component of $\mathcal{M}$. Then $\mathcal{M} \models \Theta MN = \Omega$ by Lemma 14(i). But then from $\mathcal{M} \models \mathbf{S} = \Theta_2(\Theta MN)$ and $\mathcal{M} \models \Theta_2\Omega = \mathbf{K}$ we derive that $\mathcal{M} \models \mathbf{S} = \mathbf{K}$, which contradicts the non-triviality of $\mathcal{M}$. Hence each denotation of a $\beta\eta$-normal form belongs to exactly one connected component. The second part of the statement follows directly from Lemma 14(ii). ◀

5 Subtractivity and orderings

The inspiration for the subtractive equations comes from a general algebraic framework, developed by Ursini [14], called *subtractivity*. Salibra in [12] investigated the weaker notion of *semi-subtractivity*, linking it to properties of po-models of λ-calculus. Here we follow that path illustrating the stronger properties of subtractivity.

We start the section briefly reviewing the connection established by Selinger in [13] between the absolute unorderability and the validity of certain Mal'cev-type conditions.

Let $\mathbf{A}$ be an algebra of some variety $\mathcal{V}$ (i.e., equational class). A preorder $\leq$ on $\mathbf{A}$ is *compatible* if it is monotone in each coordinate of every function symbol of $\mathcal{V}$. Then we have: (i) $\mathbf{A}$ is *unorderable* if it admits only equality as a compatible partial order; (ii) $\mathbf{A}$ is *absolutely unorderable* if, for every algebra $\mathbf{B} \in \mathcal{V}$ and every embedding $f : \mathbf{A} \to \mathbf{B}$ (i.e., injective homomorphism), the algebra $\mathbf{B}$ is unorderable.

Let $\mathcal{V}$ be a variety, $\mathbf{A} \in \mathcal{V}$ and X be a set of indeterminates. We denote by $\mathbf{A}[X]$ the free extension of $\mathbf{A}$ in the variety $\mathcal{V}$. The algebra $\mathbf{A}[X]$ is defined up to isomorphism by the following universal mapping properties: (1) $A \cup X \subseteq A[X]$; (2) $\mathbf{A}[X] \in \mathcal{V}$; (3) for every $\mathbf{B} \in \mathcal{V}$, homomorphism $h : \mathbf{A} \to \mathbf{B}$ and every function $f : X \to B$, there exists a unique homomorphism $f : \mathbf{A}[X] \to \mathbf{B}$ extending h and f. When $X = \{x_1, \dots, x_n\}$ is finite, we write $\mathbf{A}[x_1, \dots, x_n]$ for $\mathbf{A}[X]$.

The following result by Selinger [13] characterises those algebras which are absolutely unorderable.

▶ **Theorem 20.** *Let $\mathcal{V}$ be a variety. An algebra $\mathbf{A} \in \mathcal{V}$ is absolutely unorderable if, and only if, there exist a natural number $n \geq 1$ and ternary terms $p_1, \dots, p_n$ in the type of $\mathcal{V}$ such that the algebra $\mathbf{A}[x, y]$ satisfies the following identities:*

$$\begin{array}{rcll} x & = & p_1(x, y, y); & \\ p_i(x, x, y) & = & p_{i+1}(x, y, y) & (i = 1, \dots, n-1); \\ p_n(x, x, y) & = & y. & \end{array}$$

In the case the variety $\mathcal{V}$ has a constant 0, then we can relativise the Mal'cev identities as follows:

$$\begin{array}{rcll} 0 & = & p_1(0, y, y); & \\ p_i(0, 0, y) & = & p_{i+1}(0, y, y) & (i = 1, \dots, n-1); \\ p_n(0, 0, y) & = & y. & \end{array}$$

This suggests that the absolute unorderability relative to the element 0 can be expressed by the following identities defining n-subtractivity.

▶ **Definition 21.** Let $\mathcal{V}$ be a variety of algebras with a constant 0. We say that $\mathcal{V}$ is n-subtractive ($n \geq 2$) if there exist $n-1$ binary terms $s_1(x,y), \dots, s_{n-1}(x,y)$ such that $\mathcal{V}$ satisfies the following identities:

$$\begin{array}{rcll} 0 & = & s_1(x,x) & \\ s_i(x,0) & = & s_{i+1}(x,x) & (i = 1, \dots, n-2); \\ s_{n-1}(x,0) & = & x. & \end{array}$$

Then Ursini's subtractivity (see Definition 5) means 2-subtractivity.

Every model of the two equations (π) and (ϕ) is subtractive, when we define the binary operator $s_1(x,y)$ defining subtractivity as the λ-term Θxy. As a consequence of the consistency of the λ-theory $\lambda\pi\phi$, it follows that there exists a non-trivial subtractive variety of combinatory algebras.

▶ **Definition 22.** An algebra $\mathbf{A}$ is 0-*unorderable* if, for every compatible partial order $\leq$ on A and every $a \neq 0 \in A$, neither $0 \leq a$ nor $a \leq 0$.

▶ **Definition 23.** Let $\mathcal{V}$ be a variety. An algebra $\mathbf{A} \in \mathcal{V}$ is said to be *absolutely* 0-*unorderable* if, for any algebra $\mathbf{B} \in \mathcal{V}$ and embedding $f : \mathbf{A} \to \mathbf{B}$, $\mathbf{B}$ is 0-unorderable.

Let R_1 (resp. R_2) be the smallest compatible preorder on $\mathbf{A}[x]$ such that xR_10 (resp. $0R_2x$).

▶ **Lemma 24.** *Let $\mathcal{V}$ be a variety. An algebra $\mathbf{A} \in \mathcal{V}$ is absolutely 0-unorderable iff $0R_1x$ and xR_20.*

Proof. Assume that $\mathbf{A}$ is not absolutely 0-unorderable. Then there exists an embedding $f : \mathbf{A} \to \mathbf{B} \in \mathcal{V}$, where $\mathbf{B}$ has a non-trivial partial ordering $\leq$ and there exists an element $b \neq 0 \in B$ such that either $0 \leq b$ or $b \leq 0$. Consider the unique homomorphism $g : \mathbf{A}[x] \to \mathbf{B}$ extending f such that $g(x) = b$. Define aSc in $\mathbf{A}[x]$ iff $g(a) \leq g(c)$ in $\mathbf{B}$. We have that S is a compatible preorder on $\mathbf{A}[x]$ such that either $0Sx$ or $xS0$ but not both (!). If $xS0$, then $R_1 \subseteq S$ but not $0R_1x$. If $0Sx$, then $R_2 \subseteq S$ but not xR_20. ◀

Note that, as a consequence of Lemma 24, if $\mathbf{A}$ is absolutely 0-unorderable, then $R_1 = R_2$.

▶ **Theorem 25.** *Let $\mathcal{V}$ be a variety. An algebra $\mathbf{A} \in \mathcal{V}$ is absolutely 0-unorderable if, and only if, the free extension $\mathbf{A}[x]$ of $\mathbf{A}$ is n-subtractive for some $n \geq 2$.*

Proof. The argument is similar to Selinger's proof of [13, Theorem 3.4]. Define a relation $\prec$ on $\mathbf{A}[x]$ as follows: $t \prec u$ iff there exists a polynomial $p(x,y) \in \mathbf{A}[x,y]$ such that $\mathbf{A}[x] \models t = p(x,x)$ and $\mathbf{A}[x] \models p(x,0) = u$.

We start by showing that tR_1u iff $t \prec^* u$.

($\Rightarrow$) The relation $\prec$ is compatible and contains the pair $(x, 0)$ since the polynomial $p(x,y) \equiv y$ witnesses $x \prec 0$. Hence by its minimality, R_1 is contained in $\prec^*$.

($\Leftarrow$) On the other hand suppose $t \prec u$ and let $p(x,y) \in \mathbf{A}[x,y]$ be such that $\mathbf{A}[x] \models t = p(x,x)$ and $\mathbf{A}[x] \models p(x,0) = u$. Then $t = p(x,x)R_1p(x,0) = u$ by compatibility and the fact that xR_10. Finally the transitivity of R_1 implies that $\prec^* \subseteq R_1$.

By Lemma 24 and the above paragraph, if $\mathbf{A}$ is absolutely 0-unorderable, then there are $p_1, \dots, p_{n-1} \in \mathbf{A}[x,y]$ such that $p_1 \prec \dots \prec p_{n-1}$. These polynomials witness n-subtractivity.

Conversely if $\mathbf{A}$ is n-subtractive, then $0 \prec^* x$ and hence $\mathbf{A}$ is absolutely 0-unorderable. ◀

▶ **Corollary 26.** *(i) The term model of the λ-theory $\lambda\pi\phi$ is absolutely $|\Omega|$-unorderable in the variety of combinatory algebras.*

(ii) If $\mathcal{M}$ is a λ-model such that $Th(\mathcal{M}) \supseteq \lambda\pi\phi$, then $\mathcal{M}$ is absolutely $|\Omega|$-unorderable in the variety of combinatory algebras.

We would like to conclude this paper by remarking that Ursini [14] has shown that subtractive algebras have a good theory of ideals. We recall that ideals in general algebras generalize normal subgroups, ideals in rings, filters in Boolean or Heyting algebras, ideals in Banach algebra, in l-groups, etc. One feature of subtractive varieties is that their ideals are exactly the congruence classes of 0, but one does not have the usual one-one correspondence ideals-congruences: mapping a congruence θ to its equivalence class $0/\theta$ only establishes a lattice homomorphism between the congruence lattice and the ideal lattice. This points to another feature: the join of two congruences is a tricky thing to deal with. The join of two ideals in a subtractive algebra behaves nicely: for I, J ideals, we have that $b \in I \vee J$ iff for some $a \in I$, $s(b, a) \in J$. Thanks to the consistency of the subtractive equations with λ-calculus, the theory of ideals for subtractive varieties can be applied to all λ-theories extending $\lambda\pi\phi$.

References

1 Barendregt H.P., The lambda calculus: Its syntax and semantics, North-Holland Publishing Co., Amsterdam, 1984.

2 Berarducci A., *Infinite lambda-calculus and nonsensible models*, Logic and Algebra, Lecture Notes in Pure and Applied Mathematics, Marcel Dekker Inc., (1996), 339–378.

3 Burris S. and H.P. Sankappanavar, A course in universal algebra, Springer-Verlag, Berlin, 1981.

4 Duda J., *On two schemes applied to Mal'cev type theorems*, Ann. Univ. Sci. Budapest, Section Math. **26** (1983), 39–45.

5 Honsell F. and Plotkin G.D., *On the completeness of order-theoretic models of the λ-calculus*, Information and Computation **207**(5) (2009), 583–594.

6 Honsell F. and Ronchi della Rocca S., *An approximation theorem for topological λ-models and the topological incompleteness of λ-calculus*, Journal Computer and System Science **45** (1992), 49–75.

7 Jacopini, G., *A condition for identifying two elements of whatever model of combinatory logic*, in "Lambda Calculus and Computer Science" (C. Bohm, Ed.), Springer-Verlag, Berlin (1975), pp. 213–219.

8 Kuper J., *On the Jacopini technique*, Information and Computation **138** (1997), 101–123.

9 Lusin S. and Salibra A., *The lattice of lambda theories.* Journal of Logic and Computation **14** (2004), 373–394.

10 Manzonetto G. and Salibra A., *Applying universal algebra to lambda calculus*, J. Log. Comput., **20**(4) (2010), 877–915.

11 Plotkin G.D., *On a question of H. Friedman*, Information and Computation **126** (1996), 74–77.

12 Salibra A., *Topological incompleteness and order incompleteness of the lambda calculus*, ACM TOCL **4**(3) (2003).

13 Selinger P., *Order-incompleteness and finite lambda reduction models*, Theoretical Computer Science **309** (2003), 43–63.

14 Ursini A., *On subtractive varieties, I*, Algebra Universalis, **31**(2) (1994), 204–222.

Faster Algorithms for Alternating Refinement Relations

Krishnendu Chatterjee[1], Siddhesh Chaubal[2], and Pritish Kamath[2]

1 IST Austria (Institute of Science and Technology Austria)
2 IIT Bombay

Abstract

One central issue in the formal design and analysis of reactive systems is the notion of *refinement* that asks whether all behaviors of the implementation is allowed by the specification. The local interpretation of behavior leads to the notion of *simulation*. Alternating transition systems (ATSs) provide a general model for composite reactive systems, and the simulation relation for ATSs is known as alternating simulation. The simulation relation for fair transition systems is called fair simulation. In this work our main contributions are as follows: (1) We present an improved algorithm for fair simulation with Büchi fairness constraints; our algorithm requires $O(n^3 \cdot m)$ time as compared to the previous known $O(n^6)$-time algorithm, where n is the number of states and m is the number of transitions. (2) We present a game based algorithm for alternating simulation that requires $O(m^2)$-time as compared to the previous known $O((n \cdot m)^2)$-time algorithm, where n is the number of states and m is the size of transition relation. (3) We present an iterative algorithm for alternating simulation that matches the time complexity of the game based algorithm, but is more space efficient than the game based algorithm.

1998 ACM Subject Classification D.2.4 Formal methods

Keywords and phrases Simulation and fair simulation, Alternating simulation, Graph games

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.167

1 Introduction

Simulation relation and extensions. One central issue in formal design and analysis of reactive systems is the notion of refinement relations. The refinement relation (system A refines system A') intuitively means that every behavioral option of A (the implementation) is allowed by A' (the specification). The *local* interpretation of behavorial option in terms of successor states leads to refinement as *simulation* [15]. The simulation relation enjoys many appealing properties, such as it has a denotational characterization, it has a logical characterization and it can be computed in polynomial time (as compared to trace containment which is PSPACE-complete). While the notion of simulation was originally developed for transition systems [15], it has many important extensions. Two prominent extensions are as follows: (a) extension for composite systems and (b) extension for fair transition systems.

Alternating simulation relation. Composite reactive systems can be viewed as multi-agent systems [16, 9], where each possible step of the system corresponds to a possible move in a game which may involve some or all component moves. We model multi-agent systems as *alternating transition systems* (ATSs) [1]. In general a multi-agent system consists of a set I of agents, but for algorithmic purposes for simulation we always consider a subset $I' \subseteq I$ of agents against the rest, and thus we will only consider two-agent systems (one agent is the collection I' of agents, and the other is the collection of the rest of the agents). Consider the composite systems $A||B$ and $A'||B$, in environment B. The relation that A refines A'

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 167–182
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

without constraining the environment B is expressed by generalizing the simulation relation to *alternating simulation relation* [2]. Alternating simulation also enjoys the appealing properties of denotational and logical characterization along with polynomial time computability. We refer the readers to [2] for an excellent exposition of alternating simulation and its applications in design and analysis of composite reactive systems. We briefly discuss some applications of alternating simulation relation. Given a composite system with many components, the problem of refinement of a component (i.e., a component C can be replaced with its implementation C') without affecting the correctness of the composite system is an alternating simulation problem. Similarly, refinement for open reactive systems is also an alternating simulation problem. Finally, graph games provide the mathematical framework to analyze many important problems in computer science, specially in relation to logic, as there is a close connection of automata and graph games (see [17, 8] for details). Alternating simulation provides the technique for state space reduction for graph games, which is a pre-requisite for efficient algorithmic analysis of graph games. Thus computing alternating simulation for ATSs is a core algorithmic question in the formal analysis of composite systems, as well as in the heart of efficient algorithmic analysis of problems related to logic in computer science.

Fair simulation relation. Fair transition systems are extension of transition systems with fairness constraint. A *liveness* (or weak fairness or Büchi fairness) constraint consists of a set B of live states, and requires that runs of the system visit some live state infinitely often. In general the fairness constraint can be a strong fairness constraint instead of a liveness constraint. The notion of simulation was extended to fair transition systems as *fair simulation* [11]. It was shown in [11] that fair simulation also enjoys the appealing properties of denotational and logical characterization, and polynomial time computability (see [11] for many other important properties and discussion on fair simulation). Again the computation of fair simulation with Büchi fairness constraints is an important algorithmic problem for design and analysis of reactive systems with liveness requirements.

Our contributions. In this work we improve the algorithmic complexities of computing fair simulation with Büchi fairness constraints and alternating simulation. In the descriptions below we will denote by n the size of the state space of systems, and by m the size of the transition relation. Our main contributions are summarized below.

1. *Fair simulation.* First we extend the notion of fair simulation to alternating fair simulation for ATSs with Büchi fairness constraints. There are two natural ways of extending the definition of fair simulation to alternating fair simulation, and we show that both the definitions coincide. We present an algorithm to compute the alternating fair simulation relation by a reduction to a game with parity objectives with three priorities. As a special case of our algorithm for fair simulation, we show that the fair simulation relation can be computed in $O(n^3 \cdot m)$ time, as compared to the previous known $O(n^6)$-time algorithm of [11]. Observe that m is at most $O(n^2)$ and thus the worst case running time of our algorithm is $O(n^5)$. Moreover, in many practical examples systems have constant out-degree (for examples see [5]) (i.e., $m = O(n)$), and then our algorithm requires $O(n^4)$ time.
2. *Game based alternating simulation.* We present a game based algorithm for alternating simulation. Our algorithm is based on a reduction to a game with reachability objectives, and requires $O(m^2)$ time, as compared to the previous known algorithm that requires $O((n \cdot m)^2)$ time [2]. One key step of the reduction is to construct the game graph in time linear in the size of the game graph.
3. *Iterative algorithm for alternating simulation.* We present an iterative algorithm to com-

pute the alternating simulation relation. The time complexity of the iterative algorithm matches the time complexity of the game based algorithm, however, the iterative algorithm is more space efficient. (see paragraph on space complexity of Section 4.2 for the detailed comparision). Moreover, both the game based algorithm and the iterative algorithm when specialized to transition systems match the best known algorithms to compute the simulation relation.

We remark that the game based algorithms we obtain for alternating fair simulation and alternating simulation are reductions to standard two-player games on graphs with parity objectives (with three priorities) and reachability objectives. Since such games are well-studied, standard algorithms developed for games can now be used for computation of refinement relations. Our key technical contribution is establishing the correctness of the efficient reductions, and showing that the game graphs can be constructed in linear time in the size of the game graphs. For the iterative algorithm we establish an alternative characterization of alternating simulation, and present an iterative algorithm that simultaneously prunes two relations, without explicitly constructing game graphs (thus saving space), to compute the relation obtained by the alternative characterization. Proofs omitted due to lack of space are available in [4].

2 Definitions

In this section we present all the relevant definitions, and the previous best known results. We present definitions of labeled transition systems (Kripke structures), labeled alternating transitions systems (ATS), fair simulation, and alternating simulation. All the simulation relations we will define are closed under union (i.e., if two relations are simulation relations, then so is their union), and we will consider the maximum simulation relation. We also present relevant definitions for graph games that will be later used for the improved results.

▶ **Definition 1** (Labeled transition systems (TS)). A labeled *transition system (TS)* (Kripke structure) is a tuple $K = \langle \Sigma, W, \widehat{w}, R, L \rangle$, where Σ is a finite set of observations; W is a finite set of states and $\widehat{w}$ is the initial state; $R \subseteq W \times W$ is the transition relation; and $L : W \to \Sigma$ is the labeling function that maps each state to an observation. For technical convenience we assume that for all $w \in W$ there exists $w' \in W$ such that $(w, w') \in R$.

Runs, fairness constraint, and fair transition systems. For a TS K and a state $w \in W$, a w-run of K is an infinite sequence $\overline{w} = w_0, w_1, w_2, \ldots$ of states such that $w_0 = w$ and $R(w_i, w_{i+1})$ for all $i \geq 0$. We write $\mathsf{Inf}(\overline{w})$ for the set of states that occur infinitely often in the run $\overline{w}$. A run of K is a $\widehat{w}$-run for the initial state $\widehat{w}$. In this work we will consider *Büchi fairness constraints*, and a Büchi fairness constraint is specified as a set $F \subseteq W$ of Büchi states, and defines the fair set of runs, where a run $\overline{w}$ is *fair* iff $\mathsf{Inf}(\overline{w}) \cap F \neq \emptyset$ (i.e., the run visits F infinitely often). A *fair transition system* $\mathcal{K} = \langle K, F \rangle$ consists of a TS K and a Büchi fairness constraint $F \subseteq W$ for K. We consider two TSs $K_1 = \langle \Sigma, W_1, \widehat{w}_1, R_1, L_1 \rangle$ and $K_2 = \langle \Sigma, W_2, \widehat{w}_2, R_2, L_2 \rangle$ over the same alphabet, and the two fair TSs $\mathcal{K}_1 = \langle K_1, F_1 \rangle$ and $\mathcal{K}_2 = \langle K_2, F_2 \rangle$. We now define the fair simulation between $\mathcal{K}_1$ and $\mathcal{K}_2$ [11].

▶ **Definition 2** (Fair simulation). A binary relation $S \subseteq W_1 \times W_2$ is a *fair simulation* of $\mathcal{K}_1$ by $\mathcal{K}_2$ if the following two conditions hold for all $(w_1, w_2) \in W_1 \times W_2$:

1. If $S(w_1, w_2)$, then $L_1(w_1) = L_2(w_2)$.
2. There exists a strategy $\tau : (W_1 \times W_2)^+ \times W_1 \to W_2$ such that if $S(w_1, w_2)$ and $\overline{w} = u_0, u_1, u_2, \ldots$ is a fair w_1-run of $\mathcal{K}_1$, then the following conditions hold: (a) the outcome $\tau[\overline{w}] = u'_0, u'_1, u'_2, \ldots$ is a fair w_2-run of $\mathcal{K}_2$ (where the outcome $\tau[\overline{w}]$ is defined as follows:

for all $i \geq 0$ we have $u'_i = \tau((u_0, u'_0), (u_1, u'_1), \ldots, (u_{i-1}, u'_{i-1}), u_i))$; and (b) the outcome $\tau[\overline{w}]$ S-matches $\overline{w}$; that is, $S(u_i, u'_i)$ for all $i \geq 0$. We say τ is a *witness* to the fair simulation S.

We denote by $\preceq_{\mathsf{fair}}$ the maximum fair simulation relation between $\mathcal{K}_1$ and $\mathcal{K}_2$. We say that the fair TS $\mathcal{K}_2$ *fairly simulates* the fair TS $\mathcal{K}_1$ iff $(\widehat{w}_1, \widehat{w}_2) \in \preceq_{\mathsf{fair}}$.

We have the following result for fair simulation from [11] (see item 1 of Theorem 4.2 from [11]).

▶ **Theorem 3.** *Given two fair TSs $\mathcal{K}_1$ and $\mathcal{K}_2$, the problem of whether $\mathcal{K}_2$ fairly simulates $\mathcal{K}_1$ can be decided in time $O((|W_1| + |W_2|) \cdot (|R_1| + |R_2|) + (|W_1| \cdot |W_2|)^3)$.*

▶ **Definition 4** (Labeled alternating transition systems (ATS))**.** A labeled *alternating transitions system (ATS)* is a tuple $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$, where (i) Σ is a finite set of observations; (ii) W is a finite set of states with $\widehat{w}$ the initial state; (iii) A_i is a finite set of actions for Agent i, for $i \in \{1, 2\}$; (iv) $P_i : W \to 2^{A_i} \setminus \emptyset$ assigns to every state w in W the non-empty set of actions available to Agent i at w, for $i \in \{1, 2\}$; (v) $L : W \to \Sigma$ is the labeling function that maps every state to an observation; and (vi) $\delta : W \times A_1 \times A_2 \to W$ is the transition relation that given a state and the joint actions gives the next state.

Observe that a TS can be considered as a special case of ATS with A_2 singleton (say $A_2 = \{\bot\}$), and the transition relation R of a TS is described by the transition relation $\delta : W \times A_1 \times \{\bot\} \to W$ of the ATS.

▶ **Definition 5** (Alternating simulation)**.** Given two ATS, $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta' \rangle$ a binary relation $S \subseteq W \times W'$ is an alternating simulation from $\mathcal{K}$ to $\mathcal{K}'$ if for all states w and w' with $(w, w') \in S$, the following conditions hold :

1. $L(w) = L'(w')$
2. For every action $a \in P_1(w)$, there exists an action $a' \in P'_1(w')$ such that for every action $b' \in P'_2(w')$, there exists an action $b \in P_2(w)$ such that $(\delta(w, a, b), \delta'(w', a', b')) \in S$.

We denote by $\preceq_{\mathsf{altsim}}$ the maximum alternating simulation relation between K and K'. We say that the ATS K' *simulates* the ATS K iff $(\widehat{w}, \widehat{w}') \in \preceq_{\mathsf{altsim}}$.

The following result was shown in [2] (see proof of Theorem 3 of [2]).

▶ **Theorem 6.** *For two ATSs K and K', the alternating simulation relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W|^2 \cdot |W'|^2 \cdot |A_1| \cdot |A'_1| \cdot |A_2| \cdot |A'_2|)$.*

In the following section we will present an extension of the notion of fair simulation for TSs to alternating fair simulation for ATSs, and present improved algorithms to compute $\preceq_{\mathsf{fair}}$ and $\preceq_{\mathsf{altsim}}$. Some of our algorithms will be based on reduction to two-player games on graphs. We present the required definitions below.

Two-player Game graphs. A *two-player game graph* $G = ((V, E), (V_1, V_2))$ consists of a directed graph (V, E) with a set V of n vertices and a set E of m edges, and a partition (V_1, V_2) of V into two sets. The vertices in V_1 are *player 1 vertices*, where player 1 chooses the outgoing edges; and the vertices in V_2 are *player 2 vertices*, where player 2 (the adversary to player 1) chooses the outgoing edges. For a vertex $u \in V$, we write $\mathsf{O}ut(u) = \{v \in V \mid (u, v) \in E\}$ for the set of successor vertices of u and $\mathsf{I}n(u) = \{v \in V \mid (v, u) \in E\}$ for the set of incoming edges of u. We assume that every vertex has at least one out-going edge. i.e., $\mathsf{O}ut(u)$ is non-empty for all vertices $u \in V$.

Plays. A game is played by two players: player 1 and player 2, who form an infinite path in the game graph by moving a token along edges. They start by placing the token on an initial vertex, and then they take moves indefinitely in the following way. If the token is on a vertex in V_1, then player 1 moves the token along one of the edges going out of the vertex. If the token is on a vertex in V_2, then player 2 does likewise. The result is an infinite path in the game graph, called a *play*. We write Ω for the set of all plays.

Strategies. A strategy for a player is a rule that specifies how to extend plays. Formally, a *strategy* α for player 1 is a function α: $V^* \cdot V_1 \to V$ such that for all $w \in V^*$ and all $v \in V_1$ we have $\alpha(w \cdot v) \in \mathsf{O}ut(v)$, and analogously for player 2 strategies. We write $\mathcal{A}$ and $\mathcal{B}$ for the sets of all strategies for player 1 and player 2, respectively. A *memoryless* strategy for player 1 is independent of the history and depends only on the current state, and can be described as a function $\alpha : V_1 \to V$, and similarly for player 2. Given a starting vertex $v \in V$, a strategy $\alpha \in \mathcal{A}$ for player 1, and a strategy $\beta \in \mathcal{B}$ for player 2, there is a unique play, denoted $\omega(v, \alpha, \beta) = \langle v_0, v_1, v_2, \ldots \rangle$, which is defined as follows: $v_0 = v$ and for all $k \geq 0$, if $v_k \in V_1$, then $\alpha(v_k) = v_{k+1}$, and if $v_k \in V_2$, then $\beta(v_k) = v_{k+1}$. We say a play ω is *consistent* with a strategy of a player, if there is a strategy of the opponent such that given both the strategies the unique play is ω.

Objectives. An objective Φ for a game graph is a desired subset of plays. For a play $\omega = \langle v_0, v_1, v_2, \ldots \rangle \in \Omega$, we define $\mathsf{Inf}(\omega) = \{v \in V \mid v_k = v \text{ for infinitely many } k \geq 0\}$ to be the set of vertices that occur infinitely often in ω. We define reachability, safety and parity objectives with three priorities.

1. *Reachability and safety objectives.* Given a set $T \subseteq V$ of vertices, the reachability objective Reach(T) requires that some vertex in T be visited, and dually, the safety objective Safe(F) requires that only vertices in F be visited. Formally, the sets of winning plays are $\text{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \exists k \geq 0.\ v_k \in T\}$ and $\text{Safe}(F) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \forall k \geq 0.\ v_k \in F\}$. The reachability and safety objectives are dual in the sense that $\text{Reach}(T) = \Omega \setminus \text{Safe}(V \setminus T)$.
2. *Parity objectives with three priorities.* Consider a priority function $p : V \to \{0, 1, 2\}$ that maps every vertex to a priority either 0, 1 or 2. The parity objective requires that the minimum priority visited infinitely often is even. In other words, the objectives require that either vertices with priority 0 are visited infinitely often, or vertices with priority 1 are visited finitely often. Formally the set of winning plays is $\text{Parity}(p) = \big\{\omega \mid \mathsf{Inf}(\omega) \cap p^{-1}(0) \neq \emptyset \text{ or } \mathsf{Inf}(\omega) \cap p^{-1}(1) = \emptyset\big\}$.

Winning strategies and sets. Given an objective $\Phi \subseteq \Omega$ for player 1, a strategy $\alpha \in \mathcal{A}$ is a *winning strategy* for player 1 from a vertex v if for all player 2 strategies $\beta \in \mathcal{B}$ the play $\omega(v, \alpha, \beta)$ is winning, i.e., $\omega(v, \alpha, \beta) \in \Phi$. The winning strategies for player 2 are defined analogously by switching the role of player 1 and player 2 in the above definition. A vertex $v \in V$ is winning for player 1 with respect to the objective Φ if player 1 has a winning strategy from v. Formally, the set of *winning vertices for player 1* with respect to the objective Φ is the set $W_1(\Phi) = \{v \in V \mid \exists \alpha \in \mathcal{A}.\ \forall \beta \in \mathcal{B}.\ \omega(v, \alpha, \beta) \in \Phi\}$. Analogously, the set of all winning vertices for player 2 with respect to an objective $\Psi \subseteq \Omega$ is $W_2(\Psi) = \{v \in V \mid \exists \beta \in \mathcal{B}.\ \forall \alpha \in \mathcal{A}.\ \omega(v, \alpha, \beta) \in \Psi\}$.

▶ **Theorem 7** (Determinacy and complexity). *The following assertions hold.*

1. *For all game graphs $G = ((V, E), (V_1, V_2))$, all objectives Φ for player 1 where Φ is reachability, safety, or parity objectives with three priorities, and the complementary objective $\Psi = \Omega \setminus \Phi$ for player 2, we have $W_1(\Phi) = V \setminus W_2(\Psi)$; and memoryless winning strategies exist for both players from their respective winning set [7].*
2. *The winning set $W_1(\Phi)$ can be computed in linear time ($O(|V| + |E|)$) for reachability*

and safety objectives Φ [12, 3]; and in quadratic time $(O(|V| \cdot |E|))$ for parity objectives with three priorities [13].

3 Fair Alternating Simulation

In this section we will present two definitions of fair alternating simulation, show their equivalence, present algorithms for solving fair alternating simulations, and our algorithms specialized to fair simulation will improve the bound of the previous algorithm (Theorem 3). Similar to fair TSs, a *fair ATS* $\mathcal{K} = \langle K, F\rangle$ consists of an ATS K and a Büchi fairness constraint F for K.

To extend the definition of fair simulation to fair alternating simulation we consider the notion of strategies for ATSs. Consider two ATSs $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta\rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta'\rangle$ and the corresponding fair ATSs $\mathcal{K} = \langle K, F\rangle$ and $\mathcal{K}' = \langle K', F'\rangle$. We use the following notations:

- $\tau : (W \times W')^+ \to A_1$ is a strategy employed by Agent 1 in $\mathcal{K}$. The aim of the strategy is to choose transitions in $\mathcal{K}$ to make it difficult for Agent 1 in $\mathcal{K}'$ to match them. The strategy acts on the past run on both systems.
- $\tau' : (W \times W')^+ \times A_1 \to A'_1$ is a strategy employed by Agent 1 in $\mathcal{K}'$. The aim of this strategy is to match actions in $\mathcal{K}'$ to those made by Agent 1 in $\mathcal{K}$. The strategy acts on the past run on both the systems, as well as the action chosen by Agent 1 in $\mathcal{K}$.
- $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ is a strategy employed by Agent 2 in $\mathcal{K}'$. The aim of this strategy is to choose actions in $\mathcal{K}'$ to make it difficult for Agent 2 to match them in $\mathcal{K}$. The strategy acts on the past run of both the systems, as well as the actions chosen by Agent 1 in $\mathcal{K}$ and $\mathcal{K}'$.
- $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A'_2 \to A_2$ is a strategy employed by Agent 2 in $\mathcal{K}$. Intuitively, the aim of this strategy of Agent 2 is to choose actions in $\mathcal{K}$ to show that Agent 1 is not as powerful in $\mathcal{K}$ as in $\mathcal{K}'$, i.e., in some sense the strategy of Agent 2 will witness that the strategy of Agent 1 in $\mathcal{K}$ does not satisfy certain desired property. The strategy acts on the past run of both the systems, as well as the actions chosen by Agent 1 in $\mathcal{K}$ and both the agents in $\mathcal{K}'$.
- $\rho(w, w', \tau, \tau', \xi, \xi')$ is the run that emerges in $\mathcal{K}$ if the game starts with $\mathcal{K}$ on state w, $\mathcal{K}'$ on state w' and the agents employ strategies τ, τ', ξ and ξ' as described above, and $\rho'(w, w', \tau, \tau', \xi, \xi')$ is the corresponding run that emerges in $\mathcal{K}'$.

▶ **Definition 8** (Weak fair alternating simulation)**.** A binary relation $S \subseteq W \times W'$ is a *weak fair alternating simulation (WFAS)* of $\mathcal{K}$ by $\mathcal{K}'$ if the following two conditions hold for all $(w, w') \in W \times W'$:

1. If $S(w, w')$, then $L(w) = L'(w')$.
2. There exists a strategy $\tau' : (W \times W')^+ \times A_1 \to A'_1$ for Agent 1 in $\mathcal{K}'$, such that for all strategies $\tau : (W \times W')^+ \to A_1$ for Agent 1 in $\mathcal{K}$, there exists a strategy $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A_2 \to A'_2$ for Agent 2 in $\mathcal{K}$, such that for all strategies $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ for Agent 2 on $\mathcal{K}'$, if $S(w, w')$ and $\rho(w, w', \tau, \tau', \xi, \xi')$ is a fair w-run of $\mathcal{K}$, then
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ is a fair w'-run of $\mathcal{K}'$; and
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ S-matches $\rho(w, w', \tau, \tau', \xi, \xi')$.

We denote by $\preceq^{\mathsf{weak}}_{\mathsf{fairalt}}$ the maximum WFAS relation between $\mathcal{K}$ and $\mathcal{K}'$. We say that the fair ATS $\mathcal{K}'$ *weak-fair-alternate simulates* the fair ATS $\mathcal{K}$ iff $(\widehat{w}, \widehat{w}') \in \preceq^{\mathsf{weak}}_{\mathsf{fairalt}}$.

▶ **Definition 9** (Strong fair alternating simulation)**.** A binary relation $S \subseteq W \times W'$ is a *strong fair alternating simulation (SFAS)* of $\mathcal{K}$ by $\mathcal{K}'$ if the following two conditions hold for all $(w, w') \in W \times W'$:

1. If $S(w, w')$, then $L(w) = L'(w')$.
2. There exist strategies $\tau' : (W \times W')^+ \times A_1 \to A'_1$ for Agent 1 in $\mathcal{K}'$ and $\xi : (W \times W')^+ \times A_1 \times A'_1 \times A_2 \to A'_2$ for Agent 2 in $\mathcal{K}$, such that for all strategies $\tau : (W \times W')^+ \to A_1$ for Agent 1 in $\mathcal{K}$ and $\xi' : (W \times W')^+ \times A_1 \times A'_1 \to A'_2$ for Agent 2 on $\mathcal{K}'$, if $S(w, w')$ and $\rho(w, w', \tau, \tau', \xi, \xi')$ is a fair w-run of $\mathcal{K}$, then
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ is a fair w'-run of $\mathcal{K}'$; and
   - $\rho'(w, w', \tau, \tau', \xi, \xi')$ S-matches $\rho(w, w', \tau, \tau', \xi, \xi')$.

We denote by $\preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$ the maximum SFAS relation between $\mathcal{K}$ and $\mathcal{K}'$. We say that the fair ATS $\mathcal{K}'$ *strong-fair-alternate simulates* the fair ATS $\mathcal{K}$ iff $(\widehat{w}, \widehat{w}') \in \preceq^{\mathsf{strong}}_{\mathsf{fairalt}}$.

The difference in the definitions of weak and strong alternating fair simulation is in the order of the quantifiers in the strategies. In the weak version the quantifier order is exists forall exists forall, whereas in the strong version the order is exists exists forall forall. Thus strong fair alternating simulation implies weak fair alternating simulation. We will show that both the definitions coincide and present algorithms to compute the maximum fair alternating simulation. Also observe that both the weak and strong version coincide with fair simulation for TSs. We will present a reduction of weak and strong fair alternating simulation problem to games with parity objectives with three priorities. We now present a few notations related to the reduction.

Successor sets. Given an ATS K, for a state w and an action $a \in P_1(w)$, let $\mathsf{Succ}(w, a) = \{w' \mid \exists b \in P_2(w) \text{ such that } w' = \delta(w, a, b)\}$ denote the possible successors of w given an action a of Agent 1 (i.e., successor set of w and a). Let $\mathsf{Succ}(K) = \{\mathsf{Succ}(w, a) \mid w \in W, a \in P_1(w)\}$ denote the set of all possible successor sets. Note that $|\mathsf{Succ}(K)| \leq |W| \cdot |A_1|$.

Game construction. Let $K = \langle \Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta \rangle$ and $K' = \langle \Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta' \rangle$ be two ATSs, and let $\mathcal{K} = \langle K, F \rangle$ and $\mathcal{K}' = \langle K', F' \rangle$ be the two corresponding fair ATSs. We will construct a game graph $G = ((V, E), (V_1, V_2))$ with a parity objective. Before the construction we assume that from every state $w \in K$ there is an Agent-1 strategy to ensure fairness in K. The assumption is without loss of generality because if there is no such strategy from w, then trivially all states w' with same label as w simulate w (as Agent 2 can falsify the fairness from w). The states in K from which fairness cannot be ensured can be identified with a quadratic time pre-processing step in K (solving Büchi games), and hence we assume that in all remaining states in K fairness can be ensured. The game construction is as follows:

- *Player 1 vertices:* $V_1 = \{\langle w, w' \rangle \mid w \in W, w' \in W' \text{ such that } L(w) = L'(w')\} \cup \big(\mathsf{Succ}(K) \times \mathsf{Succ}(K')\big) \cup \{\frownie\}$
- *Player 2 vertices:* $V_2 = \mathsf{Succ}(K) \times W' \times \{\#, \$\}$
- *Edges.* We specify the edges as the following union: $E = E_1 \cup E_2 \cup E_3 \cup E^1_4 \cup E^2_4 \cup E_5$

$$\begin{aligned}
E_1 &= \{(\langle w, w' \rangle, \langle \mathsf{Succ}(w, a), w', \# \rangle) \mid \langle w, w' \rangle \in V_1, a \in P_1(w)\} \\
E_2 &= \{(\langle T, w', \# \rangle, \langle T, \mathsf{Succ}(w', a') \rangle) \mid \langle T, w', \# \rangle \in V_2, a' \in P'_1(w')\} \\
E_3 &= \{(\langle T, T' \rangle, \langle T, r', \$ \rangle) \mid \langle T, T' \rangle \in V_1, r' \in T'\} \\
E^1_4 &= \{(\langle T, r', \$ \rangle, \langle r, r' \rangle) \mid \langle T, r', \$ \rangle \in V_2, r \in T, L(r) = L'(r')\} \\
E^2_4 &= \{(\langle T, r', \$ \rangle, \frownie) \mid \langle T, r', \$ \rangle \in V_2 \text{ such that } \forall r \in T \cdot L(r) \neq L'(r')\} \\
E_5 &= \{(\frownie, \frownie)\}
\end{aligned}$$

The intuitive description of the game graph is as follows: (i) the player 1 vertices are either state pairs $\langle w, w' \rangle$ with same label, or pairs $\langle T, T' \rangle$ of successor sets, or a state ☹; and (ii) the player 2 vertices are tuples $\langle T, w', \bowtie \rangle$ where T is a successor set in $\mathsf{Succ}(K)$, w' a state in K' and $\bowtie \in \{\#, \$\}$. The edges are described as follows: (i) E_1 describes that in vertices $\langle w, w' \rangle$ player 1 can choose an action $a \in P_1(w)$, and then the next vertex is the player 2 vertex $\langle \mathsf{Succ}(w, a), w', \# \rangle$; (ii) E_2 describes that in vertices $\langle T, w', \# \rangle$ player 2 can choose an action $a' \in P_1(w')$ and then the next vertex is $\langle T, \mathsf{Succ}(w', a') \rangle$; (iii) E_3 describes that in states $\langle T, T' \rangle$ player 1 can choose a state $r' \in T'$ (which intuitively corresponds to an action $b' \in P_2'(w')$) and then the next vertex is $\langle T, r', \$ \rangle$; (iv) the edges $E_4^1 \cup E_4^2$ describes that in states $\langle T, r', \$ \rangle$ player 2 can either choose a state $r \in T$ that matches the label of r' and then the next vertex is the player 1 vertex $\langle r, r' \rangle$ (edges E_4^1) or if there is no match, then the next vertex is ☹; and (v) finally E_5 specifies that the vertex ☹ is an absorbing (sink) vertex with only self-loop. The three-priority parity objective Φ^* for player 2 with the priority function p is specified as follows: for vertices $v \in (W \times F') \cap V_1$ we have $p(v) = 0$; for vertices $v \in ((F \times W' \setminus W \times F') \cap V_1) \cup \{☹\}$ we have $p(v) = 1$; and all other vertices have priority 2. The following proposition establishes the equivalence of the winning set for player 2 with strong and weak fair alternating simulation.

▶ Proposition 10. Let $\mathsf{Win}_2 = \{(w_1, w_2) \mid \langle w_1, w_2 \rangle \in V_1, \langle w_1, w_2 \rangle \in W_2(\Phi^*)\}$ be the winning set for player 2. Then we have $\mathsf{Win}_2 = \preceq_{\mathsf{fairalt}}^{\mathsf{weak}} = \preceq_{\mathsf{fairalt}}^{\mathsf{strong}}$.

▶ **Lemma 11.** *For the game graph constructed for fair alternating simulation we have* $|V_1| + |V_2| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot |A_1'|)$*; and* $|E| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot (|A_1'| \cdot |A_2'| + |A_2|))$*.*

The above lemma bounds the size of the game, and we require that the game graph can be constructed in time quadratic in the size of the game graph and in the following section we will present a more efficient (than quadratic) construction of the game graph. Proposition 10, along with the complexity to solve parity games with three priorities gives us the following theorem. The result for fair simulation follows as a special case and the details are presented in [4].

▶ **Theorem 12.** *We have* $\preceq_{\mathsf{fairalt}}^{\mathsf{weak}} = \preceq_{\mathsf{fairalt}}^{\mathsf{strong}}$*, the relation* $\preceq_{\mathsf{fairalt}}^{\mathsf{strong}}$ *can be computed in time* $O(|W|^2 \cdot |W'|^2 \cdot |A_1|^2 \cdot |A_1'| \cdot (|A_1'| \cdot |A_2'| + |A_2|))$ *for two fair ATSs* $\mathcal{K}$ *and* $\mathcal{K}'$*. The fair simulation relation* $\preceq_{\mathsf{fair}}$ *can be computed in time* $O(|W| \cdot |W'| \cdot (|W| \cdot |R'| + |W'| \cdot |R|))$ *for two fair TSs* $\mathcal{K}$ *and* $\mathcal{K}'$*.*

▶ Remark. We consider the complexity of fair simulation, and let $n = |W| = |W'|$ and $m = |R| = |R'|$. The previous algorithm of [11] requires time $O(n^6)$ and our algorithm requires time $O(n^3 \cdot m)$. Since m is at most n^2, our algorithm takes in worst case time $O(n^5)$ and in most practical cases we have $m = O(n)$ and then our algorithm requires $O(n^4)$ time as compared to the previous known $O(n^6)$ algorithm.

4 Alternating Simulation

In this section we will present two algorithms to compute the maximum alternating simulation relation for two ATSs K and K'. The first algorithm for the problem was presented in [2] and we refer to the algorithm as the basic algorithm. The basic algorithm iteratively considers pairs of states and examines if they are already witnessed to be not in the alternating simulation relation, removes them and continues until a fix-point is reached (see Theorem 3 of [2]). The correctness of the basic algorithm was shown in [2], and the time complexity of the algorithm is $O(|W|^2 \cdot |W'|^2 \cdot |A_1| \cdot |A_1'| \cdot |A_2| \cdot |A_2'|)$ (see fuller version for further explanation).

4.1 Improved Algorithm Through Games

In this section we present an improved algorithm for alternating simulation by reduction to reachability-safety games.

Game construction. Given two ATSs $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$ and $K' = (\Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta')$, we construct a game graph $G = ((V, E), (V_1, V_2))$ as follows:

- *Player 1 vertices:* $V_1 = (W \times W') \cup \big(\mathsf{Succ}(K) \times \mathsf{Succ}(K')\big)$;
- *Player 2 vertices:* $V_2 = \mathsf{Succ}(K) \times W' \times \{\#, \$\}$;
- *Edges:* The edge set E is specified as the following union: $E = E_1 \cup E_2 \cup E_3 \cup E_4$

$$\begin{aligned}
E_1 &= \{(\langle w, w'\rangle, \langle \mathsf{Succ}(w, a), w', \#\rangle) \mid w \in W, w' \in W', a \in P_1(w)\} \\
E_2 &= \{(\langle T, w', \#\rangle, \langle T, \mathsf{Succ}(w', a')\rangle) \mid T \in \mathsf{Succ}(K), w' \in W', a' \in P'_1(w')\} \\
E_3 &= \{(\langle T, T'\rangle, \langle T, r', \$\rangle) \mid T \in \mathsf{Succ}(K), T' \in \mathsf{Succ}(K'), r' \in T'\} \\
E_4 &= \{(\langle T, r', \$\rangle, \langle r, r'\rangle) \mid T \in \mathsf{Succ}(K), r' \in W', r \in T\}
\end{aligned}$$

Let $T = \{\langle w, w'\rangle \mid L(w) \neq L'(w')\}$ be the state pairs that does not match by the labeling function, and let $F = V \setminus T$. The objective for player 1 is to reach T (i.e., $\text{Reach}(T)$) and the objective for player 2 is the safety objective $\text{Safe}(F)$. In the following proposition we establish the connection of the winning set for player 2 and $\preceq_{\mathsf{altsim}}$.

▶ Proposition 13. Let $\mathsf{Win}_2 = \{(w, w') \mid w \in W, w' \in W', \langle w, w'\rangle \in W_2(\text{Safe}(F))\}$ Then we have $\mathsf{Win}_2 = \preceq_{\mathsf{altsim}}$.

The algorithmic analysis will be completed in two steps: (1) estimating the size of the game graph; and (2) analyzing the complexity to construct the game graph from the ATSs.

▶ **Lemma 14.** *For the game graph constructed for alternating simulation, we have* $|V_1| + |V_2| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot |A'_1|)$ *and* $|E| \leq O(|W| \cdot |W'| \cdot |A_1| \cdot (|A'_1| \cdot |A'_2| + |A_2|))$.

Game graph construction complexity. We now show that the game graph can be constructed in time linear in the size of the game graph. The data strucutre for the game graph is as follows: we map every vertex in $V_1 \cup V_2$ to a unique integer, and construct the list of edges. Given this data structure for the game graph, the winning sets for reachability and safety objectives can be computed in linear time [3, 12]. We now present the details of the construction of the game graph data structure.

Basic requirements. We start with some basic facts. For two sets A and B, if we have two bijective functions $f_A : A \leftrightarrow \{0, \ldots, |A| - 1\}$ and $f_B : B \leftrightarrow \{0, \ldots, |B| - 1\}$, then we can assign a unique integer to elements of $A \times B$ in time $O(|A| \cdot |B|)$. Since it is easy to construct bijective functions for W and W', we need to construct such bijective functions for $\mathsf{Succ}(K)$ and $\mathsf{Succ}(K')$ to ensure that every vertex has a unique integer. We will present data structure that would achieve the following: (i) construct bijective function $f_K : \mathsf{Succ}(K) \leftrightarrow \{0, \ldots, |\mathsf{Succ}(K)| - 1\}$; (ii) construct function $h_K : W \times A_1 \to \{0, \ldots, |\mathsf{Succ}(K)| - 1\}$ such that for all $w \in W$ and $a \in P_1(w)$ we have $h_K((w, a)) = f_K(\mathsf{Succ}(w, a))$, i.e., it gives the unique number for the successor set of w and action a; (iii) construct function $g_K : \{0, 1, \ldots, |\mathsf{Succ}(K)| - 1\} \to 2^W$ such that for all $T \in \mathsf{Succ}(K)$ we have $g_K(f_K(T))$ is the list of states in T. We will construct the same for K', and also ensure that for all T we compute $g_K(f_K(T))$ in time proportional to the size of T. We first argue how the above functions are sufficient to construct every edge in constant time: (i) edges in E_1 can be constructed by considering state pairs $\langle w, w'\rangle$, actions $a \in P_1(w)$, and with the function $h_K((w, a))$ we add the required edge, and the result for edges E_2 is similar with the function $h_{K'}$; (ii) edges in E_3 and E_4 are generated using the function g_K that gives the list of states for $g_K(f_K(T))$ in time proportional to the size of T. Hence every edge can be generated in constant time, given the functions, and it follows that with the above functions the game construction is achieved in linear time. We now present the data structure to support the above functions.

Binary tree data structure. Observe that $\mathsf{Succ}(K)$ is a set such that each element is a successor set (i.e., elements are set of states). Without efficient data structure the requirements for the functions f_K, h_K, and g_K cannot be achieved. The data structure we use is a *binary tree data structure*. We assume that states in W are uniquely numbered from 1 to $|W|$. Consider a binary tree, such that every leaf has depth $|W|$, i.e., the length of the path from root to a leaf is $|W|$. Each path from the root to a leaf represents a set — every path consists of a $|W|$ length sequence of *left* and *right* choices. Consider a path π in the binary tree, and the path π represents a subset W_π of W as follows: if the i-th step of π is *left*, then $w_i \notin W_\pi$, if the i-th step is *right*, then $w_i \in W_\pi$. Thus, $\mathsf{Succ}(K)$ is the collection of all sets represented by paths (from root to leaves) in this tree. We have several steps and we describe them below.

1. *Creation of binary tree.* The binary tree BT is created as follows. Initially the tree BT is empty. For all $w \in W$ and all $a \in P_1(w)$ we generate the set $\mathsf{Succ}((w,a))$ as a Boolean array Ar of length $|W|$ such that $\mathsf{Ar}[i] = 1$ if $w_i \in \mathsf{Succ}(w,a)$ and 0 otherwise. We use the array Ar to add the set $\mathsf{Succ}((w,a))$ to BT as follows: we proceed from the root, if $\mathsf{Ar}[0] = 0$ we add left edge, else the right edge, and proceed with $\mathsf{Ar}[1]$ and so on. For every $w \in W$ and $a \in P_1(w)$, the array Ar is generated by going over actions in $P_2(w)$, and the addition of the set $\mathsf{Succ}(w,a)$ to the tree is achieved in $O(|W|)$ time. The initialization of array Ar also requires time $O(|W|)$. Hence the total time required is $O(|W| \cdot |A_1| \cdot (|W| + |A_2|))$. The tree has at most $|W| \cdot |A_1|$ leaves and hence the size of the tree is $O(|W|^2 \cdot |A_1|)$.
2. *The functions f_K, g_K and h_K.* Let Lf denote the leaves of the tree BT, and note that every leaf represents an element of $\mathsf{Succ}(K)$. We do a DFT (depth-first traversal) of the tree BT and assign every leaf the number according to the order of leaves in which it appears in the DFT. Hence the function f_K is constructed in time $O(|W|^2 \cdot |A_1|)$. Moreover, when we construct the function f_K, we create an array GAr of lists for the function g_K. If a leaf is assigned number i by f_K, we go from the leaf to the root and find the set $T \in \mathsf{Succ}(K)$ that the leaf represents and $\mathsf{GAr}[i]$ is the list of states in T. Hence the construction of g_K takes time at most $O(|W| \cdot |A_1| \cdot |W|)$. The function h_K is stored as a two-dimensional array of integers with rows indexed by numbers from 0 to $|W| - 1$, and columns by numbers 0 to $|A_1| - 1$. For a state w and action a, we generate the Boolean array Ar, and use the array Ar to traverse BT, obtain the leaf for $\mathsf{Succ}((w,a))$, and assign $h_K((w,a)) = f_K(\mathsf{Succ}(w,a))$. It follows that h_K is generated in time $O(|W| \cdot |A_1| \cdot (|W| + |A_2|))$.

From the above graph construction, Proposition 13, Lemma 14, and the linear time algorithms to solve games with reachability and safety objectives we have the following result for computing alternating simulation.

▶ **Theorem 15.** *The relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W| \cdot |W'| \cdot |A_1| \cdot (|A'_1| \cdot |A'_2| + |A_2|) + |W|^2 \cdot |A_1| + |W'|^2 \cdot |A'_1|)$ for two ATSs K and K'. The relation $\preceq_{\mathsf{altsim}}$ can be computed in time $O(|W| \cdot |R'| + |W'| \cdot |R|)$ for two TSs K and K'.*

The result for the special case of TSs is obtained by noticing that for TSs we have both $|V|$ and $|E|$ at most $|W| \cdot |R'| + |W'| \cdot |R|$ (see [4] for details), and our algorithm matches the complexity of the best known algorithm of [10] for simulation for transition systems. Let us denote by $n = |W|$ and $n' = |W'|$ the size of the state spaces, and by $m = |W| \cdot |A_1| \cdot |A_2|$ and $m' = |W'| \cdot |A_1'| \cdot |A_2'|$ the size of the transition relations. Then the basic algorithm requires $O(n \cdot n' \cdot m \cdot m')$ time, whereas our algorithm requires at most $O(m \cdot m' + n \cdot m + n' \cdot m')$ time, and when $n = n'$ and $m = m'$, then the basic algorithm requires $O((n \cdot m)^2)$ time and our algorithm takes $O(m^2)$ time.

4.2 Iterative Algorithm

In this section we will present an iterative algorithm for alternating simulation. For our algorithm we will first present a new and alternative characterization of alternating simulation through successor set simulation.

▶ **Definition 16** (Successor set simulation)**.** Given two ATSs $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$ and $K' = (\Sigma, W', \widehat{w}', A_1', A_2', P_1', P_2', L', \delta')$, a relation $\approxeq \subseteq W \times W'$ is a *successor set simulation* from K to K', if there exists a companion relation $\approxeq^S \subseteq \mathsf{Succ}(K') \times \mathsf{Succ}(K)$, such that the following conditions hold:

- for all $(w, w') \in \approxeq$ we have $L(w) = L'(w')$;
- if $(w, w') \in \approxeq$, then for all actions $a \in P_1(w)$, there exists an action $a' \in P_1'(w')$ such that $(\mathsf{Succ}(w', a'), \mathsf{Succ}(w, a)) \in \approxeq^S$; and
- if $(T', T) \in \approxeq^S$, then for all $r' \in T'$, there exists $r \in T$ such that $(r, r') \in \approxeq$.

We denote by $\approxeq^*$ the maximum successor set simulation.

The following lemma shows that successor set simulation and alternating simulation coincide. We present the iterative algorithm to compute the maximum successor set simulation $\approxeq^*$.

▶ **Lemma 17.** *The following assertions hold: (1) Every successor set simulation is an alternating simulation, and every alternating simulation is a successor set simulation. (2) We have* $\approxeq^* = \preceq_{\mathsf{altsim}}$.

We will now present our iterative algorithm to compute $\approxeq^*$, and we will denote by $\approxeq^S$ the witness companion relation of $\approxeq^*$. Our algorithm will use the following graph construction: Given an ATS K, we will construct the graph $G_K = (V_K, E_K)$ as follows: (1) $V_K = W \cup \mathsf{Succ}(K)$, where W is the set of states; and (2) $E_K = \{(w, \mathsf{Succ}(w, a)) \mid w \in W \wedge a \in P_1(w)\} \cup \{(T, r) \mid T \in \mathsf{Succ}(K) \wedge r \in T\}$. The graph G_K can be constructed in time $O(|W|^2 \cdot |A_1|)$ using the binary tree data structure described earlier. Our algorithm will use the standard notation of Pre and Post: given a graph $G = (V, E)$, for a set U of states, $\mathsf{Post}(U) = \{v \mid \exists u \in U, (u, v) \in E\}$ is the set of successor states of U, and similarly, $\mathsf{Pre}(U) = \{v \mid \exists u \in U, (v, u) \in E\}$ is the set of predecessor states. If $U = \{q\}$ is singleton, we will write $\mathsf{Post}(q)$ instead of $\mathsf{Post}(\{q\})$. Note that in the graph G_K for the state $T \in \mathsf{Succ}(K)$ we have $\mathsf{Post}(T) = \{q \mid q \in T\} = T$. Given ATSs K and K' our algorithm will work simultaneously on the graphs G_K and $G_{K'}$ using three data structures, namely, sim, count and remove for the relation $\approxeq^*$ (resp. sim^S, count^S and remove^S for the companion relation $\approxeq^S$). The data structures are as follows: (1) Intuitively sim will be an overapproximation of $\approxeq^*$, and will be maintained as a two-dimensional Boolean array so that whenever the i,j-th entry is false, then we have a witness that the j-th state w_j' of K' does not simulate the i-th state w_i of K (similary we have sim^S over $\mathsf{Succ}(K')$ and $\mathsf{Succ}(K)$ for the relation $\approxeq^S$). (2) The data structure count is two-dimensional array, such that for a state $w' \in W'$ and $T \in \mathsf{Succ}(K)$ we have $\mathsf{count}(w', T)$ is the number of elements in the intersection of the successor states of w' and the set of all states that T simulates according to sim^S; and we also have similar array count^S for T, w' elements. (3) Finally, the data structure remove is a list of sets, where for every $w' \in W'$ we have $\mathsf{remove}(w')$ is a set such that every element of the set belongs to $\mathsf{Succ}(K)$. Similarly for every $T \in \mathsf{Succ}(K)$ we have $\mathsf{remove}^S(T)$ is a set of states. Intuitively the interpretation of remove data structure will be as follows: if $T \in \mathsf{Succ}(K)$ belongs to $\mathsf{remove}(w')$, then no element w of T is simulated by w'. Our algorithm will always maintain sim (resp. sim^S) as overapproximation of $\approxeq^*$ (resp. $\approxeq^S$), and will iteratively prune them. Our algorithm is iterative and we denote by $\mathsf{prevsim}$ (resp. $\mathsf{prevsim}^S$) the sim (resp. sim^S) of the previous iteration. To give an intuitive idea of the invariants maintained by the algorithm (Algorithm 1) let us denote by $\mathsf{sim}(w)$ the set of w' such that $\mathsf{sim}(w, w')$ is true, and let us denote by $\mathsf{invsim}(w')$ the inverse of $\mathsf{sim}(w')$, i.e., the set of states w such that (w, w')-th element of sim is true (similar notation for $\mathsf{invprevsim}(w'), \mathsf{invsim}^S(T)$ and $\mathsf{invprevsim}^S(T)$). The algorithm will ensure the following invariants at different steps:

1. For $w \in W, w' \in W'$ and $T \in \mathsf{Succ}(K), T' \in \mathsf{Succ}(K')$,
   a. if $\mathsf{sim}(w, w')$ is false, then $(w, w') \notin \approxeq^*$;
   b. similarly, if $\mathsf{sim}^S(T', T)$ is false, then $(T', T) \notin \approxeq^S$.
2. For $w' \in W'$ and $T \in \mathsf{Succ}(K)$,
   a. $\mathsf{count}(w', T) = |\mathsf{Post}(w') \cap \mathsf{invsim}^S(T)|$; and
   b. $\mathsf{count}(T, w') = |\mathsf{Post}(T) \cap \mathsf{invsim}(w')| = |T \cap \mathsf{invsim}(w')|$
3. For $w' \in W'$ and $T \in \mathsf{Succ}(K)$,
   a. $\mathsf{remove}(w') = \mathsf{Pre}(\mathsf{invprevsim}(w')) \setminus \mathsf{Pre}(\mathsf{invsim}(w'))$
   b. $\mathsf{remove}(T) = \mathsf{Pre}(\mathsf{invprevsim}^S(T)) \setminus \mathsf{Pre}(\mathsf{invsim}^S(T))$.

The algorithm has two phases: the initialization phase, where the data structures are initialized; and then a while loop. The while loop consists of two parts: one is pruning of sim and the other is the pruning of sim^S and both the pruning steps are similar. The initialization phase initializes the data structures and is described in Steps 1, 2, and 3 of Algorithm 1. Then the algorithm calls the two pruning steps in a while loop. The condition of the while loop checks whether $\mathsf{prevsim}$ and sim are the same, and it is done in constant time by simply checking whether remove is empty. We now describe one of the pruning procedures and the other is similar. The pruning step is similar to the pruning step of the algorithm of [10] for simulation on transition systems. We describe the pruning procedure PruneSimStrSucc. For every state $w' \in W'$ such that the set $\mathsf{remove}(w')$ is non-empty, we run a for loop. In the for loop we first obtain the predecessors T' of w' in $G_{K'}$ (each predecessor belongs to $\mathsf{Succ}(K')$) and an element T from $\mathsf{remove}(w')$. If $\mathsf{sim}^S(T', T)$ is true, then we do the following steps: (i) We set $\mathsf{sim}^S(T', T)$ to false, because we know that there does not exist any element $w \in T$ such that w' simulates w. (ii) Then for all s' that are predecessors of T' in $G_{K'}$ we decrement $\mathsf{count}(s', T)$, and if the count is zero, then we add s' to the remove set of T. Finally we set the remove set of w' to $\emptyset$. The description of PruneSimStr to prune sim is similar.

Correctness. Our correctness proof will be in two steps. First we will show that invariant 1 (both about sim and sim^S) and invariant 2 (both about count and count^S) are true at the beginning of step 4.1. The invariant 3.(a) (on remove) is true after the procedure call PruneSimStr (step 4.4) and invariant 3.(b) (on remove^S) is true after the procedure call PruneSimStrSucc (step 4.3). We will then argue that these invariants ensure correctness of the algorithm.

Maintaining invariants. We first consider invaraint 1, and focus on invariant 1.(b) (as the other case is symmetric). In procedure PruneSimStrSucc when we set $\mathsf{sim}^S(T', T)$ to false, we need to show that $(T', T) \notin \approxeq^S$. The argument is as follows: when we set $\mathsf{sim}^S(T', T)$

Algorithm 1 Iterative Algorithm

```
Input: $K = (\Sigma, W, \widehat{w}, A_1, A_2, P_1, P_2, L, \delta)$, $K' = (\Sigma, W', \widehat{w}', A'_1, A'_2, P'_1, P'_2, L', \delta')$.
Output: $\approxeq^*$.
1. Initialize sim and sim$^S$:
    1.1. for all $w \in W, w' \in W'$
            prevsim$(w, w') \leftarrow$ true;
            if $L(w) = L'(w')$, then sim$(w, w') \leftarrow$ true;
            else sim$(w, w') \leftarrow$ false;
    1.2. for all $T \in$ Succ$(K)$ and $T' \in$ Succ$(K')$
            prevsim$^S(T', T) =$ sim$^S(T', T) \leftarrow$ true;
2. Initialize count and count$^S$:
    2.1. for all $w' \in W'$ and $T \in$ Succ$(K)$
            count$(w', T) \leftarrow |$Post$(w') \cap$ invsim$^S(T)| = |$Post$(w')|$;
            count$^S(T, w') \leftarrow |$Post$(T) \cap$ invsim$(w')|$;
3. Initialize remove and remove$^S$:
    3.1. for all $w' \in W'$
            remove$(w') \leftarrow$ Succ$(K) \setminus$ Pre(invsim$(w')$);
    3.2. for all $T \in$ Succ$(K)$
            remove$^S(T) \leftarrow \emptyset$;
Pruning while loop:
4. while prevsim $\neq$ sim
    4.1 prevsim $\leftarrow$ sim;
    4.2 prevsim$^S \leftarrow$ sim$^S$;
    4.3 Procedure PruneSimStrSucc;
    4.4 Procedure PruneSimStr;
5. return $\{(w, w') \in W \times W' \mid$ sim$(w, w')$ is true$\}$;
```

to false, we know that since $T \in$ remove(w') we have count$^S(T, w') = 0$ (i.e., Post$(T) \cap$ invsim$(w') = \emptyset$). This implies that for every $w \in T$ we have that w' does not simulate w. Also note that since count^S is never incremented, if it reaches zero, it remains zero. This proves the correctness of invariant 1.(b) (and similar argument holds for invariant 1.(a)). The correctness for invariant 2.(a) and 2.(b) is as follows: whenever we decrement count(s', T) we have set sim$^S(T', T)$ to false, and T' was earlier both in Post(s') as well as in invsim$^S(T)$, and is now removed from invsim$^S(T)$. Hence from the set Post$(s') \cap$ invsim$^S(T)$ we remove the element T' and its cardinality decreases by 1. This establishes correctness of invariant 2.(a) (and invariant 2.(b) is similar). Finally we consider invariant 3.(a): when we add s' to remove$^S(T)$, then we know that count(s', T) was decremented to zero, which means T' belongs to invprevsim$^S(T)$, but not to invsim$^S(T)$. Thus s' belongs to Pre(invprevsim$^S(T)$) (since s' belongs to Pre(T')), but not to Pre(invsim$^S(T)$). This shows that s' belongs to remove$^S(T)$, and establishes correctness of the desired invariant (argument for invariant 3.(b) is similar).

Invariants to correctness. The initialization part ensures that sim is an overapproximation of $\approxeq^*$ and it follows from invariant 1 that the output is an overapproximation of $\approxeq^*$. Similarly we also have that sim^S in the end is an overapproximation of $\approxeq^S$. To complete the correctness proof, let sim and sim^S be the result when the while loop iterations end. We will now show that sim and sim^S are witnesses to satisfy successor set simulation. We know that when

Algorithm 2 Procedure PruneSimStrSucc

```
1. forall $w' \in W'$ such that $\mathsf{remove}(w') \neq \emptyset$
    1.1. forall $T' \in \mathsf{Pre}(w')$ and $T \in \mathsf{remove}(w')$
        1.1.1 if $(\mathsf{sim}^S(T', T))$
                $\mathsf{sim}^S(T', T) \leftarrow$ false;
                1.1.1.A. forall $(s' \in \mathsf{Pre}(T'))$
                        $\mathsf{count}(s', T) \leftarrow \mathsf{count}(s', T) - 1$;
                        if $(\mathsf{count}(s', T) = 0\ )$
                            $\mathsf{remove}^S(T) \leftarrow \mathsf{remove}^S(T) \cup \{s'\}$;
    1.2. $\mathsf{remove}(w') \leftarrow \emptyset$;
```

Algorithm 3 Procedure PruneSimStr

```
1. forall $T \in \mathsf{Succ}(K)$ such that $\mathsf{remove}^S(T) \neq \emptyset$
    1.1. forall $w \in \mathsf{Pre}(T)$ and $w' \in \mathsf{remove}^S(T)$
        1.1.1 if $(\mathsf{sim}(w, w'))$
                $\mathsf{sim}(w, w') \leftarrow$ false;
                1.1.1.A. forall $(D \in \mathsf{Pre}(w))$
                        $\mathsf{count}^S(D, w') \leftarrow \mathsf{count}^S(D, w') - 1$;
                        if $(\mathsf{count}^S(D, w') = 0\ )$
                            $\mathsf{remove}(w') \leftarrow \mathsf{remove}(w') \cup \{D\}$;
    1.2. $\mathsf{remove}^S(T) \leftarrow \emptyset$;
```

the algorithm terminates, $\mathsf{remove}(w') = \emptyset$ for every $w' \in W'$, and $\mathsf{remove}^S(T) = \emptyset$ for every $T \in \mathsf{Succ}(K)$ (this follows since $\mathsf{sim} = \mathsf{prevsim}$). To show that sim and sim^S are witness to satisfy successor set simulation, we need to show the following two properties: (i) If $\mathsf{sim}(w, w')$ is true, then for every $a \in P_1(w)$, there exists $a' \in P'_1(w')$ such that $\mathsf{sim}^S(\mathsf{Succ}(w', a'), \mathsf{Succ}(w, a))$ is true. (ii) If $\mathsf{sim}^S(T', T)$ is true, then for every $s' \in T'$, there exists $s \in T$ such that $\mathsf{sim}(s, s')$ is true. The property (i) holds because for every $a \in P_1(w)$, we have that $\mathsf{count}(w', T) > 0$, where $T = \mathsf{Succ}(w, a)$, (because otherwise, w' would have been inserted in $\mathsf{remove}(T)$, but since $\mathsf{remove}(T)$ is empty, consequently $\mathsf{sim}(w, w')$ must have been made false). Hence we have that $\mathsf{Post}(w') \cap \mathsf{invsim}^S(T)$ is non-empty and hence there exists $T' \in \mathsf{Post}(w')$ such that $\mathsf{sim}^S(T', T)$ is true. Similar argument works for (ii). Thus we have established that sim is both an overapproximation of $\approxeq^*$ and also a witness successor set relation. Since $\approxeq^*$ is the maximum successor set relation, it follows that Algorithm 1 correctly computes $\approxeq^* = \preceq_{\mathsf{altsim}}$ ($\approxeq^* = \preceq_{\mathsf{altsim}}$ by Lemma 17).

Space complexity. We now argue that the space complexity of the iterative algorithm is superior as compared to the game based algorithm. We first show that the space taken by Algorithm 1 is $O(|W|^2 \cdot |A_1| + |W'|^2 \cdot |A'_1| + |W| \cdot |W'| \cdot |A_1| \cdot |A'_1|)$. For the iterative algorithm, the space requirements are, (i) sim and sim^S require at most $O(|W| \cdot |W'|)$ and $O(|W| \cdot |W'| \cdot |A_1| \cdot |A'_1|)$ space, respectively; (ii) count and count^S require at most $O(|W| \cdot |W'| \cdot |A_1|)$ space each; (iii) remove and remove^S maintained as an array of sets require at most $O(|W| \cdot |W'| \cdot |A_1|)$, space each. Also, for the construction of graphs G_K and $G_{K'}$ using the binary tree data structure as described earlier, the space required is at most $O(|W|^2 \cdot |A_1|)$ and $O(|W'|^2 \cdot |A'_1|)$, respectively. As compared to the space requirement of the iterative algorithm, the game based algorithm requires to store the entire game graph and requires at least $O(|W| \cdot |W'| \cdot |A_1| \cdot |A_1'| \cdot |A_2'|)$ space (to store edges in E_3) as well as space $O(|W|^2 \cdot |A_1| + |W'|^2 \cdot |A_1'|)$ for the binary tree data structure. The iterative algorithm can be viewed as an efficient simultaneous pruning algorithm that does not explicitly construct the game graph (and thus saves at least a factor of $|A_2'|$ in terms of space). We now show that the iterative algorithm along with being space efficient matches the time complexity of the game based algorithm.

Time complexity. The data structures for sim (also sim^S) and count (also count^S) are as described earlier. We store remove and remove^S as a list of lists (i.e., it is a list of sets, and sets are stored as lists). It is easy to verify that all the non-loop operations take unit cost, and thus for the time complexity, we need to estimate the number of times the different loops could run. Also the analysis of the initialization steps are straight forward, and we present the analysis of the loops below: (1) The **while** loop (Step 4) of Algorithm 1 can run for at most $|W| \cdot |W'|$ iterations because in every iteration (except the last iteration) at least one entry of sim changes from true to false (otherwise the iteration stops), and sim has $|W| \cdot |W'|$-entries. (2) The **forall** loop (Step 1) in Algorithm 2 can overall run for at most $|W'| \cdot |W| \cdot |A_1|$ iterations. This is because elements of $\mathsf{remove}(w')$ are from $\mathsf{Succ}(K)$ and elements T from $\mathsf{Succ}(K)$ are included in $\mathsf{remove}(w')$ at most once (when $\mathsf{count}^S(T, w')$ is set to zero, and once $\mathsf{count}^S(T, w')$ is set to zero, it remains zero). Thus $\mathsf{remove}(w')$ can be non-empty at most $|\mathsf{Succ}(K)|$ times, and hence the loop runs at most $|W| \cdot |A_1|$ times for states $w' \in W'$. (3) The **forall** loop (Step 1.1) in Algorithm 2 can overall run for at most $|W'| \cdot |A_1'| \cdot |A_2'| \cdot |W| \cdot |A_1|$ iterations. The reasoning is as follows: for every edge $(T', w') \in G_{K'}$ and $T \in \mathsf{Succ}(K)$ the loop runs at most once (since every T is included in $\mathsf{remove}(w')$ at most once). Hence the number of times the loop runs is at most the number of edges in $G_{K'}$ (at most $|W'| \cdot |A_1'| \cdot |A_2'|$) times the number of elements in $\mathsf{Succ}(K)$ (at most $|W| \cdot |A_1|$). Thus overall the number of iterations of Step 1.1 of Algorithm 2 is at most $|W'| \cdot |A_1'| \cdot |W| \cdot |A_1|$. (4) The **forall** loop (Step 1.1.1.A) in Algorithm 2 can overall run for at most $|W'| \cdot |A_1'| \cdot |A_2'| \cdot |W| \cdot |A_1|$ iterations because every edge (s', T') in $G_{K'}$ would be iterated at most once for every $T \in \mathsf{Succ}(K)$ (as for every T, T' we set $\mathsf{sim}^S(T, T')$ false at most once, and the loop gets executed when such an entry is set to false). The analysis of the following items (5), (6), and (7), are similar to (2), (3), and (4), respectively. (5) The **forall** loop (Step 1) in Algorithm 3 can overall run for at most $|W| \cdot |A_1| \cdot |W'|$ iterations, because $\mathsf{remove}^S(T)$ can be non-empty at most $|W'|$ times (i.e., the number of different T is at most $|\mathsf{Succ}(K)| = |W| \cdot |A_1|$). (6) The **forall** loop (Step 1.1) in Algorithm 3 can overall run for at most $|W| \cdot |A_1| \cdot |A_2| \cdot |W'|$ iterations because every edge (w, T) in G_K can be iterated over at most once for every w' (the number of edges in G_K is $|W| \cdot |A_1| \cdot |A_2|$ and number of states w' is at most $|W'|$). (7) The **forall** loop (Step 1.1.1.A) in Algorithm 3 can overall run for at most $|W| \cdot |A_1| \cdot |A_2| \cdot |W'|$ iterations because every edge (w, D) in G_K would be iterated over at most once for every $w' \in W'$. Adding the above terms, we get that the total time complexity is $O\big(|W| \cdot |W'| \cdot |A_1| \cdot (|A_1'| \cdot |A_2'| + |A_2|)\big)$, i.e., the time complexity matches the time complexity of the game reduction based algorithm. We also remark that for transition systems (TSs), the procedure PruneSimStrSucc coincides with PruneSimStr and our algorithm simplifies to the algorithm of [10], and thus matches the complexity of computing simulation for TSs.

▶ **Theorem 18.** *Algorithm 1 correctly computes $\preceq_{\mathsf{altsim}}$ in time $O\big(|W| \cdot |W'| \cdot |A_1| \cdot (|A_1'| \cdot |A_2'| + |A_2|) + |W|^2 \cdot |A_1| + |W'|^2 \cdot |A_1'|\big)$.*

5 Conclusion

In this work we presented faster algorithms for alternating simulation and alternating fair simulation which are core algorithmic problems in analysis of composite and open reactive systems, as well as state space reduction for graph games (that has deep connection with automata theory and logic). Moreover, our algorithms are obtained as efficient reductions to graph games with reachability and parity objectives with three priorities, and efficient implementations exist for all these problems (for example, see [14] for implementation of games with reachability and parity objectives, and [6] for specialized implementation of games with parity objectives with three priorities).

Acknowledgements. The research was supported by Austrian Science Fund (FWF) Grant No P 23499-N23 on Modern Graph Algorithmic Techniques in Formal Verification, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *JACM*, 49:672–713, 2002.

2 R. Alur, T.A. Henzinger, O. Kupferman, and M.Y. Vardi. Alternating refinement relations. In *CONCUR'98*, LNCS 1466, pages 163–178. Springer, 1998.

3 C. Beeri. On the membership problem for functional and multivalued dependencies in relational databases. *ACM Trans. on Database Systems*, 5:241–259, 1980.

4 K. Chatterjee, S. Chaubal, and P Kamath. Faster algorithms for alternating refinement relations. *CoRR*, abs/1201.4449, 2012.

5 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

6 L. de Alfaro and M. Faella. An accelerated algorithm for 3-color parity games with an application to timed games. In *CAV*, pages 108–120, 2007.

7 E.A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS'91*, pages 368–377. IEEE, 1991.

8 Y. Gurevich and L. Harrington. Trees, automata, and games. In *STOC'82*, pages 60–65. ACM Press, 1982.

9 J. Y. Halpern and R. Fagin. Modeling knowledge and action in distributed systems. *Distributed Computing*, 3:159–179, 1989.

10 M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462. IEEE, 1995.

11 T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. *I & C.*, 173:64–81, 2002.

12 N. Immerman. Number of quantifiers is better than number of tape cells. *JCSS*, 22:384–406, 1981.

13 M. Jurdzinski. Small progress measures for solving parity games. In *STACS'00*, pages 290–301. LNCS 1770, Springer, 2000.

14 M. Lange and O. Friedmann. The pgsolver collection of parity game solvers. 2009.

15 R. Milner. An algebraic definition of simulation between programs. In *Second International Joint Conference on Artificial Intelligence*, pages 481–489. The British Computer Society, 1971.

16 L.S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.

17 W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.

A Systematic Approach to Canonicity in the Classical Sequent Calculus

Kaustuv Chaudhuri[1], Stefan Hetzl[2], and Dale Miller[1]

1 **INRIA & LIX/Ecole Polytechnique, France**
`{kaustuv.chaudhuri,dale.miller}@inria.fr`
2 **Vienna University of Technology, Austria**
`hetzl@logic.at`

Abstract

The sequent calculus is often criticized for requiring proofs to contain large amounts of low-level syntactic details that can obscure the essence of a given proof. Because each inference rule introduces only a single connective, sequent proofs can separate closely related steps—such as instantiating a block of quantifiers—by irrelevant noise. Moreover, the sequential nature of sequent proofs forces proof steps that are syntactically non-interfering and permutable to nevertheless be written in some arbitrary order. The sequent calculus thus lacks a notion of *canonicity*: proofs that should be considered essentially the same may not have a common syntactic form. To fix this problem, many researchers have proposed replacing the sequent calculus with proof structures that are more parallel or geometric. Proof-nets, matings, and atomic flows are examples of such *revolutionary* formalisms. We propose, instead, an *evolutionary* approach to recover canonicity within the sequent calculus, which we illustrate for classical first-order logic. The essential element of our approach is the use of a *multi-focused* sequent calculus as the means of abstracting away the details from classical cut-free sequent proofs. We show that, among the multi-focused proofs, the *maximally multi-focused* proofs that make the foci as parallel as possible are canonical. Moreover, such proofs are isomorphic to *expansion proofs*—a well known, minimalistic, and parallel generalization of Herbrand disjunctions—for classical first-order logic. This technique is a systematic way to recover the desired essence of any sequent proof without abandoning the sequent calculus.

1998 ACM Subject Classification F.4.1 Mathematical Logic: Proof theory

Keywords and phrases Sequent Calculus, Canonicity, Classical Logic, Expansion Trees

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.183

1 Introduction

The sequent calculus, initially described by Gentzen for classical and intuitionistic first-order logic [10], has become a standard proof formalism for a wide variety of logics. One of the chief reasons for its ubiquity is that it defines provability in a logic parsimoniously and modularly, with every logical connective defined by introduction rules, and with the logical properties defined by structural rules. Sequent rules can thus be seen as the *atoms* of logical inference. Different logics can be described simply by choosing different atoms. For instance, linear logic [11] differs from classical logic by removing the structural rules of weakening and contraction, and letting the multiplicative and the additive variants of introduction rules introduce different connectives. The proof-theoretic properties of the logics can then be derived by analyzing these atoms of inference. For example, the *cut-elimination* theorem directly shows that the logic is consistent.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 183–197
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Yet, despite its success as a framework for establishing proof-theoretic properties, the sequent proofs themselves seem to obscure the "essence" of a proof. One quickly feels that sequent proofs are syntactic monsters: they record the exact sequence of inferences and detours even when it is not really relevant to the essential high level features of the proof.

The usual approach over the years to dealing with this syntactic morass of the sequent calculus—and some other proof systems with similar issues—is one of *revolution.* Instead of the sequent calculus, new proof formalisms are proposed that are supposedly free of *syntactic bureaucracy.* Usually, such formalisms are more parallel or geometric than sequent proofs. We list here several examples—not an exhaustive list—of such revolutionary proof systems.

1. The *mating method* [2] and the *connection method* [5] represent proofs as a graph structure among the literals in (an expansion of) a formula.
2. *Expansion trees* [27] record only the instantiations of quantifiers using a tree structure.
3. *Proof-nets* [11] eschew inference rules for more geometric representations of proofs in terms of *axiom linkages.*
4. *Atomic flows* [13] track only the flow of atoms in a proof and can expose the dynamics of cut-elimination.
5. Even Gentzen's *natural deduction* calculus [10] is arguably a principally different representation of proofs.

These revolutionary approaches continue by providing a means of *de-sequentializing* sequent proofs into the new formalism, and then arguing that two sequent proofs are essentially the same if they de-sequentialize to the same form. While compelling, it is worth noting that such approaches are not without problems. At a basic level, showing when a proposed structure is correct—that it actually represents a "proof"—generally requires checking global criteria such as connectedness, acyclicity, or well-scoping. Such formalisms generally lack *local* correctness criteria, wherein a partial (unfinished) proof object can be ensured to have only correct finished forms. By contrast, every instance of a rule in a (partial) sequent proof can easily be checked to be an instance of a proper rule schema.

A second and bigger issue with such revolutionary formalisms is that none of them is as general as the sequent calculus. Proof-nets, to pick an example, are only well defined for the unit-free multiplicative linear logic (*MLL*) [11]. Even adding the multiplicative units is tricky [22] and for larger fragments such as *MALL* with units the problem of finding a proof-net formalism remains open.

In this paper, we consider instead an *evolutionary* approach to extracting the essence of sequent proofs without discarding the sequent calculus. We simply add abstractions to the sequent calculus as follows.

1. Analysis of the permutation properties of sequent rules shows that some rules are invertible, and hence require no choice, while others are non-invertible and the proofs must record the choices made for them. These two classes of rules can be used to organize sequent proofs in such a way that the inference atoms coalesce into larger inference molecules – several small inference steps combine into synthetic steps or *actions.* The essential information in a proof is then moved to the action boundaries. *Focusing* [1] is the general technique for this kind of synthesis for cut-free sequent calculi, and it can be described as a simple local modification of the usual sequent rules that preserves completeness. We then simply remove unfocused proofs.
2. The standard focusing technique can be extended to allow *multi-focusing*, where multiple actions can be done *in parallel*, simultaneously. The exact order of the inferences

constituting two simultaneous actions can then be elided from sequent proofs. Proofs with the same parallel action structure are identified, which we call *action equivalence*.

3. Finally, if we insist on as much parallelism as possible, *i.e.*, on *maximal multi-focusing*, then such proofs are action-canonical. That is, two equivalent maximal multi-focused proofs can be shown to be action equivalent. Thus, for each multi-focused proof, its equivalent maximal form is action canonical.

In this paper, we apply this method to classical first-order logic. We show not only that the evolutionary approach gives us canonical sequent proofs at the level of the action abstraction but also that these proofs induce the same notion of identity as expansion proofs [27], an existing parallel (revolutionary) approach for classical first-order (and higher-order) logic. This result is surprising because it is known that expansion trees can be more compact than sequent proofs by an exponential factor [4].

In section 2, we give some background on the sequent calculus and multi-focusing. Section 3 provides the definition of expansion trees and their interconversion with sequent proofs. Section 4 presents the main technical result that maximal multi-focused proofs are isomorphic to expansion proofs. We begin with a quick summary of related work.

1.1 Related Work

1.1.1 Denotational Semantics of Classical Proofs

It is well known that cut-elimination using Gentzen's cut-reduction rules is non-confluent for *LK* proofs [12, 3, 17]. It is generally believed that classical logic lacks a denotational semantics for proofs akin to Cartesian-closed categories (CCC) for intuitionistic logic or $\star$-autonomous categories for linear logic. For example, if one tries to enrich the usual CCC semantics for intuitionistic logic with an involutive negation, then the CCC degenerates into a poset that equates all proofs of a formula (Joyal's paradox) [23].

This problem has been attacked from both the syntactic and the semantic ends. Of the syntactic approaches, one can recover confluence (up to a small equivalence relation) as well as strong normalization by fixing particular cut-reduction strategies [8]. If one refrains from fixing a reduction strategy one may still obtain a strongly normalizing though non-confluent system by using sufficiently strong local reductions [31, 32]. Another approach is to carry out cut-elimination in a more abstract formalism, similar to a proof-net, on the level of quantifiers (see [14] and [25]). The reduction in such a setting is typically not confluent and strong normalization is open [25] or known not to hold [14]. Confluence (up to the equivalence relation of having the same expansion tree) as well as normalization can be recovered for a class of proofs [19] by considering a maximal abstract reduction based on tree grammars [18] which contains all concrete reductions. Extension of these results to all proofs is open.

From the semantic end, briefly, there are two principal approaches. The first approach rejects the involutive negation, which results in negation having a computational content that can be reified in the $\lambda\mu$ calculus with a semantics in terms of control categories (see [15] for a survey). The second approach rejects the Cartesian structure for conjunctions, which requires a variant of proof-nets called *flow graphs* for the proofs and a semantics in terms of enriched Boolean categories [21, 30].

1.1.2 Cut-Free Formalisms

This paper deals with the question of recovering the essence of *cut-free* sequent proofs. There are a number of alternative approaches to this question. For example, the notion of proof-nets while well-behaved on *MLL* does not scale nicely to larger logics. Girard sketched a design of proof-nets for classical logic [12] that was subsequently fully formalized by Robinson [29], but these nets differentiate between some sequent proofs that are related by rule permutations because of the non-canonicity of weakening nodes. Similar problems also exist for the $\mathbb{B}/\mathbb{N}$-net formalisms [22] based on flow graphs, or the *combinatorial proofs* of Hughes [20]. It is possible to recover the canonicity lost with Robinson's proof-nets by removing weakening (with the use of MIX) and rigidly controlling contraction [26]. This results in *expansion nets*, which are related to expansion trees [27], but are limited to the propositional fragment.

Expansion trees, because they generalize Herbrand disjunctions, are applicable to first-order and even higher-order logics. They achieve this generality by recording only the quantifier instances in a tree structure, and therefore have an expensive correctness criterion involving checking that the deep formula for an expansion tree is a tautology. The mating method [2] or the connection method [5] represents these tautological checks using graph structures, but the correctness criteria for such structures are no less expensive to check than deciding whether the deep formula is a tautology.

To our knowledge, there has been only a single attempt to produce canonical proof structures directly in the sequent calculus, in this case for $\top$-free propositional *MALL* [7]. This attempt also used multi-focusing as its abstraction mechanism, and it is actually the first place where the concept of maximally multi-focused proofs appears in the literature. It is important to note that the notion of a maximal multi-focused proof strictly generalizes existing canonical forms in other contexts. For example, for intuitionistic logic, if one uses the focused sequent calculus *LJF* [24] with just the two negative connectives of implication and universal quantification and with negative atomic formulas, then maximal multi-focused proofs are the same as singly focused proofs. Moreover, they correspond to the familiar β-normal η-long forms of the typed λ-calculus [9].

2 Background: Sequent Calculus, Focusing, and Canonicity

We use the usual syntax for (first-order) *formulas* $(A, B, \dots)$ and connectives drawn from $\{\wedge, \top, \vee, \bot, \neg, \forall, \exists\}$. *Atomic formulas* $(a, b, \dots)$ are of the form $p(t_1, \dots, t_n)$ where p represents a predicate symbol and $t_1, \dots, t_n$ are first-order terms $(n \geq 0)$. Formulas are assumed to be identical up to α-equivalence and in negation-normal form (*i.e.*, only atomic formulas can be $\neg$-prefixed). We use *literal* to refer to either an atomic formula or a negated atomic formula. We write $(A)^\perp$ to stand for the De Morgan dual of A, and $[t/x]A$ for the capture-avoiding substitution of term t for x in A. We also write $\exists \vec{x}.\, A$ for $\exists x_1. \dots \exists x_n.\, A$, $\forall \vec{x}.\, A$ for $\forall x_1. \dots \forall x_n.\, A$, and $[\vec{t}/\vec{x}]$ for $[t_1/x_1] \cdots [t_n/x_n]$ if $\vec{x} = x_1, \dots, x_n$ and $\vec{t} = t_1, \dots, t_n$.

2.1 Sequent Calculus

We use one-sided sequents $\vdash \Gamma$ in which Γ is a multiset of formulas. Figure 1 contains the inference rules for our sequent calculus that we call *LKN*. There is no cut rule, the initial rule is restricted to atomic formulas, and all the rules except for $\exists$ are invertible. Since invertible rules are associated with the *negative* polarity in focused proof systems, we use the *N* in *LKN* to highlight the fact that is a variant of Gentzen's *LK* calculus in which most rules are invertible. The following rules are admissible in *LKN*; in these rules, A can be any formula.

$$\frac{\vdash \Gamma, A \quad \vdash \Gamma, (A)^\perp}{\vdash \Gamma}\ \textsf{cut} \qquad \frac{}{\vdash \Gamma, (A)^\perp, A}\ \textsf{arbinit} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, A}\ \textsf{weak} \qquad \frac{\vdash \Gamma}{\vdash [t/x]\Gamma}\ \textsf{subst}$$

$$\frac{}{\vdash \Gamma, \neg a, a}\ \mathsf{init} \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \wedge B}\ \wedge \qquad \frac{}{\vdash \Gamma, \top}\ \top \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \vee B}\ \vee \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}\ \bot$$

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, \forall x.\, A}\ \forall \qquad \frac{\vdash \Gamma, [t/x]A}{\vdash \Gamma, \exists x.\, A}\ \exists \qquad \frac{\vdash \Gamma, \Delta}{\vdash \Gamma}\ \mathsf{contr}$$

Notes:

1. In the $\forall$ rule, the principal formula is implicitly α-converted so x is not free in the conclusion.
2. In the contr rule, $\emptyset \neq \Delta \subseteq_{\mathrm{set}} \Gamma$. Here, $\Delta \subseteq_{\mathrm{set}} \Gamma$ denotes the set inclusion of the underlying sets of the multisets Δ and Γ.

Figure 1 Rules of *LKN*.

These admissible rules easily allow us to mimic any of the other standard inference rules for this logic in *LKN*, including Gentzen's original *LK* calculus, so completeness is immediate. Soundness is equally trivial as every rule preserves classical validity under the interpretation of a sequent $\vdash A_1, \ldots, A_n$ as the formula $A_1 \vee \cdots \vee A_n$.

The reflexive-symmetric-transitive-congruence closure of the permutation steps defines the equivalence relation $\sim$ over *LKN* proofs. One of the standard goals of proof theory is to find canonical syntactic representatives of the permutative equivalence classes for a given sequent calculus. We shall employ focusing to produce such representatives of *LKN* proofs, following a technique introduced in [7] for $\top$-free multiplicative-additive linear logic (*MALL*) using the technical device of *multi-focusing*.

There is one critical difference between the approach of [7] and that of this paper: we restrict permutation steps to cases where both of the rules being permuted have at least one premise. In other words, $\top/r$ and init/r permutation steps are impossible for any rule r; in particular, we disallow the following permutation step.

$$\frac{\dfrac{}{\vdash \Gamma, \Delta, \top}\ \top}{\vdash \Gamma, \top}\ \mathsf{contr} \qquad \longrightarrow \qquad \frac{}{\vdash \Gamma, \top}\ \top$$

If such permutation steps were to be allowed, then the induced equivalence on *LKN* proofs would equate arbitrary sub-proofs and defeat any attempt at canonicity. Observe that preventing such permutations does not affect the classical symmetries, *i.e.*, A continues to be identical to $((A)^\perp)^\perp$.

2.2 Focused Sequent Calculus

The proof-theoretic analysis of the logic programming paradigm developed in the 1980s accounted for notions of *goal-reduction* and *back-chaining* as two alternating phases in the construction of (cut-free) sequent proofs [28]. Andreoli [1] developed the notion of *focused sequent proofs* for classical linear logic as a generalization of this earlier work in logic programming. Subsequently, focused sequent calculus proofs have been written for intuitionistic and classical logics [24]. Such proof systems are increasingly being seen as general proof-theoretic tools for uncovering structures within proofs.

A focused calculus partitions formulas into *positive* and *negative polarities* based on the permutation properties of their sequent rules. Similarly, the introduction rules in a focused calculus appear in either one of two *phases*. The *asynchronous* or *negative* phase

Invertible

$$\frac{\vdash \Gamma, L \Uparrow \Delta}{\vdash \Gamma \Uparrow \Delta, L}\ \mathsf{store} \qquad \frac{\vdash \Gamma \Uparrow \Delta, A \quad \vdash \Gamma \Uparrow \Delta, B}{\vdash \Gamma \Uparrow \Delta, A \wedge B}\ \wedge \qquad \frac{}{\vdash \Gamma \Uparrow \Delta, \top}\ \top \qquad \frac{\vdash \Gamma \Uparrow \Delta, A, B}{\vdash \Gamma \Uparrow \Delta, A \vee B}\ \vee$$

$$\frac{\vdash \Gamma \Uparrow \Delta}{\vdash \Gamma \Uparrow \Delta, \bot}\ \bot \qquad \frac{\vdash \Gamma \Uparrow \Delta, A}{\vdash \Gamma \Uparrow \Delta, \forall x.\, A}\ \forall$$

Existential

$$\frac{\vdash \Gamma \Downarrow \Delta, [t/x]A}{\vdash \Gamma \Downarrow \Delta, \exists x.\, A}\ \exists$$

Structural

$$\frac{}{\vdash \Gamma, \neg a, a \Uparrow \cdot}\ \mathsf{init} \qquad \frac{\vdash \Gamma \Downarrow \Delta}{\vdash \Gamma \Uparrow \cdot}\ \mathsf{decide} \qquad \frac{\vdash \Gamma \Uparrow \Delta}{\vdash \Gamma \Downarrow \Delta}\ \mathsf{release}$$

Notes:

1. In the store rule, L is a literal or an existential formula.
2. In the $\forall$ rule, the principal formula is implicitly α-converted so x is not free in the conclusion.
3. In the decide rule, Δ contains only existential formulas and $\emptyset \neq \Delta \subseteq_{\text{set}} \Gamma$.
4. In the release rule, Δ contains no existential formulas.

Figure 2 Rules of *LKNF*.

consists of applying[1] all available invertible rules to the negative non-atomic formulas, in an arbitrary order, until none remains. The *synchronous* or *positive* phase is then launched per sequent by *focusing* on one or more positive formulas using a rule called decide. In this phase, non-invertible rules are applied to the focused formulas and, crucially, the focus is maintained on the positive subformulas in the premises of the applied rule. The positive phase persists until the focused formulas all become negative; the proof then switches back to the negative phase by a rule named release.

Formally, we will use a sequent calculus that closely resembles the *LKF* system as given in [24], with some important differences. First, *LKF* allows only a single focus formula while our calculus will allow multiple foci. (It is a simple matter to add multi-focusing to *LKF*.) A second and bigger difference is that the *LKF* proof system contains a positive and negative version of both conjunction and disjunction, while we will use only the negative versions of these connectives. This choice is motivated by our desire to model the Herbrand disjunctions underlying expansion proofs, where the propositional content is elided. The last difference is that the positive phase in *LKF* can contain instances of the initial and $\exists$-introduction rules, but for our goal of obtaining a variant of Herbrand's theorem we will need a clean separation of quantification rules and propositional rules. The critical issue is that in *LKF* there is only a single proof of $\vdash \neg p(a), \exists x.\, p(x) \Uparrow \cdot$, while there are infinitely many expansion proofs of $\neg p(a) \vee \exists x.\, p(x)$ that simply differ in their numbers of instances of the existential quantifier. One way to limit the focusing strength of *LKF* to obtain these other proofs is to replace all the occurrences of positive literals L with a *delayed* literal $(L \wedge \top)$, which is equivalent but of negative polarity.

In Figure 2 we present our focused sequent calculus *LKNF*. It can be seen as the multi-focused variant of *LKF* with only negative propositional connectives and implicitly delayed positive literals. Since the positive phase of *LKNF* only involves the existential quantifier, we rename the "positive phase" of *LKF* as the "existential phase". The two phases of *LKNF*

[1] In this paper we use "apply" to stand for a reading of an inference rule from conclusion to premises.

proofs are depicted using two different sequent forms: *negative sequents* of the form $\vdash \Gamma \Uparrow \Delta$ and *positive sequents* of the form $\vdash \Gamma \Downarrow \Delta$. In either form, Γ is a multiset of literals or existential formulas, and Δ is a multiset of arbitrary formulas. In the positive sequent $\vdash \Gamma \Downarrow \Delta$, we say that the formulas in Δ are its *foci* and we require Δ to be non-empty. We write $\vdash \Gamma \Updownarrow \Delta$ to stand for either sequent form.

The inference rules of *LKNF* are divided into three classes. The *invertible* rules all apply to negative sequents and contain no essential non-determinism. The *existential* rule is non-invertible: the witness terms must be recorded in the proof. The final class of *structural* rules includes: the init rule for initial sequents; the decide rule where a number of existential formulas are copied, possibly more than once, to the foci of a new positive phase; and the release rule to leave the positive phase when *none* of the foci is an existential formula.

LKNF is sound and complete with respect to *LKN*; to make this statement precise, we inject *LKNF* proofs to *LKN* proofs.

▶ **Definition 1.** For any *LKNF* proof π, we write $[\pi]$ to stand for that *LKN* proof that:

- replaces all sequents of the form $\vdash \Gamma \Updownarrow \Delta$ with $\vdash \Gamma, \Delta$;
- removes all instances of the rules store and release; and
- renames decide to contr in π.

▶ **Theorem 2** (*LKNF* vs. *LKN*).

1. *If π is an LKNF proof of $\vdash \Gamma \Updownarrow \Delta$, then $[\pi]$ is an LKN proof of $\vdash \Gamma, \Delta$ (soundness).*
2. *If $\vdash \Delta$ is provable in LKN, then $\vdash \cdot \Uparrow \Delta$ is provable in LKNF (completeness).*

Proof. Soundness is immediate by inspection. Completeness follows by observing that the *LKF* calculus of [24], which is complete for *LK* (and hence also for *LKN*), is simply a singly focused fragment of *LKNF* if all its connectives are negatively biased and delays are inserted as needed around literals. ◀

We can also define an equivalence over *LKNF* proofs in terms of rule permutations. The permutations in the focused setting are subtle; certain permutations such as decide/store are simply impossible. We therefore exploit the injection of definition 1 to bootstrap the *LKNF* equivalence using the *LKN* equivalence.

▶ **Definition 3.** Two *LKNF* proofs π_1 and π_2 of the same sequent are *equivalent*, written $\pi_1 \sim \pi_2$, iff $[\pi_1] \sim [\pi_2]$.

2.3 Canonicity

The main benefit of focusing is that the introduction rules of the unfocused calculus (*LKN*) coalesce into larger *synthetic rules* that represent *actions*. Every action begins at the bottom with an instance of decide, and the action ends with premises of the form $\vdash \Gamma \Uparrow \cdot$. The underlying *LKN* rules inside a single action can be freely permuted with each other, and it is not important to record their particular sequence. In other words, two equivalent *LKNF* proofs should be considered "the same" if they use the decide rules in the same way; we call such proofs *action equivalent*.

▶ **Definition 4.** Two *LKNF* proofs π_1 and π_2 of the same sequent are *action equivalent*, written $\pi_1 \cong \pi_2$, iff they are equivalent (definition 3) and are tree-isomorphic for the instances of the decide rules.

Action equivalence gives us the "essence" of cut-free focused sequent proofs. Since two action equivalent proofs have the same decide rules, one can reason about such proofs by induction on the *decision depth*—*i.e.*, the depth of the decide rules—in the *LKNF* proof. If from a proof we simply elide all but the decide rules, and record the existential witnesses along with these instances of decide, we can then obtain a canonical *synthetic* representation of the proof directly in the sequent calculus. (It is indeed possible to build a sequent calculus that uses solely synthetic sequent rules [6].)

Two equivalent *LKNF* derivations need not be action equivalent as they may perform the decide steps in a different order or with different foci. However, each equivalence class of *LKNF* proofs does have a canonical form where the foci of each decide rule are selected to be as numerous as possible.

▶ **Definition 5** (Maximality). Given an *LKNF* proof π that ends in an instance of decide, we write $\text{foci}\,(\pi)$ for the foci in the premise of that instance of decide. We say that the instance is *maximal* iff for every $\pi' \sim \pi$, it is the case that $\text{foci}\,(\pi') \subseteq_{\text{multiset}} \text{foci}\,(\pi)$. An *LKNF* proof is maximal iff every instance of decide in it is maximal.

The two main properties of maximal proofs are that equivalent maximal proofs are action equivalent, and that for every proof there is an equivalent maximal proof. This pair of results guarantees that the maximal proofs are canonical (action equivalent) representatives of their $\sim$-equivalence classes. Similar theorems have appeared in [7, 6].

▶ **Theorem 6** (Canonicity).

1. *Every LKNF proof has an equivalent maximal proof.*
2. *Two equivalent maximal LKNF proofs are action equivalent.*

Proof. Because init/contr and $\top$/contr permutations are disallowed, equivalent proofs have the same multiset union of all the foci of their decide rules. Using the consolidated form of contr/contr permutations, the foci of the instances of decide can be divided or combined as needed. Therefore, there is a *merge* operation that, starting from the bottom of an *LKNF* proof and going upwards, permutes and merges foci into the lowermost decide instances by splitting them from higher instances. This merge operation obviously terminates (by induction on the decision depth); moreover, the result is maximal by definition 5.

To see that two given equivalent maximal proofs are action equivalent, suppose the contrary. Then there is a lowermost instance of decide in the two proofs that have an incomparable multiset of foci (if they were comparable, then either one of the proofs is not maximal or they are action equivalent). Since the proofs are equivalent, these two decide rules themselves permute; hence, their foci can be merged as above, contradicting our assumption that they are maximal. ◀

▶ **Definition 7.** Theorem 6 shows that for every *LKNF* proof π there is a unique action equivalence class corresponding to the maximal proofs of π. We write $\max(\pi)$ for this class.

In other words, $\max(\pi)$ is the maximally parallel structure of decide and existential inferences corresponding to π. A simple corollary of the completeness of *LKNF* and canonicity is Herbrand's theorem for prenex formulas.

▶ **Corollary 8** (Herbrand's theorem). *The formula $\exists\vec{x}.\,A$, where A is quantifier-free, is valid if and only if there is a sequence of vectors of terms $\vec{t_1}, \dots, \vec{t_n}$ such that the disjunction $[\vec{t_1}/\vec{x}]A \vee \cdots \vee [\vec{t_n}/\vec{x}]A$ is valid.*

Proof. One direction is trivial. Suppose $\exists\vec{x}.\,A$ is valid, *i.e.*, the *LKN* sequent $\vdash \exists\vec{x}.\,A$ is provable. By theorem 2 $\vdash \cdot \Uparrow \exists\vec{x}.\,A$ is provable in *LKNF*, *i.e.*, $\vdash \exists\vec{x}.\,A \Uparrow \cdot$ is provable as only store applies to the former. Because A is quantifier-free, the decide rule can only apply to $\exists\vec{x}.\,A$; thus, the equivalent maximal proof (which exists by Theorem 6) performs only (at most) a single decide at the bottom, producing a number of focused copies of $\exists\vec{x}.\,A$. In the positive phase, the $\exists$s are removed from the foci to give the required term vectors. ◀

3 Expansion Trees

Herbrand's theorem [16] tells us that recording how quantifiers are instantiated is sufficient to describe a proof of a prenex normal formula. Gentzen [10] noticed this also in (cut-free) proofs of a prenex normal sequents via the *mid-sequent*. Miller [27] defined *expansion trees* for full higher-order logic as a structure to record such substitution information without restriction to prenex normal form. We will use a first-order version of this notion here.

▶ **Definition 9.** *Expansion trees* and a function $\mathrm{Sh}(\cdot)$ (for *shallow*) that maps an expansion tree to a formula are defined as follows:

1. A literal L is an expansion tree with $\mathrm{Sh}(L) = L$ and top node L.
2. If E_1 and E_2 are expansion trees and $\circ \in \{\wedge, \vee\}$, then $E_1 \circ E_2$ is an expansion tree with top node $\circ$ and $\mathrm{Sh}(E_1 \circ E_2) = \mathrm{Sh}(E_1) \circ \mathrm{Sh}(E_2)$.
3. If E is an expansion tree with $\mathrm{Sh}(E) = [y/x]A$ and y is not an eigenvariable of any node in E, then $\forall x.\,A +^y E$ is an expansion tree with top node $\forall x.\,A$ and $\mathrm{Sh}(\forall x.\,A +^y E) = \forall x.\,A$. The variable y is called an *eigenvariable* of its top node.
4. If $\{t_1, \ldots, t_n\}$ is a set of terms and $E_1, \ldots, E_n$ are expansion trees with $\mathrm{Sh}(E_i) = [t_i/x]A$ for $i = 1, \ldots, n$, then $E' = \exists x.\,A +^{t_1} E_1 \ldots +^{t_n} E_n$ is an expansion tree with top node $\exists x.\,A$ and $\mathrm{Sh}(E') = \exists x.\,A$. The terms $t_1, \ldots, t_n$ are known as the *expansion terms* of its top node. We allow the case where $n = 0$.

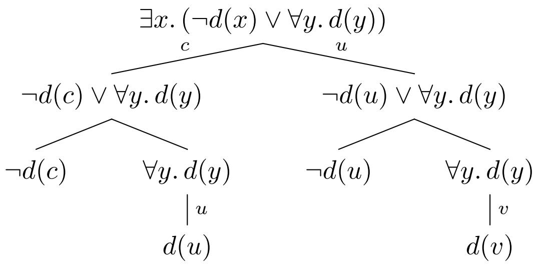
Note that the requirement of y not being an eigenvariable of any node in E in the clause for the universal node ensures that each eigenvariable appears only once in an expansion tree. In the context of proofs this condition is often formulated globally and called *regularity*. The reason for requiring this property of expansion trees is that the correctness criterion is global and hence needs globally unique variable names. In contrast, the correctness of a sequent proof is locally checkable, so the (local) eigenvariable condition is enough. We shall consider eigenvariables within expansion trees to be bound over the entire expansion tree and that systematic changes to eigenvariable names (α-conversion) result in equal trees.

There is a simple way to coerce a formula into an expansion tree: use the bound variable of a universally quantified subformula as that quantifiers eigenvariable and use the empty set of terms to expand an existentially quantified formula. Whenever we use a formula to denote an expansion tree, we shall assume that we use this coercion.

▶ **Example 10.** The expression

$$\exists x.\,(\neg d(x) \vee \forall y.\,d(y)) +^c (\neg d(c) \vee (\forall y.\,d(y) +^u d(u))) +^u (\neg d(u) \vee (\forall y.\,d(y) +^v d(v)))$$

is an expansion tree that can alternatively be written as follows.


$\exists x. (\neg d(x) \vee \forall y. d(y))$
c
u
$\neg d(c) \vee \forall y. d(y)$
$\neg d(u) \vee \forall y. d(y)$
$\neg d(c)$
$\forall y. d(y)$
$\neg d(u)$
$\forall y. d(y)$
u
v
$d(u)$
$d(v)$

So far, we have only described a basic data structure for storing quantifier instances; we still lack a correctness criterion for deciding when such a tree is a proof. For this criterion we need the following function $\mathrm{Dp}(\cdot)$ (for *deep*).

▶ **Definition 11.** For an expansion tree E, the quantifier-free formula $\mathrm{Dp}(E)$, called the *deep formula of* E, is defined as:

- $\mathrm{Dp}(E) = E$ for a literal E,
- $\mathrm{Dp}(E_1 \circ E_2) = \mathrm{Dp}(E_1) \circ \mathrm{Dp}(E_2)$, for $\circ \in \{\wedge, \vee\}$,
- $\mathrm{Dp}(\forall x. A +^y E) = \mathrm{Dp}(E)$, and
- $\mathrm{Dp}(\exists x. A +^{t_1} E_1 \ldots +^{t_n} E_n) = \bigvee_{i=1}^n \mathrm{Dp}(E_i)$. If $n = 0$ then $\mathrm{Dp}(\exists x. A) = \bot$.

In addition to considering expansion trees (of formulas) we will also consider expansion sequents (of sequents). If $S = \vdash A_1, \ldots, A_n$ is a sequent and $E_1, \ldots, E_n$ are expansion trees with $\mathrm{Sh}(E_i) = A_i$, then $\vdash E_1, \ldots, E_n$ is called an expansion sequent of S if whenever E_i and E_j share an eigenvariable then $i = j$. For an expansion sequent $\mathcal{E} = \vdash E_1, \ldots, E_n$, define $\mathrm{Dp}(\mathcal{E}) = \vdash \mathrm{Dp}(E_1), \ldots, \mathrm{Dp}(E_n)$ and $\mathrm{Sh}(\mathcal{E}) = \vdash \mathrm{Sh}(E_1), \ldots, \mathrm{Sh}(E_n)$. A second component of the correctness criterion involves the following dependency relation.

▶ **Definition 12.** Let $\mathcal{E}$ be an expansion tree or expansion sequent and let $<^0_{\mathcal{E}}$ be the binary relation on the occurrences of the expansion terms in $\mathcal{E}$ defined by $t <^0_{\mathcal{E}} s$ if there is an x which is free in s and which is the eigenvariable of a node dominated by t. Then $<_{\mathcal{E}}$, the transitive closure of $<^0_{\mathcal{E}}$, is called the *dependency relation* of $\mathcal{E}$.

In terms of the sequent calculus, $t <_{\mathcal{E}} s$ means that the inference corresponding to t must be below the inference corresponding to s.

▶ **Definition 13.** Let A be a formula (S be a sequent). An expansion tree $\mathcal{E}$ of A (or respectively an expansion sequent $\mathcal{E}$ of S) is called an *expansion proof* of A (respectively S) if $<_{\mathcal{E}}$ is acyclic and $\mathrm{Dp}(\mathcal{E})$ is a tautology.

▶ **Example 14.** Let E be the expansion tree of example 10. It has two expansion terms: c and u. We have $c <_E u$ because the node labeled with c dominates the $\forall$-node with eigenvariable u. However $u \not<_E c$, so $<_E$ is acyclic; furthermore, $\mathrm{Dp}(E) = \neg d(c) \vee d(u) \vee \neg d(u) \vee d(v)$, which is a tautology. So, E is an expansion proof of the formula $\mathrm{Sh}(E) = \exists x. (\neg d(x) \vee \forall y. d(y))$.

3.1 Expansions from Proofs

We now turn to describing how to read off an expansion proof from a sequent calculus proof. To that aim, the following merge-operation on expansion trees will be useful.

▶ **Definition 15.** Let E_1 and E_2 be expansion trees with $\mathrm{Sh}(E_1) = \mathrm{Sh}(E_2)$. Then their *merge* $E_1 \cup E_2$ is defined as follows:

1. If A is a literal then $E_1 \cup E_2 = E_1 = E_2 = A$.
2. If $E_1 = E_1' \circ E_1''$ and $E_2 = E_2' \circ E_2''$ for $\circ \in \{\wedge, \vee\}$, then $E_1 \cup E_2 = (E_1' \cup E_2') \circ (E_1'' \cup E_2'')$.
3. If $E_1 = \forall x.\, B +^{y_1} E_1'$ and $E_2 = \forall x.\, B +^{y_2} E_2'$, then $E_1 \cup E_2 = \forall x.\, B +^{y_1} (E_1' \cup [y_1/y_2]E_2')$. Alphabetic change of eigenvariable names in E_1' and E_2' might be necessary to do this merge in general.
4. If $E_1 = \exists x.\, B +^{r_1} E_{1,1} \ldots +^{r_k} E_{1,k} +^{s_1} F_1 \ldots +^{s_l} F_l$ and $E_2 = \exists x.\, B +^{r_1} E_{2,1} \ldots +^{r_k} E_{2,k} +^{t_1} G_1 \ldots +^{t_m} G_m$ where $\{s_1, \ldots, s_l\} \cap \{t_1, \ldots, t_m\} = \emptyset$, then $E_1 \cup E_2 =$

$$\exists x.\, B +^{r_1} (E_{1,1} \cup E_{2,1}) \ldots +^{r_k} (E_{1,k} \cup E_{2,k}) +^{s_1} F_1 \ldots +^{s_l} F_l +^{t_1} G_1 \ldots +^{t_m} G_m$$

The merge of expansion sequents is component-wise.

We now present an explicit mapping from *LKN* proofs to expansion proofs.

▶ **Definition 16.** Let π be an *LKN*-proof. The expansion sequent $\mathcal{E}(\pi)$ is defined by induction: if π is the initial rule with conclusion $\vdash \Gamma, \neg a, a$, then let $\mathcal{E}(\pi) = \Gamma, \neg a, a$. (It is straightforward to coerce the formulas in Γ into expansion trees.) The analogous translation is needed for the introduction rule for $\top$. The remaining cases for π are the following.

$$\text{(a)}\ \dfrac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \vdash \Gamma, A & \vdash \Gamma, B\end{array}}{\vdash \Gamma, A \wedge B}\ \wedge \qquad \text{(b)}\ \dfrac{\begin{array}{c}(\pi')\\ \vdash \Gamma, A\end{array}}{\vdash \Gamma, \forall x.\, A}\ \forall \qquad \text{(c)}\ \dfrac{\begin{array}{c}(\pi')\\ \vdash \Gamma, [t/x]A\end{array}}{\vdash \Gamma, \exists x.\, A}\ \exists \qquad \text{(d)}\ \dfrac{\begin{array}{c}(\pi')\\ \vdash \Gamma, \Delta\end{array}}{\vdash \Gamma}\ \mathsf{contr}$$

For case (a), if $\mathcal{E}(\pi_1) = \mathcal{E}_1, E_1$ and $\mathcal{E}(\pi_2) = \mathcal{E}_2, E_2$, then $\mathcal{E}(\pi) = \mathcal{E}_1 \cup \mathcal{E}_2, E_1 \wedge E_2$. Analogous definitions apply for the other propositional rules. For case (b), if $\mathcal{E}(\pi') = \mathcal{E}, E$, then $\mathcal{E}(\pi) = \mathcal{E}, \forall x.\, A +^{y} [y/x]E$ where y is not an eigenvariable of a node in $\mathcal{E}, E$. For case (c), if $\mathcal{E}(\pi') = \mathcal{E}, E$, then $\mathcal{E}(\pi) = \mathcal{E}, \exists x.\, A +^{t} E$. Finally, for case (d), let $\Gamma = A_1, \ldots, A_n$ with corresponding expansion trees $E_1, \ldots E_n$ in $\mathcal{E}(\pi')$. For $i \in \{1, \ldots, n\}$ let k_i be the number of copies of A_i in Δ and let $E_{i,1}, \ldots, E_{i,k_i}$ be the expansion trees corresponding to them. Then $\mathcal{E}(\pi) = E_1 \bigcup_{j=1}^{k_1} E_{1,j}, \ldots, E_n \bigcup_{j=1}^{k_n} E_{n,j}$.

The above definition extends to the focused setting in a straightforward way by defining $\mathcal{E}(\pi) = \mathcal{E}([\pi])$ for an *LKNF*-proof π.

▶ **Theorem 17.** *If π is an LKN- or LKNF-proof, then $\mathcal{E}(\pi)$ is an expansion proof.*

Proof. That $\mathrm{Dp}(\mathcal{E}(\pi))$ is a tautology can be shown by induction on the depth of π treating each of the cases of definition 16. Acyclicity of $<_{\mathcal{E}(\pi)}$ follows from the side condition of the $\forall$-rule and the appropriate choice of variable names in definition 16. ◀

3.2 Sequentialization

For translating expansion trees to *LKNF*-proofs we will proceed in two phases: first we translate an expansion tree to a proof in an intermediate calculus *LKNFE* which has the structure of *LKNF* but instead of working on sequents it works on expansion sequents. Secondly we map an *LKNFE*-proof π to an *LKNF*-proof $\mathrm{Sh}(\pi)$ which is defined by applying $\mathrm{Sh}(\cdot)$ to every expansion tree appearing in the proof. This operation will indeed yield a valid *LKNF*-proof as the Sh-image of a *LKNFE*-rule will be a *LKNF*-rule. In particular, the decide-rule of *LKNFE* is the following, where Δ is a choice of some instances which are present in Γ and Γ' are the remaining instances.

$$\dfrac{\vdash \Gamma' \Downarrow \Delta}{\vdash \Gamma \Uparrow \cdot}\ \mathsf{decide}$$

Formally: $\Gamma = E_1, \ldots, E_n$ where $E_i = \exists x.\, A_i +^{t_{i,1}} E_{i,1} \cdots +^{t_{i,n_i}} E_{i,n_i}$ and $\Gamma' = E'_1, \ldots, E'_n$ where $E'_i = \exists x.\, A_i +^{t_{i,1}} E_{i,1} \cdots +^{t_{i,k_i}} E_{i,k_i}$ with $0 \leq k_i \leq n_i$ and $\Delta = \Delta_1, \ldots, \Delta_n$ where $\Delta_i = \{\exists x.\, A_i +^{t_{i,j}} E_{i,j} \mid k_i < j \leq n_i\}$. The rule for existentials in *LKNFE* is:

$$\frac{\vdash \Gamma \Downarrow \Delta, E}{\vdash \Gamma \Downarrow \Delta, \exists x.\, A +^{t} E}$$

The other rules are adapted in the natural way.

When writing down expansion trees for formulas which contain blocks of quantifiers we will abbreviate using a vector notation. For example, the expansion term $\exists x.\, \exists y.\, A +^{t} (\exists y.\, [t/x]A +^{s_1} E_1 +^{s_2} E_2)$ is abbreviated as $\exists (x, y).\, A +^{(t,s_1)} E_1 +^{(t,s_2)} E_2$. If the length of a vector is irrelevant, we write $\vec{x}$ for a vector of variables and $\vec{t}$ for a vector of terms.

We distinguish proofs and derivations in a calculus. While the initial sequents of a proof are among those declared in the definition of the calculus, the initial sequents of a derivation are arbitrary. The construction of an *LKNFE*-proof from an expansion proof will be done in a phase-wise manner, the derivation containing the negative phase is defined as follows.

▶ **Definition 18** (π^-). Let $\vdash \Gamma \Uparrow \Delta$ be a focused expansion sequent where Δ consists of non-existential expansion trees only. Define the *LKNFE*-derivation $\pi^-_{\vdash \Gamma \Uparrow \Delta}$ of $\vdash \Gamma \Uparrow \Delta$ by exhaustive application of negative rules and stores. These lead to expansion sequents $\vdash \Gamma, \Delta_1 \Uparrow \cdot, \ldots, \vdash \Gamma, \Delta_n \Uparrow \cdot$ and to finishing the proof in case $n = 0$.

We now define a derivation corresponding to the positive phase in a way that will have the effect that sequentializations of expansion trees are always maximal. This property will be crucial for the main theorem of this paper.

▶ **Definition 19** (π^+). Let $\vdash \Sigma \Uparrow \cdot$ be a focused expansion sequent and define the *LKNFE*-derivation $\pi^+_{\vdash \Sigma \Uparrow \cdot}$ of $\vdash \Sigma \Uparrow \cdot$ as follows. Let $\Sigma = \Gamma, \Delta$ where Γ are the non-existential expansion trees and $\Delta = \{E_1, \ldots, E_n\}$ are the existential expansion trees of Σ. Then $E_i = \exists \vec{x}.\, A_i +^{\vec{t}_{i,1}} E_{i,1} \cdots +^{\vec{t}_{i,n_i}} E_{i,n_i}$ where A_i is a negative formula. For $i \in \{1, \ldots, n\}$ let w.l.o.g. $\{1, \ldots, k_i\} = \{j \mid 1 \leq j \leq n_i,$ all terms in $\vec{t}_{i,j}$ are $<_\Sigma$ -minimal$\}$. Define Δ'_i as $\{\exists \vec{x}.\, A_i +^{\vec{t}_{i,1}} E_{i,1}, \ldots, \exists \vec{x}.\, A_i +^{\vec{t}_{i,k_i}} E_{i,k_i}\}$ and Δ'' as $\{E''_1, \ldots, E''_n\}$ where $E''_i = \exists \vec{x}.\, A_i +^{\vec{t}_{i,k_i+1}} E_{i,k_i+1} \cdots +^{\vec{t}_{i,n_i}} E_{i,n_i}$ and apply the decide rule as

$$\frac{\vdash \Gamma, \Delta'' \Downarrow \Delta'_1, \ldots, \Delta'_n}{\vdash \Sigma \Uparrow \cdot}\ \textsf{decide}$$

Because all the expansion terms in Δ'_i are $<_\Sigma$-minimal, exhaustive application of existential inferences is possible and, followed by a release, leads to a sequent $\vdash \Gamma, \Delta'' \Uparrow \Theta$ where Θ consists of non-existential expansion trees only.

▶ **Theorem 20** (Sequentialization). *If $\mathcal{E}$ is an expansion proof, then $\vdash \Uparrow \mathrm{Sh}(\mathcal{E})$ in LKNF.*

Proof. First, let the *LKNFE*-proof $\pi_{\mathcal{E}}$ of $\vdash \cdot \Uparrow \mathcal{E}$ be

$$\begin{array}{c} (\psi) \\ \vdash \Gamma \Uparrow \Delta \\ \vdots \\ \vdash \cdot \Uparrow \mathcal{E} \end{array}$$

where Δ consists of non-existential expansion trees only and ψ is obtained by alternating instances of π^- and π^+ for appropriate expansion sequents. This construction can be carried out as $\mathrm{Dp}(\mathcal{F})$ is a tautology for every expansion sequent $\mathcal{F}$ in ψ and it terminates as the number of nodes of the current expansion sequent strictly decreases with each line of the proof. Then $\mathrm{Sh}(\pi_{\mathcal{E}})$ is indeed an *LKNF*-proof of $\vdash \cdot \Uparrow \mathrm{Sh}(\mathcal{E})$. ◀

▶ **Definition 21** (Sequentialization). The *LKNF*-proof $\mathrm{Sh}(\pi_{\mathcal{E}})$ constructed in the above proof of the sequentialization theorem will be denoted by $\mathrm{Seq}(\mathcal{E})$.

4 Equivalence

A first central observation concerning the relationship of rule permutations and expansion trees is that the former do not change the latter.

▶ **Theorem 22.** *If π_1 and π_2 are LKN-proofs with $\pi_1 \sim \pi_2$ then $\mathcal{E}(\pi_1) = \mathcal{E}(\pi_2)$.*

Proof. Instead of spelling out the proof for every rule permutation, here is just the $\wedge/\exists$-case. Here, π_1 contains a subproof of the form (a) below, where $\mathcal{E}(\pi_1') = \mathcal{E}_1, E_1, E'$ and $\mathcal{E}(\pi_1'') = \mathcal{E}_2, E_2, E''$.

$$\text{(a)}\quad \dfrac{\dfrac{\overset{(\pi_1')}{\vdash \Gamma, A, [t/x]C} \qquad \overset{(\pi_1'')}{\vdash \Gamma, B, [t/x]C}}{\vdash \Gamma, A \wedge B, [t/x]C}\,\wedge}{\vdash \Gamma, A \wedge B, \exists x.\, C}\,\exists \qquad\qquad \text{(b)}\quad \dfrac{\dfrac{\overset{(\pi_1')}{\vdash \Gamma, A, [t/x]C}}{\vdash \Gamma, A, \exists x.\, C}\,\exists \qquad \dfrac{\overset{(\pi_1'')}{\vdash \Gamma, B, [t/x]C}}{\vdash \Gamma, B, \exists x.\, C}\,\exists}{\vdash \Gamma, A \wedge B, \exists x.\, C}\,\wedge$$

By definition 16, the expansion sequent of this subproof is $\mathcal{E}_1 \cup \mathcal{E}_2, E_1 \wedge E_2, \exists x.\, C +^t (E' \cup E'')$. The corresponding subproof in π_2 has the form (b) above and the corresponding expansion sequent is $\mathcal{E}_1 \cup \mathcal{E}_2, E_1 \wedge E_2, (\exists x.\, C +^t E') \cup (\exists x.\, C +^t E'')$ which by definition 15 is equal to $\mathcal{E}_1 \cup \mathcal{E}_2, E_1 \wedge E_2, \exists x.\, C +^t (E' \cup E'')$. ◀

We now turn back to the sequentialization procedure for constructing an *LKNF*-proof from an expansion proof. The procedure used in Theorem 20 has been designed for producing only maximal proofs as shown in the following lemma.

▶ **Lemma 23.** *If $\mathcal{E}$ is an expansion proof, then* $\mathrm{Seq}(\mathcal{E})$ *is maximal.*

Proof. Suppose $\mathrm{Seq}(\mathcal{E})$ is not maximal, then it contains a subproof π ending with a decide inference s.t. there exists a proof π' with $\pi \sim \pi'$ and $\mathrm{foci}\,(\pi) \subset_{\mathrm{multiset}} \mathrm{foci}\,(\pi')$. So there is an existential formula $\exists x.\, A$ in $\mathrm{foci}\,(\pi') \setminus \mathrm{foci}\,(\pi)$ to which in $\pi_{\mathcal{E}}$ corresponds an expansion $\exists x.\, A +^t E'$. As rule permutations allow to shift down the instantiation of the expansion term t over all $\forall$-inferences, the term t must be $<_{\mathcal{F}}$-minimal for $\mathcal{F}$ being the expansion sequent corresponding to the conclusion sequent of π in $\mathrm{Seq}(\mathcal{E})$. This is a contradiction to the choice of Δ'' and Δ_i' made in definition 19. ◀

▶ **Lemma 24.** *If π is a maximal LKNF-proof, then $\pi \cong \mathrm{Seq}(\mathcal{E}(\pi))$.*

Proof. We proceed by induction on the decision depth of π. If π ends with a positive phase, it is of the form (a) below where the A_i are non-existential formulas and $\pi' \cong \mathrm{Seq}(\mathcal{E}(\pi'))$ by induction hypothesis.

$$\text{(a)}\quad \dfrac{\begin{array}{c}(\pi')\\ \vdash \Gamma' \Downarrow [\vec{t}_1/\vec{x}_1]A_1, \dots, [\vec{t}_n/\vec{x}_n]A_n\\ \vdots\\ \vdash \Gamma' \Downarrow \exists \vec{x}.\, A_1, \dots, \exists x.\, A_n\end{array}}{\vdash \Gamma \Uparrow \cdot}\ \mathsf{decide} \qquad\qquad \text{(b)}\quad \dfrac{\begin{array}{c}\overset{(\pi_1)}{\vdash \Gamma, \Delta_1 \Uparrow \cdot} \quad \cdots \quad \overset{(\pi_n)}{\vdash \Gamma, \Delta_n \Uparrow \cdot}\\ \vdots\\ \vdash \Gamma \Uparrow \Delta\end{array}}{\vdash \Gamma \Downarrow \Delta}\ \mathsf{release}$$

As π is maximal, the existential inferences in this phase are in 1-1 correspondence to the $<_{\mathcal{E}(\pi)}$-minimal expansion terms of $\mathcal{E}(\pi)$. Therefore, by definition 19, Seq creates the shown segment of π from $\mathcal{E}(\pi)$ up to permutations of the existential inferences inside this segment.

If π ends with a negative phase, then it is of the form (b) above where Δ does not contain an existential formula. If $n = 0$, then π consists only of this phase and we are done. Otherwise we have $\pi_i \cong \mathrm{Seq}(\mathcal{E}(\pi_i))$ for $i = 1, \ldots, n$ by induction hypothesis. For fixed Δ, the sequents $\vdash \Gamma, \Delta_1 \Uparrow \cdot, \ldots, \vdash \Gamma, \Delta_n \Uparrow \cdot$ are uniquely determined and there are no decide and existential inferences in the negative phase so we obtain $\pi \cong \mathrm{Seq}(\mathcal{E}(\pi))$. ◀

A maximal proof corresponding to π can be obtained via rule permutations as in the first part of theorem 6. Reading off an expansion tree from π and then re-sequentializing this tree gives an alternative way to compute a maximal proof as the following theorem shows.

▶ **Theorem 25.** *For any LKNF proof π:* $\mathrm{Seq}(\mathcal{E}(\pi)) \in \max(\pi)$.

Proof. By the first part of theorem 6 there is a $\pi' \sim \pi$ with $\pi' \in \max(\pi)$. Applying lemma 24 to π' shows that $\pi' \cong \mathrm{Seq}(\mathcal{E}(\pi'))$ and hence $\mathrm{Seq}(\mathcal{E}(\pi')) \in \max(\pi)$ but by theorem 22 we have $\mathcal{E}(\pi') = \mathcal{E}(\pi)$, so we obtain $\mathrm{Seq}(\mathcal{E}(\pi)) \in \max(\pi)$. ◀

We can now finally obtain the equivalence of expansion trees and maximal proofs with respect to the induced identity notion for proofs. This theorem is our main technical result about proofs in first-order classical logic: the abstractions of *LKNF* proofs provided by expansion trees and by maximal multi-focusing are the same.

▶ **Theorem 26.** *Let π_1, π_2 be LKNF proofs. Then $\mathcal{E}(\pi_1) = \mathcal{E}(\pi_2)$ iff* $\max(\pi_1) = \max(\pi_2)$.

Proof. For the left-to-right direction let $E = \mathcal{E}(\pi_1) = \mathcal{E}(\pi_2)$. Theorem 25 then implies that that $\mathrm{Seq}(E)$ is in both $\max(\pi_1)$ and $\max(\pi_2)$, so $\max(\pi_1) = \max(\pi_2)$. The right-to-left direction follows directly from theorem 22. ◀

5 Conclusion

We have illustrated that, instead of discarding the sequent calculus in search of canonical proof systems, sequent proofs can be systematically abstracted by (maximal) multi-focusing into canonical structures. In this paper, we have imposed a particular focusing discipline on classical sequent proofs—negatively polarized propositional connectives and delayed literals—and have then showed that maximal multi-focusing in the sequent calculus yields the parallel and minimalistic notion of proofs based on expansion trees. Our framework is obviously generative as well: there are other polarizations within classical logic and in focused proof systems for intuitionistic and linear logics. Maximal multi-focusing yields different canonical structures for these other polarizations.

Acknowledgments

This research has been funded in part by the ERC Advanced Grant ProofCert and by the ANR-FWF project STRUCTURAL.

References

1 J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

2 P. B. Andrews. Theorem-proving via general matings. *J. ACM*, 28:193–214, 1981.

3 M. Baaz and S. Hetzl. On the non-confluence of cut-elimination. *J. of Symbolic Logic*, 76(1):313–340, 2011.

4 M. Baaz, S. Hetzl, and D. Weller. On the complexity of proof deskolemization. *J. of Symbolic Logic*, 77(2):669–686, 2012.

5 W. Bibel. Matrices with connections. *J. of the ACM*, 28:633–645, 1981.

6 K. Chaudhuri. Focusing strategies in the sequent calculus of synthetic connectives. *LPAR 2008*, *LNCS* 5330, pages 467–481. Springer, Nov. 2008.

7 K. Chaudhuri, D. Miller, and A. Saurin. Canonical sequent proofs via multi-focusing. In *IFIP TCS*, *IFIP* 273, pages 383–396. Springer, Sept. 2008.

8 V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. *Journal of Symbolic Logic*, 62(3):755–807, 1997.

9 A. Felty. Transforming specifications in a dependent-type λ-calculus to specifications in an intuitionistic logic. In G. Huet and G. D. Plotkin, eds, *Logical Frameworks*. CUP 1991.

10 G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

11 J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

12 J.-Y. Girard. A new constructive logic: classical logic. *Math. Structures in Comp. Science*, 1:255–296, 1991.

13 A. Guglielmi, T. Gundersen, and M. Parigot. A proof calculus which reduces syntactic bureaucracy. In *RTA 2010*, *LIPIcs* 6, pages 135–150, July 2010.

14 W. Heijltjes. Classical proof forestry. *A. of Pure and Applied Logic*, 161:1346–1366, 2010.

15 H. Herbelin and A. Saurin. λ-calculus and Λ-calculus: a capital difference. Unpublished manuscript, 2010.

16 J. Herbrand. *Recherches sur la Théorie de la Démonstration.* PhD thesis, Paris, 1930.

17 S. Hetzl. The Computational Content of Arithmetical Proofs. to appear in volume 53 of the *Notre Dame Journal of Formal Logic.*

18 S. Hetzl. Applying Tree Languages in Proof Theory. In *Language and Automata Theory and Applications (LATA) 2012*, *LNCS* 7183, pages 301–312. Springer, 2012.

19 S. Hetzl and L. Straßburger. Herbrand-Confluence for Cut-Elimination in Classical First-Order Logic. In *Computer Science Logic (CSL) 2012.* to appear.

20 D. J. D. Hughes. Proofs without syntax. *A. of Mathematics*, 143(3):1065–1076, Nov. 2006.

21 F. Lamarche and L. Straßburger. Naming proofs in classical propositional logic. In P. Urzyczyn, editor, *TLCA 2005*, *LNCS* 3461, pages 246–261. Springer, 2005.

22 F. Lamarche and L. Straßburger. From proof nets to the free *-autonomous category. *Logical Methods in Computer Science*, 2(4:3):1–44, 2006.

23 J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Cambridge University Press, 1986.

24 C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

25 R. McKinley. Herbrand expansion proofs and proof identity. In *CL&C 2008.*

26 R. McKinley. Expansion nets: Proof-nets for propositional classical logic. In C. G. Fermüller and A. Voronkov, editors, *LPAR 2010 LNCS* 6397, pp 535–549, Indonesia, Springer.

27 D. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

28 D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

29 E. P. Robinson. Proof nets for classical logic. *J. of Logic and Comp.*, 13(5):777–797, 2003.

30 L. Straßburger. What is the problem with proof nets for classical logic? In *CiE 2010*, *LNCS* 6158, pages 406–416, Ponta Delgada, Azores, Portugal, June 2010. Springer.

31 C. Urban. *Classical Logic and Computation.* PhD thesis, University of Cambridge, 2000.

32 C. Urban and G. M. Bierman. Strong normalisation of cut-elimination in classical logic. *Fundamenta Informaticae*, 45(1–2):123–155, 2001.

ML with PTIME complexity guarantees*

Jacek Chrząszcz and Aleksy Schubert

University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland
`{chrzaszcz,alx}@mimuw.edu.pl`

Abstract

Implicit Computational Complexity is a line of research where the possibility to inference a valid property for a program implies that the program runs in particular complexity class. Soft type systems are one of the research threads within the field. We present here a soft type system with ML-like polymorphism that enjoys decidable typechecking, type inference and typability problems and gives polynomial time computational guarantees for the running time of typed programs.

1998 ACM Subject Classification F.3.3 Studies of program constructs, F.4.1 Mathematical logic

Keywords and phrases implicit computational complexity, polymorphism, soft type assignment

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.198

1 Introduction

The design of a programming language may be focused on guarantees the language gives to a programmer or a software consumer. Implicit Computational Complexity studies machine-free methods to characterise particular complexity class, e.g. PTIME, NP, PSPACE. This line of research may lead not only to an interesting programming language, but also can give new insights to the theoretical analysis of the subject class.

Soft type systems proposed by Gaboardi et al. [11, 12] that emerged from the Soft Linear Logic (SLL) of Lafont [19], are one of the proposals which makes the expected guarantees with relatively low annotation burden. One drawback of the soft type systems is that they use full second-order polymorphism to gain necessary uniformity of representation [20]. This, however, results in undecidable type-checking and type inference [6]. We regain the necessary uniformity by introduction of special constants similar in fashion to the ones used in [8] to obtain the completeness.

Linear logic brought many characterisations of complexity classes in addition to the well established proposals such as [5, 7, 9, 18, 23, 27], to mention few. This was started by Girard in [16] where the Light Linear Logic (LLL) and Elementary Linear Logic (ELL) were proposed to characterise polynomial and elementary time complexities respectively by means of the cut elimination procedure. The ideas of LLL were taken up by Asperti and Roversi [1] who designed a more flexible affine variant of Girard's logic called Light Affine Logic. A type system which is based on these ideas was presented in [4]. Another line of research on linear logic and complexity classes was started by Lafont's Soft Linear Logic (SLL) [19] which characterises polynomial time complexity and is the starting point for the Soft Type Assignment (STA) systems by Gaboardi et al. [11, 12, 14] where certain form of typability guarantees reduction of lambda terms in polynomial time. The light logic principles have been used to characterise other interesting complexity classes for instance [25] uses Light

[*] This work was partially supported by the Polish government grant no N N206 355836.

Linear Logic with additional operation + to characterise NP; LOGSPACE is characterised by different versions of Stratified Bounded Affine Logic (SBAL) in e.g. [30, 21].

The type systems that are based on linear logic employ linear modalities (e.g. ! or §) to guard the necessary restrictions. The modalities control duplication of data by marking in the type the particular function argument that is multiplied during the computation. This has effect similar to the one obtained by Bellantoni, Cook and Leivant, but in a way which results from more basic assumptions. For instance, their restriction is obtained by Baillot et al. in [2] due to prefixing of arguments with § and ! in Def. 5. Similar function has the prefixing with ! in STA by Gaboardi and Ronchi Della Rocca [14].

In this paper we present a version of STA [14] with ML-like polymorphism and useful data types such as booleans, integers and strings. The ambition of the paper is similar to the one of [2] to present a contribution close to real language. However, we move the focus here to polymorphism which is not present in the contribution of Baillot et al. nor in earlier papers on monomorphic calculi [10, 6]. The ML polymorphism in our setting is presented in a traditional way which contrasts with the presentation of [22] where a division into upper class and lower class types is used. Moreover, we use linear equations over natural numbers to express the necessary constraints, which is conceptually simpler to the approach by Dal Lago, Schöpp where E-unification is used or the approach by Baillot, Martin [3] where additional disequality constraints are used.

We observe that the set of obtained functionals, in addition to the running time guarantees, provides a natural way to program in a design pattern present in imperative programming. When the pattern is followed all the memory is allocated before any essential computation is done. In this way the dynamic allocation is no longer needed in the course of computation. This way of programming is advised both by the Java Card manufacturers, see e.g. [15, Sect. 2.4.3], and by software verification community, see e.g. [26, Sect. 3].

This paper is structured as follows. We present the syntax and semantics of the system in Sect. 2. Then we explain the primitives of the language in Sect. 3. Basic properties of MLSTA are presented in Sect. 4. The complexity guarantees of the system are proved in Sect. 5 while the decidability of type related problems in Sect. 6. We conclude in Sect. 7.

2 ML-like system MLSTA

We propose a type system which makes possible a more uniform treatment of input data. The system is inspired by ML and uses several algebraic types. Its syntax and types are defined as follows:

$$\begin{array}{rll} A ::= \alpha \mid \sigma \multimap A \mid A \otimes B \mid \mathbb{S}_i^{!j} \mid (\mathbb{S}_i^{!j})^k \boxtimes A \mid \mathbb{N}^{!j} \mid \mathbb{B} & \text{(Linear Types)} & (1)\\ \mathfrak{s} ::= !^i \forall \vec{\alpha}.A & \text{(Type Schemes)} & \\ \sigma ::= !^i A & \text{(Types)} & \\ M ::= x \mid \lambda x.M \mid M_1 M_2 \mid \texttt{let}\ x = M_1\ \texttt{in}\ M_2 \mid c & \text{(Terms)} & \end{array}$$

where $\alpha \in \mathcal{V}$, which is a countable set of type variables, $i, j, k \in \mathbb{N}$, i.e. natural numbers, with $!^0$ meaning no ! at all, and c is a constant from the set $\mathrm{Const}_{\mathrm{MLSTA}}$ listed in Fig. 2. The set of linear types generated from the nonterminal A above is denoted $\mathcal{T}_A$, the set of type schemes generated from the nonterminal $\mathfrak{s}$ is denoted $\mathcal{T}_{\mathfrak{s}}$, and the set of types generated from the nonterminal σ is denoted $\mathcal{T}_\sigma$. The contexts in this system are sets of pairs $x : \sigma$ or $x : \mathfrak{s}$.

The reduction relation contains β rules and δ rules presented in Fig. 2. The presentation of the typing rules requires a notion of a closure with respect to a context. For a type A

and a context Γ we define the *closure* of A with respect to Γ as $\mathrm{Clos}(\Gamma; A) = \forall\alpha_1, \ldots, \alpha_n.A$ where $\{\alpha_1, \ldots, \alpha_n\} = \mathrm{FTV}(A)\backslash\mathrm{FTV}(\Gamma)$. The typing rules of the system are presented in Fig. 1. To express that $\Gamma \vdash M : A$ is derivable in MLSTA we write $\Gamma \vdash_{\mathrm{MLSTA}} M : A$. We introduce a succinct notation for derivations inspired by the Church-style form:

$$\begin{aligned}\mathfrak{D} ::=& x^A \mid x^{A\leq\mathfrak{s}} \mid \langle\mathfrak{D}, x : A\rangle^w \mid \lambda x^\sigma.\mathfrak{D} \mid \mathfrak{D}_1\mathfrak{D}_2 \mid \langle\xi x_1, \ldots, x_n : \sigma.\mathfrak{D}, x : !\sigma\rangle^m \mid \langle\mathfrak{D}\rangle^{sp} \mid \\ & \texttt{let } x^{!^i\forall\vec{\alpha}.B} = \mathfrak{D}_1 \texttt{ in } \mathfrak{D}_2\end{aligned}$$

The subsequent cases in the above definition are in direct correspondence with the rules presented in Fig. 1. This correspondence makes possible to represent uniquely derivations in MLSTA with the terms defined above (still some terms generated with the grammar above have no corresponding derivation in MLSTA). We sometimes write $\langle\mathfrak{D}, \vec{x} : \vec{A}\rangle^{wn}$ to denote n-times application of the rule (w) with pairs $x_1 : A_1, \ldots, x_n : A_n$. In case a derivation $\mathfrak{D}$ ends with a judgement $\Gamma \vdash M : \sigma$ then we let $\mathfrak{D}_{term} = M$, $\mathfrak{D}_{ctxt} = \Gamma$, and $\mathfrak{D}_{type} = \sigma$.

$$\frac{\mathfrak{s} \geq A}{x : \mathfrak{s} \vdash x : A}\ (Ax) \qquad \frac{x : \mathfrak{s} \in \mathrm{Const}_{\mathrm{MLSTA}} \quad \mathfrak{s} \geq A}{\vdash x : A}\ (AxC) \qquad \frac{\Gamma \vdash M : \sigma}{\Gamma, x : A \vdash M : \sigma}\ (w)$$

$$\frac{\Gamma, x : \sigma \vdash M : A}{\Gamma \vdash \lambda x.M : \sigma \multimap A}\ (\multimap I) \qquad \frac{\Gamma \vdash M : \sigma \multimap A \qquad \Delta \vdash N : \sigma \quad \Gamma\#\Delta}{\Gamma, \Delta \vdash MN : A}\ (\multimap E)$$

$$\frac{\Gamma, x_1 : \xi, \ldots, x_n : \xi \vdash M : \tau \quad \xi \in \mathcal{T}_\sigma \cup \mathcal{T}_\mathfrak{s}}{\Gamma, x : !\xi \vdash M[x/x_1, \ldots, x/x_n] : \tau}\ (m) \qquad \frac{\Gamma \vdash M : \sigma}{!\Gamma \vdash M : !\sigma}\ (sp)$$

$$\frac{\Gamma \vdash M_1 : !^iB \qquad \Delta, x : !^i\mathrm{Clos}(\Gamma, \Delta; B) \vdash M_2 : A \quad \Gamma\#\Delta}{\Gamma, \Delta \vdash \texttt{let } x = M_1 \texttt{ in } M_2 : A}\ (let)$$

Figure 1 The typing rules of MLSTA.

3 Gentle introduction to MLSTA

The main feature of soft type systems is their ability to control multiplication of data. One piece of the mechanism is realised by the (m) rule. This rule makes explicit the demand of an operator to duplicate some portion of data. The multiplication is reflected by ! in the type. The second piece of the mechanism is realised by the (sp) rule. The latter is used when the term it types is to be directly multiplied by a substitution present in the β-reduction. Note that when a term M_1 is directly multiplied k_1 times and is used inside another term M_2 that is directly multiplied k_2 times the number of occurrences of M_1 at some point of the computation may be as high as $k_1 \cdot k_2$.

The traditional soft type assignment systems use the full polymorphism of the System F. This makes possible to conveniently define data types and iterators over them, but it leads to undecidability of the type inference and type checking problems [6]. In our proposal, we provide access to the polymorphic expressibility in a structured fashion. It is achieved through two mechanisms. The first one brings a few fixed algebraic types: strings, naturals, booleans and products with appropriate constructors, destructors and iterators. The set of algebraic types could be richer. This, however, would make the design of the model language unnecessarily complicated. The second mechanism is the traditional let-polymorphism which makes possible to define generic operations that work for different kinds of data.

The language we propose contains also a type of lists of booleans $\mathbb{S}_i^{!j}$, called here strings, which is our main recursive data structure The numbers i and j describe the complexity of the

$$\texttt{let}\ x = M_1\ \texttt{in}\ M_2 \to_\beta M_2[M_1/x]$$
$$(\lambda x.M)N \to_\beta M[N/x]$$

Constants:

The typed constants $c : \tau$ listed below belong to the set $\mathrm{Const}_{\mathrm{MLSTA}}$.

$$\begin{array}{rcl}
\multicolumn{3}{l}{\underline{\text{Product}}}\\
\langle\cdot,\cdot\rangle & : & \forall\alpha\beta.\alpha \multimap \beta \multimap \alpha\otimes\beta\\
\texttt{match} & : & \forall\alpha\beta\gamma.\alpha\otimes\beta \multimap (\alpha\multimap\beta\multimap\gamma)\multimap\gamma\\
 & & \texttt{match}\ \langle M_1, M_2\rangle\ N \to_\delta N\ M_1\ M_2\\
\multicolumn{3}{l}{\underline{\text{Booleans}}}\\
\mathbf{0} & : & \mathbb{B}\\
\mathbf{1} & : & \mathbb{B}\\
\texttt{ifte} & : & \forall\alpha.\mathbb{B}\multimap\alpha\multimap\alpha\\
 & & \texttt{ifte}\ \mathbf{0}\ M_1\ M_2 \to_\delta M_1\\
 & & \texttt{ifte}\ \mathbf{1}\ M_1\ M_2 \to_\delta M_2\\
\multicolumn{3}{l}{\underline{\text{Natural numbers}}}\\
\underline{n} & : & \mathbb{N}^{!j}\\
\texttt{add} & : & \mathbb{N}^{!j_1}\multimap\mathbb{N}^{!j_2}\multimap\mathbb{N}^{!\max(j_1,j_2)+1}\\
 & & \texttt{add}\ \underline{n}\ \underline{m} \to_\delta \underline{n+m}\\
\texttt{mul} & : & \mathbb{N}^{!j_1}\multimap\mathbb{N}^{!j_2}\multimap\mathbb{N}^{!(j_1+j_2)}\\
 & & \texttt{mul}\ \underline{n}\ \underline{m} \to_\delta \underline{n*m}\\
\texttt{iter} & : & \forall\alpha.\mathbb{N}^{!j}\multimap !^j(\alpha\multimap\alpha)\multimap\alpha\multimap\alpha\\
 & & \texttt{iter}\ \underline{n}\ F\ M \to_\delta F\ (\dots(F\ M)\dots)\quad (F \text{ applied } n \text{ times to } M)\\
\multicolumn{3}{l}{\underline{\text{Strings}}}\\
[\cdot;\dots;\cdot] & : & \mathbb{B}^i\multimap\cdots\multimap\mathbb{B}^i\multimap\mathbb{S}_i^{!j}\\
\texttt{create} & : & \mathbb{N}^{!j}\multimap !^j\mathbb{B}^i\multimap\mathbb{S}_i^{!j}\\
 & & \texttt{create}\ \underline{n}\ M \to_\delta [M;\dots;M]\quad (n \text{ copies of } M)\\
\texttt{concat} & : & \mathbb{S}_i^{!j_1}\multimap\mathbb{S}_i^{!j_2}\multimap\mathbb{S}^{!\max(j_1,j_2)+1}\\
 & & \texttt{concat}\ [M_1;\dots;M_m]\ [N_1;\dots;N_n] \to_\delta [M_1;\dots;M_m;N_1;\dots;N_n]\\
\texttt{len} & : & \mathbb{S}_i^{!j}\multimap\mathbb{N}^{!j}\\
 & & \texttt{len}\ [M_1;\dots;M_m] \to_\delta \underline{m}\\
\multicolumn{3}{l}{\underline{\text{Looping constructs}}}\\
\texttt{localvars} & : & \forall\alpha.\mathbb{S}_i^{!j}\multimap\cdots\multimap\mathbb{S}_i^{!j}\multimap\alpha\multimap(\mathbb{S}_i^{!j})^k\boxtimes\alpha\quad (k+1 \text{ arguments})\\
\mathfrak{p}_0,\dots,\mathfrak{p}_{k-1} & : & \forall\alpha.(\mathbb{S}_i^{!j})^k\boxtimes\alpha\multimap\mathbb{S}_i^{!j+1}\\
\mathfrak{p}_k & : & \forall\alpha.(\mathbb{S}_i^{!j})^k\boxtimes\alpha\multimap\alpha\\
 & & \mathfrak{p}_l(\texttt{localvars}\ M_0\dots\ M_k) \to_\delta M_l\\
\texttt{step} & : & ((\mathbb{B}^i\otimes\mathbb{B})^k\otimes\mathbb{B}^q\multimap(\mathbb{B}^i\otimes\mathbb{B}^l)^k\otimes\mathbb{B}^q)\multimap(\mathbb{S}_i^{!j})^k\boxtimes\mathbb{B}^q\multimap(\mathbb{S}_i^{!j})^k\boxtimes\mathbb{B}^q\\
 & & \qquad \text{where } l = \lceil\log(k+1)\rceil\\
 & & \texttt{step}\ F\ (\texttt{localvars}\ M_0\dots\ M_k) \to_\delta \texttt{localvars}\ M'_0\dots\ M'_k\\
 & & \qquad \text{where } F\ \langle \mathrm{hd}(M_0),\dots,\mathrm{hd}(M_{k-1}),M_k\rangle \to^*_{\beta\delta} \langle N_0,\dots,N_{k-1},M'_k\rangle\\
 & & \qquad\qquad N_i = \langle P_i, \overline{l_i}\rangle\\
 & & \qquad\qquad\quad \text{where } \bar{l} \text{ denotes the binary encoding of a number } l\\
 & & \qquad\qquad M'_j = P_{i_1} :: \cdots :: P_{i_w} :: \mathrm{tl}(M_j)\\
 & & \qquad\qquad\quad \text{where } i_1\dots i_w \text{ is the subsequence of } 0\dots k-1 \text{ such that}\\
 & & \qquad\qquad\quad \forall p\ l_{i_p} = j \text{ and } M_{i_p} \neq []\\
 & & \qquad\qquad \mathrm{hd}([]) = \langle\mathbf{0}^i,\mathbf{1}\rangle \qquad \mathrm{tl}([]) = []\\
 & & \qquad\qquad \mathrm{hd}([A_1;\dots;A_m]) = \langle A_1,\mathbf{0}\rangle \qquad \mathrm{tl}([A_1;\dots;A_m]) = [A_2;\dots;A_m]
\end{array}$$

Figure 2 MLSTA constants, their types and reduction rules of MLSTA. Note that $j, j_1, j_2 \geq 1$ when they represent the number of !'s. In addition the superscript of the form $!j$ indicates that the type has j 'hidden' bangs (they become explicit after a translation to STA, see Fig. 4).

string: i corresponds to the size of a symbol in the string — each symbol is of type $\mathbb{B}^i$, which is a shorthand for $\mathbb{B} \otimes \cdots \otimes \mathbb{B}$ (i times), while j, roughly speaking, reflects the complexity of the string creation. Strings are used in the framework to simulate PTIME computations. We provide a number of usual operators over strings. A string which combines n pieces of basic data can be obtained using the bracket construct $[a_1; \ldots ; a_n]$. We can also create a string of n copies of a particular piece of data `create` n a. The strings s_1, s_2 obtained in one or another way can be combined by concatenation done with `concat` s_1 s_2.

Note that traditional iterative operation `fold` from functional languages is missing in MLSTA. In principle we could introduce it to our language. However the traditional type of the operation imposes too strict restrictions on the type of function that operates on strings. In particular it is impossible to transform one string to another since this requires duplication of the string constructor which is prohibited in the context of linear types.

We adopt a different approach. In order to do iterative programs on strings, we introduce a `step` function, inspired by the encoding of a Turing Machine in [24]. Its functionality is to put a program fragment, represented by the first argument f, into the context of an iterative loop. Then `step` takes a tuple of several strings $(\mathbb{S}_i^{!j})^k \boxtimes \alpha$ “federated” into a context of “local variables”. This structure is created with the `localvars` operation and it is the structure over which the iteration is actually performed. In one iteration step the heads of the strings can be freely moved around or dumped, but not duplicated. Some information can also be stored in the accumulator, represented here by α. One application of `step` f maps the operation on heads done by its argument function f onto the corresponding operation on federated strings. The mapped operation can then be iterated by `iter` as many times as necessary in order to complete the whole string processing and in the end we can extract the result using one of the projections $\mathfrak{p}_0, \ldots, \mathfrak{p}_k$. Note that since the calculation is done in constant space, all allocations must occur before starting the iteration. Indeed, it would be impossible to extend any of the strings in the iterated function, as this would make types incompatible. The simplest example of `step` usage is string reversal:

```
let rev = let fr = λ ((a_0,b_0),(a_1,b_1),q). ((a_0,1̄),(a_1,1̄),q) in
  λs. 𝔭_1 (iter (len s) (step fr) (localvars s [] 0))
```

In the above example we use a few syntactic simplifications, which are straightforward to translate into core MLSTA. The federated tuple consists here of two strings and a (dummy) boolean. In the beginning, the left string is the initial string s and the right one is the empty string. The function `fr` takes two “heads”: a_0 and a_1 paired with the booleans b_0 and b_1 respectively, carrying the information if the given head is real or dummy (in case the given string is empty). Its result tells the `step` function to attach both a_0 and a_1 to the right string in the order from right to left, i.e., a_1 is attached first (where it came from) and then a_0. In case one of the strings was initially empty (here it is only possible for the right one), the head on the corresponding position would be attached with a dummy cons (observe condition $M_{i_p} \neq []$ in Fig. 2), i.e. would not be attached at all.

Now it is also possible to write the map function on strings:

```
let map = let fm = λ f ((a_1,b_1), (ar,br), (a_2,b_2)), q).
    if q then ((a_1,1̄), (ar,1̄), (a_2,2̄), if b_1 then 0 else 1)
    else ((a_1,1̄), (f ar,2̄), (a_2,2̄), 1)
  in λ f s. let n = len s in
    𝔭_2 (iter (add (mul 2̲ n) 1̲) (step (fm f)) (localvars s [] [] 0));;
```

Another syntactic simplification can be seen here: although the two branches of `if` share variables, the term can be written as linear using the trick `if b then t[a] else s[a]` $\equiv$

`(if b then λa.t[a] else λa.s[a]) a`. The function operates in two phases: in the first phase the input string (put initially on the left federated string) is reversed and placed on the middle string. In the second phase the middle string is mapped using f to the right string, and reversed back to the original order in the process. The number of iterations is $2n+1$, which is n for each phase and 1 for phase change. The phase number is encoded as a boolean, initially true ($\mathbf{0}$) and then changed to false ($\mathbf{1}$).

```
let fsel = λ ((sortedPartHead, sortedPartHasHead),
              (maxSoFar, maxUsable),
              (unsortedPartHead, unsortedPartHasHead),
              (soFarSeenHead, soFarSeenHasHead), x).
  if not maxUsable and unsortedPartHasHead and not soFarSeenHasHead then
     (* start of selection phase *)
    (sortedPartHead, 0̅), (maxSoFar, 1̅),
    (unsortedPartHead, 1̅), (soFarSeenHead, 3̅), x
  else if maxUsable and unsortedPartHasHead then (* selection phase in progress *)
    cmp maxSoFar unsortedPartHead (fun smaller greater ->
      (sortedPartHead, 0̅), (greater, 1̅),
      (smaller, 3̅), (soFarSeenHead, 3̅), x )
  else if maxUsable and not unsortedPartHasHead then (* end of selection phase *)
    if sortedPartHasHead then (* end of selection phase for the first selection *)
      (maxSoFar, 0̅), (sortedPartHead, 0̅),
      (unsortedPartHead, 2̅), (soFarSeenHead, 3̅), x
    else (* end of selection phase for other selections *)
      (sortedPartHead, 0̅), (maxSoFar, 0̅),
      (unsortedPartHead, 2̅), (soFarSeenHead, 3̅), x
  else (* preparation for the next selection phase *)
    if unsortedPartHasHead and soFarSeenHasHead then
      (sortedPartHead, 0̅), (maxSoFar, 1̅),
      (soFarSeenHead, 2̅), (unsortedPartHead, 2̅), x
    else
      (sortedPartHead, 0̅), (maxSoFar, 1̅),
      (unsortedPartHead, 2̅), (soFarSeenHead, 2̅), x

let ssort = λl. let n = len l in
  𝔭0 (iter (add (mul n n) n) (step fsel) (localvars [] [] l [] 0))
```

Figure 3 Selection sort. We encourage the reader to try to understand the algorithm herself.

It is very interesting to note that the type of a map function defined in this way is $!^{j+2}(\mathbb{B}^i \multimap \mathbb{B}^i) \multimap !\mathbb{S}_i^{!j} \multimap \mathbb{S}_i^{!j+1}$, while the type of a map function defined directly in STA corresponds to $!^j(\mathbb{B}^i \multimap \mathbb{B}^i) \multimap \mathbb{S}_i^{!j} \multimap \mathbb{S}_i^{!j+1}$. The difference comes from the fact that in MLSTA one iterates over natural numbers and in STA directly on the string itself.

Using this technique it is possible to program more complex functions on lists, e.g. sorting, in particular selection sort, as shown in Fig. 3. This example uses another syntactic trick: since boolean values can be freely multiplied using terms similar to $\mathbf{cnt} \equiv \lambda b.\mathtt{ifte}\ b\ \langle \mathbf{0}, \mathbf{0} \rangle\ \langle \mathbf{1}, \mathbf{1} \rangle$ one does not need to worry about how many times a given boolean variable is used in the term. Technically, sorting consists in running n phases of selecting the largest element from unsorted remaining part of the initial string. In each phase one needs to reverse the list twice, that is why we need $n^2 + n$ steps. It is interesting to note that the choice of applying the cons in the same order as they appeared originally comes at a cost of breaking symmetry of certain operations between two strings. Indeed, while it is straightforward to reverse a string from left to right (as in the `rev` example above), reversing it from right to left (as is done in the third case in Fig. 3) is a bit more technical.

It is worth stressing that the simulation of a Turing Machine we present below is in fact a paradigmatic example of a natural computation that can be performed in our language.

4 Properties of MLSTA

Many of the results in this paper can be obtained in a simpler way when we operate not just on any derivation, but on a derivation in a special, regular form. We start with its presentation, which is of interest not only for technical reasons but also, as usual in such cases, it indicates the presence of a few important tautologies (however, their further exploration goes beyond the topic of the paper).

▶ **Definition 1** (derivations in normal form)**.** A derivation $\mathfrak{D}$ of MLSTA is in *normal form* when $\mathfrak{D} = \langle\hat{\mathfrak{D}}, x : A\rangle^w$ with $\hat{\mathfrak{D}}$ in normal form and $x \notin \mathrm{FV}(\hat{\mathfrak{D}}_{term})$ or is in (m)-normal form.

A derivation $\mathfrak{D}$ of MLSTA is in (m)*-normal form* when $\mathfrak{D} = \langle \xi x : A.\langle\hat{\mathfrak{D}}, x : A\rangle^w, y : !^n A\rangle^{mn}$ with $\hat{\mathfrak{D}}$ in (m)-normal form and $x \notin \mathrm{FV}(\hat{\mathfrak{D}}_{term})$ or is in (sp)-normal form.

A derivation $\mathfrak{D}$ of MLSTA is in (sp)*-normal form* when $\mathfrak{D} = \langle\hat{\mathfrak{D}}\rangle^{sp}$ with $\hat{\mathfrak{D}}$ in (sp)-normal form or in logical normal form.

A derivation $\mathfrak{D}$ of MLSTA is in *logical normal form* when it is

- x^A for some variable x,
- $x^{A\leq\mathfrak{s}}$ for some variable x,
- $\lambda x^\sigma.\hat{\mathfrak{D}}$ for some variable x and $\hat{\mathfrak{D}}$ in logical normal form,
- $\lambda x^\sigma.\langle\hat{\mathfrak{D}}, x : A\rangle^w$ for some variable x and $\hat{\mathfrak{D}}$ in logical normal form with $x \notin \mathrm{FV}(\hat{\mathfrak{D}}_{term})$,
- $\lambda x^\sigma.\langle \xi x : A.\hat{\mathfrak{D}}, x : !^k A\rangle^{mk}$ for some variable x and $\hat{\mathfrak{D}}$ in logical normal form with $x \notin \mathrm{FV}(\hat{\mathfrak{D}}_{term})$,
- $\mathfrak{D}_1\mathfrak{D}_2$ where $\mathfrak{D}_1$ is in logical normal form and $\mathfrak{D}_2$ is in (sp)-normal form,
- `let` $x^{!^i\forall\vec{\alpha}.B} = \mathfrak{D}_1$ `in` $\mathfrak{D}_2$ where $\mathfrak{D}_1$ is in normal form and $\mathfrak{D}_2$ is in logical normal form.

▶ Proposition 2 (properties of derivations). If $\Gamma \vdash_{\mathrm{MLSTA}} M : \sigma$ then the judgement has a derivation in normal form.

Proof. The proof is using a special kind of reduction the normal forms of which are the defined above normal forms. ◀

As a corollary we obtain a condition that says in which way we can drop a bang in the final type of a term.

▶ **Corollary 3** (dropping final bang)**.** *If* $\Gamma \vdash_{\mathrm{MLSTA}} M : !\sigma$ *then there is a context* Γ' *such that* $!\Gamma' \subseteq \Gamma$ *and* $\Gamma' \vdash_{\mathrm{MLSTA}} M : \sigma$ *and for each* $x : \tau \in \Gamma \backslash !\Gamma'$ *we have* $x \notin \mathrm{FV}(M)$*.*

Proof. By Prop. 2 there is a derivation of $\Gamma \vdash_{\mathrm{MLSTA}} M : !\sigma$ in normal form. We observe that we can one by one remove the final (w) and (m) rules. At the end we have to arrive at an (sp) rule since no logical normal form can assign a ! type to a term. ◀

A crucial part of the subject reduction proof is the interaction between substitutions and the derivations. This is expressed in the following proposition.

▶ Proposition 4 (derivations and substitutions). If $\Gamma \vdash M : A$ then for each substitution $U : \mathcal{V} \rightharpoonup \mathcal{T}_A$ we have $U(\Gamma) \vdash M : U(A)$.

Proof. Induction over the inference of $\Gamma \vdash M : A$ by cases according to its final rule. ◀

Let expressions

$\texttt{let } x = M \texttt{ in } N = (\lambda x.M)N$

Tensor products

$A \otimes B = \forall \alpha.(A \multimap B \multimap \alpha) \multimap \alpha$

$\langle \cdot, \cdot \rangle = \lambda mnz.zmn$

$\texttt{match} = \lambda pf.pf$

Booleans

$\mathbb{B} = \forall \alpha.\alpha \multimap \alpha \multimap \alpha$

$\mathbf{0} = \lambda tf.t$

$\mathbf{1} = \lambda tf.f$

$\texttt{ifte} = \lambda bxy.bxy$

Natural numbers

$\mathbb{N}^{!j} = \forall \alpha.!^j(\alpha \multimap \alpha) \multimap \alpha \multimap \alpha$

$\underline{n} = \lambda fx.f(\dots(fx)\dots)$ (f applied n times to x)

$\texttt{add} = \lambda pqsz.ps(qsz)$

$\texttt{mul} = \lambda pqsz.p(qs)z$

$\texttt{iter} = \lambda pfx.pfx$

Strings

$\mathbb{S}_i^{!j} = \forall \alpha.!^j(\mathbb{B}^i \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha$

$[\cdot;\dots;\cdot] = \lambda a_1 \dots a_n cz.ca_1(\dots(ca_n z)\dots)$

$\texttt{create} = \lambda nacz.n(ca)z$

$\texttt{concat} = \lambda s_1 s_2 cz.s_1 c(s_2 cz)$

$\texttt{len} = \lambda sfz.s(\lambda x.f)z$

Local variables (for the sake of clarity we retain abbreviations related to $\otimes$ and $\mathbb{B}$)

$(\mathbb{S}_i^{!j})^k \boxtimes A = \forall \alpha.!^{j+1}(\mathbb{B}^i \multimap \alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)^k \otimes A$

$\texttt{localvars} = \lambda s_0 \dots s_{k-1} q \lambda c.\langle s_0 c, \dots, s_{k-1} c, q \rangle$

$\mathfrak{p}_n = \lambda v.\lambda c.\texttt{match}\ (vc)\ \lambda s_0 \dots s_{k-1} q.s_i$ for $n = 0 \dots k-1$

$\mathfrak{p}_k = \lambda v.\texttt{match}\ (v \lambda x.I)\ \lambda s_0 \dots s_{k-1} q.q$

$\texttt{step} =$
$\quad \lambda fvc.\texttt{match}\ (\mathsf{Dec}\ vc)\ \lambda c_0 a_0 b_0 t_0 \dots c_{k-1} a_{k-1} b_{k-1} t_{k-1} q.$
$\qquad \mathsf{Enc}\ c_0 \dots c_{k-1}(f\langle\langle a_0, b_0\rangle, \dots, \langle a_{k-1}, b_{k-1}\rangle, q\rangle)\langle t_0, \dots, t_{k-1}\rangle$, where

$\mathsf{Dec}\ = \lambda vc.\texttt{match}\ (v\ \texttt{F}[c])\ \lambda \tilde{s}_0 \dots \tilde{s}_{k-1} q.$
$\quad \texttt{match}\ (\tilde{s}_0 \langle \lambda az.z, \mathbf{0}^i, \mathbf{1}, \lambda z.z\rangle)\ \lambda c_0 a_0 b_0 t_0.\ \dots$
$\qquad \texttt{match}\ (\tilde{s}_{k-1} \langle \lambda az.z, \mathbf{0}^i, \mathbf{1}, \lambda z.z\rangle)\ \lambda c_{k-1} a_{k-1} b_{k-1} t_{k-1}.$
$\qquad\quad \langle c_0, a_0, b_0, t_0, \dots, c_{k-1}, a_{k-1}, b_{k-1}, t_{k-1}, q\rangle$

where $\texttt{F}[c] = \lambda az.\texttt{match}\ z\ \lambda c'a'b't'.\langle c, a, \mathbf{0}, c'a' \circ t'\rangle$

and

$\mathsf{Enc}\ = \lambda c_0 \dots c_{k-1} wv.\texttt{match}\ w\ \lambda h_0 \dots h_{k-1} q'.$
$\quad \texttt{match}\ h_0\ \lambda a_0' p_0.\ \dots\ \texttt{match}\ h_{k-1}\ \lambda a_{k-1}' p_{k-1}'.$
$\qquad \texttt{match}\ (\mathsf{Add}\ p_0 c_0 a_0'(\dots(\mathsf{Add}\ p_{k-1} c_{k-1} a_{k-1}' v)\dots))$
$\qquad\quad \lambda s_0' \dots s_{k-1}'.\langle s_0', \dots, s_{k-1}', q'\rangle$, where

$\mathsf{Add}\ = \lambda p.\texttt{match}\ p\ \texttt{CASE}_{\lceil \log(k+1) \rceil}[I_0, \dots, I_{k-1}][I]$, where

$I_n = \lambda cam.\texttt{match}\ m\ \lambda s_0 \dots s_{k-1}.\langle s_0, \dots, s_{n-1}, c\,a \circ s_n, s_{n+1} \dots, s_{k-1}\rangle$, and

$I = \lambda cam.m$, and

$$\texttt{CASE}_w[t_0, \dots, t_{n-1}][t] = \begin{cases} \lambda b_0.\texttt{ifte}\ b_0\ \texttt{CASE}_{w-1}[t_0, \dots, t_{2^{w-1}-1}][t] \\ \qquad \texttt{CASE}_{w-1}[t_{2^{w-1}}, \dots, t_{n-1}][t] & \text{if } w > 0 \text{ and } n > 2^{w-1} \\ \lambda b_0.\texttt{ifte}\ b_0\ \texttt{CASE}_{w-1}[t_0, \dots, t_{n-1}][t] \\ \qquad \texttt{CASE}_{w-1}[\,][t] & \text{if } w > 0 \text{ and } 0 < n \le 2^{w-1} \\ t_0 & \text{if } w = 0 \text{ and } n = 1 \\ \lambda b_0 \dots b_{w-1}.t & \text{if } n = 0 \end{cases}$$

Figure 4 Translation of MLSTA to STA.

The proposition above makes it possible to describe the way the type instantiation operation works in the context of derivations.

▶ Proposition 5. If $\Delta \vdash N : A$ then $\Delta \vdash N : A'$ if $\mathrm{Clos}(\Delta, A) \geq A'$.

Proof. This is an instance of Prop. 4, since if $A' = U(A)$ then $\mathrm{dom}(U) \cap \mathrm{FTV}(\Delta) = \emptyset$ and therefore $U(\Delta) = \Delta$. ◀

We can now combine the previous two statements and obtain the substitution lemma for our system.

▶ **Lemma 6** (substitution lemma). *If* $\Gamma; x : !^i \forall \overline{\alpha}.A \vdash_{\mathrm{MLSTA}} M : \tau$ *and* $\Delta \vdash_{\mathrm{MLSTA}} N : !^i A$ *where* $\overline{\alpha} \notin \mathrm{FTV}(\Delta)$*, then* $\Gamma; \Delta \vdash_{\mathrm{MLSTA}} M[N/x] : \tau$

Proof. The proof can be done almost in the same way as the proof of the Substitution Lemma 2.7 in [13], i.e., by generalising the statement to many simultaneous substitutions and proceeding by induction on the derivation by analysis of the last rule. The new/different rules in MLSTA are (Ax), (AxC) and (let).

If the last step is (Ax) then $M = x$ and we have $x : \forall\overline{\alpha}.A \vdash x : B$ with $\forall\overline{\alpha}.A \geq B$ and $\Delta \vdash N : A$ where $\overline{\alpha} \notin \mathrm{FTV}(\Delta)$. Therefore one has $\Delta \vdash N : B$, by Prop. 5, because $\mathrm{Clos}(\Delta; A) \geq B$.

It is impossible that the last step is (AxC), because the context is empty.

If the last rule is (let), the result follows easily by induction hypothesis.

Other MLSTA rules are identical to their $\mathrm{STA_B}$ counterparts. ◀

As a result of the substitution lemma we obtain the subject reduction property.

▶ **Theorem 7** (subject reduction). *If $\Gamma \vdash_{\mathrm{MLSTA}} M : A$ and $M \to_{\beta\delta} M'$ then $\Gamma \vdash_{\mathrm{MLSTA}} M' : A$.*

5 MLSTA and PTIME

Observe that the MLSTA can easily be embedded into STA, in the same fashion as usual ML can be embedded in System F [17, Section 3], see Fig. 4. This gives us polynomial guarantee on the length of reductions.

▶ **Theorem 8.** *Given a derivation $\Gamma \vdash_{\mathrm{MLSTA}} M : \sigma$, the number of reductions from M can be bounded by $|M|^{O(d)}$ where: $|M|$ is the* size *of the term M, defined as usual with one caveat — the size of a natural constant $\underline{n}$ is n; d is the* degree *of a derivation, defined as the maximum nesting of (sp) rules in the derivation.*

Proof. The translation of MLSTA into STA preserves types and degree of derivations, and guarantees that every MLSTA reduction step is translated to a number of steps in STA. The result of translation is bigger only by a linear factor from its original. In the end the polynomial bound established for STA (Theorem 15 in [14]) works for MLSTA as well. ◀

Now, we aim at a proof that each TM in PTIME can be simulated in MLSTA by a term. We start by a version of Lemma 16 and 17 from [14] for our built-in naturals and booleans:

▶ **Lemma 9** (polynomials). *Let P be a polynomial with positive coefficients in the variable x of the degree $\deg(P)$. There is a term $\underline{P}$ such that $\vdash_{\mathrm{MLSTA}} \underline{P} : !^{\deg(P)}\mathbb{N}^{!1} \multimap \mathbb{N}^{!(2\deg(P)+1)}$.*

▶ **Lemma 10** (boolean functions). *Each boolean total function $f : \{0,1\}^n \to \{0,1\}^m$ for $m, n \geq 1$ can be defined by a term $\underline{f}$ typable in MLSTA as $\vdash \underline{f} : \mathbb{B}^n \multimap \mathbb{B}^m$.*

The following theorem shows how one can encode polynomial Turing Machines in MLSTA. For simplicity, we encode only deterministic machines which move their head (left or right) at every step. Therefore a transition function can be encoded as $\delta : \mathbf{\Sigma} \times \mathbf{Q} \to \mathbf{\Sigma} \times \mathbf{Q} \times \mathbb{B}$, where $\mathbf{\Sigma}$ is the alphabet, $\mathbf{Q}$ the set of states and the last boolean value denotes the head move: $\mathbf{0}$ denotes 'left' and $\mathbf{1}$ 'right'. Since $\mathbf{\Sigma}$ and $\mathbf{Q}$ are finite, there exist sufficiently large k and k', such that $\mathbf{\Sigma} \equiv \mathbb{B}^k$ with 0^k representing blank and $\mathbf{Q} \equiv \mathbb{B}^{k'}$. Hence, according to Lemma 10, there exists $\underline{\delta} : \mathbb{B}^k \otimes \mathbb{B}^{k'} \multimap \mathbb{B}^k \otimes \mathbb{B}^{k'} \otimes \mathbb{B}$ representing δ.

▶ **Theorem 11.** *Let $\mathcal{M}$ be a Turing Machine. There is an MLSTA term $M_{\mathcal{M}}$ such that $\vdash_{\mathrm{MLSTA}} M_{\mathcal{M}} : !^d\mathbb{S}_k \multimap \mathbb{B}$ for some d where for each input s the term $M_{\mathcal{M}}\ \underline{s}$ reduces to $\mathbf{0}$ using $R(|s|)$ of reductions with R being a polynomial of degree $O(d)$ if and only if $\mathcal{M}$ accepts s. Moreover, $M_{\mathcal{M}}$ can be constructed from $\mathcal{M}$ in polynomial time.*

Proof. Let $\mathcal{M}$ be a deterministic TM with alphabet $\mathbf{\Sigma} \equiv \mathbb{B}^k$, the set of states $\mathbf{Q} \equiv \mathbb{B}^{k'}$ and transition function δ. Let S be a polynomial defining the maximal length of auxiliary tape of $\mathcal{M}$ for all input strings of a given length. Let T be a polynomial defining the maximal number of steps needed for all input strings of a given length. It is enough to construct the space using $\underline{S}$, the time using $\underline{T}$ and combine it all into the term $M_{\mathcal{M}}$ equal

$$\begin{array}{l} \lambda s.\mathtt{let}\ \mathit{time} = \underline{T}\ (\mathtt{len}\ s)\ \mathtt{in} \\ \quad \mathtt{let}\ \mathit{tape}_0 = \mathtt{concat}\ s\ (\mathtt{create}\ (\underline{S}\ (\mathtt{len}\ s))\ \mathbf{0}^k)\ \mathtt{in} \\ \quad \mathtt{let}\ \mathit{conf}_0 = \mathtt{localvars}\ [\,]\ \mathit{tape}_0\ \underline{q_0}\ \mathtt{in} \\ \quad \mathit{is\_acc}\ (\mathfrak{p}_2\ (\mathtt{iter}\ \mathit{time}\ (\mathtt{step}\ F_\delta)\ \mathit{conf}_0)) \end{array}$$

where $\mathit{is\_acc}$ is a function of type $\mathbf{Q} \multimap \mathbb{B}$ returning $\mathbf{0}$ if the state it receives as input is accepting and F_δ is a simple wrapper around $\underline{\delta}$ to match the input specification of `step`.

$$\begin{array}{l} F_\delta = \lambda\texttt{((al,bl),(a,b),q). match (}\underline{\delta}\texttt{(a,q)) }\lambda\texttt{a'q'd. ifte d ((al,}\overline{1}\texttt{),(a',}\overline{1}\texttt{),q')} \\ \qquad\qquad \texttt{ifte bl ((a',}\overline{0}\texttt{),(al,}\overline{0}\texttt{),q') ((al,}\overline{0}\texttt{),(a',}\overline{0}\texttt{),q')} \end{array}$$

The degree of the type derivation of $M_{\mathcal{M}}$, the degree of terms $M_{\mathcal{M}}\ \underline{s}$ for any s and the parameter d depend in a linear way on the degree of the polynomials T and S. By Theorem 8 each term $M_{\mathcal{M}}\ \underline{s}$ can be reduced to a normal form in the number of reductions bound by a polynomial of degree $O(d)$. ◀

6 Decidability of typechecking with ML polymorphism

MLSTA enjoys decidable typechecking, type inference and typability problems. To prove this we adapt the algorithm W also known as Hindley-Milner algorithm following [29].

The *type checking problem* (TCP) is the problem: given a term M, a type A, and a context Γ, is $\Gamma \vdash M : A$ derivable? The *type inference problem* (TIP) is the problem: given a term M and a context Γ, is there a type A such that $\Gamma \vdash M : A$ is derivable? Finally, the *typability problem* (TP) is the problem: given a term M, are there a context Γ and a type A such that $\Gamma \vdash M : A$ is derivable? We describe the way the problems can be solved with W-lin in the proof of Theorem 16.

The basic building block of the algorithm is the procedure of unification [28]. To make use of the procedure we divide the type variables $\mathcal{V}$ into two infinite disjoint parts $\mathcal{V}_v \cup \mathcal{V}_c = \mathcal{V}$. The set $\mathcal{V}_v$ contains *substitutable type variables*, called simply type variables below, and $\mathcal{V}_c$ contains variables that serve the role of constants in unification, called constants below. A substitution U that substitutes expressions on type variables (e.g. α, β etc.) is a unifier of $A \doteq A'$ when $U(A) = U(A')$. It is important to note that elements of $\mathcal{V}_v$ do not occur in the substitution applied to obtain an instance of a type in rules (Ax) and (AxC) in Fig. 1. This unification enjoys the most general unifier property, but we cannot use it directly here. Therefore we provide a special version of the most general unifier in Def. 13 below.

To express the procedure for typechecking, type inference, or typability we need a few technical definitions. We say that Γ is *full* with regard to a term M when $\mathrm{dom}(\Gamma) = \mathrm{FV}(M)$ this fact is denoted by $\mathrm{Full}(\Gamma; M)$. We say that Γ is *linear* with regard to a term M when for each $x : \sigma \in \Gamma$ the variable x occurs freely exactly once in M. This is denoted by $\mathrm{Linear}(\Gamma; M)$. The set of algebraic constants defined in Fig. 1 is denoted as $\mathrm{Const}_{\mathrm{MLSTA}}$. The algorithm we study here is presented in Fig. 5. The input for the algorithm is an environment Γ, a term M, and a type A. The output is a substitution U and a set of equations E. The situation that U, E are a valid output for Γ, M, A is denoted as $\Gamma \vdash_{\mathrm{Wl}} M : A \rightsquigarrow (U, E)$.

$$\dfrac{\beta_0, \dots, \beta_n \text{ are fresh} \qquad c : \forall \alpha_1 \cdots \alpha_n . A \in \mathrm{Const}_{\mathrm{MLSTA}}}{\vdash c : \emptyset, N \oslash A[\beta_1/\alpha_1, \dots, \beta_n/\alpha_n] \rightsquigarrow (\emptyset, \{N \doteq 0\})} \ (AxC)$$

$$\dfrac{}{x : N \oslash A \vdash x : N \oslash A \rightsquigarrow (\emptyset, \{N \doteq 0\})} \ (Ax) \qquad \dfrac{\Gamma \vdash M : N_2 \oslash A_2 \rightsquigarrow (U, E) \quad x \notin \mathrm{FV}(M)}{\Gamma, x : N_1 \oslash A_1 \vdash M : N_2 \oslash A_2 \rightsquigarrow (U, E)} \ (w)$$

$$\dfrac{\begin{array}{c} \Gamma, x : \alpha_1 \oslash \alpha_2 \vdash M : \alpha_3 \oslash \alpha_4 \rightsquigarrow (U_1, E_1) \qquad \alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4 \text{ are fresh} \\ \mathsf{mgu}(\{U_1(\alpha_0 \oslash ((\alpha_1 \oslash \alpha_2) \multimap \alpha_4)) \doteq U_1(N \oslash A)\}) = (U_2, E_2) \\ E_2' = E_1 \cup E_2 \cup \{N \doteq 0, \alpha_3 \doteq 0\} \qquad \mathrm{Full}(\Gamma; \lambda x.M) \quad \mathrm{Linear}(\Gamma; \lambda x.M) \end{array}}{\Gamma \vdash \lambda x.M : N \oslash A \rightsquigarrow (U_2 \circ U_1, E_2')} \ (\multimap I)$$

$$\dfrac{\begin{array}{c} \Gamma \vdash M : \alpha_1 \oslash ((\alpha_2 \oslash \alpha_3) \multimap N \oslash A) \rightsquigarrow (U_1, E_1) \qquad \alpha_1, \alpha_2, \alpha_3, \alpha_2', \alpha_3' \text{ are fresh} \\ \{x : \alpha_x' \oslash \alpha_x \mid x : N_x \oslash A_x \in \Delta, \alpha_x', \alpha_x \text{ are fresh}\} \vdash M' : \alpha_2' \oslash \alpha_3' \rightsquigarrow (U_2, E_2) \\ \mathsf{mgu}(\{U_1(A_x) \doteq U_2(\alpha_x) \mid x : A_x \in \Delta\} \cup \{U_1(\alpha_2 \oslash \alpha_3) \doteq U_2(\alpha_2' \oslash \alpha_3')\}) = (U_3, E_3) \\ E_4 = E_1 \cup E_2 \cup E_3 \cup \\ \{N \doteq 0, \alpha_1 \doteq 0\} \cup \{N_x \doteq \alpha_2' + \beta_x \mid x : N_x \oslash A_x \in \Delta, \beta_x \text{ are fresh}\} \\ \Gamma \# \Delta \qquad \mathrm{Full}(\Gamma, \Delta; MM') \qquad \mathrm{Linear}(\Gamma, \Delta; MM') \qquad \mathrm{Full}(\Gamma; M) \qquad \mathrm{Full}(\Delta; M') \end{array}}{\Gamma, \Delta \vdash MM' : N \oslash A \rightsquigarrow (U_3 \circ U_2 \circ U_1, E_4)} \ (\multimap E)$$

$$\dfrac{\begin{array}{c} \Gamma, x_1 : \alpha \oslash \alpha', \dots, x_n : \alpha \oslash \alpha' \vdash M : \alpha_2 \oslash \alpha_2' \rightsquigarrow (U_1, E_1) \\ \mathsf{mgu}(U_1(\alpha'' \oslash \alpha') \doteq U_1(N_1 \oslash A_1), U_1(\alpha_2 \oslash \alpha_2') \doteq U_1(N_2 \oslash A_2)) = (U_2, E_2) \\ E_3 = E_1 \cup E_2 \cup \{\alpha + 1 \doteq N_1\} \qquad \alpha, \alpha', \alpha_2, \alpha_2', \alpha'' \text{ are fresh} \\ \mathrm{Full}(\Gamma, x : N_1 \oslash A_1; M[x/x_1, \dots, x/x_n]) \quad \mathrm{Full}(\Gamma, x_1 : \alpha \oslash \alpha', \dots, x_n : \alpha \oslash \alpha'; M) \end{array}}{\Gamma, x : N_1 \oslash A_1 \vdash M[x/x_1, \dots, x/x_n] : N_2 \oslash A_2 \rightsquigarrow (U_2 \circ U_1, E_3)} \ (m)$$

$$\dfrac{\begin{array}{c} \Gamma \vdash M_1 : \alpha_1 \oslash \alpha_2 \rightsquigarrow (U_1, E_1) \qquad \alpha_1, \alpha_2, \beta_1 \dots, \beta_n \quad \text{are fresh} \\ U_1(\Delta), x_1 : \beta_1 \oslash B_1, \dots, x_n : \beta_n \oslash B_n \vdash M_2' : U_1(N' \oslash A) \rightsquigarrow (U_2, E_2) \\ M_2 = M_2'[x/x_1, \dots, x/x_n] \quad \mathsf{Fresh}(B_0, B_i) \text{ for } i = 1, \dots, n \quad B_0 = \mathrm{Clos}(U_1(\Gamma); U_1(\alpha_2)) \\ E_3 = E_1 \cup E_2 \cup \{\beta_i + \epsilon \doteq \alpha_1 + \beta_i' \mid i = 1, \dots, n, \beta_i' \text{ are fresh}\} \cup \\ \{N_x \doteq \alpha_1 + \beta_x \mid x : N_x \oslash A_x \in \Gamma, \beta_x \text{ are fresh}\} \qquad \epsilon = [n > 1] \\ \Gamma \# \Delta \quad \mathrm{Full}(\Gamma, \Delta; \mathtt{let}\ x = M_1\ \mathtt{in}\ M_2) \quad \mathrm{Linear}(\Gamma, \Delta; \mathtt{let}\ x = M_1\ \mathtt{in}\ M_2) \end{array}}{\Gamma, \Delta \vdash \mathtt{let}\ x = M_1\ \mathtt{in}\ M_2 : N \oslash A \rightsquigarrow (U_2 \circ U_1, E_3)} \ (let)$$

Figure 5 The algorithm W-lin, ($[n > 1]$ is 1 when $n > 1$ and 0 otherwise).

The intent is that in case this relation holds then for each solution U' of E the relation $U \circ U'(\Gamma) \vdash_{\mathrm{MLSTA}} M : U \circ U'(A)$ holds as well. This property does not hold directly, but it is spelled out in full technical detail by Lemma 15(4).

The actual algorithm works on types in a different syntax defined by this grammar:

$$\begin{array}{l} A ::= N \oslash C \qquad N ::= \alpha \mid N_1 + N_2 \mid n \\ C ::= \alpha \mid A_1 \multimap A_2 \mid A_1 \otimes A_2 \mid \mathbb{S}_i^{!j} \mid (\mathbb{S}_i^{!j})^k \boxtimes A \mid \mathbb{N}^{!j} \\ \mathfrak{s} ::= N \oslash \forall \vec{\alpha}.A \end{array}$$

where $n, i, j \in \mathbb{N}$. The set of types generated from the nonterminal A here is denoted as $\mathcal{T}_A^{\oslash}$, similarly generated from N is denoted as $\mathcal{T}_N^{\oslash}$, and from C — $\mathcal{T}_C^{\oslash}$, and the set of type schemes $\mathcal{T}_{\mathfrak{s}}^{\oslash}$. We use a general term $\oslash$-types to refer to elements of $\mathcal{T}_A^{\oslash}$. The elements generated from N are supposed to be expressions over natural numbers. We are free to perform any operations as soon as they are correct. For example the expression $3+4+\alpha$ is understood to be equal to $7+\alpha$. We divide the set of type variables $\mathcal{V}_v = \mathcal{V}_{vl} \cup \mathcal{V}_{vn}$ into two disjoint sets $\mathcal{V}_{vl}$ for type variables that are used to generate $\mathcal{T}_C^{\oslash}$ and $\mathcal{V}_{vn}$ for type variables that are use to generate $\mathcal{T}_N^{\oslash}$. We impose additional restriction on the substitutions below that variables in $\mathcal{V}_{vl}$ can be replaced by types from $\mathcal{T}_C^{\oslash}$ only and variables in $\mathcal{V}_{vn}$ by types from $\mathcal{T}_N^{\oslash}$ only. Types defined in (1) as $\mathcal{T}_\sigma$ can now be translated to $\mathcal{T}_A^{\oslash}$ and back using the following transformations:

▶ **Definition 12** (from types to $\oslash$-types and back)**.** We need a helper operation

We define now the transformation $(\cdot)^\bullet : \mathcal{T}_\sigma \to \mathcal{T}_A^{\oslash}$

- $(!^i\alpha)^\bullet = (i+\alpha') \oslash \alpha$, where $i \geq 0$, $\alpha' \in \mathcal{V}_{vn}$ is fresh,
- $(!^i(A \odot B))^\bullet = (i+\alpha) \oslash ((A)^\bullet \odot (B)^\bullet)$, where $i \geq 0$, $\alpha \in \mathcal{V}_{vn}$ is fresh and $\odot \in \{\multimap, \otimes, \boxtimes\}$,
- $(!^iH)^\bullet = (i+\alpha) \oslash H$ where $i \geq 0$, $\alpha \in \mathcal{V}_{vn}$ is fresh and $H \in \{\mathbb{S}_i^{!j} \mid, i, j \in \mathbb{N}\} \cup \{\mathbb{N}^{!j} \mid j \in \mathbb{N}\}$.

The transformation back $\llbracket\cdot\rrbracket : \mathcal{T}_A^{\oslash} \to \mathcal{T}_\sigma$ is defined as

- $\llbracket\alpha \oslash \alpha'\rrbracket = \alpha'$,
- $\llbracket(n) \oslash B\rrbracket =!^n\llbracket B\rrbracket$, where $!^0A = A$,
- $\llbracket\alpha \oslash (A \odot B)\rrbracket = \llbracket A\rrbracket \odot \llbracket B\rrbracket$, where $\odot \in \{\multimap, \otimes, \boxtimes\}$,
- $\llbracket\alpha \oslash H\rrbracket = H$ where $\alpha \in \mathcal{V}_{vn}$ and $H \in \{\mathbb{S}_i^{!j} \mid, i, j \in \mathbb{N}\} \cup \{\mathbb{N}^{!j} \mid j \in \mathbb{N}\}$.

Note that the translation back is correct only in case the translation in the context $\llbracket\alpha\oslash(A \multimap N\oslash C)\rrbracket$ is always applied so that $N = 0$. A substitution S is *proper wrt. a set of expressions* E when $S(N) = 0$ in the subexpressions of the form $\alpha \oslash (A \multimap N \oslash C)$ of expressions in E. The operations $(\cdot)^\bullet$ and $\llbracket\cdot\rrbracket$ extend to environments so that $(\Gamma)^* = \{x : (A)^* \mid x : A \in \Gamma\}$ where $(\cdot)^* \in \{(\cdot)^\bullet, \llbracket\cdot\rrbracket\}$.

The intuition behind the expressions presented here is that they make possible to more explicitly control the ! modalities. These must be, however, controlled in a non-standard way which cannot be handled with first-order unification techniques. The unification has the usual first-order ingredient, but to control the numbers of ! in ($\multimap$ E) and (*let*) rules we need a sort of second-order operation that can handle the presence of (*sp*) rules to obtain the type of the argument (see the definition of E_4 in the rule of ($\multimap$ E) in Fig. 5). The number cannot be handled locally since an occurrence of a variable in a different part of a derivation may require a higher number of (*sp*) that is immediately visible in the currently handled rule. Therefore, we split unification into two parts, i.e. one tractable by first-order techniques and one that operates on numerals and must be solved globally after the global information on the use of the (*sp*) rule is gathered. This separation requires a more subtle definition of the most general unifier operation. This is presented in the definition below.

▶ **Definition 13** (most general unification pair)**.** The operation $\mathsf{mgu}(\cdot) : \mathcal{E} \times \mathsf{Subst} \to \mathsf{Subst} \times \mathcal{E}_{\mathbb{N}} \cup \{\mathsf{fail}\}$, where $\mathcal{E}$ is the set of sets of pairs $A \doteq A'$ with $A, A' \in \mathcal{T}_A^{\oslash} \cup \mathcal{T}_B^{\oslash}$, Subst is the set of substitutions $\mathcal{V}_{vl} \rightharpoonup \mathcal{T}_C^{\oslash}$, and $\mathcal{E}_{\mathbb{N}}$ is the set of sets of pairs $B \doteq B'$ with $B, B' \in \mathcal{T}_B^{\oslash}$, is defined inductively as follows:

- $\mathsf{mgu}(\{A_1 \odot A_1' \doteq A_2 \boxdot A_2'\} \cup E, U_0) = \mathsf{fail}$ when $\odot \neq \boxdot$,
- $\mathsf{mgu}(\{N_1 \oslash A_1 \doteq N_2 \oslash A_2\} \cup E, U_0) = \mathsf{fail}$ when $\mathsf{mgu}(\{A_1 \doteq A_2\} \cup E, U_0) = \mathsf{fail}$,
- $\mathsf{mgu}(\{N_1 \oslash A_1 \doteq N_2 \oslash A_2\} \cup E, U_0) = (U, E' \cup \{N_1 \doteq N_2\})$ when $\mathsf{mgu}(\{A_1 \doteq A_2\} \cup E, U_0) = (U, E')$,
- $\mathsf{mgu}(\{A_1 \multimap N_1 \oslash A_1' \doteq A_2 \multimap N_2 \oslash A_2'\} \cup E, U_0) = \mathsf{fail}$ when $\mathsf{mgu}(\{A_1 \doteq A_2, A_1' \doteq A_2'\} \cup E, U_0) = \mathsf{fail}$,
- $\mathsf{mgu}(\{A_1 \multimap N_1 \oslash A_1' \doteq A_2 \multimap N_2 \oslash A_2'\} \cup E, U_0) = (U, E' \cup \{N_1 \doteq 0, N_2 \doteq 0\})$ when $\mathsf{mgu}(\{A_1 \doteq A_2, A_1' \doteq A_2'\} \cup E, U_0) = (U, E')$,
- $\mathsf{mgu}(\{N_1 \oslash A_1 \odot N_1' \oslash A_1' \doteq N_2 \oslash A_2 \odot N_2' \oslash A_2'\} \cup E, U_0) = \mathsf{fail}$ when $\mathsf{mgu}(\{A_1 \doteq A_2, A_1' \doteq A_2'\} \cup E, U_0) = \mathsf{fail}$ for $\odot \in \{\otimes, \boxtimes\}$,
- $\mathsf{mgu}(\{N_1 \oslash A_1 \odot N_1' \oslash A_1' \doteq N_2 \oslash A_2 \odot N_2' \oslash A_2'\} \cup E, U_0) = (U, E' \cup \{N_1 \doteq 0, N_2 \doteq 0, N_1' \doteq 0, N_2' \doteq 0\})$ when $\mathsf{mgu}(\{A_1 \doteq A_2, A_1' \doteq A_2'\} \cup E, U_0) = (U, E')$ for $\odot \in \{\otimes, \boxtimes\}$,
- $\mathsf{mgu}(\{C_1 \doteq C_2\} \cup E, U_0) = \mathsf{fail}$ when C_1, C_2 are different type constants
- $\mathsf{mgu}(\{C \doteq C\} \cup E, U_0) = \mathsf{mgu}(E, U_0)$ when C is a type constant,

- $\mathsf{mgu}(\{\alpha \doteq A\} \cup E, U_0) = \mathsf{fail}$ when $A \neq \alpha$ and α occurs in A,
- $\mathsf{mgu}(\{\alpha \doteq A\} \cup E, U_0) = \mathsf{mgu}(E[A/\alpha], [A/\alpha] \circ U_0)$ when $A = \alpha$ or α does not occur in A,
- $\mathsf{mgu}(\emptyset, U_0) = (U_0, \emptyset)$.

By default $\mathsf{mgu}(E) = (U, E)$ where $\mathsf{mgu}(E, \emptyset) = (U', E)$ and $U = R \circ U'$ where R is a renaming of all variables in $\mathrm{dom}(U)$ to fresh variables.

This most general unification pair enjoys the following natural property:

▶ Proposition 14 (correctness and completness of $\mathsf{mgu}(\cdot)$).

- If $\mathsf{mgu}(E) = (U, E')$ and E' is solvable with U' proper wrt. E then for each $A \doteq A' \in E$ where $A, A' \in \mathcal{T}^{\oslash}_A \cup \mathcal{T}^{\oslash}_B$ it holds that $U'(U(A)) = U'(U(A'))$.
- If there is U proper wrt. E such that $Z(U(A)) = Z(U(A'))$ for each $A \doteq A' \in E$ where $A, A' \in \mathcal{T}^{\oslash}_A \cup \mathcal{T}^{\oslash}_B$ and $Z(\alpha) = 0$ for each $\alpha \in \mathcal{V}_{vn}$ then $\mathsf{mgu}(E) = (U_1, E_1)$ and there is a substitution $U' : \mathcal{V}_{vl} \rightharpoonup \mathcal{T}^{\oslash}_A$ and a solution U'' of E_1 such that for each $\alpha \in \mathrm{dom}(U)$ the equality $Z(U(\alpha)) = Z(U'(U_1(U''(\alpha))))$ holds.

Proof. A standard proof is left to the reader. ◀

The main technical lemma that describes the operation of W-lin looks as follows.

▶ **Lemma 15** (key lemma).

1. *If* $\Gamma \vdash M : A \leadsto (U_1, E_1)$ *then* $\mathrm{FTV}(\Gamma, A) \cap \mathrm{FTV}(\{U_1(\alpha) | \alpha \in \mathrm{dom}(U_1)\}) = \emptyset$.
2. *For any term M and context Γ at most one rule in Fig. 5 can be used.*
3. *For any term M, context Γ, and type σ if a rule in Fig. 5 is applied then for each of the premises $\Gamma' \vdash M' : \sigma'$ and $x : \tau \in \Gamma'$ we have $\tau = B \oslash C$.*
4. *If W-lin started with $(\Gamma)^\bullet \vdash M : (A)^\bullet$ returns $(\Gamma)^\bullet \vdash M : (A)^\bullet \leadsto (U_1, E_1)$, and E_1 is unifiable with $U_1^{\#}$ then for $U = U_1^{\#} \circ U_1$ it holds that $[\![U(\Gamma)]\!] \vdash_{\mathrm{MLSTA}} M : [\![U(A)]\!]$. Moreover, the number of rules in the run of W-lin is the same as the number of rules different than (sp) in the resulting derivation in MLSTA.*
5. *Let Γ be a context and M a term. If there is a substitution U such that $U(\Gamma) \vdash_{\mathrm{MLSTA}} M : U(A)$ then the algorithm W-lin infers $(\Gamma)^\bullet \vdash M : (A)^\bullet \leadsto (U_1, E_1)$, the set E_1 is unifiable by $U_1^{\#}$, and there is a substitution U' such that for each variable $\beta \in \mathrm{dom}(U)$ $Z((U(\beta))^\bullet) = Z(U'(U_1^{\#}(U_1(\beta))))$. Moreover, the number of rules other than (sp) in a derivation in MLSTA is the same as the number of rules different than (sp) in the resulting run of W-lin.*
6. *For each Γ, M, A the algorithm W-lin terminates.*

▶ **Theorem 16** (decidability of TCP, TIP, and TP). *The TCP, TIP, and TP for the system MLSTA are decidable.*

Proof. We use here the algorithm W-lin. Note that is always terminating by Lemma 15(6).

In case of TCP we are given a context Γ, a term M, and a type σ. We may assume that σ does not start with ! by Corollary 3. Then we apply W-lin with the input $(\Gamma)^\bullet \vdash M : (\sigma)^\bullet$. In case this is derivable we obtain by Lemma 15(5) a pair (U, E) where E is unifiable with some $U^{\#}$. We now see that $Z(\emptyset(U^{\#}(U((\Gamma)^\bullet))) \vdash M : Z(\emptyset(U^{\#}(U((\sigma)^\bullet))))$ is derivable in MLSTA, but this is exactly the initial judgement as there are no variables in Γ, σ. In case this is not derivable by W-lin the initial judgement cannot be derivable in MLSTA by Lemma 15(4).

In case of TIP and TP we proceed in the same way, but we introduce substitutable variables for types the existence of which we have to discover, namely for the resulting type in case of TIP, and for the resulting type and the types in the context in case of TP. In case of TP we have to, in addition, guess which variables in the context should have type schemes. These variables must be packed by suitable `let` expression, essentially to manage the polymorphism in the way compatible with W-lin. ◀

7 Conclusions and Further Work

The system MLSTA we propose here can be viewed, similarly as ML in relation to System F, as a kind of interface over the system with full polymorphism, STA. The system offers a reasonable polymorphism with algebraic data structures such as naturals, booleans, and strings as well as recursion over the data types. All these features have their impredicative counterparts in STA. This view suggests a number of enhancements that can be done. One could develop the full theory of algebraic data types in our MLSTA, in particular polymorphic lists or polymorphic binary trees. Another possible improvement is to introduce more flexibility in the use of available constants. Currently, the programmer must provide the numerical parameters such as j_1, j_2 in `add` that express the level of natural numbers the addition operates in. One can extend our rule (AxC) to include the automatic calculation of the indexes. At last one can try to exploit other systems such as STA$_+$ or STA$_B$ [12] and give their ML-like versions.

References

1 Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Trans. Comput. Logic*, 3:137–175, January 2002.

2 Patrick Baillot, Marco Gaboardi, and Virgile Mogbil. A polytime functional language from light linear logic. In Andrew D. Gordon, editor, *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010*, volume 6012 of *LNCS*, pages 104–124. Springer-Verlag, 2010.

3 Patrick Baillot and Martin Hofmann. Type inference in intuitionistic linear logic. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*, PPDP '10, pages 219–230. ACM, 2010.

4 Patrick Baillot and Kazushige Terui. Light types for polynomial time computation in lambda calculus. *Information and Computation*, 207:41–62, January 2009.

5 Stephen Bellantoni and Stephen Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, December 1992.

6 Jacek Chrząszcz and Aleksy Schubert. The role of polymorphism in the characterisation of complexity by soft types. In Piotr Sankowski and Filip Murlak, editors, *Mathematical Foundations of Computer Science — 36th International Symposium, MFCS 2011*, volume 6907 of *LNCS*, pages 219–230. Springer-Verlag, 2011.

7 Alan Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the 1964 International Congress for Logic, Methodology, and the Philosophy of Science*, pages 24–30. North-Holland, 1964.

8 Paolo Coppola, Ugo Dal Lago, and Simona Ronchi Della Rocca. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science*, 4(4), 2008.

9 Ronald Fagin. *Contriburions to the model theory of'finire srructures.* PhD thesis, University of California at Berkeley, 1973.

10 Marco Gaboardi and Simona Ronchi della Rocca. Type inference for a polynomial lambda calculus. In Stefano Berardi, Ferruccio Damiani, and Ugo De'Liguoro, editors, *Types for Proofs and Programs, International Conference, TYPES 2008*, volume 5497 of *LNCS*, pages 136–152. Springer-Verlag, 2009.

11 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of PSPACE. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'08, pages 121–131. ACM, 2008.

12 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. Soft linear logic and polynomial complexity classes. *ENTCS*, 205:67–87, April 2008.

13 Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. An implicit characterization of pspace. *ACM Trans. Comput. Logic*, 13(2):18:1–18:36, April 2012.

14 Marco Gaboardi and Simona Ronchi Della Rocca. A soft type assignment system for λ-calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL*, volume 4646 of *LNCS*, pages 253–267. Springer-Verlag, 2007.

15 Gemalto. *Java Card$^{\text{TM}}$ & STK Applet Development Guidelines.* Gemalto, 2009.

16 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143:175–204, June 1998.

17 Robert Harper and John C. Mitchell. On the type structure of Standard ML. *ACM Trans. Program. Lang. Syst.*, 15(2):211–252, April 1993.

18 Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, August 1987.

19 Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, June 2004.

20 Ugo Dal Lago and Patrick Baillot. On light logics, uniform encodings and polynomial time. *Mathematical Structures in Comp. Sci.*, 16(4):713–733, August 2006.

21 Ugo Dal Lago and Ulrich Schöpp. Functional programming in sublinear space. In Andrew D. Gordon, editor, *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010*, volume 6012 of *LNCS*, pages 205–225. Springer-Verlag, 2010.

22 Ugo Dal Lago and Ulrich Schöpp. Type inference for sublinear space functional programming. In Kazunori Ueda, editor, *Programming Languages and Systems — 8th Asian Symposium, APLAS 2010*, volume 6461 of *LNCS*, pages 376–391. Springer-Verlag, 2010.

23 Daniel Leivant. Stratified functional programs and computational complexity. In Mary S. Van Deusen and Bernard Lang, editors, *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, pages 325–333. ACM, 1993.

24 Harry Mairson and Kazushige Terui. On the computational complexity of cut-elimination in linear logic. In Carlo Blundo and Cosimo Laneve, editors, *Theoretical Computer Science, 8th Italian Conference, ICTCS 2003*, volume 2841 of *LNCS*, pages 23–36. Springer-Verlag, 2003.

25 François Maurel. Nondeterministic light logics and NP-time. In Martin Hofmann, editor, *Typed Lambda Calculi and Applications, 6th International Conference, TLCA 2003*, volume 2701 of *LNCS*, pages 241–255. Springer-Verlag, 2003.

26 Wojciech Mostowski. Rigorous development of JavaCard applications. In T. Clark, A. Evans, and K. Lano, editors, *Proceedings of Fourth Workshop on Rigorous Object-Oriented Methods*, London, 2002.

27 Christos H. Papadimitriou. A note on the expressive power of Prolog. *Bulletin of the EATCS*, pages 21–22, 1985.

28 J. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

29 David A. Schmidt. *The Structure of Typed Programming Languages.* The MIT Press, 1994.

30 Ulrich Schöpp. Stratified bounded affine logic for logarithmic space. In *Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science, LICS 2007*, pages 411–420. IEEE Computer Society, 2007.

Definability of linear equation systems over groups and rings*

Anuj Dawar[1], Erich Grädel[2], Bjarki Holm[1], Eryk Kopczynski[3], and Wied Pakusa[2]

1 **University of Cambridge Computer Laboratory**
`{anuj.dawar,bjarki.holm}@cl.cam.ac.uk`

2 **Mathematical Foundations of Computer Science, RWTH Aachen University**
`{graedel,pakusa}@logic.rwth-aachen.de`

3 **Institute of Informatics, University of Warsaw**
`erykk@mimuw.edu.pl`

Abstract

Motivated by the quest for a logic for PTIME and recent insights that the descriptive complexity of problems from linear algebra is a crucial aspect of this problem, we study the solvability of linear equation systems over finite groups and rings from the viewpoint of logical (inter-)definability. All problems that we consider are decidable in polynomial time, but not expressible in fixed-point logic with counting. They also provide natural candidates for a separation of polynomial time from rank logics, which extend fixed-point logics by operators for determining the rank of definable matrices and which are sufficient for solvability problems over fields.

Based on the structure theory of finite rings, we establish logical reductions among various solvability problems. Our results indicate that *all* solvability problems for linear equation systems that separate fixed-point logic with counting from PTIME can be reduced to solvability over commutative rings. Further, we prove closure properties for classes of queries that reduce to solvability over rings. As an application, these closure properties provide normal forms for logics extended with solvability operators.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Math. Logic

Keywords and phrases finite model theory, logics with algebraic operators

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.213

1 Introduction

The quest for a logic for PTIME [10, 13] is one of the central open problems in both finite model theory and database theory. Specifically, it asks whether there is a logic in which a class of finite structures is expressible if, and only if, membership in the class is decidable in deterministic polynomial time.

Much of the research in this area has focused on the logic FPC, the extension of inflationary fixed-point logic by counting terms. In fact, FPC has been shown to capture polynomial time on many natural classes of structures, including planar graphs and structures of bounded tree-width [12, 13, 15]. Most recently, it was shown by Grohe [14] that FPC captures

* The first and third authors were supported by EPSRC grant EP/H026835/1 and the fourth and fifth authors were supported by ESF Research Networking Programme GAMES. The fourth author was also partially supported by the Polish Ministry of Science grant N N206 567840. An extended version of this paper is available at http://arxiv.org/abs/1204.3022.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 213–227
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

polynomial time on all classes of graphs with excluded minors, a result that generalises most of the previous partial capturing results. On the other side, already in 1992, Cai, Fürer and Immerman [6] constructed a query on a class of finite graphs that can be decided in polynomial time, but which is not definable by any sentence of FPC. But while this CFI query, as it is now called, is very elegant and has led to new insights in many different areas, it can hardly be called a natural problem in polynomial time. Therefore, it was often remarked that possibly all *natural* polynomial-time properties of finite structures could be expressed in FPC. However, this hope was eventually refuted in a strong sense by Atserias, Bulatov and Dawar [3] who proved that the very important problem of *solvability of linear equation systems* (over any fixed finite Abelian group) is not definable in FPC and that, indeed, the CFI query reduces to this problem. This motivates the systematic study of the relationship between finite model theory and linear algebra, and suggests that operators from linear algebra could be a source of new extensions to fixed-point logic, in an attempt to find a logical characterisation of PTIME. In [8], Dawar et al. pursued this direction of study by adding operators for expressing the rank of definable matrices over finite fields to first-order logic and fixed-point logic. They showed that fixed-point logic with rank operators (FPR) can define not only the solvability of linear equation systems over any finite field, but also the CFI query and essentially all other properties that were known to separate FPC from PTIME. However, although FPR is strictly more expressive than FPC and to date no examples are known to separate PTIME from FPR, it seems rather unlikely that FPR suffices to capture PTIME on the class of all finite structures.

A natural class of problems that might witness such a separation arises from linear equation systems over finite domains other than fields. Indeed, the results of Atserias, Bulatov and Dawar [3] imply that FPC fails to express the solvability of linear equation systems over any finite ring. On the other side, it is known that linear equation systems over finite rings can be solved in polynomial time [1], but it is unclear whether any notion of matrix rank is helpful for this purpose. We remark in this context that there are several non-equivalent notions of matrix rank over rings, but both the computability in polynomial time and the relationship to linear equation systems remains unclear. Thus, rather than matrix rank, the solvability of linear equation systems could be used directly as a source of operators (in the form of generalised quantifiers) for extending fixed-point logics.

Instead of introducing a host of new logics, with operators for various solvability problems, we set out here to investigate whether these problems are inter-definable. In other words, are they reducible to each other within FPC? Clearly, if they are, then any logic that generalises FPC and can define one, can also define the others. We thus study relations between solvability problems over (finite) rings, fields and Abelian groups in the context of logical many-to-one and Turing reductions, i.e., interpretations and generalised quantifiers. In this way, we show that solvability both over Abelian groups and over arbitrary (possibly non-commutative) rings reduces to solvability over commutative rings. We also show that solvability over commutative rings reduces to solvability over local rings, which are the basic building blocks of finite commutative rings. Finally, in the other direction, we show that solvability over rings endowed with a linear order and solvability over k-generated local rings, i.e. local rings for which the maximal ideal is generated by k elements, reduces to solvability over cyclic groups of prime-power order. These results indicate that *all* solvability problems for linear equation systems that separate FPC from PTIME can be reduced to solvability over commutative rings. Further, we prove closure properties for classes of queries that reduce to solvability over rings, and establish normal forms for first-order logic extended with operators for solvability over finite fields.

2 Background on logic and algebra

Throughout this paper, all structures (and in particular, all algebraic structures such as groups, rings and fields) are assumed to be finite. Furthermore, it is assumed that all groups are Abelian, unless otherwise noted.

2.1 Logic and structures

The logics we consider in this paper include *first-order logic* (FO) and *inflationary fixed-point logic* (FP) as well as their extensions by counting terms, which we denote by FOC and FPC, respectively. We also consider the extension of first-order logic with operators for deterministic transitive closure, which we denote by DTC. For details see [9, 10].

A *vocabulary* τ is a finite sequence of relation and constant symbols $(R_1, \dots, R_k, c_1, \dots, c_l)$ in which every R_i has an *arity* $r_i \geq 1$. A τ*-structure* $\mathbf{A} = (D(\mathbf{A}), R_1^{\mathbf{A}}, \dots, R_k^{\mathbf{A}}, c_1^{\mathbf{A}}, \dots, c_l^{\mathbf{A}})$ consists of a non-empty set $D(\mathbf{A})$, called the *domain* of $\mathbf{A}$, together with relations $R_i^{\mathbf{A}} \subseteq D(\mathbf{A})^{r_i}$ and constants $c_j^{\mathbf{A}} \in D(\mathbf{A})$ for each $i \leq k$ and $j \leq l$. Given a logic L and a vocabulary τ, we write $\mathrm{L}[\tau]$ to denote the set of τ-formulas of L. A τ-formula $\phi(\vec{x})$ with $|\vec{x}| = k$ defines a k*-ary query* that takes any τ-structure $\mathbf{A}$ to the set $\phi(\vec{x})^{\mathbf{A}} := \{\vec{a} \in D(\mathbf{A})^k \mid \mathbf{A} \models \phi[\vec{a}]\}$.

Lindström quantifiers and extensions. Let $\sigma = (R_1, \dots, R_k)$ be a vocabulary and consider a class $\mathcal{K}$ of σ-structures that is closed under isomorphism. With $\mathcal{K}$ we associate a *Lindström quantifier* $Q_{\mathcal{K}}$ whose *type* is the tuple $(r_1, \dots, r_k)$. For a logic L, we define the extension $\mathrm{L}(Q_{\mathcal{K}})$ by adding rules for constructing formulas of the kind $Q_{\mathcal{K}}\vec{x}_1 \dots \vec{x}_k \,.\, (\phi_1, \dots, \phi_k)$, where $\phi_1, \dots, \phi_k$ are formulas and each $\vec{x}_i$ has length r_i. The semantics of the quantifier $Q_{\mathcal{K}}$ is defined such that $\mathbf{A} \models Q_{\mathcal{K}}\vec{x}_1 \dots \vec{x}_k \,.\, (\phi_1, \dots, \phi_k)$ if $(D(\mathbf{A}), \phi_1(\vec{x}_1)^{\mathbf{A}}, \dots, \phi_k(\vec{x}_k)^{\mathbf{A}}) \in \mathcal{K}$ as a σ-structure (see [18, 20]). Similarly we can consider the extension of L by a collection $\mathbf{Q}$ of Lindström quantifiers. The logic $\mathrm{L}(\mathbf{Q})$ is defined by adding a rule for constructing formulas with Q, for each $Q \in \mathbf{Q}$, and the semantics is defined by considering the semantics for each quantifier $Q \in \mathbf{Q}$, as above. For $m \geq 1$, we write $\mathcal{K}_m$ to denote the m-ary vectorisation of $\mathcal{K}$. If Q_m is the Lindström quantifier associated with $\mathcal{K}_m$ then we write $\langle Q_{\mathcal{K}} \rangle := \{Q_m \mid m \in \mathbb{N}\}$ to denote the *vectorised sequence* of Lindström quantifiers associated with $\mathcal{K}$ (see [7]).

Interpretations and logical reductions. Consider signatures σ and τ and a logic L. An m*-ary* L*-interpretation of* τ *in* σ is a sequence of formulas of L in vocabulary σ consisting of: (i) a formula $\delta(\vec{x})$; (ii) a formula $\varepsilon(\vec{x}, \vec{y})$; (iii) for each relation symbol $R \in \tau$ of arity k, a formula $\phi_R(\vec{x}_1, \dots, \vec{x}_k)$; and (iv) for each constant symbol $c \in \tau$, a formula $\gamma_c(\vec{x})$, where each $\vec{x}$, $\vec{y}$ or $\vec{x}_i$ is an m-tuple of free variables. We call m the *width* of the interpretation. We say that an interpretation $\mathcal{I}$ associates a τ-structure $\mathcal{I}(\mathbf{A}) = \mathbf{B}$ to a σ-structure $\mathbf{A}$ if there is a surjective map h from the m-tuples $\delta(\vec{x}) = \{\vec{a} \in D(\mathbf{A})^m \mid \mathbf{A} \models \delta[\vec{a}]\}$ to $\mathbf{B}$ such that:

- $h(\vec{a}_1) = h(\vec{a}_2)$ if, and only if, $\mathbf{A} \models \varepsilon[\vec{a}_1, \vec{a}_2]$;
- $R^{\mathbf{B}}(h(\vec{a}_1), \dots, h(\vec{a}_k))$ if, and only if, $\mathbf{A} \models \phi_R[\vec{a}_1, \dots, \vec{a}_k]$; and
- $h(\vec{a}) = c^{\mathbf{B}}$ if, and only if, $\mathbf{A} \models \gamma_c[\vec{a}]$.

▶ **Definition 1** (Logical reductions). Let $\mathcal{C}$ be a class of σ-structures and $\mathcal{D}$ a class of τ-structures closed under isomorphism.

- $\mathcal{C}$ is said to be L-*many-to-one reducible* to $\mathcal{D}$ ($\mathcal{C} \leq_{\mathrm{L}} \mathcal{D}$) if there is an L-interpretation $\mathcal{I}$ of τ in σ such that for every σ-structure $\mathbf{A}$ it holds that $\mathbf{A} \in \mathcal{C}$ if, and only if, $\mathcal{I}(\mathbf{A}) \in \mathcal{D}$.
- $\mathcal{C}$ is said to be L-*Turing reducible* to $\mathcal{D}$ ($\mathcal{C} \leq_{\mathrm{L\text{-}T}} \mathcal{D}$) if $\mathcal{C}$ is definable in $\mathrm{L}(\langle Q_{\mathcal{D}} \rangle)$. ■

2.2 Rings and systems of linear equations

We recall some definitions from commutative and linear algebra, assuming that the reader has knowledge of basic algebra and group theory (for further details see Atiyah et al. [2]). For $m \geq 2$, we write $\mathbb{Z}_m$ to denote the ring of integers modulo m.

Commutative rings. Let $(R, \cdot, +, 1, 0)$ be a commutative ring. An element $x \in R$ is a *unit* if $xy = yx = 1$ for some $y \in R$ and we denote by $R^\times$ the set of all units. Moreover, we say that y *divides* x (written $y \mid x$) if $x = yz$ for some $z \in R$. An element $x \in R$ is *nilpotent* if $x^n = 0$ for some $n \in \mathbb{N}$, and we call the least such $n \in \mathbb{N}$ the *nilpotency* of x. The element $x \in R$ is *idempotent* if $x^2 = x$. Clearly $0, 1 \in R$ are idempotent elements, and we say that an idempotent x is *non-trivial* if $x \notin \{0, 1\}$. Two elements $x, y \in R$ are *orthogonal* if $xy = 0$.

We say that R is a *principal ideal ring* if every ideal of R is generated by a single element. An ideal $m \subseteq R$ is called *maximal* if $m \neq R$ and there is no ideal $m' \subsetneq R$ with $m \subsetneq m'$. A commutative ring R is *local* if it contains a unique maximal ideal m. We often consider *chain rings* that are both local and principal. For example, all prime rings $\mathbb{Z}_{p^n}$ are chain rings and so too are all finite fields. More generally, a *k-generated local ring* is a local ring for which the maximal ideal is generated by k elements. See McDonald [19] for further background.

▶ Remark. When we speak of a "commutative ring with a linear order", then in general the ordering does not respect the ring operations (cf. the notion of ordered rings from algebra).

Systems of linear equations. We consider systems of linear equations over groups and rings whose equations and variables are indexed by arbitrary sets, not necessarily ordered. In the following, if I, J and X are finite and non-empty sets then an $I \times J$ *matrix* over X is a function $A : I \times J \to X$. An *I-vector* over X is defined similarly as a function $\mathbf{b} : I \to X$.

A system of linear equations over a group G is a pair $(A, \mathbf{b})$ with $A \colon I \times J \to \{0, 1\}$ and $\mathbf{b} \colon I \to G$. By viewing G as a $\mathbb{Z}$-module (i.e. by defining the natural multiplication between integers and group elements respecting $1 \cdot g = g$, $(n+1) \cdot g = n \cdot g + g$, and $(n-1) \cdot g = n \cdot g - g$), we write $(A, \mathbf{b})$ as a matrix equation $A \cdot \mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ is a J-vector of variables that range over G. The system $(A, \mathbf{b})$ is said to be *solvable* if there exists a solution vector $\mathbf{c} \colon J \to G$ such that $A \cdot \mathbf{c} = \mathbf{b}$, where we define multiplication of unordered matrices and vectors in the usual way by $(A \cdot \mathbf{c})(i) = \sum_{j \in J} A(i, j) \cdot \mathbf{c}(j)$ for all $i \in I$. We represent linear equation systems over groups as finite structures over the vocabulary $\tau_{\text{les-g}} := \{G, A, b\} \cup \tau_{\text{group}}$, where $\tau_{\text{group}} := \{+, e\}$ denotes the language of groups, G is a unary relation symbol (identifying the elements of the group) and A, b are two binary relation symbols.

Similarly, a system of linear equations over a commutative ring R is a pair $(A, \mathbf{b})$ where A is an $I \times J$ matrix with entries in R and $\mathbf{b}$ is an I-vector over R. As before, we usually write $(A, \mathbf{b})$ as a matrix equation $A \cdot \mathbf{x} = \mathbf{b}$ and say that $(A, \mathbf{b})$ is solvable if there is a solution vector $\mathbf{c} : J \to R$ such that $A \cdot \mathbf{c} = \mathbf{b}$. In the case that the ring R is not commutative, we represent linear systems in the form $A_l \cdot \mathbf{x} + \mathbf{x} \cdot A_r = \mathbf{b}$.

We consider three different ways to represent linear systems over rings as relational structures. For simplicity, we restrict to commutative rings here. Firstly, we consider the case where the ring is part of the structure. Let $\tau_{\text{les-r}} := \{R, A, b\} \cup \tau_{\text{ring}}$, where $\tau_{\text{ring}} = \{+, \cdot, 1, 0\}$ is the language of rings, R is a unary relation symbol (identifying the ring elements), and A and b are ternary and binary relation symbols, respectively. Then a finite $\tau_{\text{les-r}}$-structure $\mathbf{S}$ describes the linear equation system $(A^{\mathbf{S}}, \mathbf{b}^{\mathbf{S}})$ over the ring $\mathbf{R}^{\mathbf{S}} = (R^{\mathbf{S}}, +^{\mathbf{S}}, \cdot^{\mathbf{S}})$. Secondly, we consider a similar encoding but with the additional assumption that the elements of the ring (and not the equations or variables of the equation systems) are linearly ordered. Such systems can be seen as finite structures over the vocabulary $\tau^{\leqslant}_{\text{les-r}} := \tau_{\text{les-r}} \cup \{\leqslant\}$. Finally, we consider linear equation systems over a fixed ring encoded in the vocabulary: for every ring R, we define the vocabulary $\tau_{\text{les}}(R) := \{A_r, b_r \mid r \in R\}$, where for each $r \in R$ the symbols A_r and b_r are binary and unary, respectively. A finite $\tau_{\text{les}}(R)$-structure $\mathbf{S}$ describes the linear equation system $(A, \mathbf{b})$ over R where $A(i,j) = r$ if, and only if, $(i,j) \in A_r^{\mathbf{S}}$ and similarly for $\mathbf{b}$ (assuming that the $A_r^{\mathbf{S}}$ form a partition of $I \times J$ and that the $b_r^{\mathbf{S}}$ form a partition of I).

Finally, we frequently say that two linear equation systems $\mathbf{S}$ and $\mathbf{S}'$ over a common domain X are *equivalent* if either both systems are solvable over X or neither system is solvable over X.

3 Solvability problems over different algebraic domains

It follows from the work of Atserias, Bulatov and Dawar [3] that FPC cannot express solvability of linear equation systems ('solvability problems') over any class of (finite) groups or rings. In this section we study solvability problems over such different algebraic domains in terms of logical reductions. Our main result here is to show that the solvability problem over groups (SlvAG) DTC-reduces to the corresponding problem over commutative rings (SlvCR) and that solvability over commutative rings equipped with a linear ordering ($\textsc{SlvCR}_{\leqslant}$) FP-reduces to solvability over cyclic groups (SlvCycG). Note that over any non-Abelian group, the solvability problem is NP-complete [11].

Our methods can be further adapted to show that solvability over arbitrary (not necessarily commutative) rings (SlvR) DTC-reduces to SlvCR. We then consider solvability restricted to special classes of commutative rings: local rings (SlvLR) and k-generated local rings ($\textsc{SlvLR}_k$), which generalises solvability over finite fields (SlvF). The reductions that we establish are illustrated in Figure 1.

$$\begin{tikzcd}
\textsc{SlvCycG} \arrow[r, hook] & \textsc{SlvAG} \arrow[d, "\text{DTC}"'] & \\
\textsc{SlvCR}_{\leqslant} \arrow[u, "\text{FP}"] \arrow[r, hook] & \textsc{SlvCR} \arrow[dr, "\text{FO-T}"', shift right] & \textsc{SlvR} \arrow[l, "\text{DTC}"'] \\
\textsc{SlvLR}_k \arrow[u, "\text{FP-T}"] & \textsc{SlvF} \arrow[l, hook'] & \textsc{SlvLR} \arrow[ul, hook, shift right]
\end{tikzcd}$$

Figure 1 Logical reductions between solvability problems. Curved arrows ($\hookrightarrow$) denote inclusion of one class in another.

In the remainder of this section we describe three of the outlined reductions: from commutative rings equipped with a linear order to cyclic groups, from groups to commutative rings, and finally from general rings to commutative rings. To give the reductions from commutative rings to local rings and from k-generated local rings to commutative linearly ordered rings we need to delve further into the theory of finite commutative rings, which is the subject of §4.

▶ **Theorem 2.** $\textsc{SlvCR}_{\leqslant} \leq_{\text{FP}} \textsc{SlvCycG}$.

Proof. Consider a system of linear equations $(A, \mathbf{b})$ over a commutative ring R of characteristic m and let $\leqslant$ be a linear order on R. In the following we describe a mapping that translates the system $(A, \mathbf{b})$ into a system of equations $(A^\star, \mathbf{b}^\star)$ over the cyclic group $\mathbb{Z}_m$ which is solvable if, and only if, $(A, \mathbf{b})$ has a solution over R.

Let $\{g_1, \dots, g_k\} \subseteq R$ be a (minimal) generating set for the additive group $(R, +)$ and let ℓ_i denote the order of g_i. We consider the group generated by g_i as a subgroup of $\mathbb{Z}_m$, i.e. $\langle g_i \rangle = \mathbb{Z}_{\ell_i} \cong (m/\ell_i)\,\mathbb{Z}_m \leq \mathbb{Z}_m$. Then $(R, +) \cong \bigoplus_i (m/\ell_i)\,\mathbb{Z}_m$ and we obtain a unique representation for each element $r \in R$ as $r = (r_1, \dots, r_k)$ where $r_i \in (m/\ell_i)\,\mathbb{Z}_m$. Similarly, we identify variables x ranging over R with tuples $x = (x_1, \dots, x_k)$ where x_i ranges over $(m/\ell_i)\,\mathbb{Z}_m$. Note that, in general, subgroups $(m/\ell)\,\mathbb{Z}_m$ are definable in linear systems over $\mathbb{Z}_m$: the equation $\ell \cdot x = 0$ ensures that the variable x takes values in $(m/\ell)\,\mathbb{Z}_m$.

To translate linear equations over R into equivalent equations over $\mathbb{Z}_m$, we consider the multiplication of a coefficient $r \in R$ with a variable x with respect to the chosen representation, i.e. the formal expression $r \cdot x = (r_1, \dots, r_k) \cdot (x_1, \dots, x_k)$. If we write all products $g_i \cdot g_j$ of pairs of generators as elements in $\bigoplus_i (m/\ell_i)\,\mathbb{Z}_m$, then the product $r \cdot x$ is uniquely determined as a k-tuple of the form $(\sum_i b^1_{i,r} \cdot x_i, \dots, \sum_i b^k_{i,r} \cdot x_i)$, where for every $\ell \leq k$ the coefficients $b^\ell_{1,r}, \dots, b^\ell_{k,r}$ only depend on $r = (r_1, \dots, r_k)$ and ℓ, and where x_i ranges over $(m/\ell_i)\,\mathbb{Z}_m$. Furthermore, the decomposition $\bigoplus_i (m/\ell_i)\,\mathbb{Z}_m$ allows us to handle addition component-wise. Hence, altogether we can translate each linear equation of the original system $(A, \mathbf{b})$ into k equations over $\mathbb{Z}_m$ and obtain a system of linear equations $(A^\star, \mathbf{b}^\star)$ over $\mathbb{Z}_m$ which is solvable if, and only if, the original system $(A, \mathbf{b})$ has a solution over R.

We proceed to show that the mapping $(A, \mathbf{b}) \mapsto (A^\star, \mathbf{b}^\star)$ can be expressed in FP. Here, we crucially rely on the given order on R to fix a set of generators. More specifically, as we can compute a set of generators in time polynomial in $|\,R\,|$, it follows from the Immerman-Vardi theorem [17, 22] that there is an FP-formula $\phi(x)$ such that $\phi(x)^R = \{g_1, \dots, g_k\}$ generates $(R, +)$ and $g_1 \leqslant \dots \leqslant g_k$. Having fixed a set of generators, it is obvious that the map $\iota : R \to (m/\ell_1)\,\mathbb{Z}_m \times \dots \times (m/\ell_k)\,\mathbb{Z}_m$ taking $r \mapsto (r_1, \dots, r_k)$, is FP-definable. Furthermore, the map $(l, i, r) \mapsto b^l_{i,r}$ can easily be formalised in FP, since we have $b^l_{i,r} = \sum_{j=1}^k r_j \cdot c^{ij}_l$ where c^{ij}_l is the coefficient of g_l in the expression $g_i \cdot g_j = \sum_{y=1}^k c^{ij}_y \cdot g_y$. Splitting the original system of equations component-wise into k systems of linear equations and combining them again to a single system over $\mathbb{Z}_m$ is trivial.

Finally, we note that a linear system over the *ring* $\mathbb{Z}_m$ can be reduced to an equivalent system over the *group* $\mathbb{Z}_m$, by rewriting terms ax with $a \in \mathbb{Z}_m$ as $x + x + \dots + x$ (a-times). ◀

So far, we have shown that solvability problems over linearly ordered commutative rings can be reduced to solvability problems over basic groups. This raises the question whether a translation in the other direction is also possible; that is, whether we can reduce solvability over groups to solvability over commutative rings. Essentially, such a reduction requires an interpretation of a commutative ring in a group, which is what we describe in the proof of the following theorem.

▶ **Theorem 3.** SLVAG $\leq_{\text{DTC}}$ SLVCR.

Proof. Let $(A, \mathbf{b})$ be a system of linear equations over a group $(G, +_G, e)$, where $A \in \{0,1\}^{I \times J}$ and $\mathbf{b} \in G^I$. For the reduction, we first construct a commutative ring $\phi(G)$ from G and then lift $(A, \mathbf{b})$ to a system of equations $(A^\star, \mathbf{b}^\star)$ which is solvable over $\phi(G)$ if, and only if, $(A, \mathbf{b})$ is solvable over G.

We consider G as a $\mathbb{Z}$-module in the usual way and write $\cdot_\mathbb{Z}$ for multiplication of group elements by integers. Let d be the least common multiple of the order of all group elements. Then we have $\text{ord}_G(g) \mid d$ for all $g \in G$, where $\text{ord}_G(g)$ denotes the order of g. This allows us to obtain from $\cdot_\mathbb{Z}$ a well-defined multiplication of G by elements of $\mathbb{Z}_d = \{[0]_d, \dots, [d-1]_d\}$ which commutes with group addition. We write $+_d$ and $\cdot_d$ for addition and multiplication in $\mathbb{Z}_d$, where $[0]_d$ and $[1]_d$ denote the additive and multiplicative identities, respectively. We now consider the set $G \times \mathbb{Z}_d$ as a group, with component-wise addition defined by $(g_1, m_1) + (g_2, m_2) := (g_1 +_G g_2, m_1 +_d m_2)$, for all $(g_1, m_1), (g_2, m_2) \in G \times \mathbb{Z}_d$, and identity element $0 = (e, [0]_d)$. We endow $G \times \mathbb{Z}_d$ with a multiplication $\bullet$ which is defined as $(g_1, m_1) \bullet (g_2, m_2) := \big((g_1 \cdot_\mathbb{Z} m_2 +_G g_2 \cdot_\mathbb{Z} m_1), (m_1 \cdot_d m_2)\big)$.

It is easily verified that this multiplication is associative, commutative and distributive over $+$. It follows that $\phi(G) := (G \times \mathbb{Z}_d, +, \bullet, 1, 0)$ is a commutative ring, with identity $1 = (e, [1]_d)$. For $g \in G$ and $z \in \mathbb{Z}$ we set $\overline{g} := (g, [0]_d) \in \phi(G)$ and $\overline{z} := (e, [z]_d) \in \phi(G)$. Let

$\iota : \mathbb{Z} \cup G \to \phi(G)$ be the map defined by $x \mapsto \overline{x}$. Extending ι to relations in the obvious way, we write $A^\star := \iota(A) \in \iota(\mathbb{Z}_d)^{I \times J}$ and $\mathbf{b}^\star := \iota(\mathbf{b}) \in \iota(G)^I$.

Claim. The system $(A^\star, \mathbf{b}^\star)$ is solvable over $\phi(G)$ if, and only if, $(A, \mathbf{b})$ is solvable over G.

Proof of claim. In one direction, observe that a solution $\mathbf{s}$ to $(A, \mathbf{b})$ gives the solution $\iota(\mathbf{s})$ to $(A^\star, \mathbf{b}^\star)$. For the other direction, suppose that $\mathbf{s} \in \phi(G)^J$ is a vector such that $A^\star \cdot \mathbf{s} = \mathbf{b}^\star$. Since each element $(g, [m]_d) \in \phi(G)$ can be written uniquely as $(g, [m]_d) = \overline{g} + \overline{m}$, we write $\mathbf{s} = \mathbf{s}_g + \mathbf{s}_n$, where $\mathbf{s}_g \in \iota(G)^J$ and $\mathbf{s}_n \in \iota(\mathbb{Z}_d)^J$. Observe that we have $\overline{g} \bullet \overline{m} \in \iota(G) \subseteq \phi(G)$ and $\overline{n} \bullet \overline{m} \in \iota(\mathbb{Z}_d) \subseteq \phi(G)$ for all $g \in G$ and $n, m \in \mathbb{Z}$. Hence, it follows that $A^\star \cdot \mathbf{s}_n \in \iota(\mathbb{Z}_d)^I$ and $A^\star \cdot \mathbf{s}_g \in \iota(G)^I$. Now, since $\mathbf{b}^\star \in \iota(G)^I$, we have $\mathbf{b}^\star = A^\star \cdot \mathbf{s} = A^\star \cdot \mathbf{s}_g + A^\star \cdot \mathbf{s}_n = A^\star \cdot \mathbf{s}_g$. Hence, $\mathbf{s}_g$ gives a solution to $(A, \mathbf{b})$, as required.

All that remains is to show that our reduction can be formalised as a DTC-interpretation. Essentially, this comes down to showing that the ring $\phi(G)$ can be interpreted in G by formulas of DTC. By elementary group theory, we know that for elements $g \in G$ of maximal order we have $\mathrm{ord}\,(g) = d$. It is not hard to see that the set of group elements of maximal order can be defined in DTC; hence, we can interpret $\mathbb{Z}_d$ in G. Also, it is not hard to show that the the multiplication of $\phi(G)$ is DTC-definable, which completes the proof. ◀

We conclude this section by describing a DTC-reduction from the solvability problem over general rings R to solvability over commutative rings. As a technical preparation, we first give a first-order interpretation that transforms a linear equation systems over R into an equivalent system with the following property: the linear equation system is solvable if, and only if, the solution space contains a *numerical solution*, i.e. a solution over $\mathbb{Z}$.

For convenience, we only consider left-multiplicative linear systems, which are systems of the form $A \cdot \mathbf{x} = \mathbf{b}$; however, the more general case of linear equation systems of the form $A_l \cdot \mathbf{x} + \mathbf{x} \cdot A_r = \mathbf{b}$ can be treated similarly.

▶ **Lemma 4.** *There is an* FO*-interpretation $\mathcal{I}$ of $\tau_{les\text{-}r}$ in $\tau_{les\text{-}r}$ such that for every linear system $\mathbf{S} : A \cdot \mathbf{x} = \mathbf{b}$ over R, $\mathcal{I}(\mathbf{S})$ describes a linear system $\mathbf{S}^\star : A^\star \cdot_{\mathbb{Z}} \mathbf{x}^\star = \mathbf{b}^\star$ over the $\mathbb{Z}$-module $(R, +)$ such that $\mathbf{S}$ is solvable over R if, and only if, $\mathbf{S}^\star$ has a solution over $\mathbb{Z}$.*

Proof (sketch). Let $A \in R^{I \times J}$ and $\mathbf{b} \in R^I$. For $\mathbf{S}^\star$, we introduce for each variable x_j $(j \in J)$ and each element $s \in R$ a new variable x_j^s, i.e. the index set for the variables of $\mathbf{S}^\star$ is $J \times R$. Finally, we replace all terms of the form rx_j by $\sum_{s \in R} rsx_j^s$. ◀

By Lemma 4, we can restrict to linear systems $(A, \mathbf{b})$ over the $\mathbb{Z}$-module $(R, +)$ that have numerical solutions. At this point, we reuse our construction from Theorem 3 to obtain a linear system $(A^\star, \mathbf{b}^\star)$ over the commutative ring $R^\star := \phi((R, +))$, where $A^\star := \iota(A)$ and $\mathbf{b}^\star := \iota(\mathbf{b})$. We claim that $(A^\star, \mathbf{b}^\star)$ is solvable over $R^\star$ if, and only if, $(A, \mathbf{b})$ is solvable over R. For the non-trivial direction, suppose $\mathbf{s}$ is a solution to $(A^\star, \mathbf{b}^\star)$ and decompose $\mathbf{s} = \mathbf{s}_g + \mathbf{s}_n$ into group elements and number elements, as explained in the proof of Theorem 3. Recalling that $\overline{r}_1 \bullet \overline{r}_2 = 0$ for all $r_1, r_2 \in R$, it follows that $A^\star \bullet (\mathbf{s}_g + \mathbf{s}_n) = A^\star \bullet \mathbf{s}_n = \mathbf{b}^\star$. Hence, there is a solution $\mathbf{s}_n$ to $(A^\star, \mathbf{b}^\star)$ that consists *only* of number elements, as claimed.

▶ **Theorem 5.** SlvR $\leq_{\mathrm{DTC}}$ SlvCR.

4 The structure of finite commutative rings

In this section we study structural properties of (finite) commutative rings and present the remaining reductions for solvability outlined in §3: from commutative rings to local rings, and from k-generated local rings to commutative rings with a linear order. Recall that a commutative ring R is local if it contains a unique maximal ideal m. The importance of the notion of local rings comes from the fact that they are the basic building blocks of finite commutative rings. We start by summarising some of their useful properties.

▶ **Proposition 6** (Properties of local rings). *For any finite commutative ring R we have:*

- *If R is local, then the unique maximal ideal is $m = R \setminus R^{\times}$.*
- *R is local if, and only if, all idempotent elements in R are trivial.*
- *If $x \in R$ is idempotent then $R = x \cdot R \oplus (1 - x) \cdot R$ as a direct sum of rings.*
- *If R is local then its cardinality (and hence its characteristic) is a prime power.*

By this proposition we know that finite commutative rings can be decomposed into local summands that are primary ideals generated by pairwise orthogonal idempotent elements. Indeed, this decomposition is unique (for details, see e.g. [5]).

▶ **Proposition 7** (Decomposition into local rings). *Let R be a (finite) commutative ring. Then there is a unique set $\mathcal{B}(R) \subseteq R$ of pairwise orthogonal idempotents elements for which it holds that (i) $e \cdot R$ is local for each $e \in \mathcal{B}(R)$; (ii) $\sum_{e \in \mathcal{B}(R)} e = 1$; and (iii) $R = \bigoplus_{e \in \mathcal{B}(R)} e \cdot R$.*

We next show that the ring decomposition $R = \bigoplus_{e \in \mathcal{B}(R)} e \cdot R$ is FO-definable. As a first step, we note that $\mathcal{B}(R)$ (the *base* of R) is FO-definable over R.

▶ **Lemma 8.** *There is $\phi(x) \in \mathrm{FO}(\tau_{ring})$ such that $\phi(x)^R = \mathcal{B}(R)$ for commutative rings R.*

Proof (sketch). It can be shown that $\mathcal{B}(R)$ consists precisely of those non-trivial idempotent elements of R which cannot be expressed as the sum of two orthogonal non-trivial idempotents, which is a first-order definable property. In particular, if R is local then trivially $\mathcal{B}(R) = \{1\}$. To test for locality, it suffices by Proposition 6 to check whether all idempotent elements in R are trivial and this can be expressed easily in first-order logic. ◀

The next step is to show that the canonical mapping $R \to \bigoplus_{e \in \mathcal{B}(R)} e \cdot R$ can be defined in FO. To this end, recall from Proposition 6 that for every $e \in \mathcal{B}(R)$ (indeed, for any idempotent element $e \in R$), we can decompose the ring R as $R = e \cdot R \oplus (1 - e) \cdot R$. This fact allows us to define for all base elements $e \in \mathcal{B}(R)$ the projection of elements $r \in R$ onto the summand $e \cdot R$ in first-order logic, without having to keep track of all local summands simultaneously.

▶ **Lemma 9.** *There is a formula $\psi(x, y, z) \in \mathrm{FO}(\tau_{ring})$ such that for all rings R, $e \in \mathcal{B}(R)$ and $r, s \in R$, it holds that $(R, e, r, s) \models \psi$ if, and only if, s is the projection of r onto $e \cdot R$.*

It follows that any relation over R can be decomposed in first-order logic according to the decomposition of R into local summands. In particular, a linear equation system $(A \mid \mathbf{b})$ over R is solvable if, and only if, each of the projected linear equation systems $(A^e \mid \mathbf{b}^e)$ is solvable over eR. Hence, we obtain:

▶ **Theorem 10.** $\textsc{SlvCR} \le_{FO\text{-}\mathrm{T}} \textsc{SlvLR}$.

In §3 we proved that solvability over rings with a linear ordering can be reduced in fixed-point logic to solvability over cyclic groups. This naturally raises the question: which classes of rings can be linearly ordered in fixed-point logic? By Lemma 9, we know that for this question it suffices to focus on local rings, which have a well-studied structure. The simplest type of local ring are rings of the form $\mathbb{Z}_{p^n}$ and the natural ordering of such rings can be easily defined by a formula of FP. Moreover, the same holds for finite fields as they have a cyclic multiplicative group [16]. In the following lemma, we are able to generalise these insights in a strong sense: for any fixed $k \geq 1$ we can define an ordering on the class of all local rings for which the maximal ideal is generated by at most k elements. We refer to such rings as *k-generated local rings*. Note that for $k = 1$ we obtain the notion of chain rings which include all finite fields and rings of the form $\mathbb{Z}_{p^n}$. For increasing values of k the structure of k-generated local rings becomes more and more sophisticated. For instance, the ring $R_k = \mathbb{Z}_2[X_1, \dots, X_k]/(X_1^2, \dots, X_k^2)$ is a k-generated local ring which is not $(k-1)$-generated.

▶ **Lemma 11** (Ordering k-generated local rings). *There is an* FP*-formula $\phi(x, z_1, \dots, z_k; v, w)$ such that for all k-generated local rings R there are $\alpha, \pi_1, \dots, \pi_k \in R$ such that*

$$\phi^R(\alpha/x, \vec{\pi}/\vec{z}; v, w) = \{(a, b) \in R \times R \mid (R, \alpha, \vec{\pi}; a, b) \models \phi\}, \textit{ is a linear order on } R.$$

Proof. Firstly, there are FP-formulas $\phi_u(x), \phi_m(x), \phi_g(x_1, \dots, x_k)$ that define in each k-generated local ring R the set of units, the maximal ideal m (which is the set of non-units) and the property of being a set of size k that generates m, respectively. More specifically, for all $(\pi_1, \dots, \pi_k) \in \phi_g^R$ we have that $\sum_i \pi_i R = \phi_m^R$ is the maximal ideal of R and $R^\times = \phi_u^R = R \setminus m$. In particular there is a first-order interpretation of the field $k := R/m$ in R.

The idea of the proof is to represent the elements of R as polynomial expressions of a certain kind. Let $q := |\,k\,|$ and define $\Gamma(R) := \{r \in R : r^q = r\}$. It can be seen that $\Gamma(R) \setminus \{0\}$ forms a multiplicative group which is known as the *Teichmüller coordinate set* [5]. Now, the map $\iota : \Gamma(R) \to k$ defined by $r \mapsto r + m$ is a bijection. Indeed, for two different units $r, s \in \Gamma(R)$ we have $r - s \notin m$. Otherwise, we would have $r - s = x$ for some $x \in m$ and thus $r = (s + x)^q = s + \sum_{k=1}^{q} \binom{q}{k} x^k s^{q-k}$. Since $q \in m$ and $r - s = x$ we obtain that $x = xy$ for some $y \in m$. Hence $x(1 - y) = 0$ and since $(1 - y) \in R^\times$ this means $x = 0$.

As explained above, we can define in FP an order on k by fixing a generator $\alpha \in k^\times$ of the cyclic group $k^\times$. Combining this order with ι^{-1}, we obtain an FP-definable order on $\Gamma(R)$. The importance of $\Gamma(R)$ lies in the fact that every ring element can be expressed as a polynomial expression over a set of k generators of the maximal ideal m with coefficients lying in $\Gamma(R)$. To be precise, let $\pi_1, \dots, \pi_k \in m$ be a set of generators for m, i.e. $m = \pi_1 R + \dots + \pi_k R$, where each π_i has nilpotency n_i for $1 \leq i \leq k$. We claim that we can express $r \in R$ as

$$r = \sum_{(i_1, \dots, i_k) \leq_{\text{lex}} (n_1, \dots, n_k)} a_{i_1 \cdots i_k} \pi_1^{i_1} \cdots \pi_k^{i_k}, \quad \text{with } a_{i_1 \cdots i_k} \in \Gamma(R). \tag{P}$$

To see this, consider the following recursive algorithm:

- If $r \in R^\times$, then for a unique $a \in \Gamma(R)$ we have that $r \in a + m$, so $r = a + (\pi_1 r_1 + \dots + \pi_k r_k)$ for some $r_1, \dots, r_k \in R$ and we continue with $r_1, \dots, r_k$.
- Else $r \in m$, and $r = \pi_1 r_1 + \dots + \pi_k r_k$ for some $r_1, \dots, r_k \in R$; continue with $r_1, \dots, r_k$.

Observe that for all pairs $a, b \in \Gamma(R)$ there exist elements $c \in \Gamma(R), r \in m$ such that $a\pi_1^{i_1} \cdots \pi_k^{i_k} + b\pi_1^{i_1} \cdots \pi_k^{i_k} = c\pi_1^{i_1} \cdots \pi_k^{i_k} + r\pi_1^{i_1} \cdots \pi_k^{i_k}$. Since $\pi_1^{i_1} \cdots \pi_k^{i_k} = 0$ if $i_l \geq n_l$ for some $1 \leq l \leq k$, the process is guaranteed to stop and the claim follows.

Note that this procedure neither yields a polynomial-time algorithm nor do we obtain a *unique* expression, as for instance, the choice of elements $r_1, \dots, r_k \in R$ (in both recursion steps) need not to be unique. However, knowing only the existence of an expression of this kind, we can proceed as follows. For any sequence of exponents $(\ell_1, \dots, \ell_k) \leq_{\text{lex}} (n_1, \dots, n_k)$

define the ideal $R[\ell_1, \dots, \ell_k] \trianglelefteq R$ as the set of all elements having an expression of the form (P) where $a_{i_1 \cdots i_k} = 0$ for all $(i_1, \dots, i_k) \leq_{\text{lex}} (\ell_1, \dots, \ell_k)$.

It is clear that we can define the ideal $R[\ell_1, \dots, \ell_k]$ in FP. Having this, we can use the following recursive procedure to define a unique expression of the form (P) for all $r \in R$:

- Choose the minimal $(i_1, \dots, i_k) \leq_{\text{lex}} (n_1, \dots, n_k)$ such that $r = a\pi_1^{i_1} \cdots \pi_k^{i_k} + s$ for some (minimal) $a \in \Gamma(R)$ and $s \in R[i_1, \dots, i_k]$. Continue the process with s.

Finally, the lexicographical ordering induced by the ordering on $n_1 \times \cdots \times n_k$ and the ordering on $\Gamma(R)$ yields an FP-definable order on R (with parameters for generators of $k^\times$ and m). ◀

▶ **Corollary 12.** $\textsc{SlvLR}_k \leq_{\text{FP-T}} \textsc{SlvCR}_{\leqslant} \leq_{\text{FP}} \textsc{SlvCycG}$.

5 Solvability problems under logical reductions

In the previous two sections we studied reductions between solvability problems over different algebraic domains. Here we change our perspective and investigate classes of *queries* that are reducible to solvability over a fixed commutative ring. Our motivation for this work was to study extensions of first-order logic with generalised quantifiers which express solvability problems over rings. In particular, the aim was to establish various *normal forms* for such logics. However, rather than defining a host of new logics in full detail, we state our results in this section in terms of closure properties of classes of finite structures that are themselves defined by reductions to solvability problems. We explain the connection between the specific closure properties and the corresponding logical normal forms in more detail below.

To state our main results formally, let R be a commutative ring and write $\textsc{Slv}(R)$ to denote the solvability problem over R, as a class of $\tau_{\text{les}}(R)$-structures. Let $\Sigma^{\text{qf}}_{\text{FO}}(R)$ and $\Sigma_{\text{FO}}(R)$ denote the classes of queries that are reducible to $\textsc{Slv}(R)$ under quantifier-free and first-order many-to-one reductions, respectively. Then we show that $\Sigma^{\text{qf}}_{\text{FO}}(R)$ and $\Sigma_{\text{FO}}(R)$ are closed under first-order operations for any commutative ring R, which also shows that $\Sigma^{\text{qf}}_{\text{FO}}(R)$ contains *any* FO-definable query. Furthermore, we prove that if R has prime characteristic, then $\Sigma^{\text{qf}}_{\text{FO}}(R)$ and $\Sigma_{\text{FO}}(R)$ are closed under oracle queries. Thus, if we denote by $\Sigma^{\text{T}}_{\text{FO}}(R)$ the class of queries reducible to $\textsc{Slv}(R)$ by first-order Turing reductions, then for all commutative rings R of prime characteristic the three solvability reduction classes coincide, i.e. we have $\Sigma^{\text{qf}}_{\text{FO}}(R) = \Sigma_{\text{FO}}(R) = \Sigma^{\text{T}}_{\text{FO}}(R)$.

To relate these results to logical normal forms, we let $\mathcal{D} = \textsc{Slv}(R)$ and write $\text{FOS}_{\text{R}} := \text{FO}(\langle Q_{\mathcal{D}} \rangle)$ to denote first-order logic extended by generalised Lindström quantifiers deciding solvability over R. Then the closure of $\Sigma_{\text{FO}}(R)$ under first-order operations amounts to showing that the fragment of FOS_{R} which consists of formulas without *nested* solvability quantifiers has a normal form which consists of a single application of a solvability quantifier to a first-order formula. Moreover, for the case when R has prime characteristic, the closure of $\Sigma^{\text{qf}}_{\text{FO}}(R) = \Sigma_{\text{FO}}(R)$ under first-order oracle queries amounts to showing that nesting of solvability quantifiers can be reduced to a single quantifier. It follows that FOS_{R} has a strong normal form: one application of a solvability quantifier to a *quantifier-free* formula suffices.

5.1 Closure under first-order operations

Let R be a fixed commutative ring of characteristic m. In this section we prove the closure of $\Sigma^{\text{qf}}_{\text{FO}}(R)$ and $\Sigma_{\text{FO}}(R)$ under first-order operations. To this end, we need to establish a couple of technical results. Of particular importance is the following key lemma, which gives a simple normal form for linear equation systems: up to quantifier-free reductions, we can restrict ourselves to linear systems over rings $\mathbb{Z}_m$, where the constant term of every equation is $1 \in \mathbb{Z}_m$. The proof of the lemma crucially relies on the fact that the ring R is fixed.

▶ **Lemma 13** (Normal form for linear equation systems). *There is a quantifier-free interpretation $\mathcal{I}$ of $\tau_{les}(\mathbb{Z}_m)$ in $\tau_{les}(R)$ so that for all $\tau_{les}(R)$-structures $\mathbf{S}$ it holds that*

- *$\mathcal{I}(\mathbf{S})$ is an equation system $(A, \mathbf{b})$ over $\mathbb{Z}_m$, where A is a $\{0,1\}$-matrix and $\mathbf{b} = \mathbf{1}$; and*
- *$\mathbf{S} \in \textsc{Slv}(R)$ if, and only if, $\mathcal{I}(\mathbf{S}) \in \textsc{Slv}(\mathbb{Z}_m)$.*

Proof. We describe $\mathcal{I}$ as the composition of three quantifier-free transformations: the first one maps a system $(A, \mathbf{b})$ over R to an equivalent system $(B, \mathbf{c})$ over $\mathbb{Z}_m$, where m is the characteristic of R. Secondly, $(B, \mathbf{c})$ is mapped to an equivalent system $(C, \mathbf{1})$ over $\mathbb{Z}_m$. Finally, we transform $(C, \mathbf{1})$ into an equivalent system $(D, \mathbf{1})$ over $\mathbb{Z}_m$, where D is a $\{0,1\}$-matrix. The first transformation is obtained by adapting the proof of Theorem 2. It can be seen that first-order quantifiers and fixed-point operators are not needed if R is fixed.

For the second transformation, suppose that B is an $I \times J$ matrix and $\mathbf{c}$ a vector indexed by I. We define a new linear equation system $\mathbf{T}$ which has in addition to all the variables that occur in $\mathbf{S}$, a new variable v_e for every $e \in I$ and a new variable w_r for every $r \in R$. For every element $r \in \mathbb{Z}_m$, we include in $\mathbf{T}$ the equation $(1-r)w_1 + w_r = 1$. It can be seen that this subsysem of equations has a unique solution given by $w_r = r$ for all $r \in \mathbb{Z}_m$. Finally, for every equation $\sum_{j \in J} B(e,j) \cdot x_j = \mathbf{c}(e)$ in $\mathbf{S}$ (indexed by $e \in I$) we include in $\mathbf{T}$ the two equations $v_e + \sum_{j \in J} B(e,j) \cdot x_j = 1$ and $v_e + w_{\mathbf{c}(e)} = 1$.

Finally, we translate the system $\mathbf{T} : C\mathbf{x} = \mathbf{1}$ over $\mathbb{Z}_m$ into an equivalent system over $\mathbb{Z}_m$ in which all coefficients are either 0 or 1. For each variable v in $\mathbf{T}$, the system has the m distinct variables $v_0, \dots, v_{m-1}$ together with equations $v_i = v_j$ for $i \neq j$. By replacing all terms rv by $\sum_{1 \leq i \leq r} v_i$ we obtain an equivalent system. However, in order to establish our original claim we need to rewrite the auxiliary equations of the form $v_i = v_j$ as a set of equations whose constant terms are equal to 1. To achieve this, we introduce a new variable v_j^- for each v_j, and the equation $v_j + v_j^- + w_1 = 1$. Finally, we rewrite $v_i = v_j$ as $v_i + v_j^- + w_1 = 1$. The resulting system is equivalent to $\mathbf{T}$ and has the desired form. ◀

▶ **Corollary 14.** *$\Sigma^{qf}_{\mathrm{FO}}(R) = \Sigma^{qf}_{\mathrm{FO}}(\mathbb{Z}_m)$, $\Sigma_{\mathrm{FO}}(R) = \Sigma_{\mathrm{FO}}(\mathbb{Z}_m)$ and $\Sigma^{T}_{\mathrm{FO}}(R) = \Sigma^{T}_{\mathrm{FO}}(\mathbb{Z}_m)$.*

It is a basic fact from linear algebra that solvability of a linear equation system $A \cdot \mathbf{x} = \mathbf{b}$ is invariant under applying elementary row and column operations to the augmented coefficient matrix $(A \mid \mathbf{b})$. Over fields, this insight justifies the method of Gaussian elimination, which transforms the augmented coefficient matrix of a linear system into row echelon form. Over the integers, a generalisation of this method can be used to transform a linear system into Hermite normal form. The following lemma shows that a similar normal form exists over chain rings. The proof uses the fact that in a chain ring R, divisibility is a total preorder.

▶ **Lemma 15** (Hermite normal form). *For every $k \times \ell$-matrix A over a chain ring R, there exists an invertible $k \times k$-matrix S and an $\ell \times \ell$-permutation matrix T so that*

$$SAT = \begin{pmatrix} Q \\ \mathbf{0} \end{pmatrix} \quad \text{with} \quad Q = \begin{pmatrix} a_{11} & \cdots & \star \\ 0 & \ddots & \vdots \\ \mathbf{0} & 0 & a_{kk} \end{pmatrix},$$

where $a_{11} \mid a_{22} \mid a_{33} \mid \cdots \mid a_{kk}$ and for all $1 \leq i,j \leq k$ it holds that $a_{ii} \mid a_{ij}$.

Now we are ready to prove the closure of $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R)$ and $\Sigma_{\mathrm{FO}}(R)$ under first-order operations. First of all, it can be seen that conjunction and universal quantification can be handled easily by combining independent subsystems into a single system. Thus, the only non-trivial part of the proof is to establish closure under complementation. To do that, we describe an appropriate reduction that translates from non-solvability to solvability of linear systems.

First of all, we consider the case where R has characteristic $m = p$ for a prime p. In this case we know that $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R) = \Sigma^{\mathrm{qf}}_{\mathrm{FO}}(\mathbb{Z}_p)$ and $\Sigma_{\mathrm{FO}}(R) = \Sigma_{\mathrm{FO}}(\mathbb{Z}_p)$ by Corollary 14, where $\mathbb{Z}_p$ is a finite field. Over fields, the method of Gaussian elimination guarantees that a linear equation system $(A, \mathbf{b})$ is not solvable if, and only if, for some vector $\mathbf{x}$ we have $\mathbf{x} \cdot (A \mid \mathbf{b}) = (0, \dots, 0, 1)$. In other words, the vector $\mathbf{b}$ is not in the column span of A if, and only if, the vector $(0, \dots, 0, 1)$ is in the row span of $(A \mid \mathbf{b})$. This shows that $(A \mid \mathbf{b})$ is not solvable if, and only if, the system $((A \mid \mathbf{b})^T, (0, \dots, 0, 1)^T)$ is solvable. In other words, over fields this reasoning translates the question of non-solvability to the question of solvability. In the proof of the next lemma, we generalise this approach to chain rings, which enables us to translate from non-solvability to solvability over all rings of prime-power characteristic.

▶ **Lemma 16** (Non-solvability over chain rings). *Let $(A, \mathbf{b})$ be a linear equation system over a chain ring R with maximal ideal πR and let n be the nilpotency of π. Then $(A, \mathbf{b})$ is not solvable over R if, and only if, there is a vector $\mathbf{x}$ such that $\mathbf{x} \cdot (A \mid \mathbf{b}) = (0, \dots, 0, \pi^{n-1})$.*

Proof. If such a vector $\mathbf{x}$ exists, then $(A, \mathbf{b})$ is not solvable. On the other hand, if no such $\mathbf{x}$ exists, then we apply Lemma 15 to transform the augmented matrix $(A \mid \mathbf{b})$ into Hermite normal form $(A' \mid \mathbf{b}')$ with respect to A (that is, $A' = SAT$ as in Lemma 15 and $\mathbf{b}' = S\mathbf{b}$). We claim that for every row index i, the diagonal entry a_{ii} in the transformed coefficient matrix A' divides the i-th entry of the transformed target vector $\mathbf{b}'$. Towards a contradiction, suppose that there is some a_{ii} not dividing $\mathbf{b}'_i$. Then a_{ii} is not a non-unit in R and can be written as $a_{ii} = u\pi^t$ for some unit u and $t \geq 1$. By Lemma 15, it holds that a_{ii} divides every entry in the i-th row of A' and thus we can multiply the i-th row of the augmented matrix $(A' \mid \mathbf{b}')$ by an appropriate non-unit to obtain a vector of the form $(0, \dots, 0, \pi^{n-1})$, contradicting our assumption. Hence, in the transformed augmented coefficient matrix, diagonal entries divide *all* entries in the same row, which implies solvability of $(A \mid \mathbf{b})$. ◀

Along with our previous discussion, Lemma 16 now yields the closure of $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R)$ and $\Sigma_{\mathrm{FO}}(R)$ under complementation if R has prime-power characteristic. For a linear systems $(A, \mathbf{b})$ over a non-local ring $\mathbb{Z}_m$ (i.e. m is not a prime power), we can consider the decomposition of $\mathbb{Z}_m$ into a direct sum of local rings and apply the Chinese remainder theorem.

▶ **Theorem 17.** *$\Sigma^{qf}_{\mathrm{FO}}(R)$, $\Sigma_{\mathrm{FO}}(R)$ and $\Sigma^{T}_{\mathrm{FO}}(R)$ are closed under first-order operations.*

5.2 Solvability over rings of prime characteristic

From now on we assume that the commutative ring R is of prime characteristic p. We prove that in this case, the three reduction classes $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R)$, $\Sigma_{\mathrm{FO}}(R)$ and $\Sigma^{\mathrm{T}}_{\mathrm{FO}}(R)$ coincide. First of all, we note that, by definition, we have $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R) \subseteq \Sigma_{\mathrm{FO}}(R) \subseteq \Sigma^{\mathrm{T}}_{\mathrm{FO}}(R)$. Also, since we know that solvability over R can be reduced to solvability over $\mathbb{Z}_p$ (Corollary 14), it suffices for our proof to show that $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(\mathbb{Z}_p) \supseteq \Sigma^{\mathrm{T}}_{\mathrm{FO}}(\mathbb{Z}_p)$. Furthermore, by Theorem 17 it follows that $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(\mathbb{Z}_p)$ is closed under first-order operations, so it only remains to prove closure under oracle queries. Recalling that the original motivation for this study was to establish normal forms for logics with solvability quantifiers, it can be seen that proving closure under oracle queries corresponds to showing that for every formula of $\mathrm{FOS}_{\mathrm{R}}$ with nested solvability quantifiers, where R has prime characteristic, there is an equivalent $\mathrm{FOS}_{\mathrm{R}}$-formula with no nested solvability quantifiers. Since $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R)$ is closed under first-order operations, any

FO-definable query is contained in $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(R)$; thus, we can conclude that every $\mathrm{FOS}_{\mathrm{R}}$-formula is equivalent to the single application of a solvability quantifier to a quantifier-free formula.

More specifically in terms of the classes $\Sigma^{\mathrm{qf}}_{\mathrm{FO}}(\mathbb{Z}_p)$, it can be seen that proving closure under oracle queries amounts to showing that nesting of linear equation systems can be reduced to a single system only. To formalise this, let $\mathcal{I}(\vec{x},\vec{y})$ be a quantifier-free interpretation of $\tau_{\mathrm{les}}(\mathbb{Z}_p)$ in σ with parameters $\vec{x},\vec{y}$ of length k and l, respectively. We extend the signature σ to $\sigma_X := \sigma \cup \{X\}$ and restrict our attention to those σ_X-structures $\mathbf{A}$ (with domain A) where the relation symbol X is interpreted as $X^{\mathbf{A}} = \{(\vec{a},\vec{b}) \in A^{k\times l} \mid \mathcal{I}(\vec{a},\vec{b})(\mathbf{A}) \in \textsc{Slv}(\mathbb{Z}_p)\}$. Then it remains to show that for any quantifier-free interpretation $\mathcal{O}$ of $\tau_{\mathrm{les}}(\mathbb{Z}_p)$ in σ_X, there is a quantifier-free interpretation of $\tau_{\mathrm{les}}(\mathbb{Z}_p)$ in σ that describes linear equation systems equivalent to $\mathcal{O}$. Hereafter, for any σ_X-structure $\mathbf{A}$ and tuples $\vec{a}$ and $\vec{b}$, we will refer to $\mathcal{O}(\mathbf{A})$ as an "outer" linear equation system and refer to $\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$ as an "inner" linear equation system. By applying Lemma 13 and Theorem 17, it is sufficient to consider the case where for σ_X-structures $\mathbf{A}$, $\mathcal{O}(\mathbf{A})$ describes a linear system $(M,\mathbf{1})$, where M is the $\{0,1\}$-matrix of the relation $X^{\mathbf{A}}$. For an illustration of this setup, see Figure 2.

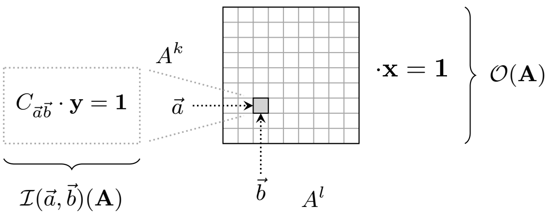

A^k
$C_{\vec{a}\vec{b}} \cdot \mathbf{y} = \mathbf{1}$
$\vec{a}$
$\cdot\mathbf{x} = \mathbf{1}$
$\mathcal{O}(\mathbf{A})$
$\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$
$\vec{b}$
A^l

Figure 2 Each entry $(\vec{a},\vec{b})$ of the coefficient matrix of the outer linear equation system $\mathcal{O}(\mathbf{A})$ is determined by the corresponding inner linear system $C_{\vec{a}\vec{b}} \cdot \mathbf{y} = \mathbf{1}$ described by $\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$: this entry is 1 if $\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$ is solvable and 0 otherwise.

▶ **Theorem 18** (Closure under oracle queries). *For $\mathcal{I}$, $\mathcal{O}$ as above, there exists a quantifier-free interpretation $\mathcal{K}$ of $\tau_{les}(\mathbb{Z}_p)$ in σ such that for all σ_X-structures $\mathbf{A}$ it holds that $\mathcal{O}(\mathbf{A}) \in \textsc{Slv}(\mathbb{Z}_p)$ if, and only if, $\mathcal{K}(\mathbf{A}) \in \textsc{Slv}(\mathbb{Z}_p)$.*

Proof. For a σ-structure $\mathbf{A}$, let M_o denote the $\{0,1\}$-coefficient matrix of the outer linear equation system $\mathcal{O}(\mathbf{A})$. Then for $(\vec{a},\vec{b}) \in A^{k\times l}$ we have $M_o(\vec{a},\vec{b}) = 1$ if, and only if, the inner linear system $\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$ is solvable. By identifying the variables of $\mathcal{O}(\mathbf{A})$ by $\{v_{\vec{b}} \mid \vec{b} \in A^l\}$, we can express the equations of $\mathcal{O}(\mathbf{A})$ as $\sum_{\vec{b}\in A^l} M_o(\vec{a},\vec{b}) \cdot v_{\vec{b}} = 1$, for $\vec{a} \in A^k$.

We begin to construct the system $\mathcal{K}(\mathbf{A})$ over the set of variables $\{v_{\vec{a},\vec{b}} \mid (\vec{a},\vec{b}) \in A^{k\times l}\}$ by including the equations $\sum_{\vec{b}\in A} v_{\vec{a},\vec{b}} = 1$ for all $\vec{a} \in A^k$. Our aim is to extend $\mathcal{K}(\mathbf{A})$ by additional equations so that in every solution to $\mathcal{K}(\mathbf{A})$, there are values $v_{\vec{b}} \in \mathbb{Z}_p$ such that for all $\vec{a} \in A^k$, we have $v_{\vec{a},\vec{b}} = M_o(\vec{a},\vec{b}) \cdot v_{\vec{b}}$. Assuming this to be true, it is immediate that $\mathcal{O}(\mathbf{A})$ is solvable if, and only if, $\mathcal{K}(\mathbf{A})$ is solvable, which is what we want to show.

In order to enforce the condition "$v_{\vec{a},\vec{b}} = M_o(\vec{a},\vec{b}) \cdot v_{\vec{b}}$" by *linear* equations, we need to introduce a number of auxilliary linear subsystems to $\mathcal{K}(\mathbf{A})$. The reason why we cannot express this condition directly by a linear equation is because $M_o(\vec{a},\vec{b})$ is determined by solvability of the inner system $\mathcal{I}(\vec{a},\vec{b})(\mathbf{A})$. Therefore, if we were to treat both the elements of $M_o(\vec{a},\vec{b})$ and the $v_{\vec{b}}$ as individual variables, then that would require to express the *non-linear* term $M_o(\vec{a},\vec{b}) \cdot v_{\vec{b}}$. To overcome this issue, we introduce new subsystems in $\mathcal{K}(\mathbf{A})$ to ensure that for all $\vec{a},\vec{b},\vec{c} \in A$:

$$\textbf{if } v_{\vec{a},\vec{b}} \neq 0 \textbf{ then } M_o(\vec{a},\vec{b}) = 1; \text{ and} \tag{$*$}$$

$$\textbf{if } v_{\vec{a},\vec{b}} \neq v_{\vec{c},\vec{b}} \textbf{ then } \{M_o(\vec{a},\vec{b}), M_o(\vec{c},\vec{b})\} = \{0,1\}. \tag{$\dagger$}$$

Assuming we have expressed $(*)$ and $(\dagger)$, it can be seen that solutions of $\mathcal{K}(\mathbf{A})$ directly translate into solutions for $\mathcal{O}(\mathbf{A})$ and vice versa. To express $(*)$ we proceed as follows: for each $(\vec{a}, \vec{b}) \in A^{k\times l}$ we introduce $\mathcal{I}(\vec{a}, \vec{b})(\mathbf{A})$ as an independent linear subsystem in $\mathcal{K}(\mathbf{A})$ in which we additionally add to each single equation the term $(v_{\vec{a},\vec{b}} + 1)$. Now, if in a solution of $\mathcal{K}(\mathbf{A})$ the variable $v_{\vec{a},\vec{b}}$ is evaluated to 0, then the subsystem corresponding to $\mathcal{I}(\vec{a}, \vec{b})(\mathbf{A})$ is trivially solvable (recall, that the target vector is $\mathbf{1}$). However, if a non-zero value is assigned to $v_{\vec{a},\vec{b}}$, then this value is a unit in $\mathbb{Z}_p$ and thereby a solution for $\mathcal{K}(\mathbf{A})$ necessarily contains a solution of the subsystem $\mathcal{I}(\vec{a}, \vec{b})(\mathbf{A})$; that is, we have $M_o(\vec{a}, \vec{b}) = 1$.

For (†) we follow a similar approach. For fixed tuples $\vec{a}$, $\vec{b}$ and $\vec{c}$, the condition on the right-hand side of (†) is a simple Boolean combination of solvability queries. Hence, by Theorem 17, this combination can be expressed by a single linear equation system. Again we embed the respective linear equation system as a subsystem in $\mathcal{K}(\mathbf{A})$ where we add to each of its equations the term $(1 + v_{\vec{a},\vec{b}} - v_{\vec{c},\vec{b}})$. With the same reasoning as above we conclude that this imposes the constraint (†) on the variables $v_{\vec{a},\vec{b}}$ and $v_{\vec{c},\vec{b}}$, which concludes the proof. ◀

▶ **Corollary 19.** *If R has prime characteristic, then $\Sigma^{qf}_{\text{FO}}(R) = \Sigma_{\text{FO}}(R) = \Sigma^{T}_{\text{FO}}(R)$.*

As explained above, our results have some important consequences. For a prime p, let us denote by FOS_p first-order logic extended by quantifiers deciding solvability over $\mathbb{Z}_p$, similar to what we have discussed before. Corresponding extensions of first-order logic by rank operators over prime fields (FOR_p) were studied by Dawar et al. [8]. Their results imply that $\text{FOS}_p = \text{FOR}_p$ over ordered structures, and that both logics have a strong normal form over ordered structures, i.e. that every formula is equivalent to a formula with only one application of a solvability or rank operator, respectively [21]. Corollary 19 allows us to generalise the latter result for FOS_p to arbitrary structures.

▶ **Corollary 20.** *Every $\phi \in \text{FOS}_p$ is equivalent to a formula with a single solvability quantifier.*

6 Discussion

Motivated by the question of finding extensions of FPC to capture larger fragments of PTIME, we have analysed the (inter-)definability of solvability problems over various classes of algebraic domains. Similar to the notion of rank logic [8] one can consider *solvability logic*, which is the extension of FPC by (generalised Lindström) quantifiers that decide solvability of linear equation systems. In this context, our results from §3 and §4 can be seen to relate fragments of solvability logic obtained by restricting quantifiers to different algebraic domains, such as Abelian groups or commutative rings. We have also identified many classes of algebraic structures over which the solvability problem reduces to the very basic problem of solvability over cyclic groups of prime-power order. This raises the question, whether a reduction even to groups of prime order is possible. In this case, solvability logic would turn out to be a fragment of rank logic.

With respect to specific algebraic domains, we proved that FPC can define a linear order on the class of all k-generated local rings, i.e. on classes of local rings for which every maximal ideal can be generated by k elements, where k is a fixed constant. Together with our results from §4, this can be used to show that all natural problems from linear algebra over (not necessarily local) k-generated rings reduce to problems over ordered rings under FP-reductions. An interesting direction of future research is to explore how far our techniques can be used to show (non-)definability in fixed-point logic of other problems from linear algebra over rings.

Finally, we mention an interesting topic of related research, which is the logical study of *permutation group membership problems* (GM for short). An instance of GM consists of a set Ω, a set of generating permutations $\pi_1, \dots, \pi_n$ on Ω and a target permutation π, and the problem is to decide whether π is generated by $\pi_1, \dots, \pi_n$. This problem is known to be decidable in polynomial time (indeed it is in NC [4]). We can show that all the solvability problems we have studied in this paper reduce to GM under first-order reductions (basically, an application of Cayley's theorem). In particular this shows that GM is not definable in FPC. By extending fixed-point logic by a suitable operator for GM we therefore obtain a logic which extends rank logics and in which all studied solvability problems are definable. This logic is worth a further study as it can uniformly express all problems from (linear) algebra that have been considered so far in the context of understanding the descriptive complexity gap between FPC and PTIME.

References

1 V. Arvind and T. C. Vijayaraghavan. Classifying problems on linear congruences and abelian permutation groups using logspace counting classes. *Comp. Compl.*, 19:57–98, 2010.

2 M. F. Atiyah and I. G. Macdonald. *Introduction to commutative algebra*, volume 29. Addison-Wesley, 1969.

3 A. Atserias, A. Bulatov, and A. Dawar. Affine systems of equations and counting infinitary logic. *Theoretical Computer Science*, 410:1666–1683, 2009.

4 L. Babai, E. Luks, and A. Seress. Permutation groups in NC. In *STOC '87*, page 409–420. ACM Press, 1987.

5 G. Bini and F. Flamini. *Finite Commutative Rings and Their Applications*. Kluwer Academic Publishers, 2002.

6 J-Y. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12(4):389–410, 1992.

7 A. Dawar. Generalized quantifiers and logical reducibilities. *J. Logic Comp.*, 5(2):213, 1995.

8 A. Dawar, M. Grohe, B. Holm, and B. Laubner. Logics with Rank Operators. In *LICS '09*, pages 113–122. IEEE Computer Society, 2009.

9 H.-D. Ebbinghaus and J. Flum. *Finite model theory.* Springer-Verlag, 2nd edition, 1999.

10 E. Grädel et. al. *Finite Model Theory and Its Applications.* Springer-Verlag, 2007.

11 Mikael Goldmann and Alexander Russell. The complexity of solving equations over finite groups. *Inf. Comput.*, 178:253–262, October 2002.

12 M. Grohe. Fixed-point logics on planar graphs. In *LICS '98*, pages 6—15, 1998.

13 M. Grohe. The quest for a logic capturing PTIME. In *LICS '08*, pages 267–271, 2008.

14 M. Grohe. Fixed-point definability and polynomial time on graph with excluded minors. In *LICS '10*, pages 179 – 188, 2010.

15 M. Grohe and J. Mariño. Definability and descriptive complexity on databases of bounded tree-width. In *ICDT '99*, volume 1540, pages 70–82. Springer-Verlag, 1999.

16 B. Holm. *Descriptive complexity of linear algebra.* PhD thesis, Univ. of Cambridge, 2010.

17 N. Immerman. Relational queries computable in polynomial time. *Inf. and Control*, 68:86–104, 1986.

18 P. Lindström. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

19 B. R. McDonald. *Finite rings with identity.* Dekker, 1974.

20 M. Otto. *Bounded Variable Logics and Counting — A Study in Finite Models*, volume 9 of *Lecture Notes in Logic.* Springer-Verlag, 1997.

21 W. Pakusa. Finite model theory with operators from linear algebra. Staatsexamensarbeit, RWTH Aachen University, 2010.

22 M. Y. Vardi. The complexity of relational query languages. In *STOC '82*, pages 137–146. ACM Press, 1982.

Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication*

Henry DeYoung[1], Luís Caires[2], Frank Pfenning[1], and Bernardo Toninho[1,2]

1 **Computer Science Department, Carnegie Mellon University**
Pittsburgh, PA, USA
{hdeyoung, fp, btoninho}@cs.cmu.edu

2 **Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa**
Lisboa, Portugal
luis.caires@di.fct.unl.pt

Abstract

Prior work has shown that intuitionistic linear logic can be seen as a session-type discipline for the π-calculus, where cut reduction in the sequent calculus corresponds to synchronous process reduction. In this paper, we exhibit a new process assignment from the *asynchronous*, polyadic π-calculus to exactly the same proof rules. Proof-theoretically, the difference between these interpretations can be understood through permutations of inference rules that preserve observational equivalence of closed processes in the synchronous case. We also show that, under this new asynchronous interpretation, cut reductions correspond to a natural asynchronous buffered session semantics, where each session is allocated a separate communication buffer.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases linear logic, cut reduction, asynchronous π-calculus, session types

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.228

1 Introduction

Ever since linear logic was originally proposed, researchers have been discovering and exploring its deep and perhaps surprising connections with concurrency in a variety of ways. Girard himself first sketched a connection of linear logic with concurrency, by giving a high-level pattern of communication that manifested itself in proof nets [13]. Others expanded upon this with further models based on proof nets and related structures, e.g., [1, 6, 5, 16]. In a different vein, two of the present authors recently developed a Curry-Howard interpretation of intuitionistic linear logic [8], where propositions are interpreted as session types [15, 14], sequent calculus proofs are interpreted as π-calculus processes, and proof reduction during cut elimination is interpreted as synchronous communication.

A natural follow-up question to this work is whether a Curry-Howard correspondence between linear logic and an *asynchronous* process calculus can be established. An answer is relevant both to the concurrency theorist and the logician. For the concurrency theorist, asynchronous communication is a more realistic (and challenging) model for concurrency, and so being able to establish properties of asynchronous processes by static typing is of great interest. For the logician, asynchrony can be seen as eliminating at least some of the

* Support was provided by the Fundação para a Ciência e a Tecnologia through the Carnegie Mellon Portugal Program, under grants SFRH/BD/33763/2009 and INTERFACES NGN-44/2009, and CITI.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 228–242
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

"bureaucracy of syntax," so that the order in which certain proof rules are applied no longer imposes artificial sequentiality.

Our novel interpretation assigns processes from the asynchronous polyadic π-calculus[1] [7, 17] to a sequent calculus formulation of DILL [3, 10] (the same proof rules as [8]), and cut reductions to asynchronous communication, as we detail in Section 2. Moreover, in Section 3 we formally determine the relationship between the prior synchronous interpretation with our asynchronous one. We show that, proof-theoretically, the fundamental difference between the two is that a class of commuting conversions that in the synchronous interpretation corresponded to only observational equivalences, now map to natural *structural* equivalences in the asynchronous π-calculus. Finally, in Section 4, we relate our asynchronous interpretation to buffered communication.

2 Linear logic as asynchronous session-typed communication

2.1 Judgmental principles

Because processes offer services along designated channels, our basic session-typing judgment is $P :: x{:}A$, meaning "process P offers service A along channel x." However, to provide new services, most processes must themselves rely on services offered by other processes. Thus, more generally, we use the hypothetical judgment

$$x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: x{:}A\,,$$

meaning "Using services A_i that are assumed to be provided along channels x_i, process P offers service A along channel x." The channels x_i and x must all be distinct, and are binding occurrences with scope over the process P. (We use the metavariable Δ and its decorated variants to stand for an arbitrary context of services.)

When two processes interact, their state changes; one now offers, and the other uses, the continuation of the initial service. Due to this change of state, our hypothetical judgment can be seen as an annotation of the intuitionistic linear sequent $A_1, \ldots, A_n \vdash A$. The context of services satisfies neither weakening nor contraction. It does satisfy exchange, however, because antecedents are uniquely labeled. Our process interpretation also handles persistent antecedents, but we postpone their introduction until Section 2.6 to keep the overhead of initial exposition lower.

The sequent calculus cut and identity rules serve to clarify the relationship between offering and using a service.

Cut as composition. In the sequent calculus, the cut rule composes a proof of lemma A with its use in the proof of theorem C:

$$\frac{\Delta \vdash A \quad \Delta', A \vdash C}{\Delta, \Delta' \vdash C}\ \mathsf{cut}$$

Because proofs should correspond to processes, this reading suggests that the process interpretation of the cut rule should compose a process that offers service A with one that uses service A. Stated differently, an offer satisfies a use. Therefore, we annotate cut as

$$\frac{\Delta \vdash P :: x{:}A \quad \Delta', x{:}A \vdash Q :: z{:}C}{\Delta, \Delta' \vdash (\nu x)(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

[1] Our interpretation in fact uses only niladic, monadic, and dyadic processes, not general polyadic communication; for brevity, however, we prefer to retain 'polyadic' as the collective term.

The process $(\nu x)(P \mid Q)$ allows to execute in parallel, as indicated by $P \mid Q$, and interact along the private channel x, as indicated by the name restriction (νx). (The occurrence of x in the name restriction (νx) is a binding occurrence, and is therefore subject to renaming.)

Identity as forwarding. The identity rule uses an antecedent to construct a proof directly:

$$\frac{}{A \vdash A}\ \mathsf{id}$$

This suggests an interpretation of id as a forwarding process, $[y \leftrightarrow x]$:

$$\frac{}{y{:}A \vdash [y \leftrightarrow x] :: x{:}A}\ \mathsf{id}$$

The process $[y \leftrightarrow x]$ forwards messages received along channel x further on to channel y, and vice versa, so that it offers A along x by directly using the service A that is available from y. Stated differently, a use is one way to fulfill an offer.

2.2 Implication as input

In our process interpretation, the linear logical connectives correspond to various basic forms of service. We adopt a verificationist perspective: the sequent calculus right rules will define what it means to offer a service, whereas the left rules show how to use a service.

Consider linear implication, written $A \multimap B$, and recall its right and left rules:

$$\frac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ {\multimap}R \qquad \frac{\Delta_1' \vdash A \quad \Delta_2', B \vdash C}{\Delta_1', \Delta_2', A \multimap B \vdash C}\ {\multimap}L$$

The right rule says that $A{\multimap}B$ is provable if B is provable using a proof of A. Correspondingly, a process that offers service $A \multimap B$ should first input a channel offering service A and then continue the session by using this service to offer service B. Conversely, a process that uses service $A \multimap B$ should behave in a complementary way: the client must first output a new channel offering service A and then continue the session by using service B.

Based on this intuition, a first attempt at an asynchronous process assignment might be:

$$\frac{\Delta, y{:}A \vdash P :: x{:}B}{\Delta \vdash x(y).P :: x{:}A \multimap B}\ {\multimap}R? \qquad \frac{\Delta_1' \vdash Q_1 :: y{:}A \quad \Delta_2', x{:}B \vdash Q_2 :: z{:}C}{\Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)(\overline{x}\langle y\rangle \mid Q_1 \mid Q_2) :: z{:}C}\ {\multimap}L?$$

The syntax $x(y).P$ denotes a blocking input along channel x that guards process P; here y is a bound name and stands for the channel that will be received by the input. The syntax $\overline{x}\langle y\rangle$ denotes an asynchronous output of y along channel x; we often think of it as a message y somewhere in transit to x. Note that, in their premises, both typing rules reuse the session channel of type $A \multimap B$, namely x, as the channel for the session continuation at type B.

Unfortunately, there are two serious problems with this assignment. First, it leaves outputs along the channel x unordered, violating the session contract. As an example, consider the alleged typing derivation

$$\frac{\Delta_1 \vdash P_1 :: y_1{:}A_1 \quad \dfrac{\Delta_2 \vdash P_2 :: y_2{:}A_2 \quad \Delta', x{:}B \vdash Q :: z{:}C}{\Delta_2, \Delta', x{:}A_2 \multimap B \vdash (\nu y_2)(\overline{x}\langle y_2\rangle \mid P_2 \mid Q) :: z{:}C}\ {\multimap}L?}{\Delta_1, \Delta_2, \Delta', x{:}A_1 \multimap (A_2 \multimap B) \vdash (\nu y_1)\big(\overline{x}\langle y_1\rangle \mid P_1 \mid (\nu y_2)(\overline{x}\langle y_2\rangle \mid P_2 \mid Q)\big) :: z{:}C}\ {\multimap}L?$$

According to the contract imposed by the session type $x{:}A_1 \multimap (A_2 \multimap B)$, a process listening on channel x should expect to receive an A_1 and then, only later, an A_2. But, under the operational semantics of the asynchronous π-calculus, a process listening on x nondeterministically receives *either* $y_1{:}A_1$ or $y_2{:}A_2$ when composed with the above process.

Second, it is possible that the output will be misdirected to the session's continuation. Because the ${\multimap}L$ rule types the continuation process Q_2 with $\Delta'_2, x{:}B \vdash Q_2 :: z{:}C$, the process Q_2 may contain an input along channel x. Because the asynchronous output $\overline{x}\langle y\rangle$ is in parallel with Q_2, it is possible that such an input might unintentionally receive $\overline{x}\langle y\rangle$.

Fortunately, there is a single, elegant fix for both problems: rather than using the same channel for the continuation, the session should continue along a fresh channel. Thus, the left rule, ${\multimap}L$, becomes

$$\frac{\Delta, y{:}A \vdash P :: x'{:}B}{\Delta \vdash x(y,x').P :: x{:}A \multimap B}\ {\multimap}R \qquad \frac{\Delta'_1 \vdash Q_1 :: y{:}A \quad \Delta'_2, x'{:}B \vdash Q_2 :: z{:}C}{\Delta'_1, \Delta'_2, x{:}A \multimap B \vdash (\nu y)(\nu x')(\overline{x}\langle y,x'\rangle \mid Q_1 \mid Q_2) :: z{:}C}\ {\multimap}L$$

It is crucial that the ${\multimap}L$ process sends both y and x', as represented by the dyadic output $\overline{x}\langle y,x'\rangle$. If the process sent only y, then its session partner would not know where to rendezvous for the session continuation. Accordingly, the right rule is a matching dyadic input. (The names y and x' are bound with scope over P in the input process $x(y,x').P$.)

By using a fresh channel for the continuation, both problems are resolved. First, outputs are now ordered. Since the ${\multimap}L$ rule types the continuation process as $\Delta'_2, x'{:}B \vdash Q_2 :: z{:}C$, a subsequent output in Q_2 will occur along channel x', not x. Because of the name restriction $(\nu x')$, the channel x' is unknown outside of this process. Thus, no other process can be listening on x' until it receives the output $\overline{x}\langle y,x'\rangle$ and learns of the new channel x', thereby imposing an order on outputs within a given session.

Pictorially, we might represent this ordering of outputs within a session as the sequence $\overline{x}\langle y_1,x'\rangle\,,\ \overline{x'}\langle y_2,x''\rangle\,,\ \overline{x''}\langle y_3,x'''\rangle\,,\ \ldots$. Because the typing discipline ensures that channels $x', x'', x''', \ldots$ are not used elsewhere, this suggests a reading of well-typed processes as using explicit communication buffers, such as the input buffer $x\langle y_1, y_2, y_3, \ldots]$ at endpoint x. This intuition will be made precise in Section 4.

Second, the problem of misdirected outputs is also resolved by using a fresh channel for the session continuation. The ${\multimap}L$ rule does not allow the previous session channel, x, to appear in the continuation process Q_2. Therefore, Q_2 will not contain inputs along x, precluding it from ever mistakenly receiving the output $\overline{x}\langle y,x'\rangle$.

Cut reduction as communication. In the linear sequent calculus, the principal cut reduction for linear implication is a local check on the coherence of the ${\multimap}R$ and ${\multimap}L$ rules:

$$\dfrac{\dfrac{\Delta, A \vdash B}{\Delta \vdash A \multimap B}\ {\multimap}R \quad \dfrac{\Delta'_1 \vdash A \quad \Delta'_2, B \vdash C}{\Delta'_1, \Delta'_2, A \multimap B \vdash C}\ {\multimap}L}{\Delta, \Delta'_1, \Delta'_2 \vdash C}\ \mathsf{cut} \quad \longrightarrow \quad \dfrac{\dfrac{\Delta'_1 \vdash A \quad \Delta, A \vdash B}{\Delta, \Delta'_1 \vdash B}\ \mathsf{cut} \quad \Delta'_2, B \vdash C}{\Delta, \Delta'_1, \Delta'_2 \vdash C}\ \mathsf{cut}$$

Under the process interpretation, cut reduction is asynchronous session-typed communication. When annotated according to the process interpretation, the above principal cut reduction for linear implication yields the process reduction

$$(\nu x)\big(x(y,x').P \mid (\nu y)(\nu x')(\overline{x}\langle y,x'\rangle \mid Q_1 \mid Q_2)\big) \longrightarrow (\nu x')\big((\nu y)(Q_1 \mid P) \mid Q_2\big)\,.$$

Modulo the structural congruences of the π-calculus (including α-renaming of bound names), this is an instance of the standard asynchronous polyadic π-calculus process reduction that drives asynchronous communication: $x(w,z).P \mid \overline{x}\langle y,x'\rangle \longrightarrow P\{y/w, x'/z\}$. This justifies our claim that cut reduction is asynchronous session-typed communication.

It is also possible to give a computational interpretation of identity expansion, the act of reducing uses of the id rule at compound types to larger proofs that appeal to the id rule at smaller types. We do not pursue it in this paper because it is not germane to our study of cut reduction as communication and commuting conversions as the basis of asynchrony. For the details of identity expansion in the synchronous case, see [9].

2.3 Multiplicative conjunction as output

Even in the intuitionistic linear sequent calculus, there is a strong flavor of duality between linear implication and multiplicative conjunction. This duality should similarly extend to the process interpretation: just as a process of type $A \multimap B$ offers an input of an A and then behaves as B, a process of type $A \otimes B$ should offer an output of an A and then behave as B.

Thus, we arrive at the following process assignment for the usual right and left rules.

$$\frac{\Delta_1 \vdash P_1 :: y{:}A \quad \Delta_2 \vdash P_2 :: x'{:}B}{\Delta_1, \Delta_2 \vdash (\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2) :: x{:}A \otimes B} \otimes R \qquad \frac{\Delta', y{:}A, x'{:}B \vdash Q :: z{:}C}{\Delta', x{:}A \otimes B \vdash x(y, x').Q :: z{:}C} \otimes L$$

Again, notice the use of a new channel, x', for the session continuation at type B. If we tried to reuse the original session channel, x, then we would again face the problems of unordered and misdirected outputs that plagued our first, failed process assignment for implication. We can verify that the $\otimes R$ and $\otimes L$ rules fit together by checking the principal cut reduction.

Cut reduction as communication. The principal cut reduction for $A \otimes B$ is

$$\frac{\dfrac{\Delta_1 \vdash A \quad \Delta_2 \vdash B}{\Delta_1, \Delta_2 \vdash A \otimes B} \otimes R \quad \dfrac{\Delta', A, B \vdash C}{\Delta', A \otimes B \vdash C} \otimes L}{\Delta_1, \Delta_2, \Delta' \vdash C} \text{cut} \quad \longrightarrow \quad \frac{\Delta_2 \vdash B \quad \dfrac{\Delta_1 \vdash A \quad \Delta', A, B \vdash C}{\Delta_1, \Delta', B \vdash C} \text{cut}}{\Delta_1, \Delta_2, \Delta' \vdash C} \text{cut}$$

The same reduction can be carried out under the process interpretation, yielding

$$(\nu x)\big((\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2) \mid x(y, x').Q\big) \longrightarrow (\nu x')\big(P_2 \mid (\nu y)(P_1 \mid Q)\big) .$$

Once again, modulo structural congruence, this is an instance of the standard asynchronous polyadic communication rule.

2.4 Multiplicative unit as termination

Because it is the unit of $\otimes$, it is often helpful to view the proposition $\mathbf{1}$ as the nullary form of $\otimes$. For instance, the inference rules for $\mathbf{1}$ are nullary versions of the rules for $A \otimes B$. We can extend this to our process interpretation:

$$\frac{}{\cdot \vdash \overline{x}\langle\rangle :: x{:}\mathbf{1}} \mathbf{1}R \qquad \frac{\Delta' \vdash Q :: z{:}C}{\Delta', x{:}\mathbf{1} \vdash x().Q :: z{:}C} \mathbf{1}L$$

The right rule outputs an empty message and has no continuation; the left rule inputs the empty message. Because the right rule has no continuation, the empty message serves as a session termination signal: the process will not offer any further service.

As the left rule is given, a process using a terminated session must block until it receives the termination signal, because the prefix $x()$ guards the continuation Q. We can enable more parallelism by modifying the left rule:

$$\frac{\Delta' \vdash Q :: z{:}C}{\Delta', x{:}\mathbf{1} \vdash x().\mathbf{0} \mid Q :: z{:}C} \mathbf{1}L$$

The left rule's continuation, Q, can now run in parallel while waiting to receive the termination signal. The $\mathbf{1}L$ rule is no longer an exact nullary version of the $\otimes L$ rule, but the process assignment is still in bijective correspondence with the proof rules.

Cut reduction as communication. The principal cut reduction at type $\mathbf{1}$ is

$$\frac{\dfrac{}{\cdot \vdash \mathbf{1}} \mathbf{1}R \quad \dfrac{\Delta' \vdash C}{\Delta', \mathbf{1} \vdash C} \mathbf{1}L}{\Delta' \vdash C} \text{cut} \quad \longrightarrow \quad \Delta' \vdash C$$

When annotated according to the process interpretation, we can extract the process reduction $(\nu x)\big(\overline{x}\langle\rangle \mid (x().\mathbf{0} \mid Q)\big) \longrightarrow Q$, which, modulo structural congruences, is an instance of the asynchronous π-calculus reduction that matches a nullary output with a nullary input.

2.5 Additive conjunction and disjunction as choice

Processes should be able to offer the client a choice between two services, A and B. The client selects one of the services and then uses the selected service. This type of external choice corresponds to the additive conjunction $A \mathbin{\&} B$. We can assign processes to the rules:

$$\frac{\Delta \vdash P_1 :: x_1'{:}A \quad \Delta \vdash P_2 :: x_2'{:}B}{\Delta \vdash x.\mathsf{case}((x_1').P_1, (x_2').P_2) :: x{:}A \mathbin{\&} B}\ \&R$$

$$\frac{\Delta', x_1'{:}A \vdash Q :: z{:}C}{\Delta', x{:}A \mathbin{\&} B \vdash (\nu x_1')(x.\mathsf{inl}\langle x_1'\rangle \mid Q) :: z{:}C}\ \&L_1 \qquad \frac{\Delta', x_2'{:}B \vdash Q :: z{:}C}{\Delta', x{:}A \mathbin{\&} B \vdash (\nu x_2')(x.\mathsf{inr}\langle x_2'\rangle \mid Q) :: z{:}C}\ \&L_2$$

In the left rules, the client asynchronously sends his selection (either inl or inr) and a new channel at which the session should continue. In the right rule, the server must be prepared for either selection; it behaves like a case, waiting for a client's selection and then continuing accordingly.

The principal cut reductions and process reductions match: the process reductions are

$$(\nu x)\big(x.\mathsf{case}((x_1').P_1, (x_2').P_2) \mid (\nu x_1')(x.\mathsf{inl}\langle x_1'\rangle \mid Q)\big) \longrightarrow (\nu x_1')(P_1 \mid Q)$$
$$(\nu x)\big(x.\mathsf{case}((x_1').P_1, (x_2').P_2) \mid (\nu x_2')(x.\mathsf{inr}\langle x_2'\rangle \mid Q)\big) \longrightarrow (\nu x_2')(P_2 \mid Q)\,.$$

External choice is dual to internal choice, where a process offers one of two possible services with the choice at its own discretion. Because $A \& B$ is dual to the additive disjunction $A \oplus B$ in linear logic, $A \oplus B$ should be interpreted as internal choice and uses the same process constructs in a dual manner. Due to space constraints, we omit the details here.

2.6 Exponential as persistent service

Thus far, we have focused on the purely linear fragment of intuitionistic linear logic, but we can also give an asynchronous process interpretation of the 'of course!' exponential. In the judgmental formulation of intuitionistic linear logic, the reusable antecedents provided by the 'of course!' exponential are expressed with a new judgment, A valid, that is subject to weakening, contraction, and exchange. To streamline notation, validity antecedents are usually written in a separate zone of the sequent, as in dual intuitionistic linear logic [3, 10]. With the process annotations added, the sequent becomes

$$u_1{:}B_1, \ldots, u_m{:}B_m; x_1{:}A_1, \ldots, x_n{:}A_n \vdash P :: x{:}A\,,$$

where $u_1{:}B_1, \ldots, u_m{:}B_m$ are the reusable, validity antecedents. We use the metavariable Γ to stand for an arbitrary context of validity antecedents. To match the new form for sequents, all of the previously presented inference rules are extended to include a context Γ in the conclusion and all premises.

2.6.1 Judgmental principles

A proposition A is valid if, and only if, A is true without linear antecedents. There are two new judgmental rules: a cut principle for validity and a rule relating validity to linear truth.

$$\frac{\Gamma; \cdot \vdash A \quad \Gamma, A; \Delta' \vdash C}{\Gamma; \Delta' \vdash C}\ \mathsf{cut!} \qquad \frac{\Gamma, A; \Delta', A \vdash C}{\Gamma, A; \Delta' \vdash C}\ \mathsf{copy}$$

What process interpretation should we give to validity and its cut! and copy rules? Because validity antecedents persist throughout a proof we will interpret $u{:}A$ as a server that persistently provides service A. Specifically:

$$\frac{\Gamma; \cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta' \vdash (\nu u)(!u(y).P \mid Q) :: z{:}C}\ \textsf{cut!} \qquad \frac{\Gamma, u{:}A; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma, u{:}A; \Delta' \vdash (\nu x)(\overline{u}\langle x\rangle \mid Q) :: z{:}C}\ \textsf{copy}$$

The cut! rule shows that a persistent offer of service A is made by the replicated input $!u(y).P$. According to the copy rule, the client process, Q, can obtain service A by asynchronously sending the server, u, a new channel; the server spawns a copy of service A at that channel, and the server continues to be available for future requests.

Note that, in contrast with all previous rules, the copy rule's premise does *not* use a renamed persistent channel in the continuation. From an operational perspective, this is because persistent servers do not directly participate in long-lived sessions with clients. Instead, they just receive individual messages from various clients and spawn linear sessions to do the real work. Alternatively, from a proof-theoretic perspective, this is because there is a commuting conversion between *any* adjacent copy inferences.

Cut reduction as communication. The server's act of spawning a copy of service A is reflected in the process interpretation of the cut reduction that arises when cut! meets copy. When processes annotate the cut reduction

$$\frac{\Gamma; \cdot \vdash A \quad \dfrac{\Gamma, A; \Delta', A \vdash C}{\Gamma, A; \Delta' \vdash C}\ \textsf{copy}}{\Gamma; \Delta' \vdash C}\ \textsf{cut!} \quad \longrightarrow \quad \frac{\Gamma; \cdot \vdash A \quad \dfrac{\Gamma; \cdot \vdash A \quad \Gamma, A; \Delta', A \vdash C}{\Gamma; \Delta', A \vdash C}\ \textsf{cut!}}{\Gamma; \Delta' \vdash C}\ \textsf{cut}$$

we obtain the following process reduction, which matches an output with a replicated input.

$$(\nu u)\big(!u(y).P \mid (\nu x)(\overline{u}\langle x\rangle \mid Q)\big) \longrightarrow (\nu x)\big(P\{x/y\} \mid (\nu u)(!u(y).P \mid Q)\big)$$

2.6.2 Right and left rules

In a judgmental formulation of the linear sequent calculus, the right and left rules for the 'of course!' connective, written $!A$, are:

$$\frac{\Gamma; \cdot \vdash A}{\Gamma; \cdot \vdash !A}\ !R \qquad \frac{\Gamma, A; \Delta' \vdash C}{\Gamma; \Delta', !A \vdash C}\ !L$$

How does $!A$ relate to validity as persistent service? Essentially, we will interpret a service $!A$ as one that creates a persistent server that offers A. The process assignment that we use is

$$\frac{\Gamma; \cdot \vdash P :: y{:}A}{\Gamma; \cdot \vdash (\nu u)(\overline{x}\langle u\rangle \mid !u(y).P) :: x{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}!A \vdash x(u).Q :: z{:}C}\ !L$$

The right rule says that a process offering $!A$ along x first chooses a new, persistent name, u, for itself and registers that channel by sending it to its session partner. The process then persistently provides service A by offering a replicated input at u. Conversely, the left rule says that a process that uses $!A$ must input a persistent channel, u, and may thereafter treat u as the name of a persistent server offering A.

Our process interpretation of the $!R$ and $!L$ rules departs significantly from the prior synchronous interpretations [8, 9], in ways that are orthogonal to asynchrony. Previously, the $!R$ rule was interpreted as a replicated input along a linear channel and the $!L$ rule was interpreted as either an implicit [8] or explicit [9] substitution. We contend that our process assignment is proof-theoretically more pleasing: now, *all* of the non-invertible right and left rules [2] ($\multimap L$, $\otimes R$, $\& L_i$, $\oplus R_i$, and $!R$) have an output flavor, and *all* of the invertible right and left rules ($\multimap R$, $\otimes L$, $\& R$, $\oplus L$, and $!L$) have an input flavor.

Cut reduction as communication. Once again, the principal cut reduction corresponds to a process reduction. The principal cut reduction at type $!A$ is

$$\dfrac{\dfrac{\Gamma; \cdot \vdash A}{\Gamma; \cdot \vdash !A}\,!R \qquad \dfrac{\Gamma, A; \Delta' \vdash C}{\Gamma; \Delta', !A \vdash C}\,!L}{\Gamma; \Delta' \vdash C}\,\mathsf{cut} \quad \longrightarrow \quad \dfrac{\Gamma; \cdot \vdash A \quad \Gamma, A; \Delta' \vdash C}{\Gamma; \Delta' \vdash C}\,\mathsf{cut!}$$

When this cut reduction is annotated with processes, we obtain the process reduction

$$(\nu x)\big((\nu u)(\overline{x}\langle u\rangle \mid !u(y).P) \mid x(u).Q\big) \longrightarrow (\nu u)(!u(y).P \mid Q)\,.$$

Thus, cut reduction at $!A$ corresponds to registering a server's persistent name with its client.

3 Relationship between synchronous and asynchronous process interpretations

Prior work has shown that the intuitionistic linear sequent calculus can be seen as a session-type discipline for the *synchronous* π-calculus [8, 9]. In the previous section, we presented a new process assignment from the *asynchronous*, polyadic π-calculus to exactly the same proof rules. This section serves to make precise the claim that, proof-theoretically, the difference between these interpretations lies in the commuting conversions that are permitted.

3.1 A synchronous, polyadic process interpretation

The first step in making our claim precise is to reconsider the synchronous process interpretation from [8, 9]. There, all outputs were represented as prefixes guarding a continuation process, as is standard in the synchronous π-calculus. For example, the assignment for the $\otimes R$ rule was a synchronous monadic output and the $\otimes L$ rule was a monadic input:

$$\dfrac{\Gamma; \Delta_1 \vdash P_1 :: y{:}A \quad \Gamma; \Delta_2 \vdash P_2 :: x{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu y)\overline{x}\langle y\rangle.(P_1 \mid P_2) :: x{:}A \otimes B}\,\otimes R \qquad \dfrac{\Gamma; \Delta', y{:}A, x{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \otimes B \vdash x(y).Q :: z{:}C}\,\otimes L$$

These processes reuse the session channel x as the channel for the session's continuation. There is no possibility of unordered or misdirected outputs here because P_1 and P_2 may not execute until the output guard, $\overline{x}\langle y\rangle$, synchronizes with another input process, $x(y).Q$.

Instead of relying on this implicit convention, we could modify the synchronous process assignment to be explicit about reusing the session channel. The $\otimes R$ and $\otimes L$ rules would thus become dyadic outputs and inputs, respectively, with the output explicitly transmitting x as a channel to be used for the session continuation. But, in fact, once dyadic outputs and inputs are used, there is no technical advantage to reusing the session channel, and we may as well use a fresh channel for the session continuation. Figure 1 presents a polyadic synchronous process interpretation in this style.

There is a very strong operational equivalence between this polyadic interpretation and the monadic synchronous interpretation from [9] (there is also a correspondence with [8], if we match their rules for $\mathbf{1}$). For example, consider the principal cut reduction at type $A \otimes B$. Under the monadic synchronous process assignment, it corresponds to

$$(\nu x)\big((\nu y)\overline{x}\langle y\rangle.(P_1 \mid P_2) \mid x(y).Q\big) \longrightarrow (\nu x)\big(P_2 \mid (\nu y)(P_1 \mid Q)\big)\,.$$

$$\frac{}{\Gamma; y{:}A \vdash [y \leftrightarrow x] :: x{:}A}\ \mathsf{id} \qquad \frac{\Gamma; \Delta \vdash P :: x{:}A \quad \Gamma; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma; \Delta, \Delta' \vdash (\nu x)(P \mid Q) :: z{:}C}\ \mathsf{cut}$$

$$\frac{\Gamma; \cdot \vdash P :: y{:}A \quad \Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta' \vdash (\nu u)(!u(y).P \mid Q) :: z{:}C}\ \mathsf{cut!} \qquad \frac{\Gamma, u{:}A; \Delta', x{:}A \vdash Q :: z{:}C}{\Gamma, u{:}A; \Delta' \vdash (\nu x)\overline{u}\langle x \rangle.Q :: z{:}C}\ \mathsf{copy}$$

$$\frac{\Gamma; \Delta, y{:}A \vdash P :: x'{:}B}{\Gamma; \Delta \vdash x(y, x').P :: x{:}A \multimap B}\ {\multimap}R \qquad \frac{\Gamma; \Delta_1' \vdash Q_1 :: y{:}A \quad \Gamma; \Delta_2', x'{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)(\nu x')\overline{x}\langle y, x' \rangle.(Q_1 \mid Q_2) :: z{:}C}\ {\multimap}L$$

$$\frac{\Gamma; \Delta_1 \vdash P_1 :: y{:}A \quad \Gamma; \Delta_2 \vdash P_2 :: x'{:}B}{\Gamma; \Delta_1, \Delta_2 \vdash (\nu y)(\nu x')\overline{x}\langle y, x' \rangle.(P_1 \mid P_2) :: x{:}A \otimes B}\ {\otimes}R \qquad \frac{\Gamma; \Delta', y{:}A, x'{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \otimes B \vdash x(y, x').Q :: z{:}C}\ {\otimes}L$$

$$\frac{}{\cdot \vdash \overline{x}\langle\rangle.\mathbf{0} :: x{:}\mathbf{1}}\ \mathbf{1}R \qquad \frac{\Gamma; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}\mathbf{1} \vdash x().Q :: z{:}C}\ \mathbf{1}L$$

$$\frac{\Gamma; \Delta \vdash P_1 :: x_1'{:}A \quad \Gamma; \Delta \vdash P_2 :: x_2'{:}B}{\Gamma; \Delta \vdash x.\mathsf{case}((x_1').P_1, (x_2').P_2) :: x{:}A \mathbin{\&} B}\ {\&}R$$

$$\frac{\Gamma; \Delta', x_1'{:}A \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \mathbin{\&} B \vdash x.\mathsf{inl}\langle x_1' \rangle; Q :: z{:}C}\ {\&}L_1 \qquad \frac{\Gamma; \Delta', x_2'{:}B \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}A \mathbin{\&} B \vdash x.\mathsf{inr}\langle x_2' \rangle; Q :: z{:}C}\ {\&}L_2$$

$$\frac{\Gamma; \Delta \vdash P :: x_1'{:}A}{\Gamma; \Delta \vdash x.\mathsf{inl}\langle x_1' \rangle; P :: x{:}A \oplus B}\ {\oplus}R_1 \qquad \frac{\Gamma; \Delta \vdash P :: x_2'{:}B}{\Gamma; \Delta \vdash x.\mathsf{inr}\langle x_2' \rangle; P :: x{:}A \oplus B}\ {\oplus}R_2$$

$$\frac{\Gamma; \Delta', x_1'{:}A \vdash Q_1 :: z{:}C \quad \Gamma; \Delta', x_2'{:}B \vdash Q_2 :: z{:}C}{\Gamma; \Delta', x{:}A \oplus B \vdash x.\mathsf{case}((x_1').Q_1, (x_2').Q_2) :: z{:}C}\ {\oplus}L$$

$$\frac{\Gamma; \cdot \vdash P :: y{:}A}{\Gamma; \cdot \vdash (\nu u)\overline{x}\langle u \rangle.!u(y).P :: x{:}!A}\ !R \qquad \frac{\Gamma, u{:}A; \Delta' \vdash Q :: z{:}C}{\Gamma; \Delta', x{:}!A \vdash x(u).Q :: z{:}C}\ !L$$

Figure 1 A polyadic synchronous process assignment that is equivalent to one from [9].

Under the polyadic synchronous process assignment, the same cut reduction corresponds to

$$\begin{aligned}&(\nu x)\big((\nu y)(\nu x')\overline{x}\langle y, x' \rangle.(P_1 \mid P_2\{x'/x\}) \mid x(y, x').Q\{x'/x\}\big)\\&\quad \longrightarrow (\nu x')\big(P_2\{x'/x\} \mid (\nu y)(P_1 \mid Q\{x'/x\})\big)\,.\end{aligned}$$

Typing guarantees that x' is not free in P_1. Thus, by α-renaming x' to x, we obtain the same process after reduction as in the monadic synchronous assignment. In this way, there is a tight operational correspondence between the monadic and polyadic synchronous assignments.

3.2 Commuting conversions as process equivalences

We can now turn to relating the interpretation of commuting conversions under the asynchronous and synchronous process assignments. Having shown that the synchronous polyadic process assignment of Figure 1 is equivalent to the synchronous monadic assignment of [9], we can compare our asynchronous assignment from Section 2 with the synchronous polyadic assignment and know that the comparison extends to the synchronous monadic assignment.

In proof theory, commuting conversions describe structural equivalences among proofs. The following is an example of one commuting conversion between two adjacent cut inferences.

$$\frac{\dfrac{\Gamma; \Delta_1 \vdash B \quad \Gamma; \Delta_2, B \vdash A}{\Gamma; \Delta_1, \Delta_2 \vdash A}\ \mathsf{cut} \qquad \Gamma; \Delta', A \vdash C}{\Gamma; \Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut} \quad \equiv \quad \frac{\Gamma; \Delta_1 \vdash B \qquad \dfrac{\Gamma; \Delta_2, B \vdash A \quad \Gamma; \Delta', A \vdash C}{\Gamma; \Delta_2, \Delta', B \vdash C}\ \mathsf{cut}}{\Gamma; \Delta_1, \Delta_2, \Delta' \vdash C}\ \mathsf{cut}$$

To what do such commuting conversions correspond under the synchronous and asynchronous process assignments? The commuting conversions can be sorted into three classes.

Class 1. Some commuting conversions correspond to structural equivalences under *both* the synchronous and asynchronous assignments. For example, both assignments interpret cut with the same process. Thus, if we annotate the above $\mathsf{cut}/\mathsf{cut}$ conversion accordingly, it can be read as the following structural equivalence on processes:

$$(\nu x)\big((\nu y)(P_1 \mid P_2) \mid Q\big) \equiv (\nu y)\big(P_1 \mid (\nu x)(P_2 \mid Q)\big)\,, \text{ if } x \notin \mathsf{fn}(P_1) \text{ and } y \notin \mathsf{fn}(Q).$$

This equivalence is derivable from more basic laws of the (synchronous and asynchronous) π-calculus structural congruence, such as associativity and commutativity of parallel composition and scope extrusion of name restrictions. (The side condition on the free names, denoted $\mathsf{fn}(-)$, is only necessary in the untyped π-calculus; here it is guaranteed by typing.)

Class 2. Most commuting conversions do *not* yield structural process equivalences under the synchronous interpretation. For example, one conversion between $\otimes R$ and $\multimap L$ is:

$$\dfrac{\Gamma;\Delta \vdash A_1 \qquad \dfrac{\Gamma;\Delta_1' \vdash B_1 \quad \Gamma;\Delta_2', B_2 \vdash A_2}{\Gamma;\Delta_1',\Delta_2', B_1 \multimap B_2 \vdash A_2}\multimap L}{\Gamma;\Delta,\Delta_1',\Delta_2', B_1 \multimap B_2 \vdash A_1 \otimes A_2}\otimes R \;\equiv\; \dfrac{\Gamma;\Delta_1' \vdash B_1 \qquad \dfrac{\Gamma;\Delta \vdash A_1 \quad \Gamma;\Delta_2', B_2 \vdash A_2}{\Gamma;\Delta,\Delta_2', B_2 \vdash A_1 \otimes A_2}\otimes R}{\Gamma;\Delta,\Delta_1',\Delta_2', B_1 \multimap B_2 \vdash A_1 \otimes A_2}\multimap L$$

Under the synchronous interpretation, this commuting conversion does not correspond to a structural process equivalence of the synchronous polyadic π-calculus because it reorders two blocking outputs:

$$\begin{aligned}&(\nu w)(\nu z')\overline{z}\langle w, z'\rangle.\big(P \mid (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.(Q_1 \mid Q_2)\big)\\ &\quad\not\equiv (\nu y)(\nu x')\overline{x}\langle y, x'\rangle.\big(Q_1 \mid (\nu w)(\nu z')\overline{z}\langle w, z'\rangle.(P \mid Q_2)\big)\,.\end{aligned}$$

However, when composed with closed processes as required by the Γ and $\Delta, \Delta_1', \Delta_2', x{:}B_1 \multimap B_2$ contexts, these two processes are *observationally* equivalent, according to (a simple dyadic extension of) typed context bisimilarity as defined by Pérez et al. [20]. Essentially, the reason is this: When composed with the required processes, only the actions along $z{:}A_1 \otimes A_2$, namely $(\nu w)(\nu z')\overline{z}\langle w, z'\rangle$, are observable—all other interactions, such as $(\nu y)(\nu x')\overline{x}\langle y, x'\rangle$, are internal to the closed process—and so the reordering cannot be detected.

On the other hand, under our asynchronous process interpretation, this $\otimes R/\multimap L$ commuting conversion can be read as the structural process equivalence

$$\begin{aligned}&(\nu w)(\nu z')\big(\overline{z}\langle w, z'\rangle \mid P \mid (\nu y)(\nu x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid Q_2)\big)\\ &\equiv (\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid Q_1 \mid (\nu w)(\nu z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2)\big), \text{ if } w, z' \notin \mathsf{fn}(Q_1) \text{ and } y, x' \notin \mathsf{fn}(P).\end{aligned}$$

Once again, this equivalence is derivable from laws of structural congruence that are standard in the asynchronous π-calculus.

We contend that our asynchronous interpretation is therefore proof-theoretically more pleasing: it maps certain structural equivalences of proofs to standard *structural* equivalences of processes, whereas the synchronous interpretation only mapped these proof equivalences to strictly weaker observational equivalences on processes.

Class 3. Another class of commuting conversions are those that involve rules to which input-flavored processes are assigned. These conversions do not correspond to structural process equivalences under *either* the synchronous or asynchronous assignments. For example, one such conversion is between $\otimes R$ and $\otimes L$:

$$\dfrac{\Gamma;\Delta_1 \vdash A_1 \quad \dfrac{\Gamma;\Delta_2,B_1,B_2 \vdash A_2}{\Gamma;\Delta_2,B_1\otimes B_2 \vdash A_2}\otimes L}{\Gamma;\Delta_1,\Delta_2,B_1\otimes B_2 \vdash A_1\otimes A_2}\otimes R \equiv \dfrac{\dfrac{\Gamma;\Delta_1\vdash A_1 \quad \Gamma;\Delta_2,B_1,B_2\vdash A_2}{\Gamma;\Delta_1,\Delta_2,B_1,B_2\vdash A_1\otimes A_2}\otimes R}{\Gamma;\Delta_1,\Delta_2,B_1\otimes B_2\vdash A_1\otimes A_2}\otimes L$$

Under the synchronous assignment, this conversion is only an observational equivalence:

$$(\nu w)(\nu z')\overline{z}\langle w,z'\rangle.\big(P \mid x(y,x').Q\big) \approx x(y,x').\big((\nu w)(\nu z')\overline{z}\langle w,z'\rangle.(P\mid Q)\big)\,.$$

The justification is similar to the previous one: When composed with closed processes required by Γ and $\Delta_1,\Delta_2,B_1\otimes B_2$, only the actions along $z{:}A_1\otimes A_2$, namely $(\nu w)(\nu z')\overline{z}\langle w,z'\rangle$, are observable because all other interactions, such as $x(y,x')$, are internal to the closed process.

Similarly, under the asynchronous process assignment, this commuting conversion does *not* yield a structural equivalence because permuting the input outward blocks actions in P:

$$(\nu w)(\nu z')\big(\overline{z}\langle w,z'\rangle \mid P \mid x(y,x').Q\big) \not\equiv x(y,x').(\nu w)(\nu z')(\overline{z}\langle w,z'\rangle \mid P \mid Q)\,.$$

This exposes a fundamental asymmetry between outputs, which can be interpreted asynchronously and give rise to very natural structural commutation laws, and inputs, which, in a process calculus, are inherently points of synchronization and cannot obey *structural* commutation laws since that would defeat the purpose of synchronization points.

With the above intuition for the three classes of conversions, we obtain the following.

▶ **Theorem 1.** *For each commuting conversion listed in Figure 2, its asynchronous process interpretation is a standard structural equivalence.*

4 Correspondence with an asynchronous buffered session semantics

In Section 2, we presented a novel Curry-Howard interpretation of intuitionistic linear logic as an asynchronous session-type system. Asynchronous outputs were represented abstractly as free-floating messages waiting to be received by an input process. However, in keeping with practical implementations of asynchronous communication, existing asynchronous session-type systems use explicitly buffered communication channels [12, 11, 18]. To relate our Curry-Howard interpretation to existing asynchronous session-type systems, we now show that the use of fresh channels for session continuations serves as an encoding of FIFO buffers. First, we must present a π-calculus with explicit two-sided FIFO buffers.

4.1 A π-calculus with explicit two-sided FIFO buffers

Syntax and structural congruence. Syntactically, this calculus extends the (synchronous) π-calculus with a FIFO buffer (i.e., queue) construct, $x[m_k,\dots,m_1\rangle z$. It represents an input buffer at endpoint z that holds the sequence $m_1,\dots,m_k$ of messages that have been sent by the peer endpoint x. A message m has one of several forms: a linear channel, y; a termination signal, fin; left and right selectors, inl and inr; or registration of a persistent channel, $!u$. We assume that a message sent by endpoint x immediately arrives at the tail of its peer endpoint's input buffer. It will also be useful to adopt $z\langle m_1,\dots,m_k]x$ as alternate notation for the queue $x[m_k,\dots,m_1\rangle z$.

In addition to the usual basic laws of π-calculus structural congruence, we include the equivalence $x[\rangle z \equiv x\langle]z$. This expresses that an empty queue remains uncommitted to its direction—either endpoint may place a message onto the empty queue.

Class 2:

$(\mathsf{cut}/{\multimap}L/-), ({\multimap}L/-/\mathsf{cut}_1)$	$(\nu x)((\nu w, y')(\overline{y}\langle w, y'\rangle \mid P_1 \mid P_2) \mid Q)$ $\quad \equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid P_1 \mid (\nu x)(P_2 \mid Q))$
$(\mathsf{cut}/\mathbf{1}L/-), (\mathbf{1}L/\mathsf{cut}_1)$	$(\nu x)((y().\mathbf{0} \mid P) \mid Q) \equiv y().\mathbf{0} \mid (\nu x)(P \mid Q)$
$(\mathsf{cut}/\&L_i/-), (\&L_i/\mathsf{cut}_1)$	$(\nu x)((\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid P) \mid Q) \equiv (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/\mathsf{copy}/-), (\mathsf{copy}/\mathsf{cut}_1)$	$(\nu x)((\nu y)(\overline{u}\langle y\rangle \mid P) \mid Q) \equiv (\nu y)(\overline{u}\langle y\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/-/{\multimap}L_1), (\mathsf{cut}/-/\otimes R_1),$ $({\multimap}L/\mathsf{cut}/-), (\otimes R/\mathsf{cut}/-)$	$(\nu x)(P \mid (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid Q_2))$ $\quad \equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid (\nu x)(P \mid Q_1) \mid Q_2)$
$(\mathsf{cut}/-/{\multimap}L_2), (\mathsf{cut}/-/\otimes R_2),$ $({\multimap}L/-/\mathsf{cut}_2), (\otimes R/-/\mathsf{cut})$	$(\nu x)(P \mid (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid Q_2))$ $\quad \equiv (\nu w, y')(\overline{y}\langle w, y'\rangle \mid Q_1 \mid (\nu x)(P \mid Q_2))$
$(\mathsf{cut}/-/\mathbf{1}L), (\mathbf{1}L/\mathsf{cut}_2)$	$(\nu x)(P \mid (y().\mathbf{0} \mid Q)) \equiv y().\mathbf{0} \mid (\nu x)(P \mid Q)$
$(\mathsf{cut}/-/\&L_i), (\mathsf{cut}/-/\oplus R_i),$ $(\&L_i/\mathsf{cut}_2), (\oplus R_i/\mathsf{cut}_2)$	$(\nu x)(P \mid (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid Q)) \equiv (\nu y')(y.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle y'\rangle \mid (\nu x)(P \mid Q))$
$(\mathsf{cut}/-/\mathsf{copy}), (\mathsf{copy}/\mathsf{cut}_2)$	$(\nu x)(P \mid (\nu y)(\overline{u}\langle y\rangle \mid Q)) \equiv (\nu y)(\overline{u}\langle y\rangle \mid (\nu x)(P \mid Q))$
$({\multimap}L/\mathsf{cut!}/-), (\otimes R/\mathsf{cut!}/-)$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid (\nu u)(!u(y).P_1 \mid P_2) \mid Q)$ $\quad \equiv (\nu u)(!u(y).P_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_2 \mid Q))$
$({\multimap}L/-/\mathsf{cut!}), (\otimes R/-/\mathsf{cut!})$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid (\nu u)(!u(y).Q_1 \mid Q_2))$ $\quad \equiv (\nu u)(!u(y).Q_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2))$
$(\mathsf{cut!}/-/\mathbf{1}L), (\mathbf{1}L/\mathsf{cut!})$	$(\nu u)(!u(y).P \mid (x().\mathbf{0} \mid Q)) \equiv x().\mathbf{0} \mid (\nu u)(!u(y).P \mid Q)$
$(\mathsf{cut!}/-/\&L_i), (\mathsf{cut!}/-/\oplus R_i),$ $(\&L_i/\mathsf{cut!}), (\oplus R_i/\mathsf{cut!})$	$(\nu u)(!u(y).P \mid (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid Q))$ $\quad \equiv (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid (\nu u)(!u(y).P \mid Q))$
$(\mathsf{cut!}/-/\mathsf{copy}), (\mathsf{copy}/\mathsf{cut!})$	$(\nu u)(!u(y).P \mid (\nu x)(\overline{v}\langle x\rangle \mid Q)) \equiv (\nu x)(\overline{v}\langle x\rangle \mid (\nu u)(!u(y).P \mid Q))$
$({\multimap}L/{\multimap}L/-), ({\multimap}L/-/{\multimap}L_1),$ $({\multimap}L/-/\otimes R_1), (\otimes R/{\multimap}L/-)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_1 \mid P_2) \mid Q)$ $\quad \equiv (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P_1 \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_2 \mid Q))$
$({\multimap}L/\mathbf{1}L/-), (\otimes R/\mathbf{1}L/-),$ $(\mathbf{1}L/{\multimap}L_1), (\mathbf{1}L/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (z().\mathbf{0} \mid P) \mid Q) \equiv z().\mathbf{0} \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q)$
$({\multimap}L/\&L_i/-), (\otimes R/\&L_i/-),$ $(\&L_i/{\multimap}L_1), (\&L_i/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P) \mid Q)$ $\quad \equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$({\multimap}L/\mathsf{copy}/-), (\otimes R/\mathsf{copy}/-),$ $(\mathsf{copy}/{\multimap}L_1), (\mathsf{copy}/\otimes R_1)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid (\nu z)(\overline{u}\langle z\rangle \mid P) \mid Q)$ $\quad \equiv (\nu z)(\overline{u}\langle z\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$({\multimap}L/-/{\multimap}L_2), ({\multimap}L/-/\otimes R_2),$ $(\otimes R/-/{\multimap}L)$	$(\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid Q_2))$ $\quad \equiv (\nu y, x')(\overline{x}\langle y, x'\rangle \mid Q_1 \mid (\nu w, z')(\overline{z}\langle w, z'\rangle \mid P \mid Q_2))$
$({\multimap}L/-/\mathbf{1}L), (\otimes R/-/\mathbf{1}L),$ $(\mathbf{1}L/{\multimap}L_2), (\mathbf{1}L/\otimes R_2)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (z().\mathbf{0} \mid Q)) \equiv z().\mathbf{0} \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q)$
$({\multimap}L/-/\&L_i), ({\multimap}L/-/\oplus R_i),$ $(\otimes R/-/\&L_i), (\&L_i/{\multimap}L_2),$ $(\&L_i/\otimes R_2), (\oplus R_i/{\multimap}L)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid Q))$ $\quad \equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$({\multimap}L/-/\mathsf{copy}), (\otimes R/-/\mathsf{copy}),$ $(\mathsf{copy}/{\multimap}L_2), (\mathsf{copy}/\otimes R_2)$	$(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid (\nu z)(\overline{u}\langle z\rangle \mid Q))$ $\quad \equiv (\nu z)(\overline{u}\langle z\rangle \mid (\nu y, x')(\overline{x}\langle y, x'\rangle \mid P \mid Q))$
$(\mathbf{1}L/\mathbf{1}L)$	$x().\mathbf{0} \mid (z().\mathbf{0} \mid P) \equiv z().\mathbf{0} \mid (x().\mathbf{0} \mid P)$
$(\mathbf{1}L/\&L_i), (\mathbf{1}L/\oplus R_i),$ $(\&L_i/\mathbf{1}L), (\oplus R_i/\mathbf{1}L)$	$x().\mathbf{0} \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P) \equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (x().\mathbf{0} \mid P))$
$(\mathbf{1}L/\mathsf{copy}), (\mathsf{copy}/\mathbf{1}L)$	$x().\mathbf{0} \mid (\nu z)(\overline{u}\langle z\rangle \mid P) \equiv (\nu z)(\overline{u}\langle z\rangle \mid (x().\mathbf{0} \mid P))$
$(\&L_i/\&L_j), (\&L_i/\oplus R_j),$ $(\oplus R_i/\&L_j)$	$(\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid P))$ $\quad \equiv (\nu z')(z.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle z'\rangle \mid (\nu x')(x.\mathsf{in}[\mathsf{l}/\mathsf{r}]\langle x'\rangle \mid P))$
$(\mathsf{copy}/\mathsf{copy})$	$(\nu x)(\overline{u}\langle x\rangle \mid (\nu z)(\overline{v}\langle z\rangle \mid P)) \equiv (\nu z)(\overline{v}\langle z\rangle \mid (\nu x)(\overline{u}\langle x\rangle \mid P))$

Figure 2 Asynchronous structural equivalences that arise from commuting conversions.

Syntax and structural congruence:

$$\begin{aligned} m, n &::= y \mid \mathsf{fin} \mid \mathsf{inl} \mid \mathsf{inr} \mid !u \\ P, Q &::= (\nu x)P \mid (P \mid Q) \mid \mathbf{0} \mid \overline{x}\langle y\rangle.P \mid x(y).P \mid \overline{x}\langle\rangle.\mathbf{0} \mid x().P \mid x.\mathsf{inl}; P \mid x.\mathsf{inr}; P \mid x.\mathsf{case}(P, Q) \\ &\quad \mid \overline{x}\langle u\rangle.P \mid x(u).P \mid \overline{u}\langle x\rangle \mid !u(y).P \mid x[\vec{m}\rangle z \end{aligned}$$

All π-calculus laws of structural congruence, plus $x[\rangle z \equiv x\langle] z$.

Untyped reductions:

$$\begin{array}{lll} \overline{x}\langle y\rangle.P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[y, \vec{m}\rangle z & (\text{S-}\textsc{ch}) & x[\vec{m}, y\rangle z \mid z(w).Q \longrightarrow x[\vec{m}\rangle z \mid Q\{y/w\} & (\text{R-}\textsc{ch}) \\ \overline{x}\langle\rangle.\mathbf{0} \mid x[\vec{m}\rangle z \longrightarrow x[\mathsf{fin}, \vec{m}\rangle z & (\text{S-}\textsc{fin}) & x[\mathsf{fin}\rangle z \mid z().Q \longrightarrow Q & (\text{R-}\textsc{fin}) \\ x.\mathsf{inl}; P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[\mathsf{inl}, \vec{m}\rangle z & (\text{S-}\textsc{inl}) & x[\vec{m}, \mathsf{inl}\rangle z \mid z.\mathsf{case}(Q_1, Q_2) \longrightarrow x[\vec{m}\rangle z \mid Q_1 & (\text{R-}\textsc{inl}) \\ x.\mathsf{inr}; P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[\mathsf{inr}, \vec{m}\rangle z & (\text{S-}\textsc{inr}) & x[\vec{m}, \mathsf{inr}\rangle z \mid z.\mathsf{case}(Q_1, Q_2) \longrightarrow x[\vec{m}\rangle z \mid Q_2 & (\text{R-}\textsc{inr}) \\ \overline{x}\langle u\rangle.P \mid x[\vec{m}\rangle z \longrightarrow P \mid x[!u, \vec{m}\rangle z & (\text{S-!}\textsc{ch}) & x[!u\rangle z \mid z(v).Q \longrightarrow Q\{u/v\} & (\text{R-!}\textsc{ch}) \\ & & !u(y).P \mid \overline{u}\langle x\rangle.Q \longrightarrow !u(y).P \mid P\{x/y\} \mid Q & (\textsc{Rep}) \end{array}$$

Notational definitions:

$$\begin{array}{ll} (\nu z)(P \mid z[\rangle x) \triangleq P\{x/z\} & \overline{x}\langle y\rangle.P \triangleq (\nu x')(\overline{x}\langle y, x'\rangle \mid P\{x'/x\}) \\ (\nu z)\big(P_2 \mid (\nu y)(P_1 \mid z[\vec{m}, y\rangle x)\big) & x(y).P \triangleq x(y, x').P\{x'/x\} \\ \quad \triangleq (\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid P_1 \mid (\nu z)(P_2 \mid z[\vec{m}\rangle x')\big) & \\ (\nu z)(z[\mathsf{fin}\rangle x) \triangleq \overline{x}\langle\rangle & \overline{x}\langle\rangle.\mathbf{0} \triangleq \overline{x}\langle\rangle \\ (\nu z)(\nu u)(P \mid z[!u\rangle x) \triangleq (\nu u)(\overline{x}\langle u\rangle \mid P) & \overline{x}\langle u\rangle.P \triangleq \overline{x}\langle u\rangle \mid P \\ (\nu z)(P \mid z[\vec{m}, \mathsf{inl}\rangle x) \triangleq (\nu x')\big(x.\mathsf{inl}\langle x'\rangle \mid (\nu z)(P \mid z[\vec{m}\rangle x')\big) & x.\mathsf{inl}; P \triangleq (\nu x')(x.\mathsf{inl}\langle x'\rangle \mid P\{x'/x\}) \\ (\nu z)(P \mid z[\vec{m}, \mathsf{inr}\rangle x) \triangleq (\nu x')\big(x.\mathsf{inr}\langle x'\rangle \mid (\nu z)(P \mid z[\vec{m}\rangle x')\big) & x.\mathsf{inr}; P \triangleq (\nu x')(x.\mathsf{inr}\langle x'\rangle \mid P\{x'/x\}) \\ & x.\mathsf{case}(P_1, P_2) \\ & \quad \triangleq x.\mathsf{case}((x_1').P_1\{x_1'/x\}, (x_2').P_2\{x_2'/x\}) \end{array}$$

Figure 3 A π-calculus with explicit two-sided FIFO buffers.

Reduction semantics. The reductions are given in Figure 3. Reductions S-ch, S-fin, S-inl, S-inr, and S-!ch show that an output from endpoint x can always be placed at the tail of its peer endpoint's input buffer. Thus, outputs are non-blocking. Conversely, reductions R-ch, R-fin, R-inl, R-inr, and R-!ch show how the peer endpoint z responds to these messages using inputs and cases. Note that receipt of a fin termination message (R-fin) causes the buffer to be deallocated. Similarly, receipt of a persistent channel (R-!ch) deallocates the buffer because persistent channels spawn linear sessions rather than establishing a persistent pattern of communication in their own right.

4.2 Typing and well-typed reductions for buffered processes

In our asynchronous process assignment, the use of fresh channels for session continuations orders outputs within a session into a queue: the sequence $\overline{x}\langle y_1, x'\rangle$, $\overline{x'}\langle y_2, x''\rangle$, $\overline{x''}\langle y_3, x'''\rangle$, ... can be read as a queue, $x\langle y_1, y_2, y_3, \dots]$. This intuition allows us to treat buffered processes as notational definitions for polyadic asynchronous processes, as shown in Figure 3.

To type processes in the calculus with explicit buffers, we expand the definitions and type the resulting process according the polyadic asynchronous process assignment. For instance, to type the process $(\nu z)\big(\overline{z}\langle\rangle.\mathbf{0} \mid (\nu y)(P_1 \mid z[y\rangle x)\big)$, we would expand the definitions, typing $(\nu y)(\nu x')\big(\overline{x}\langle y, x'\rangle \mid P_1 \mid \overline{x'}\langle\rangle\big)$ instead. The reductions with explicit buffers also correspond to reductions in the polyadic asynchronous process assignment:

▶ **Theorem 2.** *Well-typed reductions respect the definitions from Figure 3.*

Proof. As a representative example, consider the well-typed reductions derived from (S-CH) and (R-CH). The well-typed reduction corresponding to (R-CH) is

$$(\nu x)\big((\nu z)(P_2 \mid (\nu y)(P_1 \mid z[\vec{m}, y\rangle x)) \mid x(y).Q\big) \longrightarrow (\nu x)\big((\nu z)(P_2 \mid z[\vec{m}\rangle x) \mid (\nu y)(P_1 \mid Q)\big)\,.$$

By expanding according to the notational definitions from Figure 3, the reducing process is $(\nu x)\big((\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid (\nu z)(P_2 \mid z[\vec{m}\rangle x')) \mid x(y, x').Q\{x'/x\}\big)$. By the principal cut at $\otimes$ type, this reduces to $(\nu x')\big((\nu z)(P_2 \mid z[\vec{m}\rangle x') \mid (\nu y)(P_1 \mid Q\{x'/x\})\big)$, which, modulo α-conversion (since x' is not free in P_1 or P_2), is the same as the direct result.

The well-typed reduction corresponding to (S-CH) is

$$(\nu x)\big((\nu z)((\nu y)\overline{z}\langle y\rangle.(P_1 \mid P_2) \mid z[\vec{m}\rangle x) \mid Q\big) \longrightarrow (\nu x)\big((\nu z)(P_2 \mid (\nu y)(P_1 \mid z[y, \vec{m}\rangle x)) \mid Q\big)\,.$$

We can show by induction on the length of $\vec{m}$ that these processes are structurally equivalent when the definitions from Figure 3 are applied. The inductive case is straightforward. In the base case, it suffices to show that $(\nu z)((\nu y)\overline{z}\langle y\rangle.(P_1 \mid P_2) \mid z[\rangle x)$ and $(\nu z)(P_2 \mid (\nu y)(P_1 \mid z[y\rangle x))$ are structurally equivalent when the definitions are expanded. The former expands to $(\nu y, x')(\overline{x}\langle y, x'\rangle \mid P_1 \mid P_2\{x'/z\})$ and so does the latter. ◀

This result shows that our asynchronous polyadic process assignment does indeed faithfully represent buffered asynchronous session types.

5 Related Work

The connections between linear logic and concurrency have been studied by both the logic and concurrency theory communities. Abramsky gave a process algebraic interpretation of classical linear logic proofs [1]. Since then, some work has taken a "propositions as types" approach. Two of the present authors proposed an interpretation of linear logic in which *synchronous* π-calculus session-typed processes are intuitionistic linear logic proofs [8], giving rise to several interesting extensions and applications [9].

The connections between *asynchronous* process algebras and linear logic are also not new. Honda and Laurent [16] show a correspondence between polarized proof nets and typings for the asynchronous polyadic π-calculus. In contrast to our work, they consider the much simpler IO-type system, rather than session types. Moreover, they use classical proof nets, whereas we capture asynchrony while remaining in a sequent calculus.

Bellin and Scott [6] also give a process interpretation of classical linear logic using proof nets. They modify the synchronous π-calculus by adding structural laws that allow for arbitrary prefix commutations. This greatly simplifies the match with the proof theory but also makes the development somewhat artificial from the process calculus perspective. In contrast, we compromise between the two worlds in an arguably more satisfying way.

Beauxis et al. [4] showed that, in an untyped setting, the asynchronous π-calculus corresponds to using bags for buffered communication. Buffers for session-typed asynchrony have been considered for binary session types in a functional language [12], an object-oriented language [11], and for multiparty sessions [18]. The systems of [12, 11] are similar to ours but lack a clean logical interpretation which we get from our operational correspondence results.

6 Conclusion

In this paper, we have exhibited a novel process assignment from the asynchronous, polyadic π-calculus to the proof rules of intuitionistic linear logic (Section 2). By allowing non-blocking outputs, our asynchronous interpretation exposes additional parallelism inherent in linear logic that remained hidden in the prior synchronous interpretation [8, 9]. Proof-theoretically, this arises from a better match between proof equivalences and process equivalences (Section 3).

As future work, we would like to further study the behavioral theory of the asynchronous process assignment. With this understanding, it should then be possible to relate the synchronous and the asynchronous assignments by developing a form of delayed bisimulation for synchronous processes. We would also like to extend the asynchronous assignment to *multiparty* session types [18]; we conjecture that hybrid logic [19] might prove useful.

References

1 S. Abramsky. Computational interpretations of linear logic. *Theoret. Comput. Sci.*, 111(1–2):3–57, 1993.

2 J-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Logic Comput.*, 2(3):197–347, 1992.

3 A. Barber. Dual intuitionistic linear logic. Technical Report LFCS-96-347, Univ. of Edinburgh, 1996.

4 R. Beauxis, C. Palamidessi, and F. D. Valencia. On the asynchronous nature of the asynchronous π-calculus. In *Concurrency, Graphs and Models*, pages 473–492, 2008.

5 E. Beffara. A concurrent model for linear logic. In *21st Ann. Conf. Math. Found. Program. Semantics*, pages 147–168, 2006.

6 G. Bellin and P. Scott. On the π-calculus and linear logic. *Theoret. Comput. Sci.*, 135(1):11–65, 1994.

7 G. Boudol. Asynchrony and the π-calculus. Rapport de recherche RR-1702, INRIA, 1992.

8 L. Caires and F. Pfenning. Session types as intuitionistic linear propositions. In *21st Int. Conf. Concur. Theory*, pages 222–236. LNCS 6269, 2010.

9 L. Caires, F. Pfenning, and B. Toninho. Towards concurrent type theory. In *8th ACM SIGPLAN Workshop on Types in Language Design and Implementation*, pages 1–12, 2012.

10 B-Y. E. Chang, K. Chaudhuri, and F. Pfenning. A judgmental analysis of linear logic. Technical Report CMU-CS-03-131R, Carnegie Mellon Univ., 2003.

11 M. Dezani-Ciancaglini, S. Drossopoulou, D. Mostrous, and N. Yoshida. Objects and session types. *Inform. and Comput.*, 207(5):595–641, 2009.

12 S. Gay and V. T. Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Programming*, 20(1):19–50, 2010.

13 J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–102, 1987.

14 M. Giunti and V. T. Vasconcelos. A linear account of session types in the π-calculus. In *21st Int. Conf. Concur. Theory*, pages 432–446. LNCS 6269, 2010.

15 K. Honda. Types for dyadic interaction. In *4th Int. Conf. Concur. Theory*, pages 509–523. LNCS 715, 1993.

16 K. Honda and O. Laurent. An exact correspondence between a typed π-calculus and polarised proof-nets. *Theoret. Comput. Sci.*, 411(22–24):2223–2238, 2010.

17 K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *5th Eur. Conf. Object-Oriented Programming*, pages 133–147. LNCS 512, 1991.

18 K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *35th ACM SIGPLAN-SIGACT Symp. Prin. Program. Lang.*, pages 273–284, 2008.

19 T. Murphy, VII. *Modal Types for Mobile Code.* PhD thesis, Carnegie Mellon Univ., January 2008. Available as technical report CMU-CS-08-126.

20 J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho. Linear logical relations for session-based concurrency. In *22nd Eur. Symp. Program.*, pages 539–558. LNCS 7211, 2012.

Bounded Combinatory Logic

Boris Düdder[1], Moritz Martens[1], Jakob Rehof[1], and Paweł Urzyczyn*[2]

1 **Department of Computer Science**
Technical University of Dortmund
Dortmund, Germany
{boris.duedder,moritz.martens,jakob.rehof}@cs.tu-dortmund.de

2 **Institute of Informatics,**
University of Warsaw
Warszawa, Poland
urzy@mimuw.edu.pl

Abstract

In combinatory logic one usually assumes a fixed set of basic combinators (axiom schemes), usually **K** and **S**. In this setting the set of provable formulas (inhabited types) is PSPACE-complete in simple types and undecidable in intersection types. When arbitrary sets of axiom schemes are considered, the inhabitation problem is undecidable even in simple types (this is known as Linial-Post theorem).

Bounded combinatory logic (BCL_k) arises from combinatory logic by imposing the bound k on the depth of types (formulae) which may be substituted for type variables in axiom schemes. We consider the inhabitation (provability) problem for BCL_k: Given an arbitrary set of typed combinators and a type τ, is there a combinatory term of type τ in k-bounded combinatory logic?

Our main result is that the problem is $(k+2)$-EXPTIME complete for BCL_k with intersection types, for every fixed k (and hence non-elementary when k is a parameter). We also show that the problem is EXPTIME-complete for simple types, for all k.

Theoretically, our results give new insight into the expressive power of intersection types. From an application perspective, our results are useful as a foundation for composition synthesis based on combinatory logic.

1998 ACM Subject Classification F.4.1 Mathematical Logic, I.2.2 Automatic Programming

Keywords and phrases Intersection types, Inhabitation, Composition synthesis

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.243

1 Introduction

In standard combinatory logic (see, e.g., [5]), one usually considers a fixed set of typed combinators (a combinatory basis), for example $\mathbf{S} : (\alpha \to (\beta \to \gamma)) \to (\alpha \to \beta) \to (\alpha \to \gamma)$ and $\mathbf{K} : \alpha \to \beta \to \alpha$. Under the propositions-as-types correspondence, combinator types correspond to axiom schemes of propositional logic in a Hilbert-style proof system, with modus ponens and a rule of axiom scheme instantiation as the principles of deduction. The schematic interpretation of axioms corresponds to implicit polymorphism of combinator types, where type variables $(\alpha, \beta, \gamma, \ldots)$ may be instantiated with arbitrary types. Thus, the combinator **K** has types $\tau \to \sigma \to \tau$ for all τ and σ.

* Partly supported by MNiSW grant N N206 355836.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 243–258
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider *bounded combinatory logic* ($\textsc{bcl}_k$), which arises from combinatory logic by imposing the bound k on the depth of types (formulae) which may be substituted for type variables in axiom schemes. For example, in $\textsc{bcl}_k$ the type scheme of the combinator $\mathbf{K}$ can only be instantiated to $\tau \to \sigma \to \tau$ for τ and σ with depth $\leq k$. By imposing the bound, inhabitation becomes decidable in cases where the unbounded problem is undecidable.

Our interest in bounded combinatory logic is motivated both by theoretical concerns and from the standpoint of applications. Theoretically, we are interested in the complexity and expressive power of the system, depending on the bound. From an application perspective, we consider bounded combinatory logic as a foundation for type-based synthesis, following [8]. In the present paper we generalize from the monomorphic case of [8] to arbitrary bounded levels of polymorphism.

Bounded combinatory logic. In contrast to standard combinatory logic (see, e.g., [5]), we bound the depth of types used to instantiate types of combinators, but rather than considering a *fixed* base of combinators (for example, the base $\mathbf{S}, \mathbf{K}$) as is usual in combinatory logic, we consider the inhabitation problem *relativized* to an arbitrary set Γ of typed combinators, given as part of the input:

Given Γ and τ, is there an applicative term e such that $\Gamma \vdash_k e : \tau$?

The relativized problem is generally much harder than the fixed-base problem. For example, inhabitation in standard (unbounded) simple-typed $\mathbf{SK}$-calculus is Pspace-complete [11], whereas the unbounded relativized problem is *undecidable*, even in simple types. We recall that the latter type of problem has been considered since 1948 when Linial and Post [6] initiated a line of work studying decision problems for arbitrary propositional axiom systems (often referred to as partial propositional calculi, abbreviated PPC) answering a question posed by Tarski in 1946. They proved (among other things) that there exists a PPC with an unsolvable decision problem (Linial-Post theorem). Since then, many results have been obtained for various PPC, e.g., Gladstone [3] and Singletary [9] showed that every r.e. degree can be represented by a PPC. In 1974, Singletary [10] showed that the implicational fragment of PPC can represent every r.e. many-one degree. The problem considered there is identical to the unbounded relativized inhabitation problem for simple types.

Our main result is that the relativized inhabitation problems for $\textsc{bcl}_k$ with intersection types form an infinite hierarchy, being $(k+2)$-Exptime-complete for each fixed k. A non-elementary lower bound follows for the problem where k is taken as an input parameter. Our lower bound techniques, which may be of independent interest, expose new aspects of the expressive power of intersection types. We generically simulate alternating Turing machines operating in $\exp_{k+1}(n)$-bounded space, where $\exp_m$ denotes the iterated exponential function. For each k, we devise a numeral representation with intersection types in $\textsc{bcl}_k$ for numbers between 0 and $\exp_{k+1}(n) - 1$, and we use this system to achieve a succinct representation (exploiting k-bounded polymorphism) of the Turing tape. In contrast, we show that the k-bounded inhabitation problem is Exptime-complete for simple types, for all k.

A foundation for composition synthesis With this paper we continue the work begun in [8] on investigating limited systems of combinatory logic as a foundation for type-based synthesis (automatic synthesis of function compositions from a repository of typed functions). In [8], we proved the monomorphic inhabitation problem Exptime-complete and devised inhabitation algorithms that we have since implemented and applied to synthesis. In our applications, the set Γ models a repository, the goal type τ is considered as a specification of a desired composition, and the inhabitation algorithm automatically constructs solutions (if any) to the synthesis problem. The relativized inhabitation problem is the natural basis for applications in synthesis, where Γ models a changing repository of functions. As argued in [8], intersection types play a key role in these applications, since they can be used to specify deep semantic properties.

A limited degree of polymorphism has been found to be very useful in applications, since it allows for succinct specifications. In particular, the lowest level (BCL_0) of the hierarchy studied here turns out to be already of major importance. At this level, we are able to instantiate type variables with atoms or intersections of such. Since type structure can be atomized by introducing type names (atoms) for structured types through definitions, many interesting problems can be specified and solved in BCL_0.

As a simple example of succinctness, consider that we can represent any finite function $f : A \to B$ as an intersection type $\tau_f = \bigcap_{a \in A} a \to f(a)$, where elements of A and B are type constants. Suppose we have combinators $F_i : \tau_{f_i}$ in Γ, and we want to synthesize compositions of such functions represented as types (in some of our applications they could, for example, be refinement types [2]). We might want to introduce composition combinators of arbitary arity, say $g : (A \to A)^n \to (A \to A)$. In the monomorphic system, a function table for g would be exponentially large in n. In BCL_0, we can represent g with the single declaration $G : (\alpha_0 \to \alpha_1) \to (\alpha_1 \to \alpha_2) \to \cdots \to (\alpha_{n-1} \to \alpha_n) \to (\alpha_0 \to \alpha_n)$ in Γ. Through level-0 polymorphism, the action of g is thereby fully specified.

Interestingly, by the present results, the complexity of BCL_0 is 2-EXPTIME complete and hence comparable in complexity to other known synthesis frameworks (such as, e.g., variants of temporal logic and of propositional dynamic logic). It is also interesting to observe that the lower bound techniques of the present paper appear to reveal a methodology by which inhabitation of intersection types can be used to express a form of logic programming at the type level, which appears to be useful in synthesis. Space limitations preclude us from going into further details here, and we report on our experience in synthesis in a separate paper.

2 Preliminaries

Types: Type expressions, ranged over by τ, σ etc., are defined by

$$\tau \ ::= \ a \mid \tau \to \tau \mid \tau \cap \tau$$

where $a, b, c, \ldots$ range over *atoms* comprising of *type constants*, drawn from a finite set $\mathbb{A}$ including the constant ω, and *type variables*, drawn from a disjoint denumerable set $\mathbb{V}$ ranged over by α, β etc. We let $\mathbb{T}$ denote the set of all types.

As usual, types are taken modulo commutativity ($\tau \cap \sigma = \sigma \cap \tau$), associativity ($(\tau \cap \sigma) \cap \rho = \tau \cap (\sigma \cap \rho)$), and idempotency ($\tau \cap \tau = \tau$). As a matter of notational convention, function types associate to the right, and $\cap$ binds stronger than $\to$. A type *environment* Γ is a finite set of type assumptions of the form $x : \tau$. We let $\mathit{Dm}(\Gamma)$ and $\mathit{Rn}(\Gamma)$ denote the domain and range of Γ. Let $\mathit{Var}(\tau)$, $\mathit{Cnst}(\tau)$ and $\mathit{At}(\tau)$ denote, respectively, the set of variables, the set of constants and the set of atoms occurring in τ, and we extend the definitions to environments, written $\mathit{Var}(\Gamma)$, $\mathit{Cnst}(\Gamma)$ and $\mathit{At}(\Gamma)$ in the standard way.

A type $\tau \cap \sigma$ is said to have τ and σ as *components*. For an intersection of several components we sometimes write $\bigcap_{i=1}^{n} \tau_i$ or $\bigcap_{i \in I} \tau_i$ or $\bigcap\{\tau_i \mid i \in I\}$, where the empty intersection is identified with ω.

Subtyping: Subtyping $\leq$ is the least preorder (reflexive and transitive relation) on $\mathbb{T}$, with

$$\sigma \leq \omega, \quad \omega \leq \omega \to \omega, \quad \sigma \cap \tau \leq \sigma, \quad \sigma \cap \tau \leq \tau, \quad \sigma \leq \sigma \cap \sigma;$$
$$(\sigma \to \tau) \cap (\sigma \to \rho) \leq \sigma \to \tau \cap \rho;$$
If $\sigma \leq \sigma'$ and $\tau \leq \tau'$ then $\sigma \cap \tau \leq \sigma' \cap \tau'$ and $\sigma' \to \tau \leq \sigma \to \tau'$.

We identify σ and τ when $\sigma \leq \tau$ and $\tau \leq \sigma$. The following distributivity properties follow from the axioms of subtyping:

$$(\sigma \to \tau) \cap (\sigma \to \rho) = \sigma \to (\tau \cap \rho) \qquad (\sigma \to \tau) \cap (\sigma' \to \tau') \leq (\sigma \cap \sigma') \to (\tau \cap \tau')$$

Paths: If $\tau = \tau_1 \to \cdots \to \tau_m \to \sigma$, then we write $\sigma = \mathit{tgt}_m(\tau)$ and $\tau_i = \mathit{arg}_i(\tau)$, for $i \leq m$. If $\mathit{arg}_i(\tau) = \rho$ for all i we also write $\tau = \rho^m \to \sigma$. A type of the form $\tau_1 \to \cdots \to \tau_m \to a$, where $a \neq \omega$ is an atom,[1] is called a *path of length m*. A type τ is *organized* if it is a (possibly empty) intersection of paths (those are called *paths in* τ). Note that premises in an organized type do not have to be organized, i.e., organized is not necessarily normalized [4].

▶ **Lemma 1.** *Every type τ is equal to an organized type $\overline{\tau}$, computable in polynomial time.*

Proof. Define $\overline{a} = a$ if a is an atom and let $\overline{\tau \cap \sigma} = \overline{\tau} \cap \overline{\sigma}$. If $\overline{\sigma} = \bigcap_{i \in I} \sigma_i$ then take $\overline{\tau \to \sigma} = \bigcap_{i \in I}(\tau \to \sigma_i)$. ◀

Sets of paths: For an organized type σ, we let $\mathbb{P}_m(\sigma)$ denote the set of all paths in σ of length m or more. We extend the definition to arbitrary τ by implicitly organizing τ, i.e., we write $\mathbb{P}_m(\tau)$ as a shorthand for $\mathbb{P}_m(\overline{\tau})$.

Type size: The *size* of a type τ, denoted $|\tau|$, is defined to be the number of nodes in the syntax tree of τ (this is identical to the textual size of τ). The *path length* of a type τ is denoted $\|\tau\|$ and is defined to be the maximal length of a path in τ.

Substitutions: A *substitution* is a function $S : \mathbb{V} \to \mathbb{T}$ such that S is the identity everywhere but on a finite subset of $\mathbb{V}$. For a substitution S, we define the *support* of S, written $\mathit{Supp}(S)$, as $\mathit{Supp}(S) = \{\alpha \in \mathbb{V} \mid \alpha \neq S(\alpha)\}$. We may write $S : V \to \mathbb{T}$ when V is a finite subset of $\mathbb{V}$ with $\mathit{Supp}(S) \subseteq V$. We write $\mathit{At}(S)$ to denote the set $\{\mathit{At}(S(\alpha)) \mid \alpha \in \mathit{Supp}(S)\}$. A substitution S is tacitly lifted to a function on types, $S : \mathbb{T} \to \mathbb{T}$, by homomorphic extension. Finally, a *constant-function* is a map $c : \mathbb{A} \to \mathbb{A}$ such that $c(\omega) = \omega$. Constant-functions are tacitly lifted to functions $c : \mathbb{T} \to \mathbb{T}$.

The following property, probably first stated in [1], is often called *beta-soundness*. Note that the converse is trivially true.

▶ **Lemma 2.** *Let a_j, for $j \in J$, be atoms.*

1. *If $\bigcap_{i \in I}(\sigma_i \to \tau_i) \cap \bigcap_{j \in J} a_j \leq \alpha$ then $\alpha = a_j$, for some $j \in J$.*
2. *If $\bigcap_{i \in I}(\sigma_i \to \tau_i) \cap \bigcap_{j \in J} a_j \leq \sigma \to \tau$, where $\sigma \to \tau \neq \omega$, then the set $\{i \in I \mid \sigma \leq \sigma_i\}$ is nonempty and $\bigcap\{\tau_i \mid \sigma \leq \sigma_i\} \leq \tau$.*

▶ **Lemma 3.** *Let $\bigcap_{i \in I} \tau_i \leq \beta_1 \to \cdots \to \beta_m \to p$, where τ_i are paths. Then there is an $i \in I$ such that $\tau_i = \alpha_1 \to \cdots \to \alpha_m \to p$ and $\beta_j \leq \alpha_j$, for all $j \leq m$.*

Proof. Induction with respect to m, using the beta soundness (Lemma 2). ◀

[1] Observe that $\tau_1 \to \cdots \to \tau_m \to \omega = \omega$.

▶ **Lemma 4.** *Let S be a substitution and let c be a constant-function. Then $\sigma \leq \tau$ implies $S(\sigma) \leq S(\tau)$ and $c(\sigma) \leq c(\tau)$.*

Proof. Induction with respect to the definition of $\sigma \leq \tau$. ◀

Alternating Turing Machines

An *alternating Turing machine* is a tuple $\mathcal{M} = (\Sigma, Q, q_0, q_a, q_r, \Delta)$. The set of states $Q = Q_\exists \uplus Q_\forall$ is partitioned into a set $Q_\exists$ of existential states and a set $Q_\forall$ of universal states. There is an initial state $q_0 \in Q$, an accepting state $q_a \in Q_\forall$, and a rejecting state $q_r \in Q_\exists$. We take $\Sigma = \{0, 1, \sqcup\}$, where $\sqcup$ is the blank symbol (used to initialize the tape but not written by the machine). The transition relation Δ satisfies

$$\Delta \subseteq \Sigma \times Q \times \Sigma \times Q \times \{\text{L}, \text{R}\},$$

where $h \in \{\text{L}, \text{R}\}$ are the moves of the machine head (left and right). For $b \in \Sigma$ and $q \in Q$, we write $\Delta(b, q) = \{(c, p, h) \mid (b, q, c, p, h) \in \Delta\}$. We assume $\Delta(b, q_a) = \Delta(b, q_r) = \emptyset$, for all $b \in \Sigma$, and $\Delta(b, q) \neq \emptyset$ for $q \in Q \setminus \{q_a, q_r\}$. A *configuration* of $\mathcal{M}$ is a word wqw' with $q \in Q$ and $w, w' \in \Sigma^*$. The *successor* relation $\mathcal{C} \Rightarrow \mathcal{C}'$ on configurations is defined as usual [7], according to Δ. We classify a configuration wqw' as *existential*, *universal*, *accepting* etc., according to q. The notion of *eventually accepting* configuration is defined by induction:[2]

- An accepting configuration is eventually accepting.
- If $\mathcal{C}$ is existential and some successor of $\mathcal{C}$ is eventually accepting then so is $\mathcal{C}$.
- If $\mathcal{C}$ is universal and all successors of $\mathcal{C}$ are eventually accepting then so is $\mathcal{C}$.

3 Bounded combinatory logic

▶ **Definition 5.** (Levels) Given a type τ we define the *level* of τ, written $\ell(\tau)$, as follows.

$$\begin{array}{lcl} \ell(a) & = & 0, \text{ for } a \in \mathbb{A} \cup \mathbb{V}; \\ \ell(\tau \to \sigma) & = & 1 + \max\{\ell(\tau), \ell(\sigma)\}; \\ \ell(\bigcap_{i=1}^{n} \tau_i) & = & \max\{\ell(\tau_i) \mid i = 1, \ldots, n\}. \end{array}$$

The level of a substitution S, written $\ell(S)$, is defined as

$$\ell(S) = \max\{\ell(S(\alpha)) \mid \alpha \in \mathbb{V}\}.$$

A *level-k type* is a type τ with $\ell(\tau) \leq k$, and a *level-k substitution* is a substitution S with $\ell(S) \leq k$. For $k \geq 0$, we let $\mathbb{T}_k$ denote the set of all level-k types. For a subset A of atomic types, we let $\mathbb{T}_k(A)$ denote the set of level-k types with atoms (leaves) in the set A. ◀

Notice that the level of a type is independent from the width (number of arguments) of intersections. Notice also that $\ell(S)$ is completely determined by the restriction of S to $\mathit{Supp}(S)$: if $\mathit{Supp}(S) = \emptyset$, then $\ell(S) = 0$, and if $\mathit{Supp}(S) \neq \emptyset$, then $\ell(S) = \max\{\ell(S(\alpha)) \mid \alpha \in \mathit{Supp}(S)\}$. Finally, we have $\ell(S \circ S') \leq \ell(S) + \ell(S')$.

Type assignment: For each $k \geq 0$ the system $\text{BCL}_k(\to, \cap)$ (*k-bounded combinatory logic with intersection types*) is defined by the type assigment rules shown in Figure 1. In rule (var), the

[2] Formally we define the set of all eventually accepting configurations as the smallest set satisfying the appropriate closure conditions.

condition $\ell(S) \leq k$ is understood as a side condition to the axiom $\Gamma, x : \tau \vdash_k x : S(\tau)$. The restriction to *simple types* (types without $\cap$) is called $\textsc{bcl}_k(\to)$ and is defined by the rules (var), ($\to$E) and ($\leq$), where τ and τ' range over simple types, by dropping all axioms from the subtyping relation that involve $\cap$, and by considering only substitutions S mapping type variables to simple types. Recall from [8] *finite combinatory logic with intersection types*, denoted FCL. This system can be presented as the restriction of $\textsc{bcl}_k$ in which the (var) rule is simplified to the axiom $\Gamma, x : \tau \vdash x : \tau$.

In this paper we are addressing the following *relativized inhabitation problem:*

Given Γ and τ, is there an applicative term e such that $\Gamma \vdash_k e : \tau$?

$$\frac{[\ell(S) \leq k]}{\Gamma, x : \tau \vdash_k x : S(\tau)}\,(\text{var}) \qquad \frac{\Gamma \vdash_k e : \tau \to \tau' \quad \Gamma \vdash_k e' : \tau}{\Gamma \vdash_k (e\ e') : \tau'}\,(\to\text{E})$$

$$\frac{\Gamma \vdash_k e : \tau_1 \quad \Gamma \vdash_k e : \tau_2}{\Gamma \vdash_k e : \tau_1 \cap \tau_2}\,(\cap\text{I}) \qquad \frac{\Gamma \vdash_k e : \tau \quad \tau \leq \tau'}{\Gamma \vdash_k e : \tau'}\,(\leq)$$

Figure 1 Bounded combinatory logic $\textsc{bcl}_k$.

Algorithm

In this section we formulate an algorithm to decide the relativized inhabitation problem for $\textsc{bcl}_k$, and derive the $(k + 2)$-EXPTIME upper bound.

▶ **Lemma 6.** *Let $\Gamma \vdash_k e : \tau$ and let S be a level-m substitution. Then there exists a derivation of $\Gamma \vdash_{k+m} e : S(\tau)$ of the same depth.*

Proof. Induction with respect to the derivation of $\Gamma \vdash_k e : \tau$. ◀

▶ **Lemma 7.** *Let $\Gamma \vdash_k e : \tau$ and let c be a constant-function such that c is the identity on $Cnst(\Gamma)$. Then there exists a derivation of $\Gamma \vdash_k e : c(\tau)$ of the same depth.*

Proof. Induction with respect to the derivation of $\Gamma \vdash_k e : \tau$. In case the derivation ends with rule ($\leq$), we use Lemma 4 and apply the induction hypothesis. ◀

Let $At_\omega(\Gamma, \tau) = At(\Gamma) \cup At(\tau) \cup \{\omega\}$. The following proposition shows that, in order to solve an inhabitation question $\Gamma \vdash_k ? : \tau$, one needs only consider rule (var) restricted to substitutions of the form $S : Var(\Gamma) \to \mathbb{T}_k(At_\omega(\Gamma, \tau))$.

We say that a substitution S *occurs* in a derivation $\mathcal{D}$, whenever S is used in an application of rule (var) in $\mathcal{D}$.

▶ **Proposition 8.** *If $\Gamma \vdash_k e : \tau$, then there exists a derivation $\mathcal{D}$ of $\Gamma \vdash_k e : \tau$ such that every substitution S occurring in $\mathcal{D}$ satisfies the conditions*

1. *$Supp(S) \subseteq Var(\Gamma)$;*
2. *$At(S) \subseteq At_\omega(\Gamma, \tau)$.*

Proof. By induction with respect to derivations, using Lemmas 6 and 7. ◀

The following lemma shows that inhabitation in $\text{BCL}_k(\to,\cap)$ is equivalent to inhabitation in FCL modulo expansion of the type environment. Given a number k, an environment Γ and a type τ, define for each $x \in Dm(\Gamma)$ the set of substitutions

$$\mathcal{S}_x^{(\Gamma,\tau,k)} = Var(\Gamma(x)) \to \mathbb{T}_k(At_\omega(\Gamma,\tau))$$

and define the environment $\Gamma^{(\tau,k)}$ with domain $Dm(\Gamma)$ so that, for $x \in Dm(\Gamma)$,

$$\Gamma^{(\tau,k)}(x) = \bigcap\{S(\Gamma(x)) \mid S \in \mathcal{S}_x^{(\Gamma,\tau,k)}\}$$

▶ **Lemma 9** (Expansion). *One has* $\Gamma \vdash_k e : \tau$ *in* $\text{BCL}_k(\to,\cap)$ *iff* $\Gamma^{(\tau,k)} \vdash e : \tau$ *in* FCL.

Proof. If $\Gamma \vdash_k e : \tau$ by a derivation $\mathcal{D}$, consider each application of rule (var) of the form $\Gamma', x : \sigma \vdash_k x : S(\sigma)$, occurring in $\mathcal{D}$. By Proposition 8, we can assume that S is a member of the set $\mathcal{S}_x^{(\Gamma,\tau,k)}$. Hence, one has $\Gamma^{(\tau,k)} \vdash x : S(\sigma)$ in FCL, by an application of rule (var), followed by an application of rule ($\leq$). It follows that $\Gamma^{(\tau,k)} \vdash e : \tau$ holds in FCL.

For the implication in the other direction, consider that one has in $\text{BCL}_k(\to,\cap)$

$$\Gamma \vdash_k x : \bigcap\{S(\Gamma(x)) \mid S \in \mathcal{S}_x^{(\Gamma,\tau,k)}\}$$

for all $x \in Dm(\Gamma)$, by multiple applications of rule (var), followed by rule ($\cap$I). ◀

▶ **Lemma 10** (Path Lemma for FCL [8]). *The following are equivalent conditions:*

1. $\Gamma \vdash x\ e_1 \ldots e_m : \tau$;
2. *There exists a set* P *of paths in* $\mathbb{P}_m(\Gamma(x))$ *such that*
   - **a.** $\bigcap_{\pi \in P} tgt_m(\pi) \ \leq\ \tau$;
   - **b.** $\Gamma \vdash e_i : \bigcap_{\pi \in P} arg_i(\pi)$, *for all* $i \leq m$.

▶ **Lemma 11** (Path Lemma for $\text{BCL}_k(\to,\cap)$). *The following are equivalent conditions:*

1. $\Gamma \vdash_k x\ e_1 \ldots e_m : \tau$;
2. *There exists a set* P *of paths in* $\mathbb{P}_m(\bigcap\{S(\Gamma(x)) \mid S \in \mathcal{S}_x^{(\Gamma,\tau,k)}\})$ *such that*
   - **a.** $\bigcap_{\pi \in P} tgt_m(\pi) \ \leq\ \tau$;
   - **b.** $\Gamma \vdash_k e_i : \bigcap_{\pi \in P} arg_i(\pi)$, *for all* $i \leq m$.

Proof. Immediate, by Lemma 9 and Lemma 10. ◀

The following corollary will be used later.

▶ **Corollary 12.** *Let* $\Gamma(x) = \bigcap_{j \in J}(\tau_1^j \to \cdots \to \tau_m^j \to \sigma^j)$. *If* $\Gamma \vdash x\ e_1 \ldots e_m : \tau$ *then there are substitutions* S_ℓ, *for* $\ell \in L$, *and numbers* j_ℓ *such that*

1. $\bigcap_{\ell \in L} S_\ell(\sigma^{j_\ell}) \leq \tau$;
2. $\Gamma \vdash_k e_i : \bigcap_{\ell \in L} S_\ell(\tau_i^{j_\ell})$.

Let $\exp_k$ be the iterated exponential function, given by $\exp_0(n) = n, \exp_{k+1}(n) = 2^{\exp_k(n)}$. The lemma below can be shown by an elementary counting argument.

▶ **Lemma 13.** *For every* k, *there is a polynomial* $p(n)$ *such that the number of level-*k *types over* n *atoms is at most* $\exp_{k+1}(p(n))$, *and the size of such types is at most* $\exp_k(p(n))$. *The number and size of* simple *level-*k *types (for a fixed* k*) is respectively bounded by a polynomial and a constant.*

▶ **Theorem 14.** *Inhabitation in* $\text{BCL}_k(\to,\cap)$ *is in* $(k+2)$-EXPTIME.

Proof. The alternating Turing machine shown in Figure 2 is a decision procedure for inhabitation in $\text{BCL}_k(\to,\cap)$ for each $k \geq 0$, being a direct alternating implementation of Lemma 11. In Figure 2 we use shorthand notation for instruction sequences starting from existential states (CHOOSE ...) and instruction sequences starting from universal states ($\text{FORALL}(i = 1 \ldots k)\, S_i$). A command of the form CHOOSE $x \in S$ branches from an existential state to successor states in which x gets assigned distinct elements of S. A command of the form $\text{FORALL}(i = 1 \ldots k)\, S_i$ branches from a universal state to successor states from which each instruction sequence S_i is executed.

The machine operates in bounded space, because, for all Γ, τ, k, x, the set $\mathcal{S}_x^{(\Gamma,\tau,k)}$ is finite. More precisely, it follows from Lemma 13 that the size of $\mathcal{S}_x^{(\Gamma,\tau,k)}$ can be bounded by $\exp_{k+1}(p(n))$, and the size of each level-k type can be bounded by $\exp_k(p(n))$, for some polynomial $p(n)$. It follows that the types σ' (Figure 2, line 2) can be written down in space bounded by $\exp_{k+1}(p(n))$, and hence the algorithm is bounded in alternating space $\exp_{k+1}(p(n))$. By the identity $\text{ASPACE}(f(n)) = \text{DTIME}(2^{\mathcal{O}(f(n))})$ inhabitation is therefore in $(k+2)$-EXPTIME. ◀

```
     Input :  $\Gamma, \tau, k$
     loop :
1    CHOOSE $(x : \sigma) \in \Gamma$;
2    $\sigma' := \bigcap\{S(\sigma) \mid S \in \mathcal{S}_x^{(\Gamma,\tau,k)}\}$;
3    CHOOSE $m \in \{0, \ldots, \|\sigma'\|\}$;
4    CHOOSE $P \subseteq \mathbb{P}_m(\sigma')$;

5    IF $(\bigcap_{\pi \in P} tgt_m(\pi) \leq \tau)$ THEN
6       IF $(m = 0)$ THEN ACCEPT;
7       ELSE
8            FORALL$(i = 1 \ldots m)$
9                 $\tau := \bigcap_{\pi \in P} arg_i(\pi)$;
10           GOTO loop;
```

Figure 2 Alternating Turing machine deciding inhabitation in BCL_k.

4 Simple types, $\text{BCL}_k(\to)$

The upper bound for simple types is obtained as a special case of the analysis in Section 3.

▶ **Theorem 15.** *Inhabitation in $\text{BCL}_k(\to)$ is in* EXPTIME, *for all k.*

Proof. The proof uses the same argument as the proof of Theorem 14. The difference is that now we only substitute simple types. Under this restriction, the machine of Figure 2 operates in alternating polynomial space, because all types of the form $S(\sigma)$ are of linear size. ◀

▶ **Theorem 16.** *For every $k \geq 0$, the inhabitation problem for $\text{BCL}_k(\to)$ is* EXPTIME-*complete.*

Proof. Take an alternating TM, working in polynomial space $p(n)$. We use fresh type atoms to represent every state and tape symbol. A configuration $\mathcal{C} = wqw'$, where $w = b_1 \dots b_{m-1}$ and $w' = b_m \dots b_{p(n)}$ is encoded as a type $\varphi_{\mathcal{C}} = b_1 \to \dots \to b_{m-1} \to q \to b_m \to \dots \to b_{p(n)}$. We define an environment Γ so that, for all $\mathcal{C}$,

$$\mathcal{C} \text{ is eventually accepting} \qquad \text{if and only if} \qquad \Gamma \vdash \varphi_{\mathcal{C}}\,. \qquad (*)$$

We put into Γ polymorphic patterns $\alpha_1 \to \dots \to \alpha_{m-1} \to q_a \to \alpha_m \to \dots \to \alpha_{p(n)}$ for accepting configurations, and types representing machine moves, as we now define.

For any q, b, the patterns $\zeta_{bqm}(\vec{\alpha}) = \alpha_1 \to \dots \to \alpha_{m-1} \to q \to b \to \alpha_{m+1} \to \dots \to \alpha_{p(n)}$ represents all configurations where $\Delta(b,q)$ is applicable. Let $\Delta(b,q) = \{(c_j,p_j,h_j) \mid j \leq r\}$, and for $j \leq r$, let $\eta_{bqmj}(\vec{\alpha})$ represent the j-th successor configuration. For example, if $h_j = \mathsf{R}$ then $\eta_{bqmj}(\vec{\alpha}) = \alpha_1 \to \dots \to \alpha_{m-1} \to c_j \to p_j \to \alpha_{m+1} \to \dots \to \alpha_{p(n)}$.
If $\mathcal{C} = b_1 \dots b_{m-1} q b b_{m+1} \dots b_{p(n)}$ then there exists exactly one substitution S (mapping each α_i to b_i) such that $S(\zeta_{bqm}) = \varphi_{\mathcal{C}}$. In addition, if $\mathcal{D}_1, \dots, \mathcal{D}_r$ are all the successor configurations of $\mathcal{C}$ then we have $S(\eta_{bqmj}) = \varphi_{\mathcal{D}_j}$. Now if q is an existential state then we include in Γ all types of the form $\eta_{bqmj} \to \zeta_{bqm}$. For a universal q, we let Γ contain just one type, namely $\eta_{bqm1} \to \dots \to \eta_{bqmr} \to \zeta_{bqm}$.

The "only if" part of (*) can now be proved by induction with respect to the definition of acceptance. In the "if" part we use induction with respect to proofs. ◀

5 Lower bound for intersection types

In this section we fix a number K and an $\exp_{K+1}(n)$-space bounded alternating Turing machine $\mathcal{M}$. In what follows it is assumed that $k \leq K$, whenever level k is considered. The basic idea is to represent a configuration of $\mathcal{M}$ by, essentially, a type of the form

$$\bigcap_{i=0}^{\exp_{K+1}(n)-1} \mathsf{Cell}(a_i, q, \langle m \rangle_K, \langle i \rangle_K),$$

where $a_i \in \Sigma$, $q \in Q$, $0 \leq m \leq \exp_{K+1}(n) - 1$. Each component $\mathsf{Cell}(a_i, q, \langle m \rangle_K, \langle i \rangle_K)$ represents one of the tape cells, where a_i represents the symbol in the i-th cell, q represents the current state, type $\langle m \rangle_K$ represents the address (number) of the cell which is under the current ATM head position, and $\langle i \rangle_K$ represents the address of the cell itself. Notice that the types q and $\langle m \rangle_K$ are identical across all the components of the type (i.e., across all indexes i). The adresses $\langle i \rangle_K$ impose a numerical order on the cell representations, so that we can represent a tape consisting of a sequence of cells. Moreover, we can use these addresses to compute the head position of the ATM (moving left or right of the current cell address).

Since we need a representation which is polynomial bounded in the size of the ATM input, we cannot represent such types explicitly in our reduction. In order to achieve a succinct (polynomial sized) representation, we exploit polymorphism. The basic insight in the reduction is to represent the large configuration types *implicitly*, as polymorphic types $\mathsf{Cell}(\alpha, q, \beta, \gamma)$, and to arrange the environment Γ coding the behavior of $\mathcal{M}$ in such a way that large expansions (under polymorphic instantiation) of such types become forced into the explicit form shown. As in the proof of Theorem 16, the basic strategy for coding the ATM behavior is to represent a computation sequence $\mathcal{C}_1\mathcal{C}_2 \cdots \mathcal{C}_m$ by a sequence of forced inhabitation goals in reverse order of implication, by (essentially) having the implications $[\mathcal{C}_{i+1}] \to [\mathcal{C}_i]$ in Γ such that asking for inhabitation of $[\mathcal{C}_i]$ forces the inhabitation of $[\mathcal{C}_{i+1}]$ (letting $[\mathcal{C}]$ denote the type representing the configuration $\mathcal{C}$).

Predicates

The *predicates* we use are certain type patterns serving as "containers" for their arguments. The idea is that a predicate like $F(\tau,\sigma)$ encodes a pair of types τ and σ and a "flag" F in a unique way. This is achieved by making sure that type $F(\tau,\sigma)$ is large enough to never be substituted for a variable. In addition, τ and σ are placed inside $F(\tau,\sigma)$ several times to avoid unwanted subtyping.

Some auxiliary notation for the beginning. Write $F^{[1]}$ for F and $F^{[n+1]}$ for $F^{[n]} \to F$. For instance, $F^{[4]} = ((F \to F) \to F) \to F$. Also let $\Omega_\tau = (\tau \to \tau) \to \tau \to \tau$.

Let $N > K$ be a fixed number. Type $F(\tau_1,\tau_2,\tau_3,\tau_4)$ (a predicate of four arguments) is defined using a dedicated type constant F (the *predicate identifier*), as follows:

$$F(\tau_1,\tau_2,\tau_3,\tau_4) = (((F^{[N]} \to \Omega_{\tau_1}) \to \Omega_{\tau_2}) \to \Omega_{\tau_3}) \to \Omega_{\tau_4}.$$

Predicates of fewer arguments are defined by repeating the last one, e.g. $G(\tau,\sigma)$ will stand for $G(\tau,\sigma,\sigma,\sigma)$. In what follows, the word "predicate" may refer to any $F(\tau_1,\ldots,\tau_4)$.

The level of $F(\tau_1,\ldots,\tau_4)$ is larger than K, and therefore types of the form $F(\tau_1,\ldots,\tau_4)$ never occur in the range of a substitution. Further properties are as follows:

▶ **Lemma 17.** *For all types τ, σ and all predicates Φ_i and Φ:*

1. *If $\bigcap_{i\in I} \Omega_{\tau_i} \le \Omega_\sigma$ then $\tau_i = \sigma$, for some i.*
2. *If $\bigcap_{i\in I} \Phi_i \le \Phi$ then $\Phi = \Phi_i$, for some i.*

Proof. Use Lemma 2. Details omitted. ◀

In our construction we use the following forms of predicates (for $k \le K$ and $j \le n$):

- Unary: $\mathsf{Zero}_k(\alpha)$, $\mathsf{z}_k(\alpha)$, $\mathsf{m}_k(\alpha)$, $\mathsf{Max}_k(\alpha)$, $\mathsf{Num}_k(\alpha)$, $\mathsf{n}^k(\alpha)$, $\mathsf{Num}^j(\alpha)$, $\mathsf{Bit}(\alpha)$, $\mathsf{Tape}^j(\alpha)$.
- Binary: $\mathsf{Succ}_k(\alpha,\beta)$, $\mathsf{Diff}_k(\alpha,\beta)$, $\mathsf{d}_k(\alpha,\beta)$, $\mathsf{n}_k(\alpha,\beta)$.
- Ternary: $\mathsf{R}_k(\alpha,\beta,\gamma)$, $\mathsf{L}_k(\alpha,\beta,\gamma)$.
- Quaternary: $\mathsf{Cell}(\alpha,\beta,\gamma,\delta)$.

In addition to that we also have the following constants (for $j \le n$):

- 0, 1, 0_j, 1_j, $\bullet$.

and special constants for all internal states and tape symbols of the machine.

Intersection type numerals

Fix a natural number n. Let $\mathbb{B}[n]$ denote the union of n copies of $\mathbb{B} = \{0,1\}$, written $\mathbb{B}[n] = \{0_1,\ldots,0_n\} \cup \{1_1,\ldots,1_n\}$. We let b range over $\mathbb{B}$ and we let $\mathfrak{b}$ range over $\mathbb{B}[n]$. The sets of *level-k numerals* ($k \ge 0$), denoted $\mathcal{N}_k$, are constructed from $\mathbb{B}[n]$ by induction:

- $\mathcal{N}_0 = \{\bigcap_{i=1}^n \mathfrak{b}_i \mid \mathfrak{b}_i \in \{0_i, 1_i\} \text{ for } i = 1\ldots n\}$
- $\mathcal{N}_{k+1} = \{\bigcap_{\tau\in\mathcal{N}_k} (\tau \to b_\tau) \mid b_\tau \in \{0,1\},\ \text{for } \tau \in \mathcal{N}_k\}$

Clearly, the size of $\mathcal{N}_k$ is $\exp_{k+1}(n)$. The value of a numeral $\sigma \in \mathcal{N}_k$ is denoted $[\![\sigma]\!]$ and is defined by induction with respect to k:

- $k = 0$: $[\![\bigcap_{i=1}^n \mathfrak{b}_i]\!] = \sum_{i=1}^n [\![\mathfrak{b}_i]\!] \times 2^{i-1}$, with $[\![0_i]\!] = 0$ and $[\![1_i]\!] = 1$
- $k > 0$: $[\![\bigcap_{\sigma\in\mathcal{N}_k} (\tau \to b_\tau)]\!] = \sum_{\tau\in\mathcal{N}_k} b_\tau \times 2^{[\![\tau]\!]}$

For instance, if $n = 4$ then the value of $0_1 \cap 1_2 \cap 0_3 \cap 1_4$ is $2+8 = 10$. And if $n = 2$ then the value of $((0_1 \cap 0_2) \to 0) \cap ((0_1 \cap 1_2) \to 1) \cap ((1_1 \cap 0_2) \to 0) \cap ((1_1 \cap 1_2) \to 1)$ is 10 as well.

It is easy to prove by induction that for $\sigma \in \mathcal{N}_k$ we have $0 \leq [\![\sigma]\!] \leq \exp_{k+1}(n) - 1$, and for $k > 0$ we can write σ canonically as $\sigma = \bigcap_{i=0}^{\exp_k(n)-1}(\tau_i \to b_i)$, where $[\![\tau_i]\!] = i$ and $b_i \in \mathbb{B}$, and with $[\![\sigma]\!] = \sum_{i=0}^{\exp_k(n)-1} b_i \times 2^i$.

It is also straightforward to see that, for any x between 0 and $\exp_{k+1}(n) - 1$, there is *exactly one* $\sigma \in \mathcal{N}_k$ with $[\![\sigma]\!] = x$. We use the notation $\sigma = \langle x \rangle_k$.

The encoding

Our goal is to define a BCL_K type environment Γ, representing the behavior of the machine $\mathcal{M}$. The environment Γ consists of several groups of declarations, to handle predicates over numerals, the tape, and the transition function. Note that each type σ in Γ is an intersection which has a component of the form $(\bullet^m \to \bullet)$, for some m, and that all other components are arrows of m arguments, ending with predicates of the same identifier F. We then say that σ, and the corresponding combinator, is *m-ary*, and that F is the *target identifier* of σ.

▶ **Lemma 18.** *If x is m-ary and $\Gamma \vdash_K xe_1 \ldots e_r : \bullet$ then $r = m$.*

Proof. If $\Gamma \vdash_K xe_1 \ldots e_r : \bullet$ then by Lemma 11 we have $\bigcap_{\pi \in P} \mathit{tgt}_r(\pi) \leq \bullet$, for some set P of paths in types of the form $S(\Gamma(x))$. The only such path is $\bullet^m \to \bullet$, whence $m = r$. ◀

▶ **Lemma 19.** *Let $\Gamma \vdash_K e : F(\tau_1, \ldots, \tau_4) \cap \bullet$, where $F(\tau_1, \ldots, \tau_4)$ is a predicate. Then $e = xe_1 \ldots e_m$, for some m-ary combinator x with target identifier F. More precisely, $\Gamma(x)$ has the form $\xi \cap (\zeta_1 \to \cdots \to \zeta_m \to F(\rho_1, \ldots, \rho_4))$, and there is a substitution S such that $S(\rho_i) = \tau_i$, for $i = 1, \ldots, 4$, and $\Gamma \vdash_K e_i : S(\zeta_i)$, for $i = 1, \ldots, m$.*

Proof. The term e must be of the form $e = xe_1 \ldots e_r$, where x is a combinator of some arity m in Γ. It follows from Lemma 18 that $m = r$, and from Corollary 12 we obtain that $\bigcap_{\ell \in L} \Phi_\ell \cap \bullet \leq F(\tau_1, \ldots, \tau_4) \cap \bullet$, where Φ_ℓ are predicates with the same target G. Since $\bullet$ is a constant, we actually have $\bigcap_{\ell \in L} \Phi_\ell \leq F(\tau_1, \ldots, \tau_4)$. By Lemma 17, one of Φ_ℓ must be equal to $F(\tau_1, \ldots, \tau_4)$, in particular $F = G$. Note that Φ_ℓ is obtained as $S(\mathit{tgt}_m(\phi))$, for some component ϕ of $\Gamma(x)$, and this S is the substitution required by the lemma. ◀

Numeral predicates

The declarations shown in Figure 3 and Figure 4 are included in Γ, for every $k < K$. Together they specify the way numerals are handled at each level k. The predicates are defined inductively with respect to k. Thus, in Figure 3 we define the base predicates for numerals in $\mathcal{N}_0$, whereas Figure 4 contains definitions for predicates at all higher levels $k+1$. These latter definitions may inductively refer to definitions at lower levels (for example, in Figure 4, the declaration for the combinator $\mathbf{N}_{k+1}$ refers to the lower level predicate Zero_k).

Turing machine

Now we turn to the actual machine simulation. Declarations in Figure 5 are used to "create" the initial configuration with input word $a_1 \ldots a_n$ and with further tape cells filled with blanks up to length $\exp_{K+1}(n)$. Tape cells are identified by numbers from 0 to $\exp_{K+1}(n) - 1$.

$$
\begin{array}{lcl}
\mathbf{Z}_0 & : & \mathsf{Zero}_0(0_1 \cap 0_2 \cap \cdots \cap 0_n) \cap \bullet \\
\mathbf{M}_0 & : & \mathsf{Max}_0(1_1 \cap 1_2 \cap \cdots \cap 1_n) \cap \bullet \\
 & & \\
\mathbf{N}_0 & : & [\mathsf{n}^2(\alpha) \to \mathsf{Num}_0(1_1 \cap \alpha)] \cap [\mathsf{n}^2(\alpha) \to \mathsf{Num}_0(0_1 \cap \alpha)] \cap [\bullet \to \bullet] \\
\mathbf{n}_0^2 & : & [\mathsf{n}^3(\alpha) \to \mathsf{n}^2(1_2 \cap \alpha)] \cap [\mathsf{n}^3(\alpha) \to \mathsf{n}^2(0_2 \cap \alpha)] \cap [\bullet \to \bullet] \\
\ldots & : & \ldots \\
\mathbf{n}_0^n & : & \mathsf{n}^n(1_n) \cap \mathsf{n}^n(0_n) \cap \bullet \\
 & & \\
\mathbf{D}_0 & : & [\mathsf{d}_0(\alpha,\beta) \to \mathsf{Num}_0(\alpha) \to \mathsf{Num}_0(\beta) \to \mathsf{Diff}_0(\alpha,\beta)] \cap [\bullet \to \bullet \to \bullet \to \bullet] \\
\mathbf{d}_0 & : & \bigcap_{i=1}^n (\mathsf{d}_0(0_i \cap \alpha, 1_i \cap \beta) \cap \mathsf{d}_0(1_i \cap \alpha, 0_i \cap \beta)) \cap \bullet \\
 & & \\
\mathbf{S}_0 & : & [\mathsf{Num}_0(0_1 \cap \alpha) \to \mathsf{Num}_0(1_1 \cap \alpha) \to \mathsf{Succ}_0(0_1 \cap \alpha, 1_1 \cap \alpha)] \cap \\
 & & [\mathsf{Num}_0(1_1 \cap 0_2 \cap \alpha) \to \mathsf{Num}_0(0_1 \cap 1_2 \cap \alpha) \to \mathsf{Succ}_0(1_1 \cap 0_2 \cap \alpha, 0_1 \cap 1_2 \cap \alpha)] \cap \\
 & & \ldots \cap \\
 & & [\mathsf{Num}_0(1_1 \cap 1_2 \cap \cdots \cap 1_{n-1} \cap 0_n) \to \\
 & & \quad \mathsf{Num}_0(0_1 \cap 0_2 \cap \cdots \cap 0_{n-1} \cap 1_n) \to \\
 & & \qquad \mathsf{Succ}_0(1_1 \cap 1_2 \cap \cdots \cap 1_{n-1} \cap 0_n, 0_1 \cap 0_2 \cap \cdots \cap 0_{n-1} \cap 1_n)] \cap \\
 & & [\bullet \to \bullet \to \bullet]
\end{array}
$$

Figure 3 Numeral predicates, level 0.

Before we define the core part of our coding, we introduce one more notational convention. A multiple implication $\tau_1 \to \tau_2 \to \cdots \to \tau_m \to \tau$ is sometimes written as $(\tau_1, \ldots, \tau_m) \to \tau$. We extend this style by using informal abbreviations for sequences of premises. For instance, type $\tau_1 \to \tau_2 \to \tau_3 \to \sigma_1 \to \sigma_2 \to \sigma_3 \to \tau$ may be written as $A \to B \to \tau$, where $A = (\tau_1, \tau_2, \tau_3)$ and $B = (\sigma_1, \sigma_2, \sigma_3)$.

Given q and b, let $\Delta(b,q) = \{(c_i, p_i, h_i) \mid i = 1, \ldots, r\}$. By $\mathsf{V}^{qbi}(\delta)$ and $\mathsf{U}^{qbi}(\alpha,\delta,\gamma)$ we abbreviate triples of types used to represent the transition defined by (c_i, p_i, h_i). The role of $\mathsf{V}^{qbi}(\delta)$ is to encode the action at the presently scanned tape cell, while $\mathsf{U}^{qbi}(\alpha,\delta,\gamma)$ applies to all other tape cells. Assume first that $h_i = \textsc{l}$. Then we define:

$$
\begin{array}{lcl}
\mathsf{V}^{qbi}(\delta) & = & (\mathsf{Succ}_K(\beta,\delta),\ \mathsf{Diff}_K(\xi,\zeta),\ \mathsf{Cell}(c_i,p_i,\beta,\delta)), \\
\mathsf{U}^{qbi}(\alpha,\delta,\gamma) & = & (\mathsf{Succ}_K(\beta,\delta),\ \mathsf{Diff}_K(\gamma,\delta),\ \mathsf{Cell}(\alpha,p_i,\beta,\gamma)).
\end{array}
$$

If $h_i = \textsc{r}$ then the definition is altered as follows:

$$
\begin{array}{lcl}
\mathsf{V}^{qbi}(\delta) & = & (\mathsf{Succ}_K(\delta,\beta),\ \mathsf{Diff}_K(\xi,\zeta),\ \mathsf{Cell}(c_i,p_i,\beta,\delta)), \\
\mathsf{U}^{qbi}(\alpha,\delta,\gamma) & = & (\mathsf{Succ}_K(\delta,\beta),\ \mathsf{Diff}_K(\gamma,\delta),\ \mathsf{Cell}(\alpha,p_i,\beta,\gamma)).
\end{array}
$$

Now, if q is an existential state then for every $i \leq r$ there is a combinator

$$
\begin{array}{lcl}
\mathbf{Step}^{qbi} & : & [\mathsf{V}^{qbi}(\delta) \ \to \ \mathsf{Cell}(b,q,\delta,\delta)] \cap \\
 & & [\mathsf{U}^{qbi}(\alpha,\delta,\gamma) \to \mathsf{Cell}(\alpha,q,\delta,\gamma)] \cap \\
 & & [\bullet^3 \ \to \ \bullet]
\end{array}
$$

For universal q, we declare one combinator $\mathbf{Step}^{qb}$:

$$
\begin{array}{lcl}
\mathbf{Step}^{qb} & : & [\mathsf{V}^{qb1}(\delta) \ \to \ \cdots \ \to \ \mathsf{V}^{qbr}(\delta) \ \to \ \mathsf{Cell}(b,q,\delta,\delta)] \cap \\
 & & [\mathsf{U}^{qb1}(\alpha,\delta,\gamma) \to \cdots \to \mathsf{U}^{qbr}(\alpha,\delta,\gamma) \to \mathsf{Cell}(\alpha,q,\delta,\gamma)] \cap \\
 & & [\bullet^3 \ \to \ \cdots \ \to \ \bullet^3 \ \to \ \bullet]
\end{array}
$$

$$
\begin{array}{lcl}
\mathbf{B} & : & \mathsf{Bit}(0) \cap \mathsf{Bit}(1) \cap \bullet \\
\mathbf{Z}_{k+1} & : & [\mathsf{Num}_{k+1}(\alpha) \to \mathsf{z}_{k+1}(\alpha) \to \mathsf{Zero}_{k+1}(\alpha)] \cap [\bullet \to \bullet \to \bullet] \\
\mathbf{z}_{k+1} & : & [\mathsf{z}_{k+1}(\alpha) \to \mathsf{z}_{k+1}((\beta \to 0) \cap \alpha)] \cap [\bullet \to \bullet] \\
\mathbf{z}'_{k+1} & : & \mathsf{z}_{k+1}(\beta \to 0) \cap \bullet \\
 & & \\
\mathbf{M}_{k+1} & : & [\mathsf{Num}_{k+1}(\alpha) \to \mathsf{m}_{k+1}(\alpha) \to \mathsf{Max}_{k+1}(\alpha)] \cap [\bullet \to \bullet \to \bullet] \\
\mathbf{m}_{k+1} & : & [\mathsf{m}_{k+1}(\alpha) \to \mathsf{m}_{k+1}((\beta \to 1) \cap \alpha)] \cap [\bullet \to \bullet] \\
\mathbf{m}'_{k+1} & : & \mathsf{m}_{k+1}(\beta \to 1) \cap \bullet \\
 & & \\
\mathbf{N}_{k+1} & : & [\mathsf{Bit}(\gamma) \to \mathsf{n}_{k+1}(\beta \to \gamma, \alpha) \to \mathsf{Zero}_k(\beta) \to \mathsf{Num}_{k+1}((\beta \to \gamma) \cap \alpha)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \\
\mathbf{n}_{k+1} & : & [\mathsf{Bit}(\varepsilon) \to \mathsf{Succ}_k(\beta, \delta) \to \mathsf{n}_{k+1}(\delta \to \varepsilon, \alpha) \to \mathsf{n}_{k+1}(\beta \to \gamma, (\delta \to \varepsilon) \cap \alpha)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \\
\mathbf{n}'_{k+1} & : & [\mathsf{Bit}(\varepsilon) \to \mathsf{Succ}_k(\beta, \delta) \to \mathsf{Max}_k(\delta) \to \mathsf{n}_{k+1}(\beta \to \gamma, \delta \to \varepsilon)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \\
 & & \\
\mathbf{D}_{k+1} & : & [\mathsf{d}_{k+1}(\alpha, \beta) \to \mathsf{Num}_{k+1}(\alpha) \to \mathsf{Num}_{k+1}(\beta) \to \mathsf{Diff}_{k+1}(\alpha, \beta)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \\
\mathbf{d}_{k+1} & : & \mathsf{d}_{k+1}((\gamma \to 1) \cap \alpha, (\gamma \to 0) \cap \beta) \cap \mathsf{d}_{k+1}((\delta \to 0) \cap \alpha, (\delta \to 1) \cap \beta) \cap \bullet \\
 & & \\
\mathbf{S}_{k+1} & : & [\mathsf{R}_{k+1}(\beta, \alpha, \gamma) \to \mathsf{Zero}_k(\beta) \to \mathsf{Succ}_{k+1}((\beta \to 0) \cap \alpha, (\beta \to 1) \cap \gamma)] \cap \\
 & & [\mathsf{L}_{k+1}(\beta, \alpha, \gamma) \to \mathsf{Zero}_k(\beta) \to \mathsf{Succ}_{k+1}((\beta \to 1) \cap \alpha, (\beta \to 0) \cap \gamma)] \cap \\
 & & [\bullet \to \bullet \to \bullet] \\
\mathbf{s}_{k+1} & : & [\mathsf{Succ}_k(\beta, \delta) \to \mathsf{L}_{k+1}(\delta, \alpha, \gamma) \to \mathsf{L}_{k+1}(\beta, (\delta \to 1) \cap \alpha, (\delta \to 0) \cap \gamma)] \cap \\
 & & [\mathsf{Succ}_k(\beta, \delta) \to \mathsf{R}_{k+1}(\delta, \alpha, \gamma) \to \mathsf{L}_{k+1}(\beta, (\delta \to 0) \cap \alpha, (\delta \to 1) \cap \gamma)] \cap \\
 & & [\mathsf{Succ}_k(\beta, \delta) \to \mathsf{R}_{k+1}(\delta, \alpha, \gamma) \to \mathsf{R}_{k+1}(\beta, (\delta \to 0) \cap \alpha, (\delta \to 0) \cap \gamma)] \cap \\
 & & [\mathsf{Succ}_k(\beta, \delta) \to \mathsf{R}_{k+1}(\delta, \alpha, \gamma) \to \mathsf{R}_{k+1}(\beta, (\delta \to 1) \cap \alpha, (\delta \to 1) \cap \gamma)] \cap \\
 & & [\bullet \to \bullet \to \bullet] \\
\mathbf{s}'_{k+1} & : & [\mathsf{Max}_k(\delta) \to \mathsf{Succ}_k(\beta, \delta) \to \mathsf{R}_{k+1}(\beta, \delta \to 0, \delta \to 0)] \cap \\
 & & [\mathsf{Max}_k(\delta) \to \mathsf{Succ}_k(\beta, \delta) \to \mathsf{R}_{k+1}(\beta, \delta \to 1, \delta \to 1)] \cap \\
 & & [\mathsf{Max}_k(\delta) \to \mathsf{Succ}_k(\beta, \delta) \to \mathsf{L}_{k+1}(\beta, \delta \to 0, \delta \to 1)] \cap \\
 & & [\bullet \to \bullet \to \bullet]
\end{array}
$$

Figure 4 Numeral predicates, level $k + 1$.

Properties of the coding

We now collect the main properties of our coding. The first two lemmas state that our numeral system works properly.

▶ **Lemma 20.** *For every $k \le K$ there are terms $Zero_k$, Max_k, Num_k, $Diff_k$, $Succ_k$, such that for all types σ and τ:*

1. *If $\sigma = \langle 0 \rangle_k$ then $\Gamma \vdash_K Zero_k : \mathsf{Zero}_k(\sigma) \cap \bullet$.*
2. *If $\sigma = \langle \exp_{k+1}(n) - 1 \rangle_k$ then $\Gamma \vdash_K Max_k : \mathsf{Max}_k(\sigma) \cap \bullet$.*
3. *If $\sigma \in \mathcal{N}_k$ then $\Gamma \vdash_K Num_k : \mathsf{Num}_k(\sigma) \cap \bullet$.*
4. *If $\sigma, \tau \in \mathcal{N}_k$, and $[\![\sigma]\!] \neq [\![\tau]\!]$ then $\Gamma \vdash_K Diff_k : \mathsf{Diff}_k(\sigma, \tau) \cap \bullet$.*
5. *If $\sigma, \tau \in \mathcal{N}_k$, and $[\![\sigma]\!] + 1 = [\![\tau]\!]$ then $\Gamma \vdash_K Succ_k : \mathsf{Succ}_k(\sigma, \tau) \cap \bullet$.*

$$
\begin{array}{lcl}
\mathbf{Init} & : & [\mathsf{Zero}_K(\alpha) \to \mathsf{Cell}(a_1,\ q_0,\ \alpha,\ \alpha) \cap \mathsf{Tape}^1(\alpha) \to \mathsf{Tape}] \cap [\bullet \to \bullet \to \bullet] \\
\mathbf{init}^i & : & [\mathsf{Zero}_K(\gamma) \to \mathsf{Succ}_K(\alpha,\beta) \to \mathsf{Tape}^{i+1}(\beta) \cap \mathsf{Cell}(a_i,\ q_0,\ \gamma,\ \beta) \to \mathsf{Tape}^i(\alpha)] \cap \\
 & & [\mathsf{Zero}_K(\gamma) \to \mathsf{Succ}_K(\alpha,\beta) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \qquad\qquad\qquad\qquad\qquad\qquad \text{(for all } i < n) \\
\mathbf{init}^n & : & [\mathsf{Zero}_K(\gamma) \to \mathsf{Succ}_K(\alpha,\beta) \to \mathsf{Tape}^n(\beta)\ \cap\ \mathsf{Cell}(\sqcup,\ q_0,\ \gamma,\ \beta)\ \to \mathsf{Tape}^n(\alpha)] \cap \\
 & & [\mathsf{Zero}_K(\gamma) \to \mathsf{Succ}_K(\alpha,\beta) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon)] \cap \\
 & & [\bullet \to \bullet \to \bullet \to \bullet] \\
\mathbf{finit} & : & [\mathsf{Max}_K(\alpha) \to \bullet \to \mathsf{Tape}^n(\alpha)] \cap \\
 & & [\mathsf{Max}_K(\alpha) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon) \to \mathsf{Cell}(\eta,\ q_0,\ \delta,\ \varepsilon)]\ \cap [\bullet \to \bullet \to \bullet]
\end{array}
$$

Figure 5 Initial configuration under construction.

Proof. Beginning with $k = 0$, we have $\mathit{Num}_0 = \mathbf{N}_0(\mathbf{n}_0^2(\mathbf{n}_0^3(\ldots(\mathbf{n}_0^{n-1}(\mathbf{n}_0^n))\ldots)))$, $\mathit{Zero}_0 = \mathbf{Z}_0$, $\mathit{Max}_0 = \mathbf{M}_0$, $\mathit{Diff}_0 = \mathbf{D}_0\mathbf{d}_0\mathit{Num}_0\mathit{Num}_0$, and $\mathit{Succ}_0 = \mathbf{S}_0\mathit{Num}_0\mathit{Num}_0$. Take $\mathit{max} = \exp_{k+1}(n)$ and for $k \geq 0$ define $\mathit{Num}_{k+1} = \mathbf{N}_{k+1}\mathbf{B}((\mathbf{n}_{k+1}\mathbf{B}\mathit{Succ}_k)^{\mathit{max}-2}(\mathbf{n}'_{k+1}\mathbf{B}\mathit{Succ}_k\mathit{Max}_k))\mathit{Zero}_k$, $\mathit{Zero}_{k+1} = \mathbf{Z}_{k+1}\mathit{Num}_{k+1}(\mathbf{z}_{k+1}^{\mathit{max}-1}(\mathbf{z}'_{k+1}))$, and $\mathit{Max}_{k+1} = \mathbf{M}_{k+1}\mathit{Num}_{k+1}(\mathbf{m}_{k+1}^{\mathit{max}-1}(\mathbf{m}'_{k+1}))$. Now we can define successor $\mathit{Succ}_{k+1} = \mathbf{S}_{k+1}((\mathbf{s}_{k+1}\mathit{Succ}_k)^{\mathit{max}-2}(\mathbf{s}'_{k+1}\mathit{Max}_k\mathit{Succ}_k))\mathit{Zero}_k$, and the last term we need is $\mathit{Diff}_{k+1} = \mathbf{D}_{k+1}\mathbf{d}_{k+1}\mathit{Num}_{k+1}\mathit{Num}_{k+1}$. ◀

▶ **Lemma 21.** *For every $k \leq K$ and every e:*

1. *If* $\Gamma \vdash_K e : \mathsf{Zero}_k(\sigma) \cap \bullet$ *then* $\sigma = \langle\, 0 \,\rangle_k$.
2. *If* $\Gamma \vdash_K e : \mathsf{Max}_k(\sigma) \cap \bullet$ *then* $\sigma = \langle \exp_{k+1}(n) - 1 \,\rangle_k$.
3. *If* $\Gamma \vdash_K e : \mathsf{Num}_k(\sigma) \cap \bullet$ *then* $\sigma \in \mathcal{N}_k$.
4. *If* $\Gamma \vdash_K e : \mathsf{Diff}_k(\sigma,\tau) \cap \bullet$ *then* $\sigma, \tau \in \mathcal{N}_k$*, and* $[\![\sigma]\!] \neq [\![\tau]\!]$.
5. *If* $\Gamma \vdash_K e : \mathsf{Succ}_k(\sigma,\tau) \cap \bullet$ *then* $\sigma, \tau \in \mathcal{N}_k$*, and* $[\![\sigma]\!] + 1 = [\![\tau]\!]$.

Proof. The proof is by induction with respect to k, and we show the five claims in the order of their numbers. Of the ten possible cases we consider $\Gamma \vdash_K e : \mathsf{Num}_{k+1}(\sigma) \cap \bullet$ as an example. It follows from Lemma 19 that $e = \mathbf{N}_{k+1}e_1e_2e_3$, and $\sigma = S((\beta \to \gamma) \cap \alpha)$ and we can derive $\Gamma \vdash_K e_1 : \mathsf{Bit}(S(\gamma)) \cap \bullet$, $\Gamma \vdash_K e_2 : \mathbf{n}_{k+1}(S(\beta \to \gamma), S(\alpha)) \cap \bullet$, and $\Gamma \vdash_K e_3 : \mathbf{Zero}_{k+1}(S(\beta)) \cap \bullet$, for some S. Then $S(\beta) = \langle\, 0 \,\rangle_k$ and $S(\gamma)$ is 0 or 1. We prove by induction that $\Gamma \vdash_K e' : \mathbf{n}_{k+1}(\varphi, \tau)$ implies $\varphi = \langle\, i \,\rangle_k \to \varphi'$ and $\tau = \bigcap_{j>i}\langle\, j \,\rangle_k \to b_j$, for some i, and conclude that $\sigma = \bigcap_{j\geq 0}\langle\, j \,\rangle_k \to b_j$, i.e., that σ is indeed a numeral. ◀

Let $\mathcal{C} = wqw'$ be a configuration of our machine $\mathcal{M}$. Assume that $w = b_0 \ldots b_{h-1}$ and $w' = b_h \ldots b_{\exp_{K+1}(n)-1}$. That is, the address of the currently scanned tape cell is h. We take the following type to be the encoding of $\mathcal{C}$:

$$[\mathcal{C}] \quad = \quad \bigcap_{i=0}^{\exp_{K+1}(n)-1} \mathsf{Cell}(b_i, q, \langle h \rangle_K, \langle i \rangle_K).$$

Now let $\mathcal{C}_0$ be the initial configuration for input $a_1 \ldots a_n$. (Thus $b_i = a_{i+1}$, for $i < n$.)

▶ **Lemma 22.** *The intersection* $\mathsf{Tape} \cap \bullet$ *is inhabited in* Γ *iff so is* $[\mathcal{C}_0] \cap \bullet$.

Proof. Suppose that $\Gamma \vdash e : [\mathcal{C}_0] \cap \bullet$. If $T_i = \mathbf{init}^i\,\mathit{Zero}_K\mathit{Succ}_K$, for $i = 1, \ldots, n$, then

$$\Gamma \vdash \mathbf{Init}\,\mathit{Zero}_K(T_1(T_2(\cdots(T_{n-1}(T_n^{m-n}(\mathbf{finit}\,\mathit{Max}_K\,e)))\cdots))) : \mathsf{Tape} \cap \bullet.$$

On the other hand, if $\Gamma \vdash e : \mathsf{Tape} \cap \bullet$ then $e = xe_1 \dots e_m$, where x is m-ary (Lemma 18). Since $\Gamma \vdash e : \mathsf{Tape}$, the only possibility is that $e = \mathbf{Init}\, e_1 e_2$, where $\Gamma \vdash e_1 : \mathit{Zero}_K(\langle 0 \rangle_K)$ and $\Gamma \vdash e_2 : \mathsf{Tape}^1(\langle 0 \rangle_K) \cap \mathsf{Cell}(a_1, q_0, \langle 0 \rangle_K, \langle 0 \rangle_K \rangle$. We prove by induction wrt r that $e_2 = T_1(T_2(\cdots(e')\cdots))$, where e' has type $\bullet \cap \mathsf{Tape}^\ell(\langle r \rangle) \cap \bigcap_{i \leq r} \mathsf{Cell}(b_i, q_0, \langle 0 \rangle, \langle i \rangle)$, and $\ell = \min\{r+1, n\}$. For $r = \exp_{K+1}(n) - 1$, term e' is of type $\bullet \cap \mathsf{Tape}^n(\langle r \rangle_K) \cap [C_0]$. ◀

▶ **Lemma 23.** *A configuration $\mathcal{C}$ is eventually accepting iff* $\Gamma \vdash [\mathcal{C}] \cap \bullet$.

Proof. The "only if" part goes by induction with respect to the definition of acceptance. If $\mathcal{C}$ is an accepting configuration (universal without successors) then we have a declaration

$$\mathbf{Step}^{q_a b} \ : \ \mathsf{Cell}(b, q_a, \delta, \delta) \ \cap \ \mathsf{Cell}(\alpha, q_a, \delta, \gamma) \ \cap \ \bullet,$$

for appropriate b, whence $\Gamma \vdash \mathbf{Step}^{q_a b} : [\mathcal{C}] \cap \bullet$. Let $\mathcal{C} = wqbw'$ be existential, with q at address t. If $\mathcal{C} \to \mathcal{C}'$, with $\mathcal{C}'$ eventually accepting then, by the induction hypothesis, $[\mathcal{C}'] \cap \bullet$ is inhabited. Assume for example that $\mathcal{C}'$ is obtained from $\mathcal{C}$ using a triple $(c_i, p_i, h_i) \in \Delta(b, q)$, with $h_i = \textsc{l}$. Then $[\mathcal{C}']$ differs from $[\mathcal{C}]$ in that we have $\mathsf{Cell}(c_i, p_i, \langle t-1 \rangle, \langle t \rangle)$ instead of $\mathsf{Cell}(b, q, \langle t \rangle, \langle t \rangle)$ and $\mathsf{Cell}(b_j, p_i, \langle t-1 \rangle, \langle j \rangle)$ instead of $\mathsf{Cell}(b_j, q, \langle t \rangle, \langle j \rangle)$, for all $j \neq t$. It follows that $\Gamma \vdash \mathbf{Step}^{qbi} \mathit{Succ}_k \mathit{Diff}_k\ e : [\mathcal{C}] \cap \bullet$, where Succ_k and Diff_k are defined as in Lemma 20 for appropriate k, and e is an inhabitant of $[\mathcal{C}'] \cap \bullet$.

In the universal case, we build an inhabitant of $[\mathcal{C}] \cap \bullet$ as

$$\mathbf{Step}^{qb} \mathit{Succ}_k \mathit{Diff}_k\ e_1 \dots \mathit{Succ}_k \mathit{Diff}_k\ e_r$$

where Succ_k and Diff_k are as above, and $e_1, \dots, e_r$ prove the codes of all successor configurations.

The proof from right to left is by induction with respect to length of inhabitants. Let $\Gamma \vdash e : [\mathcal{C}] \cap \bullet$. If e is a single combinator then $e = \mathbf{Step}^{q_a b}$, by Lemma 19. Otherwise $e = xe_1 \dots e_m$, for an m-ary x. It is possible that $e = \mathbf{init}^i e_0 e_1 e_2$ or $\mathbf{finit}\, e_1 e_2$, but then e_2 also proves $[\mathcal{C}] \cap \bullet$. Therefore the shortest inhabitant must begin with $\mathbf{Step}^{qbi}$ or $\mathbf{Step}^{qb}$, and we proceed as in the proof of Lemmas 21 and 22, using Lemma 19 as a basic tool. ◀

▶ **Theorem 24.** *For every $k \geq 0$, the relativized inhabitation problem for* $\textsc{bcl}_k(\to, \cap)$ *is complete in* $(k{+}2)$-EXPTIME.

Proof. By a routine padding argument[3] it suffices to prove that the halting problem for $\exp_{k+1}(n)$-space bounded ATM's is reducible to inhabitation in $\textsc{bcl}_k(\to, \cap)$. The latter claim follows from Lemmas 22 and 23: to determine if $\mathcal{M}$ accepts the input it is enough to ask if $\Gamma \vdash \bullet \cap \mathsf{Tape}$. ◀

References

1 H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.

2 T. Freeman and F. Pfenning. Refinement types for ML. In *ACM Conference on Programming Language Design and Implementation (PLDI)*, pages 268–277. ACM, 1991.

3 M. D. Gladstone. Some ways of constructing a propositional calculus of any required degree of unsolvability. *Transactions of the American Mathematical Society*, 118:195–210, 1965.

[3] If $L \in \mathrm{DTIME}(\exp_{k+1}(p(n)))$ then $L \leq_{\log} \{w\#^{p(n)-|w|} \mid w \in L\} \in \mathrm{DTIME}(\exp_{k+1}(n))$.

4 J. R. Hindley. The simple semantics for Coppo-Dezani-Sallé types. In M. Dezani-Ciancaglini and U. Montanari, editors, *International Symposium on Programming*, volume 137 of *LNCS*, pages 212–226. Springer, 1982.

5 J. R. Hindley and J. P. Seldin. *Lambda-calculus and Combinators, an Introduction.* Cambridge University Press, 2008.

6 L. Linial and E. L. Post. Recursive unsolvability of the deducibility, Tarski's completeness and independence of axioms problems of propositional calculus. *Bulletin of the American Mathematical Society*, 55:50, 1949.

7 Ch. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

8 J. Rehof and P. Urzyczyn. Finite combinatory logic with intersection types. In C.-H. Luke Ong, editor, *TLCA*, volume 6690 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2011.

9 W. E. Singletary. Recursive unsolvability of a complex of problems proposed by Post. *Journal of the Faculty of Science, University of Tokyo*, 14:25–58, 1967.

10 W. E. Singletary. Many-one degrees associated with partial propositional calculi. *Notre Dame Journal of Formal Logic*, XV(2):335–343, 1974.

11 R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theoretical Computer Science*, 9:67–72, 1979.

Collapsing non-idempotent intersection types

Thomas Ehrhard

CNRS, PPS, UMR 7126, Univ Paris Diderot, Sorbonne Paris Cité
F–75205 Paris, France

Abstract

We proved recently that the extensional collapse of the relational model of linear logic coincides with its Scott model, whose objects are preorders and morphisms are downwards closed relations. This result is obtained by the construction of a new model whose objects can be understood as preorders equipped with a realizability predicate. We present this model, which features a new duality, and explain how to use it for reducing normalization results in idempotent intersection types (usually proved by reducibility) to purely combinatorial methods. We illustrate this approach in the case of the call-by-value lambda-calculus, for which we introduce a new resource calculus, but it can be applied in the same way to many different calculi.

1998 ACM Subject Classification F.3.2

Keywords and phrases Linear logic, λ-calculus, denotational semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.259

1 Introduction

The relational model of linear logic (LL) has been introduced implicitly by Girard in [12] as a model of the λ-calculus and recognized only later as a model of LL by several authors independently. Its objects are plain sets and a morphism from X to Y is a subset of $X \times Y$. Often despised because it identifies many logical constructions of LL (most dramatically $X^\perp = X$), this model is nevertheless extremely interesting as it preserves many relevant information about programs: it is *quantitative* in the sense that the interpretation of functions allows to recover how many times an argument is used to compute a given result. For that reason, computation time can be recovered from the interpretation of terms, as shown in [3, 4]. Arbitrary fixpoints of types are quite easy to compute and therefore many interesting relational models of the pure λ-calculus and of its variants and extensions are available: call-by-value (cbv) λ-calculus, $\lambda\mu$- calculus etc. Also, the relational model provides a natural interpretation of the differential and resource λ-calculi and LL [7, 8, 9, 21, 19].

Scott semantics is of course older. It has been recognized as a model of LL a few years after Girard's discovery of LL, by Michael Huth [13, 14] and independently by Glynn Winskel [22, 23]. In this model, types are interpreted as prime algebraic complete lattices, or equivalently as preorders, since any such lattice can be presented as the set of downwards closed subsets of a preorder. The Kleisli category associated with this model of LL is (equivalent to) the category of prime-algebraic complete lattices and Scott-continuous functions. This model forgets much more information about programs than the relational model: it is purely *qualitative* in the sense that the interpretation of a function tells which parts of the arguments are used to compute a given result, but not how many times they are used.

This difference between the relational model and the Scott model of LL materializes itself in the fact that the Kleisli category of the second model is well-pointed (intuitively: morphisms can be seen as functions), whereas the Kleisli category of the first model is not. We proved in [6] that the the second model is the *extensional collapse* of the first one. The extensional collapse logical relation is a partial equivalence relation (PER) which equates morphisms if they yield equal results when applied to equal arguments. We explain now briefly how we proved that the second model is the "quotient" of the first one by the extensional collapse PER.

An object in **Rel** (the relational model) is a plain set and an object in **Pol** (the preorder model) is a structure $S = (|S|, \leq_S)$ where $|S|$ is a set (the web) and $\leq_S$ is a preorder relation on $|S|$. We can define the Scott semantics of LL in such a way that the web of the **Pol** object interpreting a formula coincide with the **Rel** object (set) interpreting the same formula. Using this fact, we build a new model of LL whose objects – called preorder with projections (pop) – are pairs $E = (\langle E\rangle, \mathsf{D}(E))$ where $\langle E\rangle$ is a preorder and $\mathsf{D}(E)$ is a subset of $\mathcal{P}(|\langle E\rangle|)$ which satisfies a closure property defined by an orthogonality relation. This allows to define a PER on $\mathcal{P}(|\langle E\rangle|)$: u and v are E-equivalent if they both belong to $\mathsf{D}(E)$ and have the same $\langle E\rangle$-downwards closure. This PER coincides with the extensional collapse PER: given pops E and F, the pop $G = (E \Rightarrow F) = (!E \multimap F)$ is such that $w, w' \subseteq |\langle G\rangle|$ are G-equivalent iff, for any $u, u' \subseteq |\langle E\rangle|$, if u and u' are E-equivalent, then $w(u)$ and $w'(u')$ are F-equivalent (applications are computed in the relational model). The $\langle G\rangle$-downwards closure of w is a morphism in the Scott model, which represents the equivalence class of w. Since conversely any downwards closed subset of $|\langle G\rangle|$ belongs to $\mathsf{D}(G)$, the Scott model coincides with the extensional collapse of the relational model. The constructions of [6] allow also to extend this result to arbitrary fixpoints of types.

Content

Indeed, just as **Rel** and **Pol**, this new category **Pop** is a model of LL where all types have least fixpoints, for a suitable notion of inclusion between pops. In the present paper, we use this property to prove an adequacy result for a Scott model of the cbv λ-calculus, which is defined as the least fixpoint $\mathcal{U}_{\mathsf{S}}$ (for a suitable order relation on preorders) of the operation $S \mapsto\ !S \multimap\ !S$. We can solve the same domain equation in **Pop** and we get an object $\mathcal{U}_{\mathsf{P}}$ for which $\langle \mathcal{U}_{\mathsf{P}}\rangle = \mathcal{U}_{\mathsf{S}}$ and $\mathcal{U}_{\mathsf{R}} = |\langle \mathcal{U}_{\mathsf{P}}\rangle|$ satisfies $\mathcal{U}_{\mathsf{R}} = !\mathcal{U}_{\mathsf{R}} \multimap !\mathcal{U}_{\mathsf{R}}$ in **Rel**. Given a term M of the cbv λ-calculus, that we assume to be closed for simplicity, we can therefore compute its relational interpretation $[M]_{\mathsf{R}}$ which is a subset of $\mathcal{U}_{\mathsf{R}}$ *which belongs to* $\mathsf{D}(\mathcal{U}_{\mathsf{P}})$ and its Scott interpretation $[M]_{\mathsf{S}}$, which is a downwards closed subset of $\mathcal{U}_{\mathsf{R}} = |\langle \mathcal{U}_{\mathsf{P}}\rangle|$ (for the preorder relation of $\langle \mathcal{U}_{\mathsf{P}}\rangle = \mathcal{U}_{\mathsf{S}}$). By induction on M, and using crucially the properties of the model **Pop**, one proves that $[M]_{\mathsf{S}} = \downarrow[M]_{\mathsf{R}}$: this is an instance of the "extensional collapse property" of this model. Now, adequacy of $\mathcal{U}_{\mathsf{R}}$ for the cbv λ-calculus (that is: if $[M]_{\mathsf{S}} \neq \emptyset$ then M reduces to a value) can be proved purely combinatorially, introducing a cbv resource λ-calculus and the fact that **Rel** satisfies a version of the Taylor formula. If $[M]_{\mathsf{S}} \neq \emptyset$ then $[M]_{\mathsf{R}} \neq \emptyset$ since $[M]_{\mathsf{S}} = \downarrow[M]_{\mathsf{R}}$ and so M reduces to a value. Whereas the standard proofs of this kind of results for Scott semantics are based on reducibility (with the noticeable exception of [2]), the present approach provides a purely semantical reduction of this result to a combinatorial argument: all the reducibility argument has been encapsulated in the model **Pop**. This approach can be used in the same way for many different calculi (standard λ-calculus, PCF, $\lambda\mu$-calculus…).

This work can be understood as relating usual idempotent intersection typing systems – points in the **Pol** model can be seen as idempotent intersection types – with non-idempotent ones, which use points of the **Rel** model as types. We adopt this viewpoint in Sections 6.1 and 7.1 where we present the semantics of the cbv λ-calculus under consideration as typing systems. Actually, in both systems, types are pairs (p, q) where p and q are finite multisets of types but the systems differ by their typing rules. The type (p, q) could also nicely be written $p \multimap q$ and "intersection" corresponds to multiset concatenation.

Notations. We use $[a_1,\dots,a_n]$ for the multiset made of $a_1,\dots,a_n$, taking multiplicities into account. We use $[]$ for the empty multiset and standard algebraic notations such as $m+m'$ of $\sum_i m_i$ for sums of multisets. We use $|m|$ for the support of the multiset m, which is the set of elements which appear at least once in m.

2 Categorical semantics of LL in a nutshell

Our main reference for categorical models of LL is the survey paper [16].

Let $\mathcal{C}$ be a Seely category. We recall briefly that such a structure consists of a category $\mathcal{C}$, whose morphisms should be thought of as linear maps, equipped with a symmetric monoidal structure for which it is closed and $*$-autonomous wrt. a dualizing object $\perp$. The monoidal product (tensor product) is denoted as $\otimes$, the linear function space object from X to Y is denoted as $X \multimap Y$. We use $\mathsf{ev} \in \mathcal{C}((X \multimap Y) \otimes X, Y)$ for the linear evaluation morphism and $\lambda(f) \in \mathcal{C}(Z, X \multimap Y)$ for the "linear curryfication" of a morphism $f \in \mathcal{C}(Z \otimes X, Y)$. The dual object $X \multimap \perp$ is denoted as $X^\perp$. Given an object X of $\mathcal{C}$ and a permutation $f \in \mathfrak{S}_n$, we use σ_f for the induced automorphism of $X^{\otimes n}$ in $\mathcal{C}$; the operation $f \to \sigma_f$ is a group homomorphism from $\mathfrak{S}_n$ to the group of automorphisms of $X^{\otimes n}$ in $\mathcal{C}$.

We also assume that $\mathcal{C}$ is cartesian, with a cartesian product denoted as $\&$ and a terminal object $\top$. By $*$-autonomy, this implies that $\mathcal{C}$ is also cocartesian; we use $\oplus$ for the coproduct and 0 for the initial object. If $\mathcal{C}$ has cartesian products of all countable families $(X_i)_{i\in I}$ of objects, we say that it is *countably cartesian*, and in that case, $\mathcal{C}$ is also countably cocartesian. If finite sums and finite products coincide, then each hom-set has a canonical commutative monoid structure and all operations defined so far (composition, tensor product, linear curryfication) are linear wrt. this structure. In that case we say that $\mathcal{C}$ is *additive*. We say that it is *countably additive* if this property extends to countable sums and products, and in that case hom-sets have countable sums. The corresponding operations are denoted using the standard mathematical notations for sums.

Last, we assume that $\mathcal{C}$ is equipped with an endofunctor $!\_$ which has a structure of comonad (unit $\mathsf{d}_X \in \mathcal{C}(!X, X)$ called *dereliction*, multiplication $\mathsf{p}_X \in \mathcal{C}(!X, !!X)$ called *digging*). Moreover, this functor must be equipped with a monoidal structure which turns it into a symmetric monoidal functor from the symmetric monoidal category $(\mathcal{C}, \&)$ to the symmetric monoidal category $(\mathcal{C}, \otimes)$: the corresponding isomorphisms $\mathsf{m} : 1 \to !\top$ and $\mathsf{m}_{X,Y} : !X \otimes !Y \to !(X \mathbin{\&} Y)$ are often called *Seely isomorphisms*. An additional diagram, relating digging and the Seely isomorphisms is required, see [16].

2.1 Structural natural transformations

Using these structures, we can define a *weakening* natural transformation $\mathsf{w}_X \in \mathcal{C}(!X, 1)$ and a *contraction* natural transformation $\mathsf{c}_X \in \mathcal{C}(!X, !X \otimes !X)$ as follows. Since $\top$ is terminal, there is a canonical morphism $\mathsf{t}_X \in \mathcal{C}(X, \top)$ and we set $\mathsf{w}_X = \mathsf{m}^{-1}\, !\mathsf{t}_X$. Similarly, we have a diagonal natural transformation $\Delta_X \in \mathcal{C}(X, X \mathbin{\&} X)$ and we set $\mathsf{c}_X = \mathsf{m}_{X,X}^{-1}\, !\Delta_X$.

One can also prove that the Kleisli category $\mathcal{C}_!$ of the comonad $!\_$ is cartesian closed, with $\&$ as cartesian product and $!X \multimap Y$ as function space object: this is a categorical version of Girard's translation of intuitionistic logic into linear logic.

We use $\mathsf{c}_X^n : !X^{\otimes n} \to !X^{\otimes n} \otimes !X^{\otimes n}$ for the generalized contraction morphism which is defined as the composition $(!X)^{\otimes n} \xrightarrow{(\mathsf{c}_X)^{\otimes n}} (!X \otimes !X)^{\otimes n} \xrightarrow{\sigma_f} (!X)^{\otimes n} \otimes (!X)^{\otimes n}$ where $f \in \mathfrak{S}_{2n}$ is the bijection which maps $2k-1$ to k and $2k$ to $n+k$ (for $k = 1,\dots,n$).

Similarly, we define a generalized weakening morphism w_X^n as the composition of morphisms $(!X)^{\otimes n} \xrightarrow{(\mathsf{w}_X)^{\otimes n}} (1)^{\otimes n} \xrightarrow{\nu} 1$ where ν is the unique canonical isomorphism induced by the monoidal structure. Given $f \in \mathcal{C}((!X)^{\otimes n}, X)$, it is standard to define $f^! \in \mathcal{C}((!X)^{\otimes n}, !X)$, using the comonad and the monoidal structure of the functor $!\_$. This operation is usually called *promotion* in LL. Given two LL models $\mathcal{C}$ and $\mathcal{D}$, an *LL functor* from $\mathcal{C}$ to $\mathcal{D}$ is a functor $F : \mathcal{C} \to \mathcal{D}$ which preserves all the structure defined above. For instance, we must have $F(f \otimes g) = F(f) \otimes F(g)$, $F(\mathsf{p}_X) = \mathsf{p}_{F(X)}$ etc.

2.2 Weak differential LL models

The notion of categorical model recalled above allows to interpret standard classical linear logic. If one wishes to interpret differential constructs as well (in the spirit of the differential λ-calculus or of differential linear logic), more structure and hypotheses are required. Basically, we need:

- that the cartesian and cocartesian category $\mathcal{C}$ be additive
- and that the model be equipped with a *codereliction* natural transformation $\overline{\mathsf{d}}_X \in \mathcal{C}(X, !X)$ such that $\mathsf{d}_X\, \overline{\mathsf{d}}_X = \mathrm{Id}_X$.

More conditions are required if one wants to interpret the full differential λ-calculus of [7] or full differential LL as presented in e.g. [18]: these conditions are a categorical axiomatization of the usual chain rule of calculus, but this rule is not required in the present paper, see [10] for a complete axiomatization. When these additional conditions hold, we say that the chain rule holds in $\mathcal{C}$.

If $\mathcal{C}$ is a weak differential LL model, we can define a coweakening morphism $\overline{\mathsf{w}}_X \in \mathcal{C}(1, !X)$ and a cocontraction morphism $\overline{\mathsf{c}}_X \in \mathcal{C}(!X \otimes !X, !X)$ as we did for w_X and c_X. Similarly we also define $\overline{\mathsf{c}}_X^n \in \mathcal{C}((!X)^{\otimes n}, !X)$. Due to the naturality of $\overline{\mathsf{d}}_X$ we have $\mathsf{w}_X\, \overline{\mathsf{d}}_X = 0$ and $\mathsf{c}_X\, \overline{\mathsf{d}}_X = \overline{\mathsf{d}}_X \otimes \overline{\mathsf{w}}_X + \overline{\mathsf{w}}_X \otimes \overline{\mathsf{d}}_X$. We also define $\mathsf{d}_X^n = \mathsf{d}_X^{\otimes n}\, \mathsf{c}_X^n \in \mathcal{C}(!X, X^{\otimes n})$ and $\overline{\mathsf{d}}_X^n = \overline{\mathsf{c}}_X^n\, \overline{\mathsf{d}}_X^{\otimes n} \in \mathcal{C}(X^{\otimes n}, !X)$.

2.3 The Taylor formula

Let $\mathcal{C}$ be a weak differential LL model which is countably additive. Remember that each hom-set $\mathcal{C}(X, Y)$ is endowed with a canonical structure of commutative monoid in which countable families are summable. We assume moreover that these monoids are idempotent. This means that, if $f \in \mathcal{C}(X, Y)$, then $f + f = f$. We say that *the Taylor formula holds in* $\mathcal{C}$ if, for any morphism $f \in \mathcal{C}(X, Y)$, we have $!f = \sum_{n=0}^{\infty} \overline{\mathsf{d}}_Y^n\, f^{\otimes n}\, \mathsf{d}_X^n$

▶ Remark. If the idempotency condition does not hold in $\mathcal{C}$, one has to require the hom-sets to have a module structure over the rig of non-negative real numbers, and the Taylor condition must be written in the more familiar way $!f = \sum_{n=0}^{\infty} \frac{1}{n!} \overline{\mathsf{d}}_Y^n\, f^{\otimes n}\, \mathsf{d}_X^n$.

▶ Remark. If the chain rule holds in $\mathcal{C}$, the Taylor condition reduces to the particular case of identity morphisms: one has just to require that $\mathrm{Id}_{!X} = \sum_{n=0}^{\infty} \frac{1}{n!} \overline{\mathsf{d}}_X^n \mathsf{d}_X^n$ (in the non-idempotent case) or $\mathrm{Id}_{!X} = \sum_{n=0}^{\infty} \overline{\mathsf{d}}_X^n \mathsf{d}_X^n$ (in the idempotent case).

3 The extensional collapse

We present the extensional collapse construction developed in [6].

3.1 The relational model of LL

The model. The base category is **Rel**, the category of sets and relations. Identities are diagonal relations and composition is the standard composition of relations. In this category, the isomorphisms are the bijections. The symmetric monoidal structure is given by $1 = \{*\}$ (arbitrary singleton set) and $X \otimes Y = X \times Y$, we do not give the monoidal isomorphisms which are obvious. This symmetric monoidal category (SMC) is closed, with $X \multimap Y = X \times Y$ and $\mathsf{ev} = \{(((a,b),a),b) \mid a \in X \text{ and } b \in Y\}$. It is $*$-autonomous with dualizing object $\bot = 1$ so that $X^\bot = X$ up to an obvious isomorphism.

Rel is countably cartesian with $\&_{i\in I} X_i = \bigcup_{i\in I}\{i\} \times X_i$ (disjoint union) and projections $\pi_i = \{((i,a),a) \mid a \in X_i\}$. It is also countably additive with $\bigoplus_{i\in I} X_i = \&_{i\in I} X_i$. The sum of a countable family of elements of $\mathbf{Rel}(X,Y)$ is its union, so that hom-sets are idempotent monoids.

The exponential functor is given by $!X = \mathcal{M}_{\mathrm{fin}}(X)$ (finite multisets of elements of X) and, if $R \in \mathbf{Rel}(X,Y)$, one sets $!R = \{([a_1,\dots,a_n],[b_1,\dots,b_n]) \mid n \in \mathbb{N} \text{ and } (a_1,b_1),\dots,(a_n,b_n) \in R\}$. The Seely isomorphism $\mathsf{m} \in \mathbf{Rel}(1,!\top)$ is $\{(*,[])\}$ and the Seely natural isomorphism $\mathsf{m}_{X,Y} \in \mathbf{Rel}(!X \otimes !Y, !(X \& Y))$ is the bijection which maps $([a_1,\dots,a_n],[b_1,\dots,b_p])$ to $[(1,a_1),\dots,(1,a_n),(2,b_1),\dots,(2,b_p)])\}$. Dereliction is $\mathsf{d}_X \in \mathbf{Rel}(!X,X)$ defined by $\mathsf{d}_X = \{([a],a) \mid a \in X\}$ and digging is $\mathsf{p}_X \in \mathbf{Rel}(!X,!!X)$ defined by $\mathsf{p}_X = \{(m_1+\cdots+m_k,[m_1,\dots,m_k]) \mid m_1,\dots,m_k \in !X\}$. As easily checked, weakening is given by $\mathsf{w}_X = \{([],*)\} \in \mathbf{Rel}(!X,1)$ and binary contraction is $\mathsf{c}_X = \{(m_1+m_2,(m_1,m_2))) \mid m_1,m_2 \in !X\}$.

This structure can also be extended to a weak differential LL model, codereliction being defined as $\overline{\mathsf{d}}_X = \{(a,[a]) \mid a \in X\} \in \mathbf{Rel}(X,!X)$. In this model, the Taylor formula holds as easily checked.

Fixpoints of types. Let $\mathbf{Rel}^{\subseteq}$ be the class of sets, ordered by inclusion. It is closed under arbitrary unions. A function $(\mathbf{Rel}^{\subseteq})^n \to \mathbf{Rel}^{\subseteq}$ is continuous if it is monotone wrt. inclusion and preserves all directed lubs. Any continuous function $\Phi : \mathbf{Rel}^{\subseteq} \to \mathbf{Rel}^{\subseteq}$ admits a least fixpoint defined as usual as $\bigcup_{n\in\mathbb{N}} \Phi^n(\emptyset)$. All the LL constructions defined above are continuous functions.

3.2 The Scott model of LL

The model. A preordered set is a pair $S = (|S|, \leq_S)$ where $|S|$ is a countable set and $\leq_S$ is a transitive and reflexive binary relation on $|S|$. We denote as $\mathcal{I}(S)$ the set of all subsets of $|S|$ which are downwards closed wrt. the $\leq_S$ relation. We set $S^{\mathsf{op}} = (|S|, \geq_S)$. We use $S \times T$ for the product preorder. Scott semantics can also be presented as a model of LL. The base category is **Pol**, the category whose objects are preordered sets and where $\mathbf{Pol}(S,T) = \mathcal{I}(S^{\mathsf{op}} \times T)$. The identity morphism at S is $\mathrm{Id}_S = \{(a,a') \in |S| \times |S| \mid a' \leq_S a\}$. Composition is just the usual composition of relations.

▶ **Lemma 1.** *There is an order isomorphism from* $\mathbf{Pol}(S,T)$ *to the set of functions* $\mathcal{I}(S) \to \mathcal{I}(T)$ *which preserve arbitrary unions, ordered under the pointwise order. This isomorphism maps the relation* R *to the function* $\xi \mapsto R\xi = \{b \in |T| \mid \exists a \in \xi\ (a,b) \in R\}$.

This is quite easy to prove, and this mapping from relation to functions is functorial. We equip **Pol** with a symmetric monoidal structure, taking $1 = (\{*\}, =)$ and $S \otimes T = S \times T$ (product preorder).

If two preorders S and S' are isomorphic as preorders through a bijection $\varphi : |S| \to |S'|$, then they are isomorphic in **Pol** by the relation $\{(a,a') \mid a' \leq_{S'} \varphi(a)\}$ but the converse is far from being true. In the first case we say that φ is a *strong isomorphism* from S to S'. The isomorphisms of the symmetric monoidal structure of **Pol** are the obvious strong ones. This SMC is closed, with $S \multimap T = S^{\mathsf{op}} \times T$ and linear evaluation $\mathsf{ev} \in \mathbf{Pol}((S \multimap T) \otimes S, T)$ given by $\mathsf{ev} = \{(((a', b), a), b') \mid a' \leq_S a \text{ and } b' \leq_T b\}$. **Pol** is $*$-autonomous with dualizing object $\perp = 1$, so that, up to an obvious strong isomorphism, $S^\perp = S^{\mathsf{op}}$. Observe that as in **Rel**, the cotensor product $\invamp$ coincides with the tensor product $\otimes$; both categories **Rel** and **Pol** are compact closed.

Pol is countably cartesian: the cartesian product of a countable family $(S_i)_{i \in I}$ of preorders is $S = \&_{i \in I} S_i$ defined by $|S| = \bigcup_{i \in I} \{i\} \times |S_i|$ preordered as follows: $(i, a) \leq_S (j, b)$ if $i = j$ and $a \leq_{S_i} b$. The projections are $\pi_i = \{((i, a), a') \mid a' \leq_{S_i} a\}$. In particular, the terminal object is $(\emptyset, \emptyset)$. The category **Pol** is therefore also cocartesian, and it is countably additive with sums of morphisms defined as unions. We define the exponential functor by $!S = (\mathcal{M}_{\mathrm{fin}}(|S|), \leq_{!S})$ where the preorder is defined by $p \leq_{!S} q$ if $\forall a \in |p| \, \exists b \in |q| \; a \leq_S b$. Given $R \in \mathbf{Pol}(S, T)$, we set $!R = \{(p, q) \in |!S| \times |!T| \mid \forall b \in |q| \, \exists a \in |p| \; (a, b) \in R\}$ and it is quite easy to check that $!R \in \mathbf{Pol}(!S, !T)$, and that this operation is functorial.

▶ Remark. The crucial point in the definition of $!S$ is that $\leq_{!S}$ does not take multiplicities into account. Indeed, there is another possible definition, for which we use another notation: we can set $!_{\mathsf{s}} S = (\mathcal{P}_{\mathrm{fin}}(|S|), \leq_{!_{\mathsf{s}} S})$, with preorder defined just as above: $\mu \leq_{!_{\mathsf{s}} S} \nu$ if $\forall a \in \mu \, \exists b \in \nu \; a \leq_S b$. The preorders $!S$ and $!_{\mathsf{s}} S$ are isomorphic (but not strongly isomorphic) through the relation $\mathsf{e}_S \in \mathbf{Pol}(!S, !_{\mathsf{s}} S)$ defined by $\mathsf{e}_S = \{(p, \mu) \mid \forall a \in \mu \, \exists a' \in |p| \; a \leq_S a'\}$. The important point is that this natural isomorphism is compatible with all the structures of both exponentials, so that the models defined by these exponentials are equivalent. We prefer to use the multiset-based construction to present the model because it is closer to the exponential of the relational model – this simplifies greatly the presentation of the extensional collapse as we shall see – but keep in mind that we could give the same definitions with the other version.

The Seely isomorphism $\mathsf{m} \in \mathbf{Pol}(1, !\top)$ is $\{(*, [])\}$ and the Seely natural isomorphism $\mathsf{m}_{S_1, S_2} \in \mathbf{Pol}(!S_1 \otimes !S_2, !(S \mathbin{\&} T))$ is $\{((p_1, p_2), q) \mid (i, a) \in |q| \Rightarrow \exists a' \in |p_i| \; a \leq_{S_i} a'\}$. Dereliction $\mathsf{d}_S \in \mathbf{Pol}(!S, S)$ is $\mathsf{d}_S = \{(p, a) \mid \exists a' \in |p| \; a \leq_S a'\}$ and digging $\mathsf{p}_S \in \mathbf{Pol}(!S, !!S)$ is $\mathsf{p}_S = \{(p, [p_1, \dots, p_n]) \mid i \in \mathbb{N} \text{ and } \forall i \; p_i \leq_{!S} p\}$. As easily checked, weakening is given by $\mathsf{w}_S = \{(p, *) \mid p \in |!S|\} \in \mathbf{Rel}(!S, 1)$ and binary contraction is $\mathsf{c}_S = \{(p, (p_1, p_2))) \mid p, p_1, p_2 \in |!S| \; p_1 \leq_{!S} p \text{ and } p_2 \leq_{!S} p\}$. Unlike the relational model, this structure cannot be extended into a weak differential LL model.

▶ Proposition 2. There is no natural transformation $\overline{\mathsf{d}}_S \in \mathbf{Pol}(S, !S)$ such that $\mathsf{d}_S \, \overline{\mathsf{d}}_S = \mathrm{Id}_S$.

Proof. We prove first that necessarily $\overline{\mathsf{d}}_S = \{(a, p) \in |S| \times |!S| \mid p \leq_{!S} [a]\}$. First, let $(a, p) \in \overline{\mathsf{d}}_S$. Let $a' \in |p|$. By definition of d_S, we have $(p, a') \in \mathsf{d}_S$, and hence $(a, a') \in \mathsf{d}_S \, \overline{\mathsf{d}}_S = \mathrm{Id}_S$. Therefore $a' \leq_S a$ and hence $p \leq_{!S} [a]$. Conversely, let $a \in |S|$. We have $(a, a) \in \mathrm{Id}_S$ and therefore there exists p such that $(a, p) \in \overline{\mathsf{d}}_S$ and $(p, a) \in \mathsf{d}_S$. By the second property, we can find $a' \in |p|$ such that $a \leq_S a'$. We have $[a] \leq_{!S} p$ and $(a, p) \in \overline{\mathsf{d}}_S \in \mathbf{Pol}(S, !S)$. Therefore $(a, [a]) \in \overline{\mathsf{d}}_S$. It follows that, for any p such that $p \leq_{!S} [a]$, one has $(a, p) \in \overline{\mathsf{d}}_S$.

Let $S = (\{0\}, =)$ and $T = (\{1, 2\}, =)$. Let $R = \{(0, 1), (0, 2)\}$, we have $R \in \mathbf{Pol}(S, T)$. Observe that $([0], [1, 2]) \in !R$ (warning: this is of course not true in **Rel**) so that $(0, [1, 2]) \in !R \, \overline{\mathsf{d}}_S$. But there is no $b \in |T|$ such that $(b, [1, 2]) \in \overline{\mathsf{d}}_T$ and hence we do not have $(0, [1, 2]) \in \overline{\mathsf{d}}_T \, R$, and this shows that $\overline{\mathsf{d}}_S$ is not a natural transformation. □

Fixpoints of types. Let S and T be preorders, we write $S \subseteq T$ if $|S| \subseteq |T|$ and, for any $a, a' \in |S|$, one has $a \leq_S a'$ iff $a \leq_T a'$. This is an order relation on the class of preorders and we use $\mathbf{Pol}^{\subseteq}$ for this partially ordered class. It is clear that any countable directed family in $\mathbf{Pol}^{\subseteq}$ has a lub and that all the LL constructions presented above are continuous. It is also clear that any continuous function $\Phi : \mathbf{Pol}^{\subseteq} \to \mathbf{Pol}^{\subseteq}$ has a least fixpoint.

3.3 The collapsing model of LL

Our last model combines the two models above. It is based on a new duality.

The model. Let S be a preorder and let $u, u' \subseteq |S|$. We write $u \perp u'$ if $u \cap u' = \emptyset \Rightarrow (\downarrow_S u) \cap u' = \emptyset$.

Observe that $(\downarrow_S u) \cap u' = \emptyset$ holds iff $(\downarrow_S u) \cap (\downarrow_{S^{\mathsf{op}}} u') = \emptyset$ so that $u \perp u'$ holds relatively to S iff $u' \perp u$ holds relatively to S^{op}. Given $D \subseteq \mathcal{P}(|S|)$, we define $D^{\perp(S)} \subseteq \mathcal{P}(|S|)$ by $D^{\perp(S)} = \{u' \subseteq |S| \mid \forall u \in D\ u \perp u'\}$. It is clear that $D \subseteq D^{\perp(S)\perp(S^{\mathsf{op}})}$ and that $D_1 \subseteq D_2 \Rightarrow {D_2}^{\perp(S)} \subseteq {D_1}^{\perp(S)}$, so that $D^{\perp(S)} = D^{\perp(S)\perp(S^{\mathsf{op}})\perp(S)}$. Observe that $\mathcal{I}(S^{\mathsf{op}}) \subseteq D^{\perp(S)} \subseteq \mathcal{P}(|S|)$ so that, when D is “closed” in the sense that $D = D^{\perp(S)\perp(S^{\mathsf{op}})}$, one has $\mathcal{I}(S) \subseteq D \subseteq \mathcal{P}(|S|)$.

The objects of the model are called *preorders with projections* (pop) and are pairs $E = (\langle E\rangle, \mathsf{D}(E))$ where $\langle E\rangle$ is a preorder and $\mathsf{D}(E) \subseteq \mathcal{P}(|\langle E\rangle|)$ satisfies $(\mathsf{D}(E))^{\perp(\langle E\rangle)\perp(\langle E\rangle^{\mathsf{op}})} \subseteq \mathsf{D}(E)$, that is $(\mathsf{D}(E))^{\perp(\langle E\rangle)\perp(\langle E\rangle^{\mathsf{op}})} = \mathsf{D}(E)$. If E is a pop, we set $E^{\perp} = (\langle E\rangle^{\mathsf{op}}, (\mathsf{D}(E))^{\perp(\langle E\rangle)})$. Let E and F be pops. One defines $E \otimes F$ by $\langle E \otimes F\rangle = \langle E\rangle \times \langle F\rangle$ and $\mathsf{D}(E \otimes F) = \{u \times v \mid u \in \mathsf{D}(E) \text{ and } v \in \mathsf{D}(F)\}^{\perp(\langle E\rangle \times \langle F\rangle)\perp(\langle E^{\perp}\rangle \times \langle F^{\perp}\rangle)}$. Let $E \multimap F = (E \otimes F^{\perp})^{\perp}$.

▶ **Lemma 3.** *Let $R \subseteq |\langle E \multimap F\rangle|$. One has $R \in \mathsf{D}(E \multimap F)$ iff any of the following equivalent conditions holds.*

- *If $u \in \mathsf{D}(E)$ and $v' \in \mathsf{D}(F^{\perp})$, then $R \cap (u \times v') = \emptyset \Rightarrow R \cap (\downarrow_{\langle E\rangle} u \times \uparrow_{\langle F\rangle} v') = \emptyset$.*
- *If $u \in \mathsf{D}(E)$, then $R\,u \in \mathsf{D}(F)$ and $R \downarrow u \subseteq \downarrow (R\,u)$.*
- *If $u \in \mathsf{D}(E)$, then $R\,u \in \mathsf{D}(F)$ and $\downarrow (R\,u) = (\downarrow_{\langle E\rangle \multimap \langle F\rangle} R)\,(\downarrow u)$.*

Proof. See [6]. □

To define the category **Pop** of pops, we set $\mathbf{Pop}(E, F) = \mathsf{D}(E \multimap F)$. By Lemma 3 $\mathrm{Id}_E = \{(a, a) \mid a \in |\langle E\rangle|\} \in \mathbf{Pop}(E, E)$, and if $Q \in \mathbf{Pop}(E, F)$ and $P \in \mathbf{Pop}(F, G)$, then $P\,Q \in \mathbf{Pop}(E, G)$ and so identities and composition of **Pop** are defined as in **Rel**. This category is $*$-autonomous: we have already defined the tensor product on objects. On morphisms, it is defined just as in **Rel**. The internal hom object $E \multimap F$ has also been defined above, and the linear evaluation relation is defined as in **Rel** again. Of course, one has to check carefully that all these relations are **Pop** morphisms, this is done in [6]. Notice that, as shown in that paper, this category is not compact closed. The category **Pop** is countably cartesian, $E = \&_{i \in I} E_i$ is defined by $\langle E\rangle = \&_{i \in I} \langle E_i\rangle$ and $w \subseteq \mathsf{D}(E)$ iff $\pi_i\, w \in \mathsf{D}(E_i)$ for each $i \in I$ (where π_i is the ith projection in the relational model). The projections morphism in **Pop** are those of the relational model. The category **Pop** is therefore also countably cocartesian, and one checks easily that it is countably additive.

We define now the exponential $!E$ of a pop E. One sets $\langle !E\rangle = !\langle E\rangle$ and therefore, we have $|\langle !E\rangle| = !|\langle E\rangle| = \mathcal{M}_{\mathrm{fin}}(|\langle E\rangle|)$ by our definition of $!E$ based on multisets and not on sets, see the remark in Section 3.2. We set $\mathsf{D}(!E) = \{u^! \mid u \in \mathsf{D}(E)\}^{\perp(\langle !E\rangle)\perp(\langle !E\rangle^{\mathsf{op}})}$, where $u^! = \mathcal{M}_{\mathrm{fin}}(u)$. Here is the main tool for dealing with this construction. See [6] for the proof.

▶ Proposition 4. Let E and F be pops and let $R \in \mathbf{Rel}(|\langle !E\rangle|, |\langle F\rangle|)$. One has $R \in \mathbf{Pop}(!E, F)$ iff, for any $u \in \mathsf{D}(E)$

- $R\,u^! \in \mathsf{D}(F)$
- $R\,(\downarrow_{\langle E\rangle} u)^! \subseteq \downarrow_{\langle F\rangle} (R\,u^!)$.

The Seely isomorphisms, and the dereliction and digging natural transformations are defined exactly as in **Rel**.

Fixpoints of types. Let E and F be pops. We write $E \subseteq F$ if $\langle E\rangle \subseteq \langle F\rangle$, $\mathsf{D}(E) \subseteq \mathsf{D}(F)$ and, for any $v \in \mathsf{D}(F)$, one has $v \cap |\langle E\rangle| \in \mathsf{D}(E)$ and $\downarrow_{\langle F\rangle} v \cap |\langle E\rangle| \subseteq \downarrow_{|\langle E\rangle|}(v \cap |\langle E\rangle|)$. This is an order relation on the class of preorders with projections, and we write $\mathbf{Pop}^{\subseteq}$ for the corresponding partially ordered class. It is shown in [6] that this partially ordered class is complete (all directed lubs exist) and we define as usual the notion of continuous function $(\mathbf{Pop}^{\subseteq})^n \to \mathbf{Pop}^{\subseteq}$, one checks that all constructions of linear logic are continuous functions, and that any continuous function $\Phi : \mathbf{Pop}^{\subseteq} \to \mathbf{Pop}^{\subseteq}$ admits a least fixpoint $\bigcup_{n=0}^{\infty} \Phi^n(\emptyset)$.

Forgetful LL functors. There is an obvious functor $\rho : \mathbf{Pop} \to \mathbf{Rel}$ defined on objects by $\rho(E) = |\langle E\rangle|$ and which is the identity on morphisms. With a preorder with projection E, we can also associate a preorder $\sigma(E) = \langle E\rangle$. This operation is extended to morphisms as follows: let $R \in \mathbf{Pop}(E, F)$, we set $\sigma(R) = \downarrow_{\langle E\rangle \multimap \langle F\rangle} R$.

▶ **Lemma 5.** *Both ρ and σ are LL functors.*

The proof can be found in [6]. The statement concerning ρ is straightforward. Concerning σ, LL functoriality is made possible by the presence of the sets $\mathsf{D}(E)$. For instance functoriality results directly from Lemma 3 and Lemma 1. But notice that, given preorders S, S' and S'' and *arbitrary* relations $R \in \mathbf{Rel}(|S|, |S'|)$ and $R' \in \mathbf{Rel}(|S'|, |S''|)$, the inclusion $(\downarrow_{S' \multimap S''} R')\,(\downarrow_{S \multimap S'} R) \subseteq \downarrow_{S \multimap S''}(R'\,R)$ does not hold in general.

▶ **Lemma 6.** *When restricted to inclusions, ρ induces a continuous function* $\mathbf{Pop}^{\subseteq} \to \mathbf{Rel}^{\subseteq}$ *and σ induces a continuous function* $\mathbf{Pop}^{\subseteq} \to \mathbf{Pol}^{\subseteq}$.

4 The cbv λ-calculus

Our syntax for the cbv λ-calculus is a slight modification of the ordinary λ-calculus syntax.

- If V is a value, then $\langle V\rangle$ is a term;
- if M and N are terms, then $M\,N$ is a term;
- if x is a variable, then x is a value;
- if M is a term and x is a variable, then $\lambda x\, M$ is a value.

We use Λ_{t} for the set of terms, Λ_{v} for the set of values and Λ_{e} for the disjoint union of these two sets, whose elements will be called expressions and denoted with letters $P, Q, \dots$. As usual, expressions are considered up to α-equivalence.

Reduction relations. One can define a general reduction relation β_{V} for this calculus: it is the contextual closure of the basic reduction rule $\langle \lambda x\, M\rangle\, \langle V\rangle\ \beta_{\mathsf{V}}\ M\,[V/x]$. We use β_{V}^{*} for the transitive closure of β_{V}. We also define a *weak* reduction relation $\hat\beta_{\mathsf{V}}$ which is included in β_{V} and which consists in reducing only redexes not occurring inside a value (that is, under a λ). It is defined by the following rules.

$$\frac{}{\langle \lambda x\, M\rangle\, \langle V\rangle\ \hat\beta_{\mathsf{V}}\ M\,[V/x]} \qquad \frac{M\ \hat\beta_{\mathsf{V}}\ M'}{M\,N\ \hat\beta_{\mathsf{V}}\ M'\,N} \qquad \frac{N\ \hat\beta_{\mathsf{V}}\ N'}{M\,N\ \hat\beta_{\mathsf{V}}\ M\,N'}$$

5 Linear-logic based models

Let $\mathcal{C}$ be an LL model. We present here a general notion of model for the cbv λ-calculus $\mathcal{C}$, which corresponds to a translation of intuitionistic logic into LL alluded to by Girard in [11] and called by him “boring”. It is compatible with the translation of the cbv λ-calculus into LL given in [15] and with other notions of model such as [20] and of course with [17] if one keeps in mind that the functor “!” defines a strong monad on the Kleisli category $\mathcal{C}_!$.

A $\mathcal{C}$-model of cbv as a triple $(U, \mathsf{app}, \mathsf{lam})$ where U is an object of $\mathcal{C}$, $\mathsf{app} \in \mathcal{C}(U, !U \multimap !U)$ and $\mathsf{lam} \in \mathcal{C}(!U \multimap !U, U)$ are such that $\mathsf{app}\,\mathsf{lam} = \mathrm{Id}_{!U \multimap !U}$.

Given an expression P and a sequence of variables $\vec{x} = (x_1, \dots, x_n)$ adapted to P (this means that the sequence is repetition-free and contains all the free variables of P), we define $[P]^{\vec{x}} \in \mathcal{C}((!U)^{\otimes n}, X)$ where $X = U$ if P is a value and $X = !U$ if P is a term. The definition is by induction on P, and we consider first the cases where P is a term.

Assume first that $P = \langle V \rangle$. By inductive hypothesis we have $[V]^{\vec{x}} : (!U)^{\otimes(n)} \to U$, and we set $[P]^{\vec{x}} = ([V]^{\vec{x}})^! : (!U)^{\otimes(n)} \to !U$. Assume next that $P = M\,N$. By inductive hypothesis, we have $[M]^{\vec{x}}, [N]^{\vec{x}} \in \mathcal{C}((!U)^{\otimes n}, !U)$. Therefore $\mathsf{app}\,\mathsf{d}_U\,[M]^{\vec{x}} \in \mathcal{C}((!U)^{\otimes n}, !U \multimap !U)$. So we set $[P]^{\vec{x}} = \mathsf{ev}\,((\mathsf{app}\,\mathsf{d}_U\,[M]^{\vec{x}}) \otimes [N]^{\vec{x}})\,\mathsf{c}_U^n \in \mathcal{C}((!U)^{\otimes n}, !U)$.

Now we interpret values. Assume first that P is a variable, so that $P = x_i$ for an uniquely determined $i \in \{1, \dots, n\}$. Then we set $[M]^{\vec{x}} = \mathsf{w}_U^{\otimes(i-1)} \otimes \mathsf{d}_U \otimes \mathsf{w}_U^{\otimes(n-i)} : (!U)^{\otimes n} \to 1^{\otimes(i-1)} \otimes U \otimes (1)^{\otimes(n-i)} \simeq U$ (we keep this isomorphism implicit). Assume last that $P = \lambda x\,M$. We can assume that x does not occur in $\vec{x}$. By inductive hypothesis, we have $[M]^{\vec{x},x} \in \mathcal{C}((!U)^{\otimes n} \otimes !U, !U)$ and hence $\lambda([M]^{\vec{x},x}) \in \mathcal{C}((!U)^{\otimes n}, !U \multimap !U)$ and we set $[P]^{\vec{x}} = (\mathsf{lam}\,\lambda([P]^{\vec{x},x})) \in \mathcal{C}((!U)^{\otimes n}, U)$.

▶ **Lemma 7** (Substitution Lemma). *Let P be an expression, x a variable and V a value. Let $\vec{x}$ which does not contain x, is adapted to V and such that $\vec{x}, x$ is adapted to E. We have $[P\,[V/x]]^{\vec{x}} = [P]^{\vec{x},x}\,((!U)^{\otimes n} \otimes ([V]^{\vec{x}})^!)\,\mathsf{c}_U^n$ where n is the length of $\vec{x}$.*

▶ **Theorem 8.** *Let $\vec{x}$ be adapted to the expressions P and P' and assume that $P\ \beta_{\mathsf{V}}\ P'$. Then $[P]^{\vec{x}} = [P']^{\vec{x}}$.*

6 A relational model and the associated type system

Let $\Phi_{\mathsf{R}} : \mathbf{Rel}^{\subseteq} \to \mathbf{Rel}^{\subseteq}$ be the continuous function defined by $\Phi_{\mathsf{R}}(X) = !X \multimap !X$. Let $\mathcal{U}_{\mathsf{R}}$ be its least fixpoint, then we have $\mathcal{U}_{\mathsf{R}} = !\mathcal{U}_{\mathsf{R}} \multimap !\mathcal{U}_{\mathsf{R}}$ so that $\mathcal{U}_{\mathsf{R}}$ is a $\mathbf{Rel}$-model of cbv with $\mathsf{app} = \mathsf{lam} = \mathrm{Id}$. An element of $\mathcal{U}_{\mathsf{R}}$ is a pair (p, q) where p and q are finite multisets of elements of $\mathcal{U}_{\mathsf{R}}$. The simplest of these elements is $\varepsilon = ([], [])$, here is another one: $([\varepsilon, \varepsilon, ([\varepsilon], [])], [\varepsilon])$.

6.1 Non-idempotent intersection types

We introduce a typing system for deriving judgments of shape $\Gamma \vdash M : m$ where M is a term, $m \in !\mathcal{U}_{\mathsf{R}}$ and Γ is a context (that is, a finite function from variables to $!\mathcal{U}_{\mathsf{R}}$) and judgments of shape $\Gamma \vdash V : a$ where V is a value and $a \in \mathcal{U}_{\mathsf{R}}$. The sum of contexts $\Gamma + \Delta$ is defined pointwise (using the sum of multisets), when Γ and Δ have the same domain. A context Γ is often written $\Gamma = (x_1 : m_1, \dots, x_n : m_n)$ where the x_i's are pairwise distinct variables and $m_1, \dots, m_n \in !\mathcal{U}_{\mathsf{R}}$. The typing rules for terms are

$$\frac{\Gamma \vdash M : [(p, q)] \qquad \Delta \vdash N : p}{\Gamma + \Delta \vdash M\,N : q} \qquad \frac{\Gamma_1 \vdash V : a_1 \qquad \cdots \qquad \Gamma_k \vdash V : a_k}{\Gamma_1 + \cdots + \Gamma_k \vdash \langle V \rangle : [a_1, \dots, a_k]}$$

The second rule conveys the intuition that $[a_1, \dots, a_k]$ represents the intersection of types $a_1, \dots, a_n$. The typing rules for values are

$$\frac{}{x_1 : [], \dots, x_n : [], x : [a] \vdash x : a} \qquad \frac{\Gamma, x : p \vdash M : q}{\Gamma \vdash \lambda x\, M : (p, q)}$$

▶ Proposition 9. Let P be an expression and let $\vec{x} = (x_1, \dots, x_n)$ be a list of variables adapted to P. Let $\vec{p} \in (!\mathcal{U}_{\mathsf{R}})^n$ and let $\alpha \in X$ (where $X = \mathcal{U}_{\mathsf{R}}$ if P is a value and $X = !\mathcal{U}_{\mathsf{R}}$ if P is a term). Then one has $(\vec{p}, \alpha) \in [P]^{\vec{x}}_{\mathsf{R}}$ iff the typing judgment $x_1 : p_1, \dots, x_n : p_n \vdash P : \alpha$ is derivable.

The proof is a simple verification, by induction on the structure of P.

6.2 A CBV resource calculus

We introduce a resource calculus whose terms can be used to denote typing derivations in the typing system described above.

6.2.1 Notation. Given a finite family $(a_i)_{i \in I}$ and a predicate P on I, we use $[a_i \mid P(i)]$ for the multiset whose elements are the a_i's such that $P(i)$ holds, taking multiplicities into account.

6.2.2 Syntax. We describe first the syntax of our resource calculus.

- If $v_1, \dots, v_n$ are simple values, then $\langle v_1, \dots, v_n \rangle$ is a simple term;
- if s and t are simple terms, then $s\,t$ is a simple term;
- if x is a variable, then x is a simple value;
- if x is a variable and s is a simple term, then $\lambda x\, s$ is a simple value.

Terms are sets of simple terms, and values are defined similarly. We speak of (simple) expressions when we don't want to be specific. We write these sets as a sums to insist on the algebraic flavor of the semantical background. The above syntactic constructs are extended to non simple expressions, by linearity. For instance, if $v = \sum_{i \in I} v_i$ and $w = \sum_{j \in J} w_j$ are values (the summands being simple), the expression $\langle v, w \rangle$ denotes $\sum_{i \in I, j \in J} \langle v_i, w_j \rangle$. And if $s = \sum_{i \in I} s_i$ is a term, then $\lambda x\, s$ denotes $\sum_{i \in I} \lambda x\, s_i$, which is a value.

Given a simple expression e and simple values $v_1, \dots, v_n$, we define the linear substitution $\partial_x(e; v_1, \dots, v_n)$, which is an expression of the same kind as e, by

$$\partial_x(e; v_1, \dots, v_n) = \begin{cases} \sum_{f \in \mathfrak{S}_n} & e\left[v_1/x_{f(1)}, \dots, v_n/x_{f(n)}\right] \\ & \text{if } n = \deg_x e \\ 0 & \text{otherwise} \end{cases}$$

where $\deg_x e$ is the number of free occurrences of x in e and $x_1, \dots, x_n$ are these occurrences (in the case $n = \deg_x e$).

6.2.3 Reduction rules. We can give now the reduction rules of the calculus. We define a reduction relation denoted as δ from simple expressions to *generally non simple* expressions by the following rules.

$$\frac{}{\langle \lambda x\, s \rangle \, \langle v_1, \dots, v_n \rangle \;\delta\; \partial_x(s; v_1, \dots, v_n)}$$

$$\frac{}{\langle v_1, \dots, v_n \rangle\, t \;\delta\; 0} \text{ if } n \neq 1 \qquad \frac{s \;\delta\; s'}{\lambda x\, s \;\delta\; \lambda x\, s'}$$

$$\frac{s \;\delta\; s'}{s\,t \;\delta\; s'\,t} \qquad \frac{t \;\delta\; t'}{s\,t \;\delta\; s\,t'} \qquad \frac{v \;\delta\; v'}{\langle v, v_1, \dots, v_n \rangle \;\delta\; \langle v', v_1, \dots, v_n \rangle}$$

This reduction can be extended to non simple expressions, but this is not needed here and is postponed to a longer version of this paper.

One defines a size function on simple expressions by $\|x\| = 0$, $\|\lambda x\, s\| = 1 + \|s\|$, $\|s\, t\| = \|s\| + \|t\|$ and $\|\langle v_1, \dots, v_k\rangle\| = \sum_{j=1}^{k} \|v_j\|$. In other words $\|e\|$ is the number of of λ's in e.

▶ **Lemma 10.** *There is no infinite sequence $(e_i)_{i\in\mathbb{N}^+}$ of simple expressions such that, for each i, $e_i\ \delta\ e'$ with $e_{i+1} \in e'$.*

Proof. $\|e_{i+1}\| < \|e_i\|$. □

6.3 Categorical denotational semantics

Let U be a $\mathcal{C}$-model of cbv, where we assume moreover that $\mathcal{C}$ is a weak differential LL model which is countably additive and where hom-sets have idempotent sums. We show how to interpret the cbv resource calculus in such a structure.

We introduce a convenient notation. Let $g_1, \dots, g_k \in \mathcal{C}((!U)^{\otimes n}, U)$. We set $\langle g_1, \dots, g_k\rangle = \bar{\mathsf{d}}_U^k\,(g_1 \otimes \cdots \otimes g_k)\,\mathsf{c}_U^{n,k}$, where $\mathsf{c}_U^{n,k} \in \mathcal{C}((!U)^{\otimes n}, ((!U)^{\otimes n})^{\otimes k})$ is an obvious generalization of c_U^n.

Given a simple expression e and an adapted sequence of variables $\vec{x}$, we define $[e]^{\vec{x}} \in \mathcal{C}((!U)^{\otimes n}, X)$ where $X = U$ if e is a value and $X = {!U}$ if e is a term. The definition is by induction on e. For the syntactical constructs which are similar to those of the cbv λ-calculus (namely: variables, application and abstraction), the interpretation is the same as in Section 5. To complete the definition we have just to define the semantics of $\langle v_1, \dots, v_k\rangle$. By inductive hypothesis we have defined $g_j = [v_j]^{\vec{x}} \in \mathcal{C}((!U)^{\otimes n}, U)$ and we set $[\langle v_1, \dots, v_k\rangle]^{\vec{x}} = \langle g_1, \dots, g_k\rangle$. If e is an expression, that is a set of simple expressions $e = \sum_{i\in I} e_i$ and a list of variables $\vec{x}$ adapted to all x_i's, we set $[e]^{\vec{x}} = \sum_{i\in I} [e_i]^{\vec{x}}$ which is well defined because we have assumed that the sum of morphisms is idempotent in $\mathcal{C}$.

▶ **Lemma 11.** *If $e\ \delta\ e'$ and $\vec{x}$ is adapted to e and e', then $[e]^{\vec{x}} = [e']^{\vec{x}}$.*

Proof. It suffices to prove the result in the case where e is simple, by induction on e. The proof uses the following property of linear substitution wrt. the interpretation (substitution lemma). Let e be a simple expression and $v_1, \dots, v_k$ be simple values. Let $\vec{x}, x$ a sequence of variable adapted to e and to all v_j's. Let n be the length of $\vec{x}$. Then we have

$$[\partial_x(e; v_1, \dots, v_k)]^{\vec{x}} = [e]^{\vec{x},x}\,(!U^{\otimes n} \otimes \left\langle [v_1]^{\vec{x}}, \dots, [v_k]^{\vec{x}}\right\rangle)\,\mathsf{c}_U^n$$

and this is proved by a simple induction on e. □

For any expression P of the cbv λ-calculus, we define a set $\mathcal{T}(P)$ of simple expressions by induction.

$$\begin{aligned}
\mathcal{T}(x) = \{x\} \qquad \mathcal{T}(\lambda x\, M) &= \{\lambda x\, s \mid s \in \mathcal{T}(M)\} \\
\mathcal{T}(M\, N) &= \{s\, t \mid s \in \mathcal{T}(M) \text{ and } t \in \mathcal{T}(N)\} \\
\mathcal{T}(\langle V\rangle) &= \{\langle v_1, \dots, v_k\rangle \mid k \in \mathbb{N} \text{ and } \forall i\, v_i \in \mathcal{T}(V)\}\,.
\end{aligned}$$

Observe that the set $\mathcal{T}(P)$ is infinite as soon as P has a subterm of shape $\langle V\rangle$.

▶ **Lemma 12.** *Let P be an expression and let V be a value. Let $e \in \mathcal{T}(P)$ and $v_1, \dots, v_k \in \mathcal{T}(V)$. Then $\partial_x(e; v_1, \dots, v_k) \subseteq \mathcal{T}(P\,[V/x])$.*

Proof. Easy induction on P. □

▶ **Lemma 13.** *If the Taylor formula holds in $\mathcal{C}$ then for any expression P and any $\vec{x}$ adapted to P we have $[P]^{\vec{x}} = \sum_{e\in\mathcal{T}(P)}[e]^{\vec{x}}$.*

Proof. Easy induction on P. □

6.4 Adequacy in Rel

▶ **Lemma 14.** *Let P, P' be expressions and let $e \in \mathcal{T}(P)$. If $P\ \hat{\beta}_{\mathsf{V}}\ P'$ then there exists $e' \subseteq \mathcal{T}(P')$ such that $e\ \delta\ e'$.*

Proof. Simple inspection, using Lemma 12. □

One has to be careful when using this lemma because, using the notations of the lemma, nothing prevents the expression e' – which is not simple in general – from being empty.

▶ **Theorem 15.** *Let P be an expression. Let $\vec{x}$ be adapted to P and let n be the length of $\vec{x}$. Let $m_1,\dots,m_n \in\ !\mathcal{U}_{\mathsf{R}}$ and let $\alpha \in X$ where $X = \mathcal{U}_{\mathsf{R}}$ if P is a value and $X = !\mathcal{U}_{\mathsf{R}}$ if P is a term. If $x_1 : m_1,\dots,x_n : m_n \vdash P : \alpha$ then P is $\hat{\beta}_{\mathsf{V}}$ strongly normalizing.*

Proof. By Proposition 9, our hypothesis means that $(\vec{m},\alpha) \in [P]^{\vec{x}}$. By Lemma 13 there exists $e \in \mathcal{T}(P)$ such that $(\vec{m},\alpha) \in [e]^{\vec{x}}$. If $P\ \hat{\beta}_{\mathsf{V}}\ P'$ there then there exists $e' \subseteq \mathcal{T}(P')$ such that $e\ \delta\ e'$ by Lemma 14. By Lemma 11 we have $(\vec{m},\alpha) \in [e']^{\vec{x}}$ and hence there exists $f \in e'$ such that $(\vec{m},\alpha) \in [f]^{\vec{x}}$ (so e' is not empty!). Therefore, for any reduction $P = P_1\ \hat{\beta}_{\mathsf{V}}\ P_2 \dots \hat{\beta}_{\mathsf{V}}\ P_l$ we can find $e_1 \in \mathcal{T}(P_1),\dots,e_l \in \mathcal{T}(P_l)$ with $\|e_1\| > \|e_2\| > \cdots > \|e_l\|$ and $(\vec{m},\alpha) \in [e_i]^{\vec{x}}$ for each i. □

7 A Scott model and the associated type system

Let $\Phi_{\mathsf{S}} : \mathbf{Pol}^{\subseteq} \to \mathbf{Pol}^{\subseteq}$ be the continuous functions defined by $\Phi_{\mathsf{S}}(S) = !S \multimap !S$. Let $\mathcal{U}_{\mathsf{S}}$ be the least fixpoint of Φ_{S}, then $\mathcal{U}_S$ (equipped with two identity morphisms) is a $\mathbf{Pol}^{\subseteq}$-model of the cbv λ-calculus. We use $\leq$ for the both preorder relations $\leq_{\mathcal{U}_{\mathsf{S}}}$ and $\leq_{!\mathcal{U}_{\mathsf{S}}}$.

It is clear that $|\mathcal{U}_{\mathsf{S}}| = \mathcal{U}_{\mathsf{R}}$, so that an element of $|\mathcal{U}_{\mathsf{S}}|$ is a pair (p,q) where p and q are finite multisets of elements of $|\mathcal{U}_{\mathsf{S}}|$. On finite multisets, the preorder is given by $p \leq_{!\mathcal{U}_{\mathsf{S}}} p'$ if $\forall a \in |p|\,\exists a' \in |p'|\ a \leq_{\mathcal{U}_{\mathsf{S}}} a'$ and on pairs, the $\mathcal{U}_{\mathsf{S}}$-preorder is given by $(p,q) \leq_{\mathcal{U}_{\mathsf{S}}} (p',q')$ if $p' \leq_{!\mathcal{U}_{\mathsf{S}}} p$ and $q \leq_{!\mathcal{U}_{\mathsf{S}}} q'$.

7.1 Idempotent intersection types

We introduce a typing system for deriving judgments of shape $\Gamma \vdash_{\mathsf{S}} M : m$ and $\Gamma \vdash_{\mathsf{S}} V : a$ with the same notations as in Section 6.1: just as in that section, the types are the elements of $|\mathcal{U}_{\mathsf{S}}| = \mathcal{U}_{\mathsf{R}}$ and the contexts associate finite multisets of types with variables. But the typing rules are different. For terms, they are given by

$$\frac{\Gamma \vdash_{\mathsf{S}} M : [(p,q)] \qquad \Gamma \vdash_{\mathsf{S}} N : p}{\Gamma \vdash_{\mathsf{S}} M\,N : q} \qquad \frac{\Gamma \vdash_{\mathsf{S}} V : a_1 \qquad \cdots \qquad \Gamma \vdash_{\mathsf{S}} V : a_k}{\Gamma \vdash_{\mathsf{S}} \langle V\rangle : [a_1,\dots,a_k]}$$

and for values, they are given by

$$\frac{[a] \leq m}{\Gamma, x : m \vdash_{\mathsf{S}} x : a} \qquad \frac{\Gamma, x : p \vdash_{\mathsf{S}} M : q}{\Gamma \vdash_{\mathsf{S}} \lambda x\,M : (p,q)}$$

Similar typing systems for cbv have already been proposed, see [5] in particular.

▶ Proposition 16. Let P be an expression and let $\vec{x} = (x_1, \dots, x_n)$ be a list of variables adapted to P. Let $\vec{m} \in (|!\mathcal{U}_\mathsf{S}|)^n$ and let $\alpha \in |S|$ (where $S = \mathcal{U}_\mathsf{S}$ if P is a value and $S = !\mathcal{U}_\mathsf{S}$ if P is a term). Then one has $(\vec{m}, \alpha) \in [P]^{\vec{x}}_\mathsf{S}$ iff the typing judgment $x_1 : m_1, \dots, x_n : m_n \vdash P : \alpha$ is derivable.

The proof is a simple verification, by induction on the structure of P.

▶ Remark. We can define a model $\mathcal{U}'_\mathsf{S}$ of cbv λ-calculus using the exponential $!_\mathsf{s}\_$ mentioned in the remark of Section 3.2, and by this remark, the models $\mathcal{U}_\mathsf{S}$ and $\mathcal{U}'_\mathsf{S}$ *are isomorphic* in **Pol**. Now the typing system associated with $\mathcal{U}'_\mathsf{S}$ is exactly the same as the system presented above, up to the fact that all multisets occurring in types should be replaced by the corresponding sets and, up to this transformation, it is equivalent to the system above. In that sense, the typing system of this section is actually an idempotent intersection typing system. And indeed, if say $\vdash_\mathsf{S} M : p$ is derivable (where M is a closed term to simplify the notations) and if p' is equivalent to p in the preorder $!\mathcal{U}_\mathsf{S}$ (in other words $p \leq p'$ and $p' \leq p$), then one can infer $\vdash_\mathsf{S} M : p'$ by an isomorphic typing derivation. This is in particular the case if p and p' differ only by the multiplicities of their subtypes, that is if $p^- = p'^-$ where $[a_1, \dots, a_n]^- = \{a_1^-, \dots, a_n^-\} \in |!_\mathsf{s}\mathcal{U}'_\mathsf{S}|$ and $(p, q)^- \in (p^-, q^-) \in |\mathcal{U}'_\mathsf{S}|$.

7.2 Adequacy in the idempotent case

One can prove an analog of Theorem 15 for this idempotent typing system, but the same technique does not apply because, as we have seen with Proposition 2, **Pol** is not a model of the cbv resource calculus. The standard method to prove adequacy in this model is by reducibility, an example of such a proof will be given in a longer version of this paper.

▶ **Theorem 17.** *Let P be an expression. Let $\vec{x}$ be adapted to P and let n be the length of $\vec{x}$. Let $m_1, \dots, m_n \in |!\mathcal{U}_\mathsf{S}|$ and let $\alpha \in |X|$ where $X = \mathcal{U}_\mathsf{S}$ if P is a value and $X = !\mathcal{U}_\mathsf{S}$ if P is a term. If $x_1 : m_1, \dots, x_n : m_n \vdash_\mathsf{S} P : \alpha$ then P is $\hat{\beta}_\mathsf{V}$ strongly normalizing.*

We show now how to use the model **Pop** and the non-idempotent adequacy result to prove this theorem.

7.3 Adequacy in the idempotent case, using preorders with projections

Let $\Phi_\mathsf{P} : \mathbf{Pop}^\subseteq \to \mathbf{Pop}^\subseteq$ be the continuous function defined by $\Phi_\mathsf{P}(E) = !E \multimap !E$. Let $\mathcal{U}_\mathsf{P}$ be the least fixpoint of Φ_P, then $\mathcal{U}_\mathsf{P}$ (equipped with two identity morphisms) is a **Pop**-model of the cbv λ-calculus.

By Lemma 6, we have $\rho(\mathcal{U}_\mathsf{P}) = \mathcal{U}_\mathsf{R}$ and $\sigma(\mathcal{U}_\mathsf{P}) = \mathcal{U}_\mathsf{S}$.

Let P be an expression and let $\vec{x}$ be a sequence of variables adapted to P, let n be the length of $\vec{x}$. Because ρ is an LL functor, we have $[P]^{\vec{x}}_\mathsf{P} = \rho([P]^{\vec{x}}_\mathsf{P}) = [P]^{\vec{x}}_\mathsf{R}$, and similarly we have $\sigma([P]^{\vec{x}}_\mathsf{P}) = [P]^{\vec{x}}_\mathsf{S}$. These properties are proved by a straightforward induction on P. As a consequence, using the definition of the functor σ, we get the following result, which relates the relational semantics of an expression to its Scott semantics.

▶ **Theorem 18.** *Let P be an expression and let $\vec{x}$ be a sequence of variables of length n, adapted to P. Then $[P]^{\vec{x}}_\mathsf{S} = \downarrow[P]^{\vec{x}}_\mathsf{R}$ where the downwards closure is taken in $(!\mathcal{U}_\mathsf{S})^{\otimes n} \multimap S$ (with $S = \mathcal{U}_\mathsf{S}$ if P is a value and $S = !\mathcal{U}_\mathsf{S}$ if P is a term).*

We prove Theorem 17. We deal with the case of a term, but the proof is of course similar for values. So assume that $x_1 : m_1, \dots, x_k : m_k \vdash_\mathsf{S} M : m$ which means that $(\vec{m}, m) \in [M]^{\vec{x}}_\mathsf{S}$ where $m_1, \dots, m_n, m \in |!\mathcal{U}_\mathsf{S}|$. Then by Theorem 18, we can find $m'_1, \dots, m'_n, m' \in |!\mathcal{U}_\mathsf{S}| = !\mathcal{U}_\mathsf{R}$

with $\forall i\ m'_i \leq_{!\mathcal{U}_{\mathsf{S}}} m_i$ and $m \leq_{!\mathcal{U}_{\mathsf{S}}} m'$ and such that $(m'_1, \ldots, m'_n, m') \in [M]^{\vec{x}}_{\mathsf{R}}$. By Theorem 15, M is $\hat{\beta}_{\mathsf{V}}$ strongly normalizing.

▶ Remark. It is quite instructive to try to prove Theorem 18 by a direct induction on derivations in the idempotent typing system of Section 7.1 and to observe that it is not as easy as one could think. Given a derivation of $\Gamma \vdash_{\mathsf{S}} P : \alpha$ we want to find a derivation $\Gamma' \vdash P : \alpha'$ such that $\alpha \leq \alpha'$ and $\Gamma' \leq \Gamma$ (this means that Γ and Γ' have same domain and that $\Gamma'(x) \leq \Gamma(x)$ for each x). Observe first that we cannot hope to have $\Gamma' = \Gamma$ and $\alpha' = \alpha$ in general, because our proof would fail on its base case (variables). Assume that the derivation ends with an application rule: $P = M\,N$, $\alpha = q$, and we have (shorter) derivations of $\Gamma \vdash_{\mathsf{S}} M : [(p, q)]$ and $\Gamma \vdash_{\mathsf{S}} N : p$. By inductive hypothesis, we can find Γ'_1, p'_1, q'_1 with $\Gamma'_1 \leq \Gamma$, $p'_1 \leq p$ and $q \leq q'_1$ such that $\Gamma'_1 \vdash M : [(p'_1, q'_1)]$ is derivable and also Γ'_2, p'_2 such that $\Gamma'_2 \leq \Gamma$, $p \leq p'_2$ and $\Gamma'_2 \vdash N : p'_2$ is derivable. To build a typing derivation for $M\,N$ in this non-idempotent system, we would need to force $p'_2 = p'_1$ but nothing in our inductive hypothesis guarantees that this is possible. It is precisely the point of the model $\mathcal{U}_{\mathsf{P}}$ in **Pop** to show that, for "well behaved" sets of the relational model (those belonging to $\mathsf{D}(\mathcal{U}_{\mathsf{S}})$) downwards closure commutes with application – this is the main content of Lemma 3 – and that the relational interpretation of cbv λ-calculus expressions are precisely such well-behaved sets.

Conclusion

We have shown how to use a purely semantical construction (the model **Pop**) to reduce the proof of an adequacy theorem usually proved by reducibility to a purely combinatorial argument and we have illustrated this approach in the cbv λ-calculus. In further work, we'll apply this approach to other languages and other notions of normalization to understand better how the reducibility structure is encoded in the model. We'll also explore the probable connections between our work and [1].

Acknowledgments

The author thanks the referees for their valuable suggestions and comments. This work has been partly supported by the international ANR-NSFC project *Locali.*

References

1 Alexis Bernadet and Stéphane Lengrand. Filter models: Non-idempotent intersection types, orthogonality and polymorphism. In *CSL*, Lecture Notes in Computer Science, pages 51–66. Springer-Verlag, 2011. To appear.

2 René David. Every unsolvable lambda-term has a decoration. In Jean-Yves Girard, editor, *TLCA*, volume 1581 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 1999.

3 Daniel De Carvalho. Execution Time of λ-Terms via Denotational Semantics and Intersection Types. Research Report RR-6638, INRIA, 2008.

4 Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. A semantic measure of the execution time in linear logic. *Theoretical Computer Science*, 412(20):1884–1902, 2011.

5 Joshua Dunfield and Frank Pfenning. Type assignment for intersections and unions in call-by-value languages. In Andrew D. Gordon, editor, *FoSSaCS*, volume 2620 of *Lecture Notes in Computer Science*, pages 250–266. Springer, 2003.

6 Thomas Ehrhard. The Scott model of Linear Logic is the extensional collapse of its relational model. *Theoretical Computer Science*, 2011. To appear. A draft version is available on http://www.pps.jussieu.fr/~ehrhard.

7 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, 2003.

8 Thomas Ehrhard and Laurent Regnier. Differential interaction nets. In *Proceedings of WoLLIC'04*, volume 103 of *Electronic Notes in Theoretical Computer Science*, pages 35–74. Elsevier Science, 2004.

9 Thomas Ehrhard and Laurent Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. Technical report, Institut de mathématiques de Luminy, 2005. submitted to Theoretical Computer Science.

10 Marcelo P. Fiore. Differential structure in models of multiplicative biadditive intuitionistic linear logic. In Simona Ronchi Della Rocca, editor, *TLCA*, volume 4583 of *Lecture Notes in Computer Science*, pages 163–177. Springer, 2007.

11 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

12 Jean-Yves Girard. Normal functors, power series and the λ-calculus. *Annals of Pure and Applied Logic*, 37:129–177, 1988.

13 Michael Huth. Linear Domains and Linear Maps. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *MFPS*, volume 802 of *Lecture Notes in Computer Science*, pages 438–453. Springer-Verlag, 1993.

14 Michael Huth, Achim Jung, and Klaus Keimel. Linear types and approximation. *Mathematical Structures in Computer Science*, 10(6):719–745, 2000.

15 John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Theoretical Computer Science*, 228(1-2):175–210, 1999.

16 Paul-André Melliès. Categorical semantics of linear logic. *Panoramas et Synthèses*, 27, 2009.

17 Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

18 Michele Pagani. The cut-elimination theorem for differential nets with promotion. In Pierre-Louis Curien, editor, *TLCA*, volume 5608 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2009.

19 Michele Pagani and Simona Ronchi Della Rocca. Solvability in resource lambda-calculus. In C.-H. Luke Ong, editor, *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 358–373. Springer-Verlag, 2010.

20 Alberto Pravato, Simona Ronchi Della Rocca, and Luca Roversi. The call-by-value lambda-calculus: a semantic investigation. *Mathematical Structures in Computer Science*, 9(5):617–650, 1999.

21 Paolo Tranquilli. Intuitionistic Differential Nets and Lambda-Calculus. *Theoretical Computer Science*, 2008. To appear.

22 Glynn Winskel. A linear metalanguage for concurrency. In Armando Martin Haeberer, editor, *AMAST*, volume 1548 of *Lecture Notes in Computer Science*, pages 42–58. Springer-Verlag, 1998.

23 Glynn Winskel. Linearity and non linearity in distributed computation. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2004.

Descriptive complexity for pictures languages

Etienne Grandjean[1] and Frédéric Olive[2]

1 Université de Caen Basse-Normandie / ENSICAEN / CNRS
GREYC – Caen, France
etienne.grandjean@info.unicaen.fr

2 Aix-Marseille Université / CNRS
LIF – Marseille, France
frederic.olive@lif.univ-mrs.fr

Abstract

This paper deals with logical characterizations of picture languages of any dimension by syntactical fragments of existential second-order logic. Two classical classes of picture languages are studied:

- the class of *recognizable* picture languages, i.e. projections of languages defined by local constraints (or tilings): it is known as the most robust class extending the class of regular languages to any dimension;
- the class of picture languages recognized on *nondeterministic cellular automata in linear time* : cellular automata are the simplest and most natural model of parallel computation and linear time is the minimal time-bounded class allowing synchronization of nondeterministic cellular automata.

We uniformly generalize to any dimension the characterization by Giammarresi et al. [7] of the class of *recognizable* picture languages in existential monadic second-order logic.

We state several logical characterizations of the class of picture languages recognized in linear time on nondeterministic cellular automata. They are the first machine-independent characterizations of complexity classes of cellular automata.

Our characterizations are essentially deduced from normalization results we prove for first-order and existential second-order logics over pictures. They are obtained in a general and uniform framework that allows to extend them to other 'regular' structures.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.3 Formal Languages

Keywords and phrases Picture languages; locality and tiling; recognizability; linear time; cellular automata; logical characterizations; second-order logic.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.274

1 Introduction

One goal of descriptive complexity is to establish logical characterizations of natural classes of problems in finite model theory. Many results in this area involve second-order logic (SO) and its restrictions, monadic second-order logic (MSO) and existential second-order logic (ESO): see *e.g.* [5, 11] for descriptive complexity of formal languages and [5, 9, 8, 11] for the one of complexity classes.

It is important to recall that the complexity class defined by a logic often depends heavily on the underlying class of structures: words, trees, graphs, ordered or unordered structures, etc. E.g., for words, a classical result by Büchi, Elgot and Trahtenbrot [2, 5, 11] states that the class of languages definable in MSO equals the class of regular languages, in short,

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 274–288
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

MSO = REG, whereas the same logic or even its existential second-order fragment EMSO can define some NP-complete problems on graphs, e.g., the 3-colorability problem.

We are interested in descriptive complexity of *picture languages.* A d-picture language is a set of d-pictures, i.e., d-dimensional words (or colored grids). First, notice the following points:

- In a series of papers culminating in [7], Giammarresi et al. proved that a 2-picture language is *recognizable*, i.e. is the projection of a local (that means: *tilable*) 2-picture language, iff it is definable in EMSO. In short: $\mathrm{REC}^2 = \mathrm{EMSO}$.
- In fact, the class REC^2 contains some NP-complete problems. More generally, one observes that, for each dimension $d \geq 1$, the class REC^d of recognizable languages *can be defined as* the class of d-picture languages recognized by nondeterministic d-dimensional *cellular automata* in *constant time*[1].

The present work originates from two questions about word/picture languages:

1. How can we generalize the proof of the above-mentioned theorem of Giammarresi et al. to any dimension? That is, can we establish the equality $\mathrm{REC}^d = \mathrm{EMSO}$ for any $d \geq 1$?
2. Can we obtain logical characterizations of time complexity classes of cellular automata[2]?

In this paper, a d-picture language is a set of d-pictures $p : [1, n]^d \to \Sigma$, for a finite alphabet Σ, i.e., d-dimensional Σ-words[3], and we use two natural representations of a d-picture p as a first-order structure:

- as a *pixel structure:* on the *pixel* domain $[1, n]^d$ where the sets $p^{-1}(s)$, $s \in \Sigma$, are encoded by *unary* relations $(Q_s)_{s\in\Sigma}$ and the underlying d-dimensional grid is encoded by d successor functions (see Definition 2);
- as a *coordinate structure:* on the *coordinate* domain $[1, n]$ where the sets $p^{-1}(s)$ are encoded by d*-ary* relations $(Q_s)_{s\in\Sigma}$; moreover, one uses the natural linear order of the coordinate domain $[1, n]$ and its associate successor function (see Definition 3).

We establish logical characterizations of two classes of d-picture languages, for all dimensions $d \geq 1$:

1. *On pixel structures:* $\mathrm{REC}^d = \mathrm{ESO}(\text{arity } 1) = \mathrm{ESO}(\text{var } 1) = \mathrm{ESO}(\forall^1, \text{arity } 1)$. That means a d-picture language is *recognizable iff* it is definable in monadic ESO (resp. in ESO with 1 first-order variable, or in monadic ESO with 1 universally quantified first-order variable) [4].
2. *On coordinate structures:* $\mathrm{NLIN}^d_{\mathrm{ca}} = \mathrm{ESO}(\text{var } d{+}1) = \mathrm{ESO}(\forall^{d+1}, \text{arity } d{+}1)$; that means a d-picture language is recognized by a nondeterministic d-dimensional cellular automaton in *linear time* (see, e.g., [3, 14]) *iff* it is definable in ESO with $d + 1$ distinct first-order variables (resp. ESO with second-order variables of arity at most $d + 1$ and a prenex first-order part of prefix $\forall^{d+1}$).

[1] That means: for such a picture language L, there is some constant integer c such that each computation stops at instant c, and $p \in L$ iff it has at least one computation that stops *with each cell in an accepting state*: see Sommerhalder et al. [13], which, to our knowledge, was the first paper involving this notion.

[2] This originates from a question that J. Mazoyer asked us in 2000 (personal communication): give a logical characterization of the linear time complexity class of nondeterministic cellular automata.

[3] More generally, the domain of a d-picture is of the "rectangular" form $[1, n_1] \times \ldots \times [1, n_d]$. For simplicity and uniformity of presentation, we have chosen to present our results in the particular case of "square" pictures of domain $[n]^d$. Fortunately, they also hold with the same proofs for general domains $[1, n_1] \times \ldots \times [1, n_d]$.

[4] It is interesting to compare this result with some results by Borchert [1].

Items 1 and 2 proceed from normalization results of, respectively, first-order and ESO logics that we prove over picture languages.

Significance of our results:

1. The *normalization equality* $\mathrm{ESO}(\text{arity } 1) = \mathrm{ESO}(\forall^1, \text{arity } 1)$ of Item 1 is a consequence of the fact that, on pixel structures (and, more generally, on structures that consist of bijective functions and unary relations), any first-order formula is equivalent to a boolean combination of *cardinality formulas* of the form: 'there exist k distinct elements x such that $\psi(x)$', where ψ is a quantifier-free formula with only *one* variable. The normalization equality explicitly expresses the local feature of EMSO on pictures – using only *one* first-order variable. Using almost exclusively logical tools, the results of Item 1 can be regarded as an explicitation/simplification (using only *one* first-order variable) and uniformisation of the (more combinatorial) proof and ideas of the main result of Giammarresi et al. [7, 6]; this allows us to generalize it to *any* dimension and, potentially, to other *regular* structures.
2. Intuitively, our characterization $\mathrm{NLIN}^d_{\mathrm{ca}} = \mathrm{ESO}(\forall^{d+1}, \text{arity } d+1)$ of Item 2 naturally reflects a *symmetry* property of the time-space diagram of any computation of a *non-deterministic* d-dimensional cellular automaton: informally, the single first-order variable representing time *cannot be distinguished* from any of the d variables that represent the d-dimensional space; in other words, the $d+1$ variables can be permuted without increasing the expressive (or computational) power of the formula. This is the sense of the inclusion $\mathrm{ESO}(\forall^{d+1}, \text{arity } d+1) \subseteq \mathrm{NLIN}^d_{\mathrm{ca}}$ whose proof is far from trivial.

2 Preliminaries

All along the paper, we denote by Σ, Γ some finite alphabets and by d a positive integer. For any positive integer n, we set $[n] := \{1, \dots, n\}$. We are interested in sets of pictures of any fixed dimension d.

▶ **Definition 1.** A *d-dimensional picture* or *d-picture* on Σ is a function $p : [n]^d \to \Sigma$ where n is a positive integer. The set $dom(p) = [n]^d$ is called the *domain* of picture p and its elements are called *points*, *pixels* or *cells* of the picture. A set of d-pictures on Σ is called a *d-dimensional language*, or *d-language*, on Σ.

Notice that 1-pictures on Σ are nothing but nonempty words on Σ.

2.1 Pictures as model theoretic structures

Along the paper, we will often describe d-languages as sets of models of logical formulas. To allow this point of view, we must settle on an encoding of d-pictures as model theoretic structures.

For logical aspects of this paper, we refer to the usual definitions and notations in logic and finite model theory (see [5] or [11], for instance). A *signature* (or *vocabulary*) σ is a finite set of relation and function symbols each of which has a fixed arity. A (finite) *structure* S of vocabulary σ, or σ-structure, consists of a finite domain D of cardinality $n \geq 1$, and, for any symbol $s \in \sigma$, an interpretation of s over D, often denoted by s for simplicity. The tuple of the interpretations of the σ-symbols over D is called the *interpretation* of σ over D and, when no confusion results, it is also denoted σ. We will often deal with *tuples* of objects. We denote them by bold letters.

Let us define the two natural representations of a picture as a logical structure.

▶ **Definition 2.** Given $p : [n]^d \to \Sigma$, we denote by *pixel*$^d(p)$ the structure

$$\mathsf{pixel}^d(p) = ([n]^d, (Q_s)_{s\in\Sigma}, (\mathrm{succ}_i)_{i\in[d]}, (\min_i)_{i\in[d]}, (\max_i)_{i\in[d]}).$$

Here:

- succ_j is the *(cyclic) successor function* according to the j^{th} dimension of $[n]^d$, mapping each $(a_1, \ldots, a_d) \in [n]^d$ on $(a_1^{(j)}, \ldots, a_d^{(j)}) \in [n]^d$, where we set : $a_j^{(j)} = a_j + 1$ if $a_j < n$, and $a_j^{(j)} = 1$ otherwise; $a_i^{(j)} = a_i$ for each $i \neq j$; in other words, for $a \in [n]^d$, $\mathrm{succ}_j(a)$ is the d-tuple $a^{(j)}$ obtained from a by 'increasing' its j^{th} component according to the cyclic successor on $[n]$;
- the $\min_i$'s, $\max_i$'s and Q_s's are the following unary (monadic) relations: $\min_i = \{a \in [n]^d : a_i = 1\}$; $\max_i = \{a \in [n]^d : a_i = n\}$; $Q_s = \{a \in [n]^d : p(a) = s\}$.

▶ **Definition 3.** Given $p : [n]^d \to \Sigma$, we denote by *coord*$^d(p)$ the structure

$$\mathsf{coord}^d(p) = \langle [n], (Q_s)_{s\in\Sigma}, <, \mathrm{succ}, \min, \max\rangle. \tag{1}$$

Here:

- each Q_s is the d-ary relation that is the set of cells of p labelled by an s, in other words: $Q_s = \{a \in [n]^d : p(a) = s\}$;
- $<$, min, max are relations of respective arities 2, 1, 1, that are respectively the sets $\{(i, j) : 1 \leq i < j \leq n\}$, $\{1\}$ and $\{n\}$;
- succ is the cyclic successor, that is: $\mathrm{succ}(i) = i + 1$ for $i < n$ and $\mathrm{succ}(n) = 1$.

For a d-language L, we set *pixel*$^d(L) = \{\mathsf{pixel}^d(p) : p \in L\}$ and *coord*$^d(L) = \{\mathsf{coord}^d(p) : p \in L\}$.

▶ Remark. Several details are irrelevant in Definitions 2 and 3, i.e. our results still hold for several variants, in particular:

- in Definition 3, the fact that the linear order $<$ and the equality $=$ are allowed or not and the fact that min, max are represented by individual constants or unary relations;
- in both definitions, the fact that the successor function(s) is/are cyclic or not and is/are completed or not by predecessor(s) function(s).

At the opposite, it is essential that, in both definitions,

- the successor(s) is/are represented by *function(s)* and not by (binary) relation(s),
- the min, max are explicitly represented.

2.2 Logics under consideration

All formulas considered hereafter belong to *relational Existential Second-Order logic*. Given a signature σ, indifferently made of relation and function symbols, a relational existential second-order formula of signature σ has the shape $\Phi \equiv \exists \mathbf{R}\varphi(\sigma, \mathbf{R})$, where $\mathbf{R} = (R_1, \ldots, R_k)$ is a tuple of relation symbols and φ is a first-order formula of signature $\sigma \cup \{\mathbf{R}\}$. We denote by ESO$^\sigma$ the class thus defined. We will often omit to mention σ for considerations on these logics that do not depend on the signature. Hence, ESO stands for the class of all formulas belonging to ESO$^\sigma$ for some σ.

We will pay great attention to several variants of ESO. In particular, we will distinguish formulas of type $\Phi \equiv \exists \mathbf{R}\varphi(\sigma, \mathbf{R})$ according to: the number of distinct first-order variables involved in φ, the arity of the second-order symbols $R \in \mathbf{R}$, and the quantifier prefix of some prenex form of φ.

With the logic $\mathrm{ESO}^\sigma(\forall^d, \mathrm{arity}\,\ell)$, we control these three parameters: it is made of formulas of which first-order part is prenex with a universal quantifier prefix of length d, and where existentially quantified relation symbols are of arity at most ℓ. In other words, $\mathrm{ESO}^\sigma(\forall^d, \mathrm{arity}\,\ell)$ collects formulas of shape $\exists\mathbf{R}\forall\mathbf{x}\theta(\sigma, \mathbf{R}, \mathbf{x})$ where θ is quantifier free, $\mathbf{x}$ is a d-tuple of first-order variables, and $\mathbf{R}$ is a tuple of relation symbols of arity at most ℓ. Relaxing some constraints of the above definition, we set:

$$\mathrm{ESO}^\sigma(\forall^d) = \bigcup_{\ell>0} \mathrm{ESO}^\sigma(\forall^d, \mathrm{arity}\,\ell) \text{ and } \mathrm{ESO}^\sigma(\mathrm{arity}\,\ell) = \bigcup_{d>0} \mathrm{ESO}^\sigma(\forall^d, \mathrm{arity}\,\ell).$$

Finally, we write $\mathrm{ESO}^\sigma(\mathrm{var}\,d)$ for the class of formulas that involve at most d first-order variables, thus focusing on the sole number of distinct first-order variables (possibly quantified several times).

3 A logical characterization of recognizable picture languages

In this section, we define the class of *local* (resp. *recognizable*) picture languages and establish the logical characterizations of the class of recognizable picture languages. In order to define a notion of locality based on sub-pictures we need to mark the border of each picture.

▶ **Definition 4.** By $\Gamma^\sharp$ we denote the alphabet $\Gamma \cup \{\sharp\}$ where $\sharp$ is a special symbol not in Γ. Let p be any d-picture of domain $[n]^d$ on Γ. The *bordered d-picture* of p, denoted by $p^\sharp$, is the function $p^\sharp : [0, n+1]^d \to \Gamma^\sharp$ defined by $p^\sharp(a) = p(a)$ if $a \in dom(p)$; $p^\sharp(a) = \sharp$ otherwise. Here, 'otherwise' means that a is on the border of $p^\sharp$, i.e. some component a_i of a is 0 or $n+1$.

Let us now define our notion of *local picture language* or *tilings language*. It is based on some sets of allowed patterns (called tiles) of the bordered pictures and is a simple generalization to any dimension of the notion of *hv-local* 2-dimensional picture language of [10] (see also [6]).

▶ **Definition 5.**
1. Given a d-picture p and an integer $j \in [d]$, two cells $a = (a_i)_{i\in[d]}$ and $b = (b_i)_{i\in[d]}$ of p are *j-adjacent* if they have the same coordinates, except the j^{th} one for which $|a_j - b_j| = 1$.
2. A *tile* for a d-language L on Γ is a pair in $(\Gamma^\sharp)^2$.
3. A picture p *is j-tiled* by a set of tiles $\Delta \subseteq (\Gamma^\sharp)^2$ if for any two j-adjacent points $a, b \in dom(p^\sharp)$: $(p^\sharp(a), p^\sharp(b)) \in \Delta$.
4. Given d sets of tiles $\Delta_1, \ldots, \Delta_d \subseteq (\Gamma^\sharp)^2$, a d-picture p *is tiled by* $(\Delta_1,\ldots,\Delta_d)$ if p is j-tiled by Δ_j for each $j \in [d]$.
5. We denote by $L(\Delta_1,\ldots,\Delta_d)$ the set of d-pictures on Γ that are tiled by $(\Delta_1,\ldots,\Delta_d)$.
6. A d-language L on Γ is *local* if there exist $\Delta_1, \ldots, \Delta_d \subseteq (\Gamma^\sharp)^2$ such that $L = L(\Delta_1,\ldots,\Delta_d)$. We then say that L is *$(\Delta_1, \ldots, \Delta_d)$-local*, or *$(\Delta_1, \ldots, \Delta_d)$-tiled*.

▶ Remark. Our notion of *locality* (that generalizes the one of [10] to any dimension) is more restrictive than the one given by Giammarresi and al. [7]. At the opposite, the *locality* notion defined by Borchert [1] is the most general one: its is defined by the presence/absence or some patterns/sub-pictures of *any size* in the picture, and, as he proved, his *locality* is equivalent to definability by some universally quantified *one-variable* first-order sentence using non cyclic *successor functions* and *minimal* and *maximal* predicates. Fortunately, the notion of *recognizability* as defined below, is a *robust* notion that remains equivalent to the one defined by either one of the locality notions of [1] and [7].

▶ **Definition 6.** A d-language L on Σ is *recognizable* if it is the projection (i.e. homomorphic image) of a *local* d-language over an alphabet Γ. It means there exist a surjective function $\pi : \Gamma \to \Sigma$ and a local d-language L_{loc} on Γ such that $L = \{\pi \circ p : \ p \in L_{\text{loc}}\}$. Because of the last item of Definition 5, one can also write: L is recognizable if there exist a surjective function $\pi : \Gamma \to \Sigma$ and d sets $\Delta_1, \ldots, \Delta_d \subseteq (\Gamma^\sharp)^2$ such that $L = \{\pi \circ p : \ p \in L(\Delta_1, \ldots, \Delta_d)\}$. We write REC^d for the class of recognizable d-languages.

A characterization of recognizable languages of dimension 2 by a fragment of existential monadic second-order logic was proved by Giammarresi et al. [7]. They established:

▶ **Theorem 7** ([7]). *For any* 2*-language* L*:* $L \in \text{REC}^2 \Leftrightarrow \mathsf{pixel}^2(L) \in \text{ESO}(\text{arity}\,1)$.

In this section, we come back to this result. We simplify its proof, refine the logic it involves, and generalize its scope to any dimension.

▶ **Theorem 8.** *For any* $d > 0$ *and any* d*-language* L*, the following assertions are equivalent:*

1. $L \in \text{REC}^d$*;*
2. $\mathsf{pixel}^d(L) \in \text{ESO}(\forall^1, \text{arity}\,1)$*;*
3. $\mathsf{pixel}^d(L) \in \text{ESO}(\text{arity}\,1)$.

Theorem 8 is a straightforward consequence of Propositions 9 and 11 below.

▶ **Proposition 9.** *For any* $d > 0$ *and any* d*-language* L *on* Σ*:* $L \in \text{REC}^d \Leftrightarrow \mathsf{pixel}^d(L) \in \text{ESO}(\forall^1, \text{arity}\,1)$.

Sketch of proof. $\boxed{\Rightarrow}$ A picture belongs to L if there exists a tiling of its domain whose projection coincides with its content. In the logic involved in the proposition, the 'arity 1' corresponds to formulating the existence of the tiling, while the $\forall^1$ is the syntactic resource needed to express that the tiling behaves as expected. Let us detail these considerations.

By Definition 5, there exist an alphabet Γ (which can be assumed disjoint from Σ), a surjective function $\pi : \Gamma \to \Sigma$ and d subsets $\Delta_1, \ldots, \Delta_d \subseteq (\Gamma^\sharp)^2$ such that L is the set $\{\pi \circ p' : p' \in L(\Delta_1, \ldots, \Delta_d)\}$.

The belonging of a picture $p' : [n]^d \to \Gamma$ to $L(\Delta_1, \ldots, \Delta_d)$ is easily expressed on $\mathsf{pixel}^d(p') = \langle [n]^d, (Q_s)_{s\in\Gamma}, \ldots\rangle$ with a first-order formula which asserts, for each dimension $i \in [d]$, that for any pixel x of p', the couple $(x, \text{succ}_i(x))$ can be tiled with some element of Δ_i. Because it deals with each cell x separately, this formula has the form $\forall x \Psi(x, (Q_s)_{s\in\Gamma})$, where Ψ is quantifier-free.

Now, a picture $p : [n]^d \to \Sigma$ belongs to L iff it results from a π-renaming of a picture $p' \in L(\Delta_1, \ldots, \Delta_d)$. It means there exists a Γ-labeling of p (that is, a tuple $(Q_s)_{s\in\Gamma}$ of subsets of $[n]^d$) corresponding to a picture of $L(\Delta_1, \ldots, \Delta_d)$ (i.e. fulfilling $\forall x \Psi(x, (Q_s)_{s\in\Gamma})$) and from which the actual Σ-labeling of p (that is, the subsets $(Q_s)_{s\in\Sigma}$) is obtained *via* π (easily expressed by a formula of the form $\forall x \Psi'(x, (Q_s)_{s\in\Sigma}, (Q_s)_{s\in\Gamma})$).

Finally, the formula $(\exists Q_s)_{s\in\Gamma} \forall x : \Psi \wedge \Psi'$ conveys the desired property and fits the required form.

$\boxed{\Leftarrow}$ In order to prove the converse implication, it is convenient to first normalize the sentences of $\text{ESO}(\forall^1, \text{arity}\,1)$. This is the role of the technical result below, which asserts that on pixel encodings, each such sentence can be rewritten in a very local form where the first-order part alludes only pairs of adjacent pixels of the bordered picture. We state it without proof (see also [1]):

▶ **Fact 10.** *On pixel structures, any $\varphi \in \mathrm{ESO}(\forall^1, \text{arity } 1)$ is equivalent to a sentence of the form:*

$$\exists \mathbf{U} \forall x \bigwedge_{i \in [d]} \left\{ \begin{array}{rcll} min_i(x) & \rightarrow & \mathsf{m}_i(x) & \wedge \\ max_i(x) & \rightarrow & \mathsf{M}_i(x) & \wedge \\ \neg max_i(x) & \rightarrow & \Psi_i(x) & \end{array} \right\}. \tag{2}$$

Here, $\mathbf{U}$ is a list of monadic relation variables and m_i, M_i, Ψ_i are quantifier-free formulas such that

- *atoms of m_i and M_i have all the form $Q(x)$;*
- *atoms of Ψ_i have all the form $Q(x)$ or $Q(succ_i(x))$,*

where, in both cases, $Q \in \{(Q_s)_{s \in \Sigma}, \mathbf{U}\}$.

Now, consider L such that $\mathsf{pixel}^d(L) \in \mathrm{ESO}(\forall^1, \text{arity } 1)$. Fact 10 ensures that $\mathsf{pixel}^d(L)$ is characterized by a sentence of the form (2) above. We have to prove that L is the projection of some local d-language L_{loc} on some alphabet Γ, that is a $(\Delta_1, \ldots, \Delta_d)$-tiled language for some $\Delta_1, \ldots, \Delta_d \subseteq \Gamma^2$. Let $U_1, \ldots, U_k$ denote the list of (distinct) elements of the set $\{(Q_s)_{s \in \Sigma}, \mathbf{U}\}$ of unary relation symbols of φ so that the first ones $U_1, \ldots, U_m$ are the Q_s's (here, $\min_i$ and $\max_i$ symbols are excluded). The trick is to put each subformula $\mathsf{m}_i(x)$, $\mathsf{M}_i(x)$ and $\Psi_i(x)$ of φ into its *complete disjunctive normal form* with respect to $U_1, \ldots, U_k$. Typically, each subformula $\Psi_i(x)$ whose atoms are of the form $U_j(x)$ or $U_j(\mathrm{succ}_i(x))$, for some $j \in [k]$, is transformed into the following 'complete disjunctive normal form':

$$\bigvee_{(\epsilon, \epsilon') \in \Delta_i} \left(\bigwedge_{j \in [k]} \epsilon_j U_j(x) \wedge \bigwedge_{j \in [k]} \epsilon'_j U_j(\mathrm{succ}_i(x)) \right).$$

Here, the following conventions are adopted:

- $\epsilon = (\epsilon_1, \ldots, \epsilon_k) \in \{0, 1\}^k$ and similarly for ϵ';
- for any atom α and any bit $\epsilon_j \in \{0, 1\}$, $\epsilon_j \alpha$ denotes the literal α if $\epsilon_j = 1$, the literal $\neg\alpha$ otherwise.

For $\epsilon \in \{0, 1\}^k$, we denote by $\Theta_\epsilon(x)$ the 'complete conjunction' $\bigwedge_{j \in [k]} \epsilon_j U_j(x)$. Intuitively, $\Theta_\epsilon(x)$ is a complete description of x and the set $\Gamma = \bigcup_{i \in [m]} \{0^{i-1} 1 0^{m-i}\} \times \{0, 1\}^{k-m}$ is the set of possible colours (remember that the Q_s's that are the U_j's for $j \in [m]$ form a partition of the domain). The complete disjunctive normal form of $\Psi_i(x)$ can be written into the suggestive form: $\bigvee_{(\epsilon, \epsilon') \in \Delta_i} (\Theta_\epsilon(x) \wedge \Theta_{\epsilon'}(\mathrm{succ}_i(x)))$.

If each subformula $\mathsf{m}_i(x)$ and $\mathsf{M}_i(x)$ of φ is similarly put into complete disjunctive normal form, that is $\bigvee_{(\sharp, \epsilon) \in \Delta_i} \Theta_\epsilon(x)$ and $\bigvee_{(\epsilon, \sharp) \in \Delta_i} \Theta_\epsilon(x)$, respectively (there is no ambiguity in our implicit definition of the Δ_i's, since $\sharp \notin \Gamma$), then the whole formula φ becomes the following equivalent formula:

$$\varphi' = \exists \mathbf{U} \forall x \bigwedge\nolimits_{i \in [d]} \left\{ \begin{array}{rcll} \min_i(x) & \rightarrow & \displaystyle\bigvee_{(\sharp, \epsilon) \in \Delta_i} \Theta_\epsilon(x) & \wedge \\ \max_i(x) & \rightarrow & \displaystyle\bigvee_{(\epsilon, \sharp) \in \Delta_i} \Theta_\epsilon(x) & \wedge \\ \neg\max_i(x) & \rightarrow & \displaystyle\bigvee_{(\epsilon, \epsilon') \in \Delta_i} (\Theta_\epsilon(x) \wedge \Theta_{\epsilon'}(\mathrm{succ}_i(x))) & \end{array} \right\}$$

Finally, let L_{loc} denote the d-language over Γ defined by the first-order sentence φ_{loc} obtained by replacing each Θ_ϵ by the new unary relation symbol Q_ϵ in the first-order part of

φ'. In other words, $\mathsf{pixel}^d(L_{\text{loc}})$ is defined by the following first-order sentence:

$$\varphi_{\text{loc}} = \forall x \bigwedge_{i\in[d]} \left\{ \begin{array}{lcll} \min_i(x) & \to & \displaystyle\bigvee_{(\sharp,\epsilon)\in\Delta_i} Q_\epsilon(x) & \wedge \\ \max_i(x) & \to & \displaystyle\bigvee_{(\epsilon,\sharp)\in\Delta_i} Q_\epsilon(x) & \wedge \\ \neg\max_i(x) & \to & \displaystyle\bigvee_{(\epsilon,\epsilon')\in\Delta_i} (\, Q_\epsilon(x) \wedge Q_{\epsilon'}(\mathrm{succ}_i(x))\,) & \end{array} \right\}$$

Hence, $L_{\text{loc}} = L(\Delta_1, \ldots, \Delta_d)$. That is, L_{loc} is indeed local and the corresponding sets of tiles are the Δ_i's of the previous formula. It is now easy to see that our initial d-language L is the projection of the local L_{loc} by the projection $\pi : \Gamma \to \Sigma$ defined as follows: $\pi(\epsilon) = s$ iff $\epsilon_i = 1$ for $i \in [m]$ and U_i is Q_s. This completes the proof. ◀

▶ **Proposition 11.** $\mathrm{ESO}(\mathrm{arity}\,1) \subseteq \mathrm{ESO}(\forall^1, \mathrm{arity}\,1)$ *on pixel structures, for any* $d > 0$.

Proof. In a pixel structure, each function symbol is interpreted as a *bijective* function (namely, a cyclic successor). It has been proved in [4, 12] that any first-order formula on such a structure can be rewritten as a so-called *cardinality formula*, that is as a boolean combination of sentences of the form $\psi^{\geq k} = \exists^{\geq k} x \psi(x)$ (for $k \geq 1$) where $\psi(x)$ is a quantifier-free formula (using the 'bijective' function symbols f and their inverses f^{-1}) with the single variable x and where the quantifier $\exists^{\geq k} x$ means 'there exist at least k elements $x \ldots$'. Therefore, it is easily seen that proving the proposition amounts to showing that each sentence of the form $\psi^{\geq k}$ or $\neg\psi^{\geq k}$ can be translated in $\mathrm{ESO}(\forall^1, \mathrm{arity}\,1)$ on pixel structures.

This is done as follows: for a given sentence $\exists^{\geq k} x \psi(x)$, we introduce new unary relations $U^{=1}, U^{=2}, \ldots, U^{=k-1}$ and $U^{\geq k}$, with the intended meaning:

A pixel $a \in [n]^d$ *belongs to* $U^{=j}$ *(resp.* $U^{\geq k}$*) iff there are exactly* j *(resp. at least* k*) pixels* $b \in [n]^d$ *lexicographically smaller than or equal to* a *such that* $\mathsf{pixel}^d(p) \models \psi(b)$.

Then we have to compel these relation symbols to fit their expected interpretations, by means of a universal first-order formula with a single variable. First, we demand the relations to form a partition of the domain:

$$(1) \quad \bigwedge_{i<j<k} \big(\neg U^{=i}(x) \vee \neg U^{=j}(x)\big) \wedge \bigwedge_{i<k} \big(\neg U^{=i}(x) \vee \neg U^{\geq k}(x)\big)\,.$$

Let us temporarily denote by $\leq_{\text{lex}}$ the lexicographic order on $[n]^d$ inherited from the natural order on $[n]$, and by $\mathrm{succ}_{\text{lex}}$, $\min_{\text{lex}}$, $\max_{\text{lex}}$ its associated successor function and unary relations corresponding to extremal elements. Then the sets described above can be defined inductively by the conjunction of the following six formulas:

$$(2) \quad (\min_{\text{lex}}(x) \wedge \neg\psi(x)) \to U^{=0}(x)$$

$$(3) \quad (\min_{\text{lex}}(x) \wedge \psi(x)) \to U^{=1}(x)$$

$$(4) \quad \bigwedge_{i<k} \big((\neg\max_{\text{lex}}(x) \wedge U^{=i}(x) \wedge \neg\psi(\mathrm{succ}_{\text{lex}}(x))\big) \to U^{=i}(\mathrm{succ}_{\text{lex}}(x))$$

$$(5) \quad \bigwedge_{i<k-1} \big((\neg\max_{\text{lex}}(x) \wedge U^{=i}(x) \wedge \psi(\mathrm{succ}_{\text{lex}}(x))\big) \to U^{=i+1}(\mathrm{succ}_{\text{lex}}(x))$$

$$(6) \quad \big((\neg\max_{\text{lex}}(x) \wedge U^{=k-1}(x) \wedge \psi(\mathrm{succ}_{\text{lex}}(x))\big) \to U^{\geq k}(\mathrm{succ}_{\text{lex}}(x))$$

(7) $\big((\neg\max_{\mathrm{lex}}(x) \wedge U^{\geq k}(x)\big) \to U^{\geq k}(\mathrm{succ}_{\mathrm{lex}}(x))$
Hence, under the hypothesis (1) $\wedge \ldots \wedge$ (7), the sentences $\psi^{\geq k}$ and $\neg\psi^{\geq k}$ are equivalent, respectively, to $\forall x(\max_{\mathrm{lex}}(x) \to U^{\geq k}(x))$ and $\forall x(\max_{\mathrm{lex}}(x) \to \neg U^{\geq k}(x))$.

To complete the proof, it remains to get rid of symbols $\mathrm{succ}_{\mathrm{lex}}$, $\min_{\mathrm{lex}}$ and $\max_{\mathrm{lex}}$ that are not allowed in our language. It is done by referring to these symbols implicitly rather than explicitly. For instance, since $\mathrm{succ}_{\mathrm{lex}}(x) = \mathrm{succ}_i\mathrm{succ}_{i+1}\ldots\mathrm{succ}_d(x)$ for the smallest $i \in [d]$ such that $\bigwedge_{j>i} \max_j(x)$, each formula φ involving $\mathrm{succ}_{\mathrm{lex}}(x)$ actually corresponds to the conjunction:

$$\bigwedge_{i\in[d]} \left((\neg\max_i(x) \wedge \bigwedge_{i<j\leq d} \max_j(x)) \to \varphi_i \right),$$

where φ_i is obtained from φ by the substitution $\mathrm{succ}_{\mathrm{lex}}(x) \rightsquigarrow \mathrm{succ}_i \ldots \mathrm{succ}_d(x)$. Similar arguments allow to get rid of $\min_{\mathrm{lex}}$ and $\max_{\mathrm{lex}}$. ◀

▶ Remark. In this proof, two crucial features of a structure of type $\mathsf{pixel}^d(p)$ are involved: its 'bijective' nature, that allows to rewrite first-order formulas as cardinality formulas; the regularity of its predefined arithmetics (the functions succ_i defined on each dimension), that endows $\mathsf{pixel}^d(p)$ with a grid structure: it enables us to implicitly define an order on the whole domain $dom(p)$ by means of first-order formulas with a single variable, which in turn allows to express cardinality formulas by 'cumulative' arguments, *via* the sets $U^{=i}$. Proposition 11 straightforwardly generalizes to all structures – and there are a lot – that fulfill these two properties.

4 A logical characterization of $\mathrm{NLIN}_{\mathsf{ca}}$

The second main concept studied in this paper is the classical notion of linear time complexity on nondeterministic cellular automata of any dimension (e.g., see [3, 14]). For simplicity of notation, we only present here the notion of *one-way d-dimensional cellular automaton*, instead of the more usual notion of *two-way d-dimensional cellular automaton*, but it is a folklore result that in the nondeterministic case, the two linear-time complexity classes so defined coincide (see [14]).

There are some technicalities in our definition of the transition function of a cellular automaton here below. This is due to the need to distinguish the different possible positions of the pixels of a picture w.r.t. its border: the one-way neighborhood of a cell $\mathbf{x} = (x_1, \ldots, x_d)$, that is the set of cells $\mathbf{y} = (y_1, \ldots, y_d)$ such that $0 \leq y_i - x_i \leq 1$ for each $i \in [d]$, may be *incomplete* according to the *position* of the cell $\mathbf{x}$ w.r.t. the border of the picture.

▶ **Definition 12.** A pixel $\mathbf{x} = (x_1, \ldots, x_d) \in [n]^d$ is *in position* $a = (a_1, \ldots, a_d) \in \{0,1\}^d$ in a picture $p : [n]^d \to \Gamma$ or in the domain $[n]^d$ if, for all $i \in [d]$, we have $a_i = 0$ if $x_i = n$ and $a_i = 1$ if $x_i < n$.

We are going to define the transition function on a pixel x of a picture p according to some 'neighborhood' (sub-picture) denoted $p_{a,x}$ whose domain, denoted by Dom_a, depends on the position a of the pixel in the picture.

▶ **Definition 13.** For each $a = (a_1, \ldots, a_d) \in \{0,1\}^d$, let us define the *a-domain* as $\mathsf{Dom}_a = [0, a_1] \times \cdots \times [0, a_d]$.

The *a-neighborhood* of some pixel $x \in [n]^d$ in position a in a picture $p : [n]^d \to \Gamma$ is the function $p_{a,x} : \mathsf{Dom}_a \to \Gamma$ defined as $p_{a,x}(b) = p(x + b)$, where $x + b$ denotes the sum of the vectors x and b.

We denote by $\mathit{neighb}_a(\Gamma)$ the set of all possible a-neighborhoods on an alphabet Γ, that is the set of functions $\nu : \mathsf{Dom}_a \to \Gamma$.

We are now ready to define the 'transition function' of a cellular automaton:

▶ **Definition 14.** A *one-way nondeterministic d-dimensional cellular automaton* (*d-automaton*, for short) over an alphabet Σ is a tuple $\mathcal{A} = (\Sigma, \Gamma, \delta, F)$, where

- the finite alphabet Γ called the *set of states* of $\mathcal{A}$ includes the *input alphabet* Σ and the set F of *accepting states*: $\Sigma, F \subseteq \Gamma$;
- δ is the (nondeterministic) *transition function* of $\mathcal{A}$: it is a family of *a-transition functions* $\delta = (\delta_a)_{a\in\{0,1\}^d}$ of the form $\delta_a : \mathsf{neighb}_a(\Gamma) \to \mathcal{P}(\Gamma)$.

Let us now define a computation.

▶ **Definition 15.** Let $\mathcal{A} = (\Sigma, \Gamma, \delta, F)$ be a d-automaton and $p, p' : [n]^d \to \Gamma$ be two d-pictures on Γ. We say that p' is a *successor* of p for $\mathcal{A}$, denoted by $p' \in \mathcal{A}(p)$, if, for each position $a \in \{0,1\}^d$ and each point x of position a in $[n]^d$, $p'(x) \in \delta_a(p_{a,x})$. The set of *$j^{th}$-successors* of p for $\mathcal{A}$, denoted by $\mathcal{A}^j(p)$, is defined inductively:

$$\mathcal{A}^0(p) = \{p\} \text{ and, for } j \geq 0,\ \mathcal{A}^{j+1}(p) = \bigcup_{p' \in \mathcal{A}^j(p)} \mathcal{A}(p').$$

▶ **Definition 16.** A *computation* of a d-automaton $\mathcal{A}$ on an input d-picture p is a sequence p_1, p_2, p_3, ... of d-pictures such that $p_1 = p$ and $p_{i+1} \in \mathcal{A}(p_i)$ for each i. A computation is *accepting* if it is finite – it has the form $p_1, p_2, \ldots, p_k$ for some k – and the cell of minimal coordinates, $1^d = (1, \ldots, 1)$, of its last configuration is in an accepting state: $p_k(1^d) \in F$.

▶ **Definition 17.** Let $\mathcal{A} = (\Sigma, \Gamma, \delta, F)$ be a d-automaton and let $T : \mathbb{N} \to \mathbb{N}$ be such that $T(n) > n$. A d-picture p on Σ is *accepted by $\mathcal{A}$ in time $T(n)$* if $\mathcal{A}$ admits an accepting computation of length $T(n)$ on p. That means, there exists a computation $p = p_1, p_2, \ldots, p_{T(n)} = \mathcal{A}^{T(n)-1}(p)$ of $\mathcal{A}$ on p such that $p_{T(n)}(1^d) \in F$. A d-language L on Σ is *accepted*, or *recognized*, by $\mathcal{A}$ *in time* $T(n)$ if it is the set of d-pictures accepted by $\mathcal{A}$ *in time* $T(n)$. That is $L = \{p : \exists p' \in \mathcal{A}^{T(n)-1}(p) \text{ such that } p'(1^d) \in F\}$.

If $T(n) = cn + c'$, for some integers c, c', then L is said to be *recognized in linear time* and we write $L \in \mathrm{NLIN}^d_{ca}$.

▶ Remark. The nondeterministic linear time class $\mathrm{NLIN}^d_{\mathrm{ca}}$ is very robust, i.e. is not modified by many changes in the definition of the automaton or in its time bound. In particular, it is a folklore result that the constants c, c' defining the bound $T(n) = cn + c'$ can be fixed arbitrarily, provided $T(n) > n$. For example, the class $\mathrm{NLIN}^d_{\mathrm{ca}}$ does not change if we take the *minimal* time $T(n) = n + 1$, called *real time*, i.e. the minimal time for that the information of any pixel of p can be communicated to the reference pixel, 1^d (see [14]).

Here is the second main result of this paper.

▶ **Theorem 18.** *For any $d > 0$ and any d-language L, the following assertions are equivalent:*

1. $L \in \mathrm{NLIN}^d_{ca}$*;*
2. $\mathit{coord}^d(L) \in \mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\, d+1)$*;*
3. $\mathit{coord}^d(L) \in \mathrm{ESO}(\mathrm{var}\, d+1)$*.*

This theorem is a straightforward consequence of Propositions 19 and 20 below.

▶ **Proposition 19.** *For any $d > 0$ and any d-language L,*

$$L \in \mathrm{NLIN}^d_{ca} \Leftrightarrow \mathit{coord}^d(L) \in \mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\,(d+1)).$$

Sketch of proof. $\boxed{\Rightarrow}$

Let $L \in \mathrm{NLIN}^d_{\mathrm{ca}}$. By the Remark preceding Theorem 18, L is recognized by a d-automaton $\mathcal{A} = (\Sigma, \Gamma, \delta, F)$ in *real time*, i.e. in time $n+1$. The sentence in $\mathrm{ESO}(\mathrm{var}\, d+1)$ that we are going to construct is of the form $\exists (R_s)_{s\in\Gamma}\ \forall \mathbf{x} \forall t\ \psi(\mathbf{x}, t)$, where:

- ψ is a quantifier-free formula that uses a list of exactly $d+1$ first-order variables $\mathbf{x} = (x_1, \ldots, x_d)$ and t. Intuitively, the d first ones represent the coordinates of any point in $dom(p) = [n]^d$ and the last one represents any of the first n instants $t \in [n]$ of the computation (the last instant $n+1$ is not explicitly represented);

- ψ uses, for each state $s \in \Gamma$, a relation symbol R_s of arity $d+1$. Intuitively, $R_s(a_1, \ldots, a_d, t)$ holds, for any $a = (a_1, \ldots, a_d) \in [n]^d$ and any $t \in [n]$, iff the state of cell a at instant t is s.

- ψ is the conjunction $\psi(\mathbf{x}, t) = \textsc{init}(\mathbf{x}, t) \wedge \textsc{step}(\mathbf{x}, t) \wedge \textsc{end}(\mathbf{x}, t)$ of three formulas whose intuitive meaning is the following.

- $\forall \mathbf{x} \forall t\ \textsc{init}(\mathbf{x}, t)$ describes the first configuration of $\mathcal{A}$, i.e. at initial instant 1, that is the input picture $p_1 = p$;
- $\forall \mathbf{x} \forall t\ \textsc{step}(\mathbf{x}, t)$ describes the computation between the instants t and $t+1$, for $t \in [n-1]$, i.e. describes the $(t+1)^{th}$ configuration p_{t+1} from the t^{th} one p_t, i.e. says $p_{t+1} \in \mathcal{A}(p_t)$;
- $\forall \mathbf{x} \forall t\ \textsc{end}(\mathbf{x}, t)$ expresses that the n^{th} configuration p_n leads to a (last) $(n+1)^{th}$ configuration $p_{n+1} \in \mathcal{A}(p_n)$ which is accepting, i.e. with an accepting state in cell 1^d: $p_{n+1}(1^d) \in F$.

Let us only give explicitly the second formula, $\textsc{step}$, which is the most central one (the last formula, $\textsc{end}$, is similar; the first one, $\textsc{init}$, is easy to construct):

$$\textsc{step}(\mathbf{x}, t) \equiv$$

$$\bigwedge_{a\in\{0,1\}^d}\ \bigwedge_{\nu\in\mathsf{neighb}_a(\Gamma)} \left\{ (\ \neg\max(t) \wedge P_a(\mathbf{x}) \wedge \bigwedge_{b\in\mathsf{Dom}_a} R_{\nu(b)}(\mathbf{x}+b, t)\ ) \to \bigoplus_{s\in\delta_a(\nu)} R_s(\mathbf{x}, \mathrm{succ}(t)) \right\}$$

Here, $\bigoplus$ denotes the exclusive disjunction. Furthermore:

- For $\mathbf{x} \in [n]^d$ and $a = (a_1, \ldots, a_d) \in \{0,1\}^d$, the formula $P_a(\mathbf{x})$ claims that the pixel $\mathbf{x}$ is in position a. Namely: $P_a(\mathbf{x}) \equiv \bigwedge_{i\in[d]} (\neg_i)\max(x_i)$, where $(\neg_i)$ is $\neg$ if $a_i = 1$, and *nothing* otherwise.
- For $b = (b_1, \ldots, b_d) \in \{0,1\}^d$, $\mathbf{x} + b$ abbreviates the tuple of terms $(\theta_1, \ldots, \theta_d)$ where, for each i, the term θ_i is x_i if $b_i = 0$, and $\mathrm{succ}(x_i)$ otherwise.

It is easy to verify that the formula $\forall \mathbf{x}\ \textsc{step}(\mathbf{x}, t)$ means $p_{t+1} \in \mathcal{A}(p_t)$, as claimed.

$\boxed{\Leftarrow}$ Assume $\mathsf{coord}^d(L) \in \mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\, d+1)$. That is, there is some sentence Φ in $\mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\, d+1)$ such that $p \in L \Leftrightarrow \mathsf{coord}^d(p) \models \Phi$. We want to prove $L \in \mathrm{NLIN}^d_{\mathrm{ca}}$, i.e. L is recognized by some d-automaton in linear time. Let us give the main idea of the proof for the simplest case $d = 1$ and a formula $\Phi \in \mathrm{ESO}(\forall^2, \mathrm{arity}\, 2)$ of the form

$$\Phi = \exists R \forall x \forall y \psi(x, y)$$

where R is a binary relation symbol and ψ is a quantifier-free formula where the only atoms in which R occurs, called R atoms, are of the following forms (1-4):

(1) $R(x, y)$; (2) $R(succ(x), y)$; (3) $R(x, succ(y))$; (4) $R(y, x)$.

First, notice that if the only atoms where R occurs are of the forms (1-3), i.e. the variables x, y only appear in this *unique* order in the arguments of R, then formula Φ has a *local* *behaviour*: points (x, y), $(succ(x), y)$ and $(x, succ(y))$ are *neighbours*, i.e. adjacent each other. This allows to construct a 1-automaton (nondeterministic cellular automaton of dimension 1) $\mathcal{A}$ that mimics Φ. Roughly, $\mathcal{A}$ successively guesses 'rows' $R(i, \ldots)$, for $i = 1, 2, \ldots n$, of R, and in the same time, it checks locally the coherence of each instantiation $\psi(i, j)$: more precisely, at instant i, the state of each cell j, $1 \leq j \leq n$, of $\mathcal{A}$ contains both values $R(i, j)$ and $R(i + 1, j)$. So, in case atoms of R are of the forms (1-3), the language L is recognized by such a 1-automaton $\mathcal{A}$ in linear time as claimed.

Now, let us consider the general case where the formula includes all the forms (1-4). Of course, the pixel of the form (4), $R(y, x)$, is not adjacent to pixels of the form (1-3) but is their symmetric (more precisely, is symmetric of $R(x, y)$) with respect to the diagonal $x = y$. The intuitive idea is to cut or to fold the 'picture' R along this diagonal: R is replaced by its two 'half pictures' denoted R_1 and R_2, that are superposed in the half square $x \leq y$ above the diagonal. More precisely, R_1 and R_2 are binary relations whose intuitive meaning is the following: for points (x, y) such that $x \leq y$, one has the equivalence $R_1(x, y) \Leftrightarrow R(x, y)$ and the equivalence $R_2(x, y) \Leftrightarrow R(y, x)$. By this transformation, each pixel $R_2(x, y)$ that represents the original pixel $R(y, x)$ lies at the same point (x, y) as pixel $R_1(x, y)$ that represents pixel $R(x, y)$, for $x \leq y$. The case $y \leq x$ is similar. This solves the problem of vicinity.

More precisely, the sentence $\Phi = \exists R \forall x \forall y \psi(x, y)$ is normalized as follows. Let $\mathsf{coherent}(x, y)$ denote the formula $x = y \to (R_1(x, y) \leftrightarrow R_2(x, y))$ whose universal closure ensures the coherence of R_1 and R_2 on the common part of R they both represent, that is the diagonal $x = y$. Using R_1 and R_2, it is not difficult to construct a formula

$$\psi'(x, y) = \mathsf{coherent}(x, y) \wedge \begin{pmatrix} x < y \to \psi_<(x, y) & \wedge \\ x = y \to \psi_=(x, y) & \wedge \\ x > y \to \psi_>(x, y) & \end{pmatrix}$$

such that the sentence $\Phi' = \exists R_1 \exists R_2 \forall x \forall y \psi'(x, y)$ in $\mathrm{ESO}(\forall^2, \mathrm{arity}\, 2)$ is equivalent to Φ. Let us describe and justify its precise form and meaning.

Table 1 Replacement of R-atoms by R_1- or R_2-atoms.

case	formula	$R(x, y)$	$R(\mathrm{succ}(x), y)$	$R(x, \mathrm{succ}(y))$	$R(y, x)$
$x < y$	$\psi_<(x, y)$	$R_1(x, y)$	$R_1(\mathrm{succ}(x), y)$	$R_1(x, \mathrm{succ}(y))$	$R_2(x, y)$
$x = y$	$\psi_=(x, y)$	$R_1(x, y)$	$R_2(x, \mathrm{succ}(y))$	$R_1(x, \mathrm{succ}(y))$	$R_1(x, y)$
$x > y$	$\psi_>(x, y)$	$R_2(y, x)$	$R_2(y, \mathrm{succ}(x))$	$R_2(\mathrm{succ}(y), x)$	$R_1(y, x)$

The formulas $\psi_<(x, y)$, $\psi_=(x, y)$ and $\psi_>(x, y)$ are obtained from formula $\psi(x, y)$ by substitution of R-atoms by R_1- or R_2-atoms according to the cases described in Table 1. It is easy to check that each replacement is correct according to its case. For instance, it is justified to replace each atom of the form $R(x, \mathrm{succ}(y))$ in ψ by $R_2(\mathrm{succ}(y), x)$ when $x > y$ (in order to obtain the formula $\psi_>(x, y)$), because when $x > y$ we get $\mathrm{succ}(y) \leq x$ and hence the equivalence $R(x, \mathrm{succ}(y)) \leftrightarrow R_2(\mathrm{succ}(y), x)$ holds, by definition of R_2.

Notice that the variables x, y always occur in this order in each R_1- or R_2-atom of the formulas $\psi_<$ and $\psi_=$ (see Table 1). At the opposite, they always occur in the reverse order y, x in the formula $\psi_>(x, y)$. This is not a problem because, by symmetry, the roles of x and y can be exchanged and the universal closure $\forall x \forall y (x > y \to \psi_>(x, y))$ is trivially equivalent to $\forall x \forall y (y > x \to \psi_>(y, x))$. So, the above sentence Φ' – and hence, the original sentence

Φ – is equivalent to the sentence denoted Φ" obtained by replacing in Φ' the subformula $x > y \to \psi_>(x,y)$ by $y > x \to \psi_>(y,x)$. By construction, relation symbols R_1, R_2 only occur in Φ in atoms of the three required 'sorted' forms: $R_i(x,y)$, $R_i(\mathrm{succ}(x),y)$ or $R_i(x,\mathrm{succ}(y))$.

Finally, to be precise, there remain two difficulties so that a 1-automaton can simulate the 'sorted' sentence Φ" in linear time, by the informal algorithm described above:

- the presence of equalities and inequalities in the sentence;
- the forms of the atoms involving input relation symbols.

It is easy to get rid of equalities and inequalities by introducing new binary relation symbols defined and used in a 'sorted' manner too (see (1-3)). Concerning the second point, we can assume, without loss of generality, that the only atoms involving the input relation symbols $(Q_s)_{s\in\Sigma}$ are of the two forms $Q_s(x)$ or $Q_s(y)$. As we do for equalities and inequalities, we can get rid of atoms of the form $Q_s(y)$ by introducing new binary ESO relation symbols : intuitively, they convey each bit $Q_s(a)$ at each point of coordinates $(a,\ldots)$ or $(\ldots,a)$; those new binary relations are also defined and used in a 'sorted' manner. The fact that all the atoms involving the input are of the form $Q_s(x)$ allows to consider this input in the initial configuration of the computation of the 1-automaton *but in no later configuration* as required. So, the sketch of proof is complete for the case $d = 1$.

For the general case, i.e. for any dimension d, the ideas and the steps of the proof are exactly the same as for $d = 1$ but the notations and details of the proof are much more technical. To give an idea, let us succinctly describe the ESO relations of arity $d+1$ introduced in the main normalization step. Here again, each ESO relation symbol R of the original sentence Φ in $\mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\, d+1)$ is replaced by – or, intuitively, 'divided into' – $(d+1)!$ new ESO relation symbol R_α of the same arity $d+1$, where α is a permutation of the set of indices $[d+1]$. The intended meaning of each relation R_α is the following: for each tuple $(a_1,\ldots,a_{d+1}) \in [n]^{d+1}$ such that $a_1 \le a_2 \ldots \le a_{d+1}$, the equivalence

$$R_\alpha(a_1,\ldots,a_{d+1}) \leftrightarrow R(a_{\alpha(1)},\ldots,a_{\alpha(d+1)})$$

holds. Then, we introduce a partition of the domain $[n]^{d+1}$ into subdomains, similar to the partition of the domain $[n]^2$ described above for $d = 1$ into the diagonal $x = y$ and the two half domains over and under the diagonal $x < y$ and $x > y$. According to the case (i.e. subdomain of the partition), this allows to replace each R atom in Φ by an atom of one of the two following sorted forms, $R_\alpha(\mathbf{x})$ and $R_\alpha(\mathbf{x}^{(i)})$, where $\mathbf{x} = (x_1,\ldots,x_{d+1})$, $1 \le i \le d+1$ and $\mathbf{x}^{(i)}$ is the tuple $\mathbf{x}$ where x_i is replaced by $\mathrm{succ}(x_i)$. Finally, the equalities and inequalities are similarly eliminated in the sentence and we normalize it with respect to the input d-ary relations $(Q_s)_{s\in\Sigma}$ by using new ESO relation symbols of arity $d+1$ to convey the input information: in the final sorted sentence all the Q_s atoms are of the unique form $Q_s(x_1,\ldots,x_d)$. For such a sorted $\mathrm{ESO}(\forall^{d+1}, \mathrm{arity}\, d+1)$-sentence Φ, it is now easy to construct a d-automaton that generalizes the automaton described above in case $d = 1$, and checks in linear time whether $\mathsf{coord}^d(p) \models \Phi$. ◀

▶ **Proposition 20.** *For any* $d > 0$, $\mathrm{ESO}(\mathrm{var}\, d) \subseteq \mathrm{ESO}(\forall^d, \mathrm{arity}\, d)$ *on coordinate structures.*

Sketch of proof. We first prove a kind of Skolemization of $\mathrm{ESO}(\mathrm{var}\, d)$-formulas, thus providing a first normalization of these formulas, in which the first-order part is *universal* and includes *the same number of first-order variables* as the initial formula. To illustrate the procedure that performs this preliminary normalisation, let us run it on a very simple first-order formula with *two* variables: $\varphi \equiv \exists x\, (\forall y U(x,y) \vee \exists y D(x,y)\,)$. We introduce three new relation symbols R_1, R_2, R_3 corresponding to the quantified subformulas of φ.

$$\begin{array}{lll} \mathrm{def}_1(R_1) \equiv \forall x: & R_1(x) & \leftrightarrow & \forall y\, U(x,y) \\ \mathrm{def}_2(R_2) \equiv \forall x: & R_2(x) & \leftrightarrow & \exists y\, D(x,y) \\ \mathrm{def}_3(R_3) \equiv \forall x: & R_3(x) & \leftrightarrow & R_1(x) \vee R_2(x) \end{array}$$

Hence our initial formula can be rewritten:

$$\exists R_1, R_2, R_3 : \left(\bigwedge_{1\le i\le 3} \mathrm{def}_i(R_i) \right) \wedge \exists x\ R_3(x). \tag{3}$$

It is easily seen that (3) can be written as a conjunction of prenex formulas, each of which involves no more than two variables and has a quantifier prefix of the shape $\forall x \forall y$ or $\forall x \exists y$ (we include in this later form the subformula $\exists x\ R_3(x)$). All in all, φ is equivalent to a formula of the form:

$$\exists R_1, R_2, R_3 : \forall x \forall y \psi(x,y,\mathbf{R}) \wedge \forall x \exists y \theta(x,y,\mathbf{R}), \tag{4}$$

where ψ and θ are quantifier-free. In order to put this conjunction under prenex form without adding a new first-order variable, we have to ”replace” the existential quantifier by a universal one. (Afterward φ, as a conjunction of formulas of prefix $\forall x, y$, could be written under the requisite shape.) To proceed, we get use of the arithmetics embedded in coordinate structures. It allows to defining a binary relation W with intended meaning: $\mathrm{W}(x,y)$ iff there exists $z \le y$ such that $\theta(x,z)$ holds. This interpretation is achieved thanks to the formula:

$$\forall x,y : \{\min(y) \to \big(\mathrm{W}(x,y) \leftrightarrow \theta(x,y)\big)\} \wedge \{\mathrm{W}(x,\mathrm{succ}(y)) \leftrightarrow \big(\theta(x,\mathrm{succ}(y)) \vee \mathrm{W}(x,y)\big)\} \tag{5}$$

Under assumption (5), the assertion $\forall x \exists y \theta(x,y)$ is equivalent to $\forall x \forall y : \max(y) \to \mathrm{W}(x,y)$. This allows to rewriting (4), and hence φ, as the $\mathrm{ESO}(\forall^2)$-formula:

$$\exists R_1, R_2, R_3, \mathrm{W}\ \big(\ (5)\ \wedge \forall x \forall y \psi(x,y,\mathbf{R})\ \wedge \forall x \forall y \big(\max(y) \to \mathrm{W}(x,y)\big).\ \big) \tag{6}$$

Thus, the above considerations allow to show the normalization $\mathrm{ESO}(\mathrm{var}\, d) = \mathrm{ESO}(\forall^d)$ on coordinate structures. It remains to prove $\mathrm{ESO}(\forall^d) = \mathrm{ESO}(\forall^d, \mathrm{arity}\, d)$. It amounts to build, for each formula Φ of type $\exists \mathbf{R} \forall x_1, \dots, x_d \varphi$, where φ is quantifier-free and $\mathbf{R}$ is a tuple of relation symbols of any arity, a formula Φ' with the same shape, but in which all relation symbols are of arity $\le d$, such that Φ and Φ' have the same models, as far as pixel structures are concerned. The possibility to replace a k-ary ($k \ge d$) relation symbol R of Φ by d-ary symbols rests in the limitation of the number of first-order variables in Φ: each atomic formula involving R has the form $R(t_1, \dots, t_k)$ where the t_i's are terms built on $x_1, \dots, x_d$. Therefore, although R is k-ary, *in each of its occurrences* it behaves as a d-ary symbol, dealing with the sole variables $x_1, \dots, x_d$. Hence, the key is to create a d-ary symbol for each occurrence of R in Φ or, more precisely, for each k-tuple of terms $(t_1, \dots, t_k)$ involved in a R-atomic formula. Let us again opt for a ‘proof-by-example’ choice and illustrate the procedure on a very simple case.

Let Φ be the $\mathrm{ESO}(\forall^2, \mathrm{arity}\, 3)$-formula $\exists R \forall x, y \varphi(x,y,R)$, where $\varphi \equiv R(x,y,x) \wedge \neg R(y,x,y)$. Introduce two new *binary* relation symbols $R_{(x,y,x)}$ and $R_{(y,x,y)}$ associated to the triple of terms (x,y,x) and (y,x,y) involved in Φ, and fix their interpretation as follows: for any $\forall a, b \in [n]$, $R_{(x,y,x)}(a,b) \Leftrightarrow R(a,b,a)$ and $R_{(y,x,y)}(a,b) \Leftrightarrow R(b,a,b)$. Then we get the equivalence: $\langle S, R\rangle \models \forall x, y : R(x,y,x) \wedge \neg R(y,x,y)$ iff $\langle S, \mathbf{R}\rangle \models \forall x, y : R_{(x,y,x)}(x,y) \wedge \neg R_{(y,x,y)}(x,y)$ which, in turn, yields the implication:

$$S \models \exists R \forall x, y : R(x,y,x) \wedge \neg R(y,x,y) \Rightarrow S \models \exists \mathbf{R} \forall x, y : R_{(x,y,x)}(x,y) \wedge \neg R_{(y,x,y)}(x,y) \tag{7}$$

The converse implication would immediatly complete the proof. Unfortunately, it does not hold, since the second formula has a model, while the first has not.

To get the right-to-left implication in (7), we have to strengthen the second formula with some assertion that compels the tuple $R_{(x,y,x)}, R_{(y,x,y)}$ to be, in some sense, the binary representation of some ternary relation. This last construction is more sophisticated than the preceding ones, and we can't detail it here. ◀

Acknowledgments

We warmly thank Gaétan Richard who has obtained in collaboration with us several results of this paper, in particular Proposition 19 and Proposition 20. We also thank the anonymous referees for their careful reading, corrections and suggestions that have helped to notably improve the readability of the paper.

References

1 Bernd Borchert. Formal language characterizations of P, NP, and PSPACE. *J. of Automata, Languages and Combinatorics*, 13(3/4):161–183, 2008.

2 J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

3 M. Delorme and J. Mazoyer. *Cellular automata: A parallel model.* Springer, 373 pages, Mathematics and Its Applications, 1998.

4 A. Durand and E. Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic*, V:1–18, 2006.

5 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 1995.

6 D. Giammarresi and A. Restivo. *Two-dimensional languages, in: Handbook of Theoretical Computer Science*, volume 3- Beyond words, chapter 4, pages 215–267. Springer-Verlag New York, 1997.

7 D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125(1):32 – 45, 1996.

8 E. Grädel, Ph. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S.Weinstein. *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. Springer, 2007.

9 N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer, 1999.

10 M. Latteux and D. Simplot. Recognizable picture languages and domino tiling. Internal Report IT-94-264, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille, France, 1994.

11 L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

12 Steven Lindell. A normal form for first-order logic over doubly-linked data structures. *Int. J. Found. Comput. Sci.*, 19(1):205–217, 2008.

13 Rudolph Sommerhalder and S. Christian van Westrhenen. Parallel language recognition in constant time by cellular automata. *Acta Inf.*, 19:397–407, 1983.

14 V. Terrier. Language recognition by cellular automata. In *Handbook of Natural Computing, Section 1, Cellular Automata.* Springer-Verlag, 2011.

Pebble Games and Linear Equations

Martin Grohe[1] **and Martin Otto**[*2]

1 **Humboldt-Universität zu Berlin, Germany**
grohe@informatik.hu-berlin.de

2 **Technische Universität Darmstadt, Germany**
otto@mathematik.tu-darmstadt.de

Abstract

We give a new, simplified and detailed account of the correspondence between levels of the Sherali–Adams relaxation of graph isomorphism and levels of pebble-game equivalence with counting (higher-dimensional Weisfeiler–Lehman colour refinement). The correspondence between basic colour refinement and fractional isomorphism, due to Ramana, Scheinerman and Ullman [18], is re-interpreted as the base level of Sherali–Adams and generalised to higher levels in this sense by Atserias and Maneva [1], who prove that the two resulting hierarchies interleave. In carrying this analysis further, we here give (a) a precise characterisation of the level k Sherali–Adams relaxation in terms of a modified counting pebble game; (b) a variant of the Sherali–Adams levels that precisely match the k-pebble counting game; (c) a proof that the interleaving between these two hierarchies is strict. We also investigate the variation based on boolean arithmetic instead of real/rational arithmetic and obtain analogous correspondences and separations for plain k-pebble equivalence (without counting). Our results are driven by considerably simplified accounts of the underlying combinatorics and linear algebra.

1998 ACM Subject Classification F.4.1, G.2.2

Keywords and phrases Finite model theory, finite variable logics, graph isomorphism, Weisfeiler-Lehman algorithm, linear programming, Sherali–Adams hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.289

1 Introduction

We study a surprising connection between equivalence in finite variable logics and a linear programming approach to the graph isomorphism problem. This connection has recently been uncovered by Atserias and Maneva [1], building on earlier work of Ramana, Scheinerman and Ullman [18] that just concerns the 2-variable case.

Finite variable logics play a central role in finite model theory. Most important for this paper are finite variable logics with counting, which have been specifically studied in connection with the question for a logical characterisation of polynomial time and in connection with the graph isomorphism problem (e.g. [4, 8, 9, 13, 14, 17]). Equivalence in finite variable logics can be characterised in terms of simple combinatorial games known as pebble games. Specifically, C^k-equivalence can be characterised by the bijective k-pebble game introduced by Hella [11]. Cai, Fürer and Immerman [4] observed that C^k-equivalence exactly corresponds to indistinguishability by the k-dimensional Weisfeiler-Lehman (WL) algorithm,[1]

[*] This Martin gratefully acknowledges the other Martin's academic hospitality at HU Berlin during his sabbatical in the winter 2011/12.

[1] The dimensions of the WL algorithm are counted differently in the literature; what we call "k-dimensional" here is sometimes called "$(k-1)$-dimensional".

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 289–304
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a combinatorial graph isomorphism algorithm introduced by Babai, who attributed it to work of Weisfeiler and Lehman in the 1970s. The 2-dimensional version of the WL algorithm precisely corresponds to an even simpler isomorphism algorithm known as colour refinement.

The isomorphisms between two graphs can be described by the integral solutions of a system of linear equations. If we have two graphs with adjacency matrices A and B, then each isomorphism from the first to the second corresponds to a permutation matrix X such that $X^tAX = B$, or equivalently

$$AX = XB. \tag{1}$$

If we view the entries of X as variables, this equation corresponds to a system of linear equations. We can add inequalities that force X to be a permutation matrix and obtain a system ISO of linear equations and inequalities whose integral solutions correspond to the isomorphisms between the two graphs. In particular, the system ISO has an integral solution if, and only if, the two graphs are isomorphic.

What happens if we drop the integrality constraints, that is, we admit arbitrary real solutions of the system ISO? We can ask for doubly stochastic matrices X satisfying equation (1). (A real matrix is *doubly stochastic* if its entries are non-negative and all row sums and column sums are one.) Ramana, Scheinerman and Ullman [18] proved a beautiful result that establishes a connection between linear algebra and logic: the system ISO has a real solution if, and only if, the colour refinement algorithm does not distinguish the two graphs with adjacency matrices A and B. Recall that the latter is equivalent to the two graphs being C^2-equivalent.

To bridge the gap between integer linear programs and their LP-relaxations, researchers in combinatorial optimisation often add additional constraints to the linear programs to bring them closer to their integer counterparts. The Sherali–Adams hierarchy [21] of relaxations gives a systematic way of doing this. For every integer linear program IL in n variables and every positive integer k, there is a *rank-k Sherali–Adams relaxation* IL(k) of IL, such that IL(1) is the standard LP-relaxation of IL where all integrality constraints are dropped and IL(n) is equivalent to IL. There is a considerable body of research studying the strength of the various levels of this and related hierarchies (e.g. [2, 3, 5, 16, 20, 19]).

Quite surprisingly, Atserias and Maneva [1] were able to lift the Ramana–Scheinerman–Ullman result, which we may now restate as an equivalence between ISO(1) and C^2-equivalence, to a close correspondence between the higher levels of the Sherali–Adams hierarchy for ISO and the logics C^k. They proved for every $k \geq 2$:

1. if ISO(k) has a (real) solution, then the two graphs are C^k-equivalent;
2. if the two graphs are C^k-equivalent, then ISO($k-1$) has a solution.

Atserias and Maneva used this results to transfer results about the logics C^k to the world of polyhedral combinatorics and combinatorial optimisation, and conversely, results about the Sherali–Adams hierarchy to logic.

Atserias and Maneva [1] left open the question whether the interleaving between the levels of the Sherali–Adams hierarchy and the finite-variable-logic hierarchy is strict or whether either the correspondence between C^k-equivalence and ISO(k) or the correspondence between C^k-equivalence and ISO($k-1$) is exact. Note that for $k = 2$ the correspondence between C^k-equivalence and ISO($k-1$) is exact by the Ramana–Scheinerman–Ullman theorem. We prove that for all $k \geq 3$ the interleaving is strict. However, we can prove an exact correspondence between ISO($k-1$) and a variant of the bijective k-pebble game that characterises C^k-equivalence. This variant, which we call the weak bijective k-pebble game, is actually equivalent to a game called ($k-1$)-sliding game by Atserias and Maneva.

Maybe most importantly, we prove that a natural combination of equalities from ISO(k) and ISO$(k-1)$ gives a linear program ISO$(k-1/2)$ that characterises C^k-equivalence exactly.

To obtain these results, we give simple new proofs of the theorems of Ramana, Scheinerman and Ullman and of Atserias and Maneva. Whereas the previous proofs use two non-trivial results from linear algebra, the Perron–Frobenius Theorem (about the eigenvalues of positive matrices) and the Birkhoff–von Neumann Theorem (stating that every doubly stochastic matrix is a convex combination of permutation matrices), our proofs only use elementary linear algebra. This makes them more transparent and less mysterious (at least to us).

In fact, the linear algebra we use is so simple that much of it can be carried out not only over the field of real numbers, but over arbitrary semirings. By using similar algebraic arguments over the boolean semiring (with disjunction as addition and conjunction as multiplication), we obtain analogous results to those for C^k-equivalence for the ordinary k-variable logic L^k, characterising L^k-equivalence, i.e., k-pebble game equivalence without counting, by systems of 'linear' equations over the boolean semiring.

For the ease of presentation, we have decided to present our results only for undirected simple graphs. It is easy to extend all results to relational structures with at most binary relations. Atserias and Maneva did this for their results, and for ours the extension works analogously. An extension to structures with relations of higher arities also seems possible, but is more complicated and comes at the price of loosing some of the elegance of the results.

Due to space limitations, we have to omit many details and proofs in this conference version of the paper. They can be found in the full version of the paper [10]. The present version of the paper contains a fairly complete account of our proof of the Ramana–Scheinerman–Ullman theorem, including the linear algebra that is also underlying they higher-dimensional results. Most proofs regarding the correspondence between the Sherali–Adams hierarchy and C^k-equivalence are omitted.

2 Finite variable logics and pebble games

We assume the reader is familiar with the basics of first-order logic FO. We almost exclusively consider first-order logic over finite graphs, which we view as finite relational structures with one binary relation. We assume graphs to be undirected and loop-free. For every positive integer k, we let L^k be the fragment of FO consisting of all formulae that contain at most k distinct variables. We let C^k be the extension of L^k by *counting quantifiers* $\exists^{\geq n}$, where$\exists^{\geq n} x\,\varphi$ means that there are at least n elements x such that φ is satisfied. L^k-*equivalence* of structures $\mathcal{A}, \mathcal{B}$ is denoted by $\mathcal{A} \equiv^k_{\mathsf{L}} \mathcal{B}$ and C^k-*equivalence* by $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$. Both equivalences can be characterised in terms of pebble games. We briefly sketch the *bijective k-pebble game* [11] that characterises C^k-equivalence. The game is played by two players on a pair $\mathcal{A}, \mathcal{B}$ of structures. A *play* of the game consists of a (possibly infinite) sequence of *rounds*. In each round, player **I** picks up one of his pebbles, and player **II** picks up her corresponding pebble. Then player **II** chooses a bijection f between $\mathcal{A}$ and $\mathcal{B}$ (if no such bijection exists, that is, if the structures have different cardinalities, player **II** immediately looses). Then player **I** places his pebble on an element a of $\mathcal{A}$, and player **II** places her pebble on $f(a)$. Note that after each round r there is a subset $p \subseteq \mathcal{A} \times \mathcal{B}$ consisting of the at most k pairs of elements on which the pairs of corresponding pebbles are placed. We call p the *position* after round r. Player **I** wins the play if every position that occurs is a local isomorphism, that is, a local mapping from $\mathcal{A}$ to $\mathcal{B}$ that is injective and preserves membership and non-membership in all relations (adjacency and non-adjacecny if $\mathcal{A}$ and $\mathcal{B}$ are graphs). Then $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$ if, and only if, player **II** has a winning strategy for the game.

C^k-equivalence also corresponds to a simple combinatorial algorithm for graph isomorphism testing known as the Weisfeiler-Lehman algorithm.

We refer the reader to the textbooks [6, 7, 12, 15] and the monograph [17] for a more thorough exposition of the material sketched here.

3 Basic combinatorics and linear algebra

We consider matrices with entries in $\mathbb{B} = \{0, 1\}$, $\mathbb{Q}$ or $\mathbb{R}$. A matrix $X \in \mathbb{R}^{m,n}$ with m rows and n columns has entry X_{ij} in row $i \in [m] = \{1, \dots, m\}$ and column $j \in [n] = \{1, \dots, n\}$. We write E_n for the n-dimensional unit matrix.

We write $X \geqslant 0$ to say that (the real or rational) matrix X has only non-negative entries, and $X > 0$ to say that all entries are strictly positive. We also speak of *non-negative* or *strictly positive matrices* in this sense. For a boolean matrix, strict positivity, $X > 0$ means that all entries are 1. A square $n \times n$-matrix is *doubly stochastic* if its entries are non-negative and if the sum of entries across every row and column is 1. *Permutation matrices* are doubly stochastic matrices over $\{0, 1\}$, with precisely one 1 in every row and in every column.

It will be useful to have the shorthand notation $X_{D_1 D_2} = 0$ for the assertion that $X_{d_1 d_2} = 0$ for all $d_1 \in D_1$, $d_2 \in D_2$.

3.1 Decomposition into irreducible blocks

With $X \in \mathbb{R}^{n,n}$ associate the directed graph $\mathrm{G}(X) := ([n], \{(i, j) \colon X_{ij} \neq 0\})$. The strongly connected components of $\mathrm{G}(X)$ induce a partition of the set $[n] = \{1, \dots, n\}$ of rows/columns of X. X is called *irreducible* if this partition has just the set $[n]$ itself.

Note that X is irreducible iff $P^t X P$ is irreducible for every permutation matrix P.

▶ **Observation 3.1.** *Let $X \in \mathbb{R}^{n,n} \geqslant 0$ with strictly positive diagonal entries. If X is irreducible, then all powers X^ℓ for $\ell \geqslant n - 1$ have non-zero entries throughout. Moreover, if X is irreducible, then so is X^ℓ for all $\ell \geqslant 1$.*

Let us call two matrices $Z, Z' \in \mathbb{R}^{n,n}$ *permutation-similar* or *S_n-similar*, $Z \sim_{S_n} Z'$, if $Z' = P^t Z P$ for some permutation matrix P, i.e., if one is obtained from the other by a coherent permutation of rows and columns.

▶ **Lemma 3.2.** *Every symmetric $Z \in \mathbb{R}^{n,n} \geqslant 0$ is permutation-similar to some block diagonal matrix $\mathrm{diag}(Z_1, \dots, Z_s)$ with irreducible blocks $Z_i \in \mathbb{R}^{n_i,n_i}$.*

The permutation matrix P corresponding to the row- and column-permutation $p \in S_n$ that puts Z into block diagonal form $P^t Z P = \mathrm{diag}(Z_1, \dots, Z_s)$ with irreducible blocks, is unique up to an outer permutation that re-arranges the block intervals $([k_i + 1, k_i + n_i])_{1 \leqslant i \leqslant s}$ where $k_i = \sum_{j<i} n_j$, and a product of inner permutations within each one of these s blocks.

The underlying partition $[n] = \dot{\bigcup}_{1 \leqslant i \leqslant s} D_i$ where $D_i := p([k_i + 1, k_i + n_i])$ for $k_i = \sum_{j<i} n_j$, is uniquely determined by Z.[2]

In the following we refer to the *partition induced by a symmetric matrix Z*.

[2] Here we regard two partitions as identical if they have the same partition sets, i.e., we ignore their indexing/enumeration.

▶ **Observation 3.3.** *In the situation of Lemma 3.2, the partition $[n] = \dot{\bigcup}_i D_i$ induced by the symmetric matrix Z is the partition of $[n]$ into the vertex sets of the connected components of $G(Z)$. Then, for every pair $i \neq j$, $Z_{D_i D_j} = 0$, while all the minors $Z_{D_i D_i}$ are irreducible.*[3]

If, moreover, Z has strictly positive diagonal entries, then the partition induced by Z is the same as that induced by Z^ℓ, for any $\ell \geqslant 1$; for $\ell \geqslant n-1$, the diagonal blocks $(Z^\ell)_{D_i D_i}$ have non-zero entries throughout: $(Z^\ell)_{D_i D_i} > 0$.

The last assertion says that for a symmetric $n \times n$ matrix Z with non-negative entries and no zeroes on the diagonal, all powers Z^ℓ for $\ell \geqslant n-1$ are *good symmetric* in the sense of the following definition.

▶ **Definition 3.4.** Let $Z \geqslant 0$ be symmetric with strictly positive diagonal. Then Z is called *good symmetric* if w.r.t. the partition $[n] = \dot{\bigcup}_i D_i$ induced by Z, all $Z_{D_i D_i} > 0$.

More generally, a not necessarily symmetric matrix $X \geqslant 0$ without null rows or columns is *good* if $Z = XX^t$ and $Z' = X^t X$ are good in the above sense.

The importance of this notion lies in the fact that, as observed above, for an arbitrary symmetric $n \times n$ matrix $Z \geqslant 0$ without zeroes on the diagonal, the partition induced by Z is the same as that induced by the good symmetric matrix $\hat{Z} := Z^{n-1}$; and, as for any good matrix, this partition can simply be read off from $\hat{Z}$: $i, j \in [n]$ are in the same partition set if, and only if, $\hat{Z}_{ij} \neq 0$.

▶ **Definition 3.5.** Consider partitions $[n] = \dot{\bigcup}_{i \in I} D_i$ and $[m] = \dot{\bigcup}_{i \in I} D'_i$ of the sets $[n]$ and $[m]$ with the same number of partition sets. We say that these two partitions are *X-related* for some matrix $X \in \mathbb{R}^{n,m}$ if

(i) $X \geqslant 0$ has no null rows or columns, and

(ii) $X_{D_i D'_j} = 0$ for every pair of distinct indices $i, j \in I$.

Note that partitions that are X-related are X^t-related in the opposite direction. More importantly, each one of the X/X^t-related partitions can be recovered from the other one through X according to

$$\begin{array}{rcl} D'_i & = & \{d' \in [m] \colon X_{dd'} > 0 \text{ for some } d \in D_i\}, \\ D_i & = & \{d \in [n] \colon X_{dd'} > 0 \text{ for some } d' \in D'_i\}. \end{array}$$

For a more algebraic treatment, we associate with the partition sets D_i of a partition $[n] = \dot{\bigcup}_{i \in I} D_i$ the *characteristic vectors* $\mathbf{d}_i$ with entries 1 and 0 according to whether the corresponding component belongs to D_i:

$$\mathbf{d}_i = \textstyle\sum_{d \in D_i} \mathbf{e}_d,$$

where $\mathbf{e}_d$ is the d-th standard basis vector. In terms of these characteristic vectors $\mathbf{d}_i$ for $[n] = \dot{\bigcup}_{i \in I} D_i$ and $\mathbf{d}'_i$ for $[m] = \dot{\bigcup}_{i \in I} D'_i$, the X/X^t-relatedness of these partitions means that

$$\begin{array}{rcl} D'_i & = & \{d' \in [m] \colon (X^t \mathbf{d}_i)_{d'} > 0\}, \\ D_i & = & \{d \in [n] \colon (X \mathbf{d}'_i)_d > 0\}. \end{array}$$

[3] Note that this does not depend on the enumeration of the partition set D_i, because irreducibility is invariant under permutation-similarity.

▶ **Lemma 3.6.** *If two partitions $[n] = \dot{\bigcup}_{i \in I} D_i$ and $[n] = \dot{\bigcup}_{i \in I} D'_i$ of the same set $[n]$ are X-related for some doubly stochastic matrix $X \in \mathbb{R}^{n,n}$, then $|D_i| = |D'_i|$ for all $i \in I$, and for the characteristic vectors $\mathbf{d}_i$ and $\mathbf{d}'_i$ of the partition sets D_i and D'_i*

$$\mathbf{d}_i = X\mathbf{d}'_i \quad \text{and} \quad \mathbf{d}'_i = X^t\mathbf{d}_i.$$

Proof. Observe that for all $d \in [n]$ we have $0 \leq (X\mathbf{d}'_i)_d = \sum_{d' \in D'_i} X_{dd'} \leq 1$. It follows immediately from the definition of X-relatedness that $(X\mathbf{d}'_i)_d = 0$ for all $d \notin D_i$. Therefore,

$$|D_i| \geqslant \sum_{d \in D_i} (X\mathbf{d}'_i)_d = \sum_{d \in [n]} (X\mathbf{d}'_i)_d = \sum_{d' \in D'_i} \sum_{d \in [n]} X_{dd'} = |D'_i|.$$

Similarly, $0 \leq (X^t\mathbf{d}_i)_{d'} \leq 1$ for $d' \in [n]$, and $|D'_i| \geq \sum_{d' \in D'_i} (X^t\mathbf{d}_i)_{d'} = |D_i|$. Together, we obtain

$$|D_i| = \sum_{d \in D_i} (X\mathbf{d}'_i)_d = |D'_i| = \sum_{d' \in D'_i} (X^t\mathbf{d}_i)_{d'}.$$

As all summands are bounded by 1, this implies $(X\mathbf{d}'_i)_d = 1$ for all $d \in D_i$ and $(X^t\mathbf{d}_i)_{d'} = 1$ for all $d' \in D_i$. ◀

▶ **Lemma 3.7.** *Let $X \geqslant 0$ be an $m \times n$ matrix without null rows or columns. Then the $m \times m$ matrix $Z := XX^t$ and the $n \times n$ matrix $Z' := X^tX$ are symmetric with positive entries on their diagonals. Moreover, the (unique) partitions of $[m]$ and $[n]$ that are induced by Z and Z', respectively, are X/X^t-related.*[4]

Proof. It is obvious that Z and Z' are symmetric with positive diagonal entries. Let partitions $[m] = \dot{\bigcup}_{i \in I} D_i$ and $[n] = \dot{\bigcup}_{i \in I'} D'_i$ be obtained from decompositions of Z and Z' into irreducible blocks. We need to show that the non-zero entries in X give rise to a coherent bijection between the index sets I and I' of the two partitions, in the sense that partition sets D_i and D'_j are related if, and only if, some pair of members $d \in D_i$ and $d' \in D'_j$ have a positive entry $X_{dd'}$. Then a re-numbering of one of these partitions will make them X-related in the sense of Definition 3.5. Recall from Observation 3.3 that the D_i are the vertex sets of the connected components of $G(XX^t)$ on $[m]$, while the D'_i the are the vertex sets of the connected components of $G(X^tX)$ on $[n]$.

Consider the uniformly directed bipartite graph $G(X)$ on $[m] \dot{\cup} [n]$ with an edge from $i \in [m]$ to $j \in [n]$ if $X_{ij} > 0$. In light of the symmetry of the whole situation w.r.t. X and X^t, it just remains to argue for instance that no $i \in [m]$ can have edges into two distinct sets of the partition $[n] = \dot{\bigcup}_{i \in I'} D'_i$. But any two target nodes of edges from one and the same $i \in [n]$ are in the same connected component of $G(X^tX)$, hence in the same partition set. ◀

In the situation of Lemma 3.7, powers of Z induce the same partitions as Z, and the partitions induced by $(Z^\ell X)(Z^\ell X)^t = Z^{2\ell+1}$ are X/X^t-related as well as $Z^\ell X/X^t Z^\ell$-related, for all $\ell \geqslant 1$.

For $\ell \geqslant n/2 - 1$, the matrix $Z^\ell X$ has no null rows or columns: else $Z^\ell X (Z^\ell X)^t = Z^{2\ell+1}$ would have to have a zero entry on the diagonal, contradicting the fact that this symmetric matrix is good symmetric in the sense of Definition 3.4. The same reasoning shows that $Z^\ell X$ is itself good in the sense of Definition 3.4.

[4] As X/X^t-relatedness refers to partitions presented with an indexing of the partition sets, we need to allow a suitable re-indexing for at least one of them, so as to match the other one.

▶ **Corollary 3.8.** *Let $X \geqslant 0$ be an $m \times n$ matrix without null rows or columns, $Z = XX^t$, $Z' = X^tX$ the associated symmetric matrices with non-zero entries on the diagonal. Then for $\ell \geqslant m-1$, the matrix $\hat{X} := Z^\ell X = X(Z')^\ell$ and its transpose $\hat{X}^t = X^tZ^\ell = (Z')^\ell X^t$ are good and relate the partitions $[m] = \dot{\bigcup}_i D_i$ and $[n] = \dot{\bigcup}_i D'_i$ induced by Z and Z', respectively.*[4] *Moreover,*

(i) $\hat{X}_{D_iD'_i} > 0$ *for all i, and*

(ii) $\hat{X}_{D_iD'_j} = 0$ *for all $i \neq j$.*

Aside: boolean vs. real arithmetic

Looking at matrices with $\{0,1\}$-entries, we may not only treat them as matrices over $\mathbb{R}$ as we have done so far, but also over other fields, or as matrices over the boolean semiring $\mathbb{B} = \{0,1\}$ with the logical operations of $\vee$ for addition and $\wedge$ for multiplication. Though not even forming a ring, boolean arithmetic yields a very natural interpretation in the context where we associate non-negative entries with edges, as we did in passage from X to $G(X)$. The 'normalisation map' $\chi \colon \mathbb{R}_{\geqslant 0} \to \{0,1\}$, $x \mapsto 1$ iff $x > 0$, relates the arithmetic of reals $x, y \geqslant 0$ to boolean arithmetic in

$$\chi(x+y) = \chi(x) \vee \chi(y) \quad \text{and} \quad \chi(xy) = \chi(x) \wedge \chi(y).$$

This is the 'logical' arithmetic that supports, for instance, arguments used in Observation 3.1: for any real $n \times n$ matrix $X \geqslant 0$, $(XX)_{ij} = \sum_k X_{ik}X_{kj} \neq 0$ iff there is at least one $k \in [n]$ for which $X_{ik} \neq 0$ and $X_{kj} \neq 0$ iff $\bigvee_{k\in[n]}(\chi(X_{ik}) \wedge \chi(X_{kj})) = 1$. It is no surprise, therefore, that several of the considerations apparently presented for real non-negative matrices above, have immediate analogues for boolean arithmetic – in fact, one could argue, that the boolean interpretation is closer to the combinatorial essence. We briefly sum up these analogues with a view to their use in the analysis of L^k-equivalence, while the real versions are related to C^k-equivalence. The boolean analogue of a doubly stochastic matrix with non-negative real entries is a matrix without null rows or columns.

Also note that the definitions of irreducibility and X-relatedness are applicable to boolean matrices without any changes. Observations 3.1 and 3.3 go through (as just indicated), and so does Lemma 3.2. For Lemma 3.6, one may look at X-related partitions of sets $[m]$ and $[n]$, where not necessarily $n = m$, by any boolean matrix X without null rows or columns, and obtains the relationship between the characteristic vectors as stated there, now in terms of boolean arithmetic – but of course we do not get any numerical equalities between the sizes of the partition sets. Lemma 3.7, finally, applies to boolean arithmetic, exactly as stated.

▶ **Lemma 3.9.** *In the sense of boolean arithmetic for matrices with entries in $\mathbb{B} = \{0,1\}$:*

(a) *Any symmetric $Z \in \mathbb{B}^{n,n}$ induces a unique partition of $[n]$ for which the diagonal minors induced by the partition sets are irreducible and the remaining blocks null; $d, d' \in [n]$ are in the same partition set if, and only if, in the sense of boolean arithmetic $(Z^\ell)_{dd'} = 1$ for any/all $\ell \geqslant n-1$.*

(b) *If two partitions (not necessarily of the same set) with the same number of partition sets are related by some boolean matrix $X \in \mathbb{B}^{m,n}$, then the characteristic vectors $(\mathbf{d}_i)_{i\in I}$ and $(\mathbf{d}'_i)_{i\in I}$ of the partitions are related by $\mathbf{d}_i = X\mathbf{d}'_i$ and $\mathbf{d}'_i = X^t\mathbf{d}_i$ in the sense of boolean arithmetic.*

(c) *For any matrix $X \in \mathbb{B}^{m,n}$ without null rows or columns, the symmetric boolean matrices $Z = XX^t$ and $Z' = X^tX$ have diagonal entries 1 and induce partitions that are X/X^t-related, and agree with the partitions induced by higher powers of Z and Z' or on the basis of $Z^\ell X$ and $X(Z')^\ell$ for any $\ell \in \mathbb{N}$. For $\ell \geqslant m-1, n-1$, the partition blocks in Z*

and Z' have entries 1 throughout, and $Z^\ell X$ and $X(Z')^\ell$ have entries 1 in all positions relating elements from matching partition sets.

▶ **Observation 3.10.** *For a symmetric boolean matrix $Z \in \mathbb{B}^{n,n}$ with $Z_{dd} = 1$ for all $d \in [n]$, the characteristic vectors $\mathbf{d}_i$ of the partition $[n] = \dot{\bigcup}_{i \in I} D_i$ induced by Z satisfy the following 'eigenvector' equation in terms of boolean arithmetic:*

$$Z\mathbf{d}_i = \mathbf{d}_i \quad \text{(boolean)}, \quad \text{for all } i \in I.$$

3.2 Eigenvalues and -vectors

▶ **Lemma 3.11.** *If $Z \in \mathbb{R}^{n,n}$ is doubly stochastic, then it has eigenvalue 1. If Z is doubly stochastic and irreducible with strictly positive diagonal entries, then the eigenspace for eigenvalue 1 has dimension 1 and is spanned by the vector $\mathbf{d} := (1, \dots, 1)^t$.*

Proof. It is obvious that $\mathbf{d}$ is an eigenvector of Z with eigenvalue 1. The eigenspace with eigenvalue 1 is contained in that of Z^{n-1}, which has entries strictly between 0 and 1 throughout if Z is irreducible with strictly positive diagonal, by Observation 3.1. For 1-dimensionality observe that all entries of $Z^{n-1}\mathbf{v}$ are convex combinations of the entries of $\mathbf{v}$ with coefficients strictly between 0 and 1. ◀

▶ **Corollary 3.12.** **(a)** *Let $Z \in \mathbb{R}^{n,n}$ be doubly stochastic with positive diagonal, and $[n] = \dot{\bigcup}_i D_i$ a partition with $Z_{D_i D_j} = 0$ for $i \neq j$ and such that the minors $Z_{D_i D_i}$ are irreducible for all i. Then the eigenspace for eigenvalue 1 of Z is the direct sum of the 1-dimensional subspaces spanned by the characteristic vectors $\mathbf{d}_i$ of the partition sets D_i.*

(b) *If $Z = X^t X \in \mathbb{R}^{n,n}$ for some doubly stochastic matrix X, then the eigenspace for eigenvalue 1 is the direct sum of the spans of the characteristic vectors $\mathbf{d}_i$ from the unique partition $[n] = \dot{\bigcup}_i D_i$ of $[n]$ induced by Z according to Lemma 3.2.*

3.3 Stable partitions

▶ **Definition 3.13.** Let $A \in \mathbb{R}^{n,n}$ and $[n] = \dot{\bigcup}_{i \in I} D_i$ a partition. We call this partition a *stable partition for A* if there are numbers $(s_{ij})_{i,j \in I}$ and $(t_{ij})_{i,j \in I}$ such that for all $i, j \in I$:

$$d \in D_i \quad \Rightarrow \quad \sum_{d' \in D_j} A_{dd'} = s_{ij} \quad \text{and} \quad \sum_{d' \in D_j} A_{d'd} = t_{ij}.$$

If there are s_{ij} such that $\sum_{d' \in D_j} A_{dd'} = s_{ij}$ for all $d \in D_i$, we call the partition *row-stable*; similarly, for t_{ij} such that $\sum_{d' \in D_j} A_{d'd} = t_{ij}$ for all $d \in D_i$, *column-stable.*

For symmetric A, column- and row-stability are equivalent (with $t_{ij} = s_{ij}$).

Note that the row and column sums in the definition are the D_i-components of $A\mathbf{d}_j$ and of $\mathbf{d}_j^t A = (A^t \mathbf{d}_j)^t$, respectively. So, for instance, row stability precisely says that for all i the vector $A\mathbf{d}_i$ is in the span of the vectors $\mathbf{d}_j$.

▶ **Lemma 3.14.** *Let $A \in \mathbb{R}^{n,n}$ commute with some symmetric matrix of the form $Z = XX^t \in \mathbb{R}^{n,n}$ for some doubly stochastic $X \in \mathbb{R}^{n,n}$. Then the partition $[n] = \dot{\bigcup}_i D_i$ of $[n]$ induced by Z according to Lemma 3.2 is stable for A.*

Proof. Using the characteristic vectors $\mathbf{d}_i$ of the partition sets again, we have $ZA\mathbf{d}_i = AZ\mathbf{d}_i = A\mathbf{d}_i$, and thus $A\mathbf{d}_i$ is an eigenvector of Z with eigenvalue 1. Hence by Corollary 3.12, it is in the span of the vectors $\mathbf{d}_j$, and this means that the partition is row stable. Column stability is established similarly. ◀

▶ **Corollary 3.15.** *Let A commute with $Z = XX^t$ and B commute with $Z' = X^tX$, where X is doubly stochastic (cf. Lemma 3.14). Then the partitions induced by Z and Z', which are X-related by Lemma 3.7, are stable for A and B, respectively.*

Aside: boolean arithmetic

We give a separate elementary proof of the analogue of Lemma 3.14 for boolean arithmetic. Here the definition of a *boolean* stable partition is this natural analogue of Definition 3.13.

▶ **Definition 3.16.** A partition $[n] = \dot{\bigcup}_{i \in I} D_i$ is *boolean stable* for $A \in \mathbb{B}^{n,n}$ if, in the sense of boolean arithmetic, $\sum_{d' \in D_j} A_{dd'}$ and $\sum_{d' \in D_j} A_{d'd}$ only depend on the set D_i for which $d \in D_i$.

Note that boolean stability implies that, for the characteristic vectors $\mathbf{d}_i$ of the partition, $(A\mathbf{d}_j)_d = \sum_{d' \in D_j} A_{dd'}$ is the same for all $d \in D_i$, so that also here $A\mathbf{d}_j$ is a boolean linear combination of the characteristic vectors $\mathbf{d}_i$.

▶ **Lemma 3.17.** *Let $A \in \mathbb{B}^{n,n}$ commute, in the sense of boolean arithmetic, with some symmetric matrix of the form $Z = XX^t \in \mathbb{B}^{n,n}$ with entries $Z_{dd} = 1$ for all $d \in [n]$. Then the partition $[n] = \dot{\bigcup}_i D_i$ induced by Z according to Lemma 3.9 is boolean stable for A.*

4 Fractional isomorphism

4.1 C^2-equivalence and linear equations

The *adjacency matrix* of graph $\mathcal{A}$ is the square matrix A with rows and columns indexed by vertices of $\mathcal{A}$ and entries $A_{aa'} = 1$ if aa' is an edge of $\mathcal{A}$ and $A_{aa'} = 0$ otherwise. By our assumption that graphs are undirected and simple, A is a symmetric square matrix with null diagonal. It will be convenient to assume that our graphs always have an initial segment $[n]$ of the positive integers as their vertex set. Then the adjacency matrices are in $\mathbb{B}^{n,n} \subseteq \mathbb{R}^{n,n}$. Throughout this subsection, we assume that $\mathcal{A}$ and $\mathcal{B}$ are graphs with vertex set $[n]$ and with adjacency matrices A, B, respectively. It will be notationally suggestive to denote typical indices of matrices $a, a', \ldots \in [n]$ when they are to be interpreted as vertices of $\mathcal{A}$, and $b, b', \ldots \in [n]$ when they are to be interpreted as vertices of $\mathcal{B}$.

Recall (from the discussion in the introduction) that two graphs $\mathcal{A}, \mathcal{B}$ are isomorphic if, and only if, there is a permutation matrix X such that $AX = XB$. We can rewrite this as the following integer linear program in the variables X_{ab} for $a, b \in [n]$.

$$\boxed{\begin{array}{lrll} \text{ISO} & \sum_{b' \in [n]} X_{ab'} = \sum_{a' \in [n]} X_{a'b} = 1, & & \\ & \sum_{a' \in [n]} A_{aa'} X_{a'b} = \sum_{b' \in [n]} X_{ab'} B_{b'b}, & & \\ & X_{ab} \geq 0 & & \text{for all } a, b \in [n]. \end{array}}$$

Then $\mathcal{A}$ and $\mathcal{B}$ are isomorphic if, and only if, ISO has an integer solution.

▶ **Definition 4.1.** Two graphs $\mathcal{A}, \mathcal{B}$ are *fractionally isomorphic*, $\mathcal{A} \approx \mathcal{B}$, if, and only if, the system ISO has a real solution.

So graphs are fractionally isomorphic if, and only if, there is a doubly stochastic matrix X such that $AX = XA$. Note that fractionally isomorphic graphs necessarily have the same number of vertices (this will be different for the boolean analogue, which cannot count).

A *stable partition* of the vertex set of an undirected graph is a stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ for its adjacency matrix in the sense of Definition 3.13. The characteristic parameters for a stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ for A are the numbers $s_{ij} = s^A_{ij}$ such that $s_{ij} = \sum_{d'\in D_j} A_{dd'}$ for all $d \in D_i$. (As A is symmetric, the parameters t_{ij} of Definition 3.13 are equal to the s_{ij}.) We call two stable partitions $\dot{\bigcup}_{i\in I} D_i$ for a matrix A and $\dot{\bigcup}_{i\in J} D'_i$ for a matrix B *equivalent* if $I = J$ and $|D_i| = |D'_i|$ for all $i \in I$ and $s^A_{ij} = s^B_{ij}$ and for all $i, j \in I$.

▶ **Lemma 4.2.** *$\mathcal{A}$ and $\mathcal{B}$ are C^2-equivalent if, and only if, there are equivalent stable partitions $\dot{\bigcup}_{i\in I} D_i$ for A and $\dot{\bigcup}_{i\in I} D'_i$ for B.*

Proof sketch. The partition of the elements $\mathcal{A}$ and $\mathcal{B}$ according to their C^2-type yields equivalent stable partitions (two elements have the same C^2-type if they satisfy the same C^2-formulae with one free variable). For the converse, it can be shown that equivalent stable partitions give player **II** a winning strategy in the bijective 2-pebble game. ◀

▶ **Theorem 4.3** (Ramana–Scheinerman–Ullman)**.** *Two graphs are C^2-equivalent if, and only if, they are fractionally isomorphic.*

Proof. In view of Lemma 4.2, it suffices to prove that $\mathcal{A}$ and $\mathcal{B}$ have equivalent stable partitions if, and only if, they are fractionally isomorphic.

For the forward direction, suppose that we have equivalent stable partitions $\dot{\bigcup}_{i\in I} D_i$ for A and $\dot{\bigcup}_{i\in J} D'_i$ for B. For all $a \in D_i, b \in D'_j$ we let $X_{ab} := \delta(i,j)/n_i$, where $n_i := |D_i| = |D'_i|$. (Here and elsewhere we use Kronecker's δ function defined by $\delta(i,j) = 1$ if $i = j$ and $\delta(i,j) = 0$ otherwise.) An easy calculation shows that this defines a doubly stochastic matrix X with $AX = XB$, that is, a solution for ISO.

For the converse implication, suppose that X is a doubly stochastic matrix such that $AX = XB$. Since A and B are symmetric, also $X^tA = BX^t$, which implies that A commutes with $Z := XX^t$ and B with $Z' := X^tX$.

From Lemma 3.14 and Corollary 3.15, the partitions $[n] = \dot{\bigcup}_{i\in I} D_i$ and $[n] = \dot{\bigcup}_{i\in I} D'_i$ that are induced by the symmetric matrices Z and Z' are X-related and stable for A and for B, respectively. We need to show that $|D_i| = |D'_i|$ and that the partitions also agree w.r.t. the parameters s_{ij}.

By Lemma 3.6 we have $|D_i| = |D'_i|$ and $\mathbf{d}_i = X\mathbf{d}'_i$ and $\mathbf{d}'_i = X^t\mathbf{d}_i$, where $\mathbf{d}_i$ and $\mathbf{d}'_i$ for $i \in I$ are the characteristic vectors of the two partitions. Thus for all $i, j \in I$,

$$(\mathbf{d}'_i)^t B\mathbf{d}'_j = (X^t\mathbf{d}_i)^t BX^t\mathbf{d}_j = \mathbf{d}^t_i XBX^t\mathbf{d}_j = \mathbf{d}^t_i AXX^t\mathbf{d}_j = \mathbf{d}^t_i AZ\mathbf{d}_j = \mathbf{d}^t_i A\mathbf{d}_j,$$

where the last equality follows from the fact that $\mathbf{d}_j$ is an eigenvector of Z with eigenvalue 1 by Corollary 3.12. Note that $\mathbf{d}^t_i A\mathbf{d}_j$ is the number of edges of $\mathcal{A}$ from D_i to D_j. By stability of the partition, we have $s^A_{ij} = \mathbf{d}^t_i A\mathbf{d}_j/|D_i|$ and similarly $s^B_{ij} = (\mathbf{d}'_i)^t B\mathbf{d}'_j/|D'_i|$, so that $s^A_{ij} = s^B_{ij}$. ◀

4.2 L^2-equivalence and boolean linear equations

W.r.t. an adjacency matrix $A \in \mathbb{B}^{n,n}$, a boolean stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ has as parameters just the boolean values ι^A_{ij} defined by $\iota^A_{ij} = 0$ if $A_{D_iD_j} = 0$ and $\iota^A_{ij} = 1$ otherwise. Boolean (row-)stability of the partition for A implies that $\iota^A_{ij} = 1$ if, and only if, for each individual $d \in D_i$ there is at least one $d' \in D_j$ such that $A_{dd'} = 1$.

To capture the situation of 2-pebble game equivalence, though, we now need to work with similar partitions that are stable both w.r.t. A and w.r.t. to the adjacency matrix A^c of the complement of the graph with adjacency matrix A. Here the complement of a graph

$\mathcal{A}$ is the graph $\mathcal{A}^c$ with the same vertex set as $\mathcal{A}$ obtained by replacing edges by non-edges and vice versa. Hence $A^c_{aa'} = 1$ if $A_{aa'} = 0$ and $a \neq a'$, and $A^c_{aa'} = 0$ otherwise. While a partition in the sense of real arithmetic is stable for A if, and only if, it is stable for A^c, this is no longer the case for boolean arithmetic. Let us call a partition that is boolean stable for both A and A^c, *boolean bi-stable* for A.

Then the following captures the situation of two graphs that are 2-pebble game equivalent. We note that 2-pebble equivalence is a very rough notion of equivalence, if we look at just simple undirected graphs – but the concepts explored here do have natural extensions to coloured, directed graphs, and form the basis for the analysis of k-pebble equivalence, which is non-trivial even for simple undirected graphs.

L^2-equivalence of two graphs does not imply that the graphs have the same size. In the following, we always assume that $\mathcal{A}, \mathcal{B}$ are graphs with vertex sets $[m], [n]$ respectively and that $A \in \mathbb{B}^{m,m}$ and $b \in \mathbb{B}^{n,n}$ are their adjacency matrices. We call two bi-stable partitions $[m] = \dot{\bigcup}_{i \in I} D_i$ for A (and A^c) and $[n] = \dot{\bigcup}_{i \in J} D'_i$ for B (and B^c) *b-equivalent* if $I = J$ and $\iota^A_{ij} = \iota^B_{ij}$ and $\iota^{A^c}_{ij} = \iota^{B^c}_{ij}$ and for all $i, j \in I$. Note that b-equivalence does not imply $|D_i| = |D'_i|$.

▶ **Lemma 4.4.** *$\mathcal{A}$ and $\mathcal{B}$ are L^2-equivalent if, and only if, there are b-equivalent bi-stable partitions $[m] = \dot{\bigcup}_{i \in I} D_i$ for A and $[n] = \dot{\bigcup}_{i \in J} D'_i$ for B.*

▶ **Definition 4.5.** $\mathcal{A}$ and $\mathcal{B}$ are *boolean isomorphic*, $\mathcal{A} \approx_{\text{bool}} \mathcal{B}$, if there is some boolean matrix X without null rows or columns such that $AX = XB$ and $A^cX = XB^c$.

▶ **Theorem 4.6.** *Two graphs are L^2-equivalent if, and only if, they are boolean isomorphic.*

5 Relaxations in the style of Sherali–Adams

In this section we refine the connection between the Sherali–Adams hierarchy of LP relaxation of the integer linear program ISO to equivalence in the finite variable counting logics. Throughout this section, our parameter $k \geqslant 2$ is the number of variables available in the logics C^k or L^k. As before, $\mathcal{A}$ and $\mathcal{B}$ are graphs with vertex sets $[m]$ and $[n]$, respectively, and A and B are their adjacency matrices.

The *level-*$(k-1)$ *Sherali–Adams relaxation* of the integer linear program ISO is the following linear program in the variables X_p for all $p \subseteq [m] \times [n]$ of size $|p| < k$. We write $p\widehat{\ }ab$ for the extension of p by the pair (a, b) (which need not be a proper extension).

$$\boxed{\begin{array}{lll}
\mathrm{ISO}(k-1) & \left.\begin{array}{l} X_\emptyset = 1 \quad \text{and} \\ X_p = \sum_{b'} X_{p\widehat{\ }ab'} = \sum_{a'} X_{p\widehat{\ }a'b} \\ \text{for } \ell := |p|+1 < k, a \in [m], b \in [n] \end{array}\right\} & \textsc{Cont}(\ell) \text{ for } \ell < k \\
 & \left.\begin{array}{l} \sum_{a'} A_{aa'} X_{p\widehat{\ }a'b} = \sum_{b'} X_{p\widehat{\ }ab'} B_{b'b} \\ \text{for } \ell := |p|+1 < k, a \in [m], b \in [n] \end{array}\right\} & \textsc{Comp}(\ell) \text{ for } \ell < k \\
 & X_p \geq 0 \text{ for } |p| \leq k-1 &
\end{array}}$$

We call the equations $\textsc{Comp}(\ell)$ for $1 \leq \ell < k$ *comaptibility equations* and the equations $\textsc{Cont}(\ell)$ for $0 \leq \ell < k$ *continuity equations*, where we let $\textsc{Cont}(0)$ be the equation $X_\emptyset = 1$. We will also consider these equations independently of $\mathrm{ISO}(k-1)$, as in the next lemma.

▶ **Lemma 5.1.** *If $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$, then there is a non-negative solution (X_p) for the combination of the continuity equations* $\textsc{Cont}(\ell)$ *of levels $\ell \leqslant k$ (!) with the compatibility equations* $\textsc{Comp}(\ell)$ *of levels $\ell < k$.*

Proof. For tuples $\mathbf{a} \in [m]^\ell$ and $\mathbf{b} \in [n]^\ell$ of length $\ell \le k$, we write $\mathrm{tp}(\mathbf{a}) = \mathrm{tp}(\mathbf{b})$ if $\mathcal{A}, \mathbf{a}$ and $\mathcal{B}, \mathbf{b}$ satisfy the same C^k-formulae $\varphi(\mathbf{x})$ (equality of C^k-types). We write $p = \mathbf{ab}$ to indicate that p consist of the pairs $a_i b_i$ of corresponding entries in these tuples.

To define the solution, we let $X_\emptyset := 1$. For $p = \mathbf{ab}$, we let $X_p = 0$ if $\mathrm{tp}(\mathbf{a}) \neq \mathrm{tp}(\mathbf{b})$, and we let $X_p := 1/\#_{\mathbf{b}'}(\mathrm{tp}(\mathbf{a}) = \mathrm{tp}(\mathbf{b}'))$ otherwise. Tedious but straightforward calculations show that this indeed defines a solution of the desired equations. ◀

Thus in particular, if $\mathcal{A}$ and $\mathcal{B}$ are C^k-equivalent then the system $\mathrm{ISO}(k-1)$ has a solution. Unfortunately, the converse does not hold (as we will see later). The solvability of $\mathrm{ISO}(k-1)$ only implies a weaker equivalence between $\mathcal{A}$ and $\mathcal{B}$, which we call $\mathsf{C}^{<k}$-equivalence. It is defined in terms of a game, the *weak bijective k-pebble game* on $\mathcal{A}, \mathcal{B}$. The game is played by two players. Positions of the game are sets $p \subseteq [m] \times [n]$ of size $|p| \le k-1$, and the initial position is $\emptyset$. A single round of the game, starting in position p, is played as follows.

1. If $|p| = k-1$, player **I** selects a pair $ab \in p$. If $|p| < k-1$, he omits this step.
2. Player **II** selects a bijection between $[m]$ and $[n]$. If no such bijection exists, i.e., if $m \neq n$, the game ends and player **II** loses.
3. Player **I** chooses a pair $a'b'$ from this bijection.
4. If $p^+ := p^\frown a'b'$ is a local isomorphism then the new position is

$$p' := \begin{cases} (p \backslash ab)^\frown a'b' & \text{if } |p| = k-1, \\ p^\frown a'b' & \text{if } |p| < k-1. \end{cases}$$

Otherwise, the play ends and player **II** loses.

Player **II** wins a play if it lasts forever. Structure $\mathcal{A}$ and $\mathcal{B}$ are $\mathsf{C}^{<k}$-*equivalent*, $\mathcal{A} \equiv^{<k}_{\mathsf{C}} \mathcal{B}$, if player **II** has a winning strategy for the game.

Note that the weak bijective k-pebble game requires more of the second player than the bijective $(k-1)$-pebble game, because p^+ rather than just p' is required to be a local isomorphism. On the other hand, it requires less than the bijective k-pebble game: the bijective k-pebble game precisely requires the second player to choose the bijection without prior knowledge of the pair ab that will be removed from the position. A strategy for player **II** in the weak version is good for the usual version if it is fully symmetric or uniform w.r.t. the pebble pair that is going to be removed. However, this is only relevant if $k \ge 3$. The weak bijective 2-pebble game and the bijective 2-pebble game are essentially the same.

The core of the proof of the following is analogous to that of Theorem 4.3.

▶ **Theorem 5.2.** *$\mathcal{A} \equiv^{<k}_{\mathsf{C}} \mathcal{B}$ if, and only if,* $\mathrm{ISO}(k-1)$ *has a solution.*

▶ Remark 5.3. The weak bijective k-pebble game is equivalent to a bisimulation-like game with $k-1$ pebbles where in each round the first player may slide a pebble along an edge of one of the graphs and the second player has to respond by sliding the corresponding pebble along an edge of the other graph. In this version, the game coresponds to the $(k-1)$-pebble sliding game introduced by Atserias and Maneva [1].

We will see in the next section that $\mathsf{C}^{<k}$-equivalence neither coincides with C^{k-1}-equivalence nor with C^k-equivalence. Thus it remains to give a characterisation of C^k-equivalence. By the previous theorem and the observation that C^k-equivalence is situated between $\mathsf{C}^{<k}$-equivalence and $\mathsf{C}^{<k+1}$-equivalence, we know that we need a linear program

that is “between” ISO$(k-1)$ and ISO(k). Surprisingly, we obtain such a linear program by combining the two in a very simple way: we take the continuity equations from ISO(k) and the compatibility equations from ISO$(k-1)$. Thus the resulting linear program, which we call ISO$(k-1/2)$, has variables X_p for all $p \subseteq [m] \times [n]$ of size $|p| \leq k$ and consists of the equations $\textsc{Cont}(\ell)$ for $\ell \leq k$ and the equations $\textsc{Comp}(\ell)$ for $\ell \leq k-1$, together with the non-negativity constraints $X_p \geq 0$. So Lemma 5.1 proves one implication of the theorem.

▶ **Theorem 5.4.** *$\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$ if, and only if,* ISO$(k-1/2)$ *has a solution.*

5.1 Boolean arithmetic and L^k-equivalence

We saw in Section 4.2 that equations, which are direct consequences of the basic continuity and compatibility equations w.r.t. the adjacency matrices A and B, may carry independent weight in their boolean interpretation. This is no surprise, because the boolean reading is much weaker, especially due to the absorptive nature of $\vee$, which unlike $+$ does not allow for inversion. $AX = XB$ for doubly stochastic X and $A, B \in \mathbb{B}^{n,n}$ implies $A^cX = XB^c$.

We now augment the boolean requirements by corresponding boolean equations that express

(a) compatibility also w.r.t. A^c and B^c, as in boolean fractional isomorphism,

(b) the new constraint $X_p = 0$ whenever p is not a local bijection.

In the presence of the continuity equations, which force monotonicity, it suffices for (b) to stipulate $X_{aa'bb'} = 0$ for all $a, a' \in [m]$, $b, b' \in [n]$ such that *not* $a = a' \Leftrightarrow b = b'$. This is captured by the constraint $\textsc{Match}(2)$ below. So we now use the following boolean version of the Sherali–Adams hierarchy ISO$(k-1)$ and ISO$(k-1/2)$ for $k \geq 2$.

B-ISO$(k-1)$

$$\left.\begin{array}{l} X_\emptyset = 1 \quad \text{and} \\ X_p = \sum_{b'} X_{p^\frown ab'} = \sum_{a'} X_{p^\frown a'b} \\ \text{for } |p| < k, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Cont}(\ell) \text{ for } \ell < k$$

$$\left.\begin{array}{l} X_{ab^\frown ab'} = 0 = X_{ab^\frown a'b} \\ \text{for } a \neq a' \in [m], b \neq b' \in [n] \end{array}\right\} \quad \textsc{Match}(2)$$

$$\left.\begin{array}{l} \sum_{a'} A_{aa'} X_{p^\frown a'b} = \sum_{b'} X_{p^\frown ab'} B_{b'b} \\ \text{for } |p| < k-1, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Comp}(\ell) \text{ for } \ell < k$$

$$\left.\begin{array}{l} \sum_{a'} A^c_{aa'} X_{p^\frown a'b} = \sum_{b'} X_{p^\frown ab'} B^c_{b'b} \\ \text{for } |p| < k-1, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Comp}(\ell)^c \text{ for } \ell < k$$

For B-ISO$(k-1/2)$ we require $\textsc{Cont}(\ell)$ for all $\ell \leqslant k$, i.e., additionally for $\ell = k$.

▶ Remark 5.5. B-ISO$(k-1)$ and B-ISO$(k-1/2)$ are systems of boolean equations, and the reader may wonder whether they can be solved efficiently. At first sight, it may seem NP-complete to solve such systems (just like boolean satisfiability). However, our systems consist of “linear” equations of the forms $\sum_{i\in I} X_i = \sum_{j \in J} X_j$ and $\sum_{i \in I} X_i = 0$ (which is actually a special case of the first for $J = \emptyset$) and $\sum_{i\in I} X_i = 1$. It is an easy exercise to prove that such systems of linear boolean equations can be solved in polynomial time.

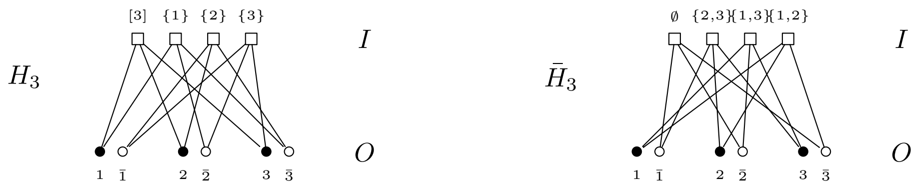

H_3
[3] {1} {2} {3}
I
O
1 $\bar{1}$ 2 $\bar{2}$ 3 $\bar{3}$
$\bar{H}_3$
∅ {2,3}{1,3}{1,2}
I
O
1 $\bar{1}$ 2 $\bar{2}$ 3 $\bar{3}$

Figure 1 The Cai–Fürer–Immerman gadgets.

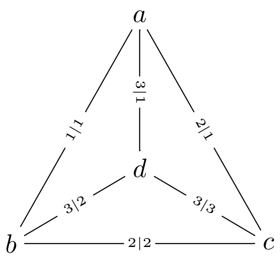


Figure 2 Structure $\mathcal{A}$.

We define a *weak k-pebble* game as a straightforward adaptation of the weak bijective k-pebble game to the setting without counting, and we denote weak k-pebble equivalence as in $\mathcal{A} \equiv_{\mathsf{L}}^{<k} \mathcal{B}$.

▶ **Theorem 5.6.** *W.r.t. boolean arithmetic:*
(a) B-ISO$(k-1)$ *has a solution if, and only if,* $\mathcal{A} \equiv_{\mathsf{L}}^{<k} \mathcal{B}$.
(b) B-ISO$(k-1/2)$ *has a solution if, and only if,* $\mathcal{A} \equiv_{\mathsf{L}}^{k} \mathcal{B}$.

6 The gap

Based on a construction due to Cai, Fürer, and Immerman [4], for $k \geqslant 3$ we construct graphs showing that $\mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B} \not\Rightarrow \mathcal{A} \equiv_{\mathsf{C}}^{k} \mathcal{B}$, and that $\mathcal{A} \equiv_{\mathsf{C}}^{k-1} \mathcal{B} \not\Rightarrow \mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B}$.

▶ **Example 6.1.** For every $k \geqslant 3$, there are graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \equiv_{\mathsf{C}}^{k-1} \mathcal{B}$ but $\mathcal{A} \not\equiv_{\mathsf{C}}^{<k} \mathcal{B}$.

We describe the graphs $\mathcal{A}$ and $\mathcal{B}$ for $k = 4$; the adaptation of the construction to other k is straightforward. The graphs are the straight and the twisted version of the Cai–Fürer–Immerman companions of the 4-clique.

We use copies of the standard degree 3 gadget H_3 and its dual $\bar{H}_3$ shown in Figure 1. We think of these as coloured graphs where the colours distinguish inner vertices (marked I) as well as outer vertices (marked O) as well as the three pairs of outer vertices. This is without loss of generality, since we may eliminate colours, e.g., by attaching simple, disjoint paths of different lengths to the members of each group of vertices. The non-trivial automorphisms of this decorated variant of H_3 and $\bar{H}_3$ precisely allow for simultaneous swaps within exactly two pairs of outer vertices.

Let $\mathcal{A}$ consist of four decorated copies of H_3, copies a, b, c, d say, that are linked by edges in corresponding outer nodes as shown in Figure 2. $\mathcal{B}$ consists of three decorated copies of H_3 (labelled a, b, c) and one of $\bar{H}_3$ (labelled d), and linked in the same manner.

It can be shown that player **I** has a winning strategy in the weak bijective 4-pebble game on $\mathcal{A}, \mathcal{B}$, whereas player **II** has a winning strategy in the bijective 3-pebble game.

▶ **Example 6.2.** For every $k \geqslant 3$, there are graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B}$ but $\mathcal{A} \not\equiv_{\mathsf{C}}^{k} \mathcal{B}$.

We describe the graphs for $k = 3$. We use variants of $\mathcal{A}$ and $\mathcal{B}$ as in the last example, but with one marked inner node: in both $\mathcal{A}$ and $\mathcal{B}$ we mark the inner node $(a, [3])$ by a new colour (which can be eliminated by attaching a path of some characteristic length, as observed above). We denote these modified structures as $\mathcal{A}_*$ and $\mathcal{B}_*$.

Then it can be shown that player **I** has a winning strategy in the bijective 3-pebble game on $\mathcal{A}_*, \mathcal{B}_*$, whereas player **II** has a winning strategy in the weak bijective 3-pebble game.

References

1 A. Atserias and E. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. In *Innovations in Theoretical Computer Science (ITCS)*, 2012.

2 D. Bienstock and N. Ozbay. Tree-width and the Sherali-Adams operator. *Discrete Optimization*, 1:13–21, 2004.

3 J. Buresh-Oppenheim, N. Galesi, S. Hoory, A. Magen, and T. Pitassi. Rank bounds and integrality gaps for cutting planes procedures. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 318–327, 2003.

4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

5 M. Charikar, K. Makarychev, and Y. Makarychev. Integrality gaps for Sherali-Adams relaxations. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, pages 283–292, 2009.

6 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 2nd edition, 1999.

7 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.

8 E. Grädel and M. Otto. Inductive definability with counting on finite structures. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M.M. Richter, editors, *Computer Science Logic, 6th Workshop, CSL '92, San Miniato 1992, Selected Papers*, volume 702 of *Lecture Notes in Computer Science*, pages 231–247. Springer-Verlag, 1993.

9 M. Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, 2010.

10 M. Grohe and M. Otto. Pebble games and linear equations, 2012. Full version of this paper. Available on arXiv, arXiv:1204.1990.

11 L. Hella. Logical hierarchies in PTIME. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 360–368, 1992.

12 N. Immerman. *Descriptive Complexity.* Springer-Verlag, 1999.

13 N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In A. Selman, editor, *Complexity theory retrospective*, pages 59–81. Springer-Verlag, 1990.

14 B. Laubner. Capturing polynomial time on interval graphs. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, pages 199–208, 2010.

15 L. Libkin. *Elements of Finite Model Theory.* Springer-Verlag, 2004.

16 C. Mathieu and A. Sinclair. Sherali-Adams relaxations of the matching polytope. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, pages 293–302, 2009.

17 M. Otto. *Bounded variable logics and counting – A study in finite models*, volume 9 of *Lecture Notes in Logic.* Springer-Verlag, 1997.

18 M. Ramana, E. Scheinerman, and D. Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 132:247–265, 1994.

19 G. Schoenebeck. Linear level Lasserre lower bounds for certain k-CSPs. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, 2008.

20 G. Schoenebeck, L. Trevisan, and M. Tulsiani. Tight integrality gaps for Lovász-Schrijver LP relaxations of vertex cover and max cut. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 302–310, 2007.

21 H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411, 1990.

Pebble Games and Linear Equations

Martin Grohe[1] and Martin Otto*[2]

1 Humboldt-Universität zu Berlin, Germany
grohe@informatik.hu-berlin.de

2 Technische Universität Darmstadt, Germany
otto@mathematik.tu-darmstadt.de

Abstract

We give a new, simplified and detailed account of the correspondence between levels of the Sherali–Adams relaxation of graph isomorphism and levels of pebble-game equivalence with counting (higher-dimensional Weisfeiler–Lehman colour refinement). The correspondence between basic colour refinement and fractional isomorphism, due to Ramana, Scheinerman and Ullman [18], is re-interpreted as the base level of Sherali–Adams and generalised to higher levels in this sense by Atserias and Maneva [1], who prove that the two resulting hierarchies interleave. In carrying this analysis further, we here give (a) a precise characterisation of the level k Sherali–Adams relaxation in terms of a modified counting pebble game; (b) a variant of the Sherali–Adams levels that precisely match the k-pebble counting game; (c) a proof that the interleaving between these two hierarchies is strict. We also investigate the variation based on boolean arithmetic instead of real/rational arithmetic and obtain analogous correspondences and separations for plain k-pebble equivalence (without counting). Our results are driven by considerably simplified accounts of the underlying combinatorics and linear algebra.

1998 ACM Subject Classification F.4.1, G.2.2

Keywords and phrases Finite model theory, finite variable logics, graph isomorphism, Weisfeiler-Lehman algorithm, linear programming, Sherali–Adams hierarchy

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.289

1 Introduction

We study a surprising connection between equivalence in finite variable logics and a linear programming approach to the graph isomorphism problem. This connection has recently been uncovered by Atserias and Maneva [1], building on earlier work of Ramana, Scheinerman and Ullman [18] that just concerns the 2-variable case.

Finite variable logics play a central role in finite model theory. Most important for this paper are finite variable logics with counting, which have been specifically studied in connection with the question for a logical characterisation of polynomial time and in connection with the graph isomorphism problem (e.g. [4, 8, 9, 13, 14, 17]). Equivalence in finite variable logics can be characterised in terms of simple combinatorial games known as pebble games. Specifically, C^k-equivalence can be characterised by the bijective k-pebble game introduced by Hella [11]. Cai, Fürer and Immerman [4] observed that C^k-equivalence exactly corresponds to indistinguishability by the k-dimensional Weisfeiler-Lehman (WL) algorithm,[1]

[*] This Martin gratefully acknowledges the other Martin's academic hospitality at HU Berlin during his sabbatical in the winter 2011/12.

[1] The dimensions of the WL algorithm are counted differently in the literature; what we call "k-dimensional" here is sometimes called "$(k-1)$-dimensional".

a combinatorial graph isomorphism algorithm introduced by Babai, who attributed it to work of Weisfeiler and Lehman in the 1970s. The 2-dimensional version of the WL algorithm precisely corresponds to an even simpler isomorphism algorithm known as colour refinement.

The isomorphisms between two graphs can be described by the integral solutions of a system of linear equations. If we have two graphs with adjacency matrices A and B, then each isomorphism from the first to the second corresponds to a permutation matrix X such that $X^tAX = B$, or equivalently

$$AX = XB. \tag{1}$$

If we view the entries of X as variables, this equation corresponds to a system of linear equations. We can add inequalities that force X to be a permutation matrix and obtain a system ISO of linear equations and inequalities whose integral solutions correspond to the isomorphisms between the two graphs. In particular, the system ISO has an integral solution if, and only if, the two graphs are isomorphic.

What happens if we drop the integrality constraints, that is, we admit arbitrary real solutions of the system ISO? We can ask for doubly stochastic matrices X satisfying equation (1). (A real matrix is *doubly stochastic* if its entries are non-negative and all row sums and column sums are one.) Ramana, Scheinerman and Ullman [18] proved a beautiful result that establishes a connection between linear algebra and logic: the system ISO has a real solution if, and only if, the colour refinement algorithm does not distinguish the two graphs with adjacency matrices A and B. Recall that the latter is equivalent to the two graphs being C^2-equivalent.

To bridge the gap between integer linear programs and their LP-relaxations, researchers in combinatorial optimisation often add additional constraints to the linear programs to bring them closer to their integer counterparts. The Sherali–Adams hierarchy [21] of relaxations gives a systematic way of doing this. For every integer linear program IL in n variables and every positive integer k, there is a *rank-k Sherali–Adams relaxation* IL(k) of IL, such that IL(1) is the standard LP-relaxation of IL where all integrality constraints are dropped and IL(n) is equivalent to IL. There is a considerable body of research studying the strength of the various levels of this and related hierarchies (e.g. [2, 3, 5, 16, 20, 19]).

Quite surprisingly, Atserias and Maneva [1] were able to lift the Ramana–Scheinerman–Ullman result, which we may now restate as an equivalence between ISO(1) and C^2-equivalence, to a close correspondence between the higher levels of the Sherali–Adams hierarchy for ISO and the logics C^k. They proved for every $k \geq 2$:

1. if ISO(k) has a (real) solution, then the two graphs are C^k-equivalent;
2. if the two graphs are C^k-equivalent, then ISO($k-1$) has a solution.

Atserias and Maneva used this results to transfer results about the logics C^k to the world of polyhedral combinatorics and combinatorial optimisation, and conversely, results about the Sherali–Adams hierarchy to logic.

Atserias and Maneva [1] left open the question whether the interleaving between the levels of the Sherali–Adams hierarchy and the finite-variable-logic hierarchy is strict or whether either the correspondence between C^k-equivalence and ISO(k) or the correspondence between C^k-equivalence and ISO($k-1$) is exact. Note that for $k = 2$ the correspondence between C^k-equivalence and ISO($k-1$) is exact by the Ramana–Scheinerman–Ullman theorem. We prove that for all $k \geq 3$ the interleaving is strict. However, we can prove an exact correspondence between ISO($k-1$) and a variant of the bijective k-pebble game that characterises C^k-equivalence. This variant, which we call the weak bijective k-pebble game, is actually equivalent to a game called ($k-1$)-sliding game by Atserias and Maneva.

Maybe most importantly, we prove that a natural combination of equalities from ISO(k) and ISO$(k-1)$ gives a linear program ISO$(k-1/2)$ that characterises C^k-equivalence exactly.

To obtain these results, we give simple new proofs of the theorems of Ramana, Scheinerman and Ullman and of Atserias and Maneva. Whereas the previous proofs use two non-trivial results from linear algebra, the Perron–Frobenius Theorem (about the eigenvalues of positive matrices) and the Birkhoff–von Neumann Theorem (stating that every doubly stochastic matrix is a convex combination of permutation matrices), our proofs only use elementary linear algebra. This makes them more transparent and less mysterious (at least to us).

In fact, the linear algebra we use is so simple that much of it can be carried out not only over the field of real numbers, but over arbitrary semirings. By using similar algebraic arguments over the boolean semiring (with disjunction as addition and conjunction as multiplication), we obtain analogous results to those for C^k-equivalence for the ordinary k-variable logic L^k, characterising L^k-equivalence, i.e., k-pebble game equivalence without counting, by systems of 'linear' equations over the boolean semiring.

For the ease of presentation, we have decided to present our results only for undirected simple graphs. It is easy to extend all results to relational structures with at most binary relations. Atserias and Maneva did this for their results, and for ours the extension works analogously. An extension to structures with relations of higher arities also seems possible, but is more complicated and comes at the price of loosing some of the elegance of the results.

Due to space limitations, we have to omit many details and proofs in this conference version of the paper. They can be found in the full version of the paper [10]. The present version of the paper contains a fairly complete account of our proof of the Ramana–Scheinerman–Ullman theorem, including the linear algebra that is also underlying they higher-dimensional results. Most proofs regarding the correspondence between the Sherali–Adams hierarchy and C^k-equivalence are omitted.

2 Finite variable logics and pebble games

We assume the reader is familiar with the basics of first-order logic FO. We almost exclusively consider first-order logic over finite graphs, which we view as finite relational structures with one binary relation. We assume graphs to be undirected and loop-free. For every positive integer k, we let L^k be the fragment of FO consisting of all formulae that contain at most k distinct variables. We let C^k be the extension of L^k by *counting quantifiers* $\exists^{\geq n}$, where$\exists^{\geq n} x\,\varphi$ means that there are at least n elements x such that φ is satisfied. L^k-*equivalence* of structures $\mathcal{A}, \mathcal{B}$ is denoted by $\mathcal{A} \equiv^k_{\mathsf{L}} \mathcal{B}$ and C^k-*equivalence* by $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$. Both equivalences can be characterised in terms of pebble games. We briefly sketch the *bijective k-pebble game* [11] that characterises C^k-equivalence. The game is played by two players on a pair $\mathcal{A}, \mathcal{B}$ of structures. A *play* of the game consists of a (possibly infinite) sequence of *rounds*. In each round, player **I** picks up one of his pebbles, and player **II** picks up her corresponding pebble. Then player **II** chooses a bijection f between $\mathcal{A}$ and $\mathcal{B}$ (if no such bijection exists, that is, if the structures have different cardinalities, player **II** immediately looses). Then player **I** places his pebble on an element a of $\mathcal{A}$, and player **II** places her pebble on $f(a)$. Note that after each round r there is a subset $p \subseteq \mathcal{A} \times \mathcal{B}$ consisting of the at most k pairs of elements on which the pairs of corresponding pebbles are placed. We call p the *position* after round r. Player **I** wins the play if every position that occurs is a local isomorphism, that is, a local mapping from $\mathcal{A}$ to $\mathcal{B}$ that is injective and preserves membership and non-membership in all relations (adjacency and non-adjacecny if $\mathcal{A}$ and $\mathcal{B}$ are graphs). Then $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$ if, and only if, player **II** has a winning strategy for the game.

C^k-equivalence also corresponds to a simple combinatorial algorithm for graph isomorphism testing known as the Weisfeiler-Lehman algorithm.

We refer the reader to the textbooks [6, 7, 12, 15] and the monograph [17] for a more thorough exposition of the material sketched here.

3 Basic combinatorics and linear algebra

We consider matrices with entries in $\mathbb{B} = \{0,1\}$, $\mathbb{Q}$ or $\mathbb{R}$. A matrix $X \in \mathbb{R}^{m,n}$ with m rows and n columns has entry X_{ij} in row $i \in [m] = \{1,\dots,m\}$ and column $j \in [n] = \{1,\dots,n\}$. We write E_n for the n-dimensional unit matrix.

We write $X \geqslant 0$ to say that (the real or rational) matrix X has only non-negative entries, and $X > 0$ to say that all entries are strictly positive. We also speak of *non-negative* or *strictly positive matrices* in this sense. For a boolean matrix, strict positivity, $X > 0$ means that all entries are 1. A square $n\times n$-matrix is *doubly stochastic* if its entries are non-negative and if the sum of entries across every row and column is 1. *Permutation matrices* are doubly stochastic matrices over $\{0,1\}$, with precisely one 1 in every row and in every column.

It will be useful to have the shorthand notation $X_{D_1D_2} = 0$ for the assertion that $X_{d_1d_2} = 0$ for all $d_1 \in D_1$, $d_2 \in D_2$.

3.1 Decomposition into irreducible blocks

With $X \in \mathbb{R}^{n,n}$ associate the directed graph $\mathrm{G}(X) := ([n], \{(i,j) \colon X_{ij} \neq 0\})$. The strongly connected components of $\mathrm{G}(X)$ induce a partition of the set $[n] = \{1,\dots,n\}$ of rows/columns of X. X is called *irreducible* if this partition has just the set $[n]$ itself.

Note that X is irreducible iff P^tXP is irreducible for every permutation matrix P.

▶ **Observation 3.1.** *Let $X \in \mathbb{R}^{n,n} \geqslant 0$ with strictly positive diagonal entries. If X is irreducible, then all powers X^ℓ for $\ell \geqslant n-1$ have non-zero entries throughout. Moreover, if X is irreducible, then so is X^ℓ for all $\ell \geqslant 1$.*

Let us call two matrices $Z, Z' \in \mathbb{R}^{n,n}$ *permutation-similar* or *S_n-similar*, $Z \sim_{S_n} Z'$, if $Z' = P^tZP$ for some permutation matrix P, i.e., if one is obtained from the other by a coherent permutation of rows and columns.

▶ **Lemma 3.2.** *Every symmetric $Z \in \mathbb{R}^{n,n} \geqslant 0$ is permutation-similar to some block diagonal matrix* $\mathrm{diag}(Z_1,\dots,Z_s)$ *with irreducible blocks $Z_i \in \mathbb{R}^{n_i,n_i}$.*

The permutation matrix P corresponding to the row- and column-permutation $p \in S_n$ that puts Z into block diagonal form $P^tZP = \mathrm{diag}(Z_1,\dots,Z_s)$ with irreducible blocks, is unique up to an outer permutation that re-arranges the block intervals $([k_i+1, k_i+n_i])_{1\leqslant i\leqslant s}$ where $k_i = \sum_{j<i} n_j$, and a product of inner permutations within each one of these s blocks.

The underlying partition $[n] = \dot{\bigcup}_{1\leqslant i\leqslant s} D_i$ where $D_i := p([k_i+1, k_i+n_i])$ for $k_i = \sum_{j<i} n_j$, is uniquely determined by Z.[2]

In the following we refer to the *partition induced by a symmetric matrix Z*.

[2] Here we regard two partitions as identical if they have the same partition sets, i.e., we ignore their indexing/enumeration.

▶ **Observation 3.3.** *In the situation of Lemma 3.2, the partition $[n] = \dot{\bigcup}_i D_i$ induced by the symmetric matrix Z is the partition of $[n]$ into the vertex sets of the connected components of $G(Z)$. Then, for every pair $i \neq j$, $Z_{D_iD_j} = 0$, while all the minors $Z_{D_iD_i}$ are irreducible.*[3]

If, moreover, Z has strictly positive diagonal entries, then the partition induced by Z is the same as that induced by Z^ℓ, for any $\ell \geqslant 1$; for $\ell \geqslant n-1$, the diagonal blocks $(Z^\ell)_{D_iD_i}$ have non-zero entries throughout: $(Z^\ell)_{D_iD_i} > 0$.

The last assertion says that for a symmetric $n \times n$ matrix Z with non-negative entries and no zeroes on the diagonal, all powers Z^ℓ for $\ell \geqslant n-1$ are *good symmetric* in the sense of the following definition.

▶ **Definition 3.4.** Let $Z \geqslant 0$ be symmetric with strictly positive diagonal. Then Z is called *good symmetric* if w.r.t. the partition $[n] = \dot{\bigcup}_i D_i$ induced by Z, all $Z_{D_iD_i} > 0$.

More generally, a not necessarily symmetric matrix $X \geqslant 0$ without null rows or columns is *good* if $Z = XX^t$ and $Z' = X^tX$ are good in the above sense.

The importance of this notion lies in the fact that, as observed above, for an arbitrary symmetric $n \times n$ matrix $Z \geqslant 0$ without zeroes on the diagonal, the partition induced by Z is the same as that induced by the good symmetric matrix $\hat{Z} := Z^{n-1}$; and, as for any good matrix, this partition can simply be read off from $\hat{Z}$: $i, j \in [n]$ are in the same partition set if, and only if, $\hat{Z}_{ij} \neq 0$.

▶ **Definition 3.5.** Consider partitions $[n] = \dot{\bigcup}_{i \in I} D_i$ and $[m] = \dot{\bigcup}_{i \in I} D'_i$ of the sets $[n]$ and $[m]$ with the same number of partition sets. We say that these two partitions are *X-related* for some matrix $X \in \mathbb{R}^{n,m}$ if

(i) $X \geqslant 0$ has no null rows or columns, and

(ii) $X_{D_iD'_j} = 0$ for every pair of distinct indices $i, j \in I$.

Note that partitions that are X-related are X^t-related in the opposite direction. More importantly, each one of the X/X^t-related partitions can be recovered from the other one through X according to

$$\begin{aligned} D'_i &= \{d' \in [m] \colon X_{dd'} > 0 \text{ for some } d \in D_i\}, \\ D_i &= \{d \in [n] \colon X_{dd'} > 0 \text{ for some } d' \in D'_i\}. \end{aligned}$$

For a more algebraic treatment, we associate with the partition sets D_i of a partition $[n] = \dot{\bigcup}_{i \in I} D_i$ the *characteristic vectors* $\mathbf{d}_i$ with entries 1 and 0 according to whether the corresponding component belongs to D_i:

$$\mathbf{d}_i = \textstyle\sum_{d \in D_i} \mathbf{e}_d,$$

where $\mathbf{e}_d$ is the d-th standard basis vector. In terms of these characteristic vectors $\mathbf{d}_i$ for $[n] = \dot{\bigcup}_{i \in I} D_i$ and $\mathbf{d}'_i$ for $[m] = \dot{\bigcup}_{i \in I} D'_i$, the X/X^t-relatedness of these partitions means that

$$\begin{aligned} D'_i &= \{d' \in [m] \colon (X^t\mathbf{d}_i)_{d'} > 0\}, \\ D_i &= \{d \in [n] \colon (X\mathbf{d}'_i)_d > 0\}. \end{aligned}$$

[3] Note that this does not depend on the enumeration of the partition set D_i, because irreducibility is invariant under permutation-similarity.

▶ **Lemma 3.6.** *If two partitions $[n] = \dot{\bigcup}_{i \in I} D_i$ and $[n] = \dot{\bigcup}_{i \in I} D'_i$ of the same set $[n]$ are X-related for some doubly stochastic matrix $X \in \mathbb{R}^{n,n}$, then $|D_i| = |D'_i|$ for all $i \in I$, and for the characteristic vectors $\mathbf{d}_i$ and $\mathbf{d}'_i$ of the partition sets D_i and D'_i*

$$\mathbf{d}_i = X\mathbf{d}'_i \quad \text{and} \quad \mathbf{d}'_i = X^t\mathbf{d}_i.$$

Proof. Observe that for all $d \in [n]$ we have $0 \leq (X\mathbf{d}'_i)_d = \sum_{d' \in D'_i} X_{dd'} \leq 1$. It follows immediately from the definition of X-relatedness that $(X\mathbf{d}'_i)_d = 0$ for all $d \notin D_i$. Therefore,

$$|D_i| \geqslant \sum_{d \in D_i} (X\mathbf{d}'_i)_d = \sum_{d \in [n]} (X\mathbf{d}'_i)_d = \sum_{d' \in D'_i} \sum_{d \in [n]} X_{dd'} = |D'_i|.$$

Similarly, $0 \leq (X^t\mathbf{d}_i)_{d'} \leq 1$ for $d' \in [n]$, and $|D'_i| \geq \sum_{d' \in D'_i} (X^t\mathbf{d}_i)_{d'} = |D_i|$. Together, we obtain

$$|D_i| = \sum_{d \in D_i} (X\mathbf{d}'_i)_d = |D'_i| = \sum_{d' \in D'_i} (X^t\mathbf{d}_i)_{d'}.$$

As all summands are bounded by 1, this implies $(X\mathbf{d}'_i)_d = 1$ for all $d \in D_i$ and $(X^t\mathbf{d}_i)_{d'} = 1$ for all $d' \in D_i$. ◀

▶ **Lemma 3.7.** *Let $X \geqslant 0$ be an $m \times n$ matrix without null rows or columns. Then the $m \times m$ matrix $Z := XX^t$ and the $n \times n$ matrix $Z' := X^tX$ are symmetric with positive entries on their diagonals. Moreover, the (unique) partitions of $[m]$ and $[n]$ that are induced by Z and Z', respectively, are X/X^t-related.*[4]

Proof. It is obvious that Z and Z' are symmetric with positive diagonal entries. Let partitions $[m] = \dot{\bigcup}_{i \in I} D_i$ and $[n] = \dot{\bigcup}_{i \in I'} D'_i$ be obtained from decompositions of Z and Z' into irreducible blocks. We need to show that the non-zero entries in X give rise to a coherent bijection between the index sets I and I' of the two partitions, in the sense that partition sets D_i and D'_j are related if, and only if, some pair of members $d \in D_i$ and $d' \in D'_j$ have a positive entry $X_{dd'}$. Then a re-numbering of one of these partitions will make them X-related in the sense of Definition 3.5. Recall from Observation 3.3 that the D_i are the vertex sets of the connected components of $G(XX^t)$ on $[m]$, while the D'_i the are the vertex sets of the connected components of $G(X^tX)$ on $[n]$.

Consider the uniformly directed bipartite graph $G(X)$ on $[m] \dot{\cup} [n]$ with an edge from $i \in [m]$ to $j \in [n]$ if $X_{ij} > 0$. In light of the symmetry of the whole situation w.r.t. X and X^t, it just remains to argue for instance that no $i \in [m]$ can have edges into two distinct sets of the partition $[n] = \dot{\bigcup}_{i \in I'} D'_i$. But any two target nodes of edges from one and the same $i \in [n]$ are in the same connected component of $G(X^tX)$, hence in the same partition set. ◀

In the situation of Lemma 3.7, powers of Z induce the same partitions as Z, and the partitions induced by $(Z^\ell X)(Z^\ell X)^t = Z^{2\ell+1}$ are X/X^t-related as well as $Z^\ell X/X^tZ^\ell$-related, for all $\ell \geqslant 1$.

For $\ell \geqslant n/2 - 1$, the matrix $Z^\ell X$ has no null rows or columns: else $Z^\ell X(Z^\ell X)^t = Z^{2\ell+1}$ would have to have a zero entry on the diagonal, contradicting the fact that this symmetric matrix is good symmetric in the sense of Definition 3.4. The same reasoning shows that $Z^\ell X$ is itself good in the sense of Definition 3.4.

[4] As X/X^t-relatedness refers to partitions presented with an indexing of the partition sets, we need to allow a suitable re-indexing for at least one of them, so as to match the other one.

▶ **Corollary 3.8.** *Let $X \geqslant 0$ be an $m \times n$ matrix without null rows or columns, $Z = XX^t$, $Z' = X^tX$ the associated symmetric matrices with non-zero entries on the diagonal. Then for $\ell \geqslant m-1$, the matrix $\hat{X} := Z^\ell X = X(Z')^\ell$ and its transpose $\hat{X}^t = X^tZ^\ell = (Z')^\ell X^t$ are good and relate the partitions $[m] = \dot{\bigcup}_i D_i$ and $[n] = \dot{\bigcup}_i D'_i$ induced by Z and Z', respectively.*[4] *Moreover,*
(i) *$\hat{X}_{D_iD'_i} > 0$ for all i, and*
(ii) *$\hat{X}_{D_iD'_j} = 0$ for all $i \neq j$.*

Aside: boolean vs. real arithmetic

Looking at matrices with $\{0,1\}$-entries, we may not only treat them as matrices over $\mathbb{R}$ as we have done so far, but also over other fields, or as matrices over the boolean semiring $\mathbb{B} = \{0,1\}$ with the logical operations of $\vee$ for addition and $\wedge$ for multiplication. Though not even forming a ring, boolean arithmetic yields a very natural interpretation in the context where we associate non-negative entries with edges, as we did in passage from X to $G(X)$. The 'normalisation map' $\chi\colon \mathbb{R}_{\geqslant 0} \to \{0,1\}$, $x \mapsto 1$ iff $x > 0$, relates the arithmetic of reals $x, y \geqslant 0$ to boolean arithmetic in

$$\chi(x+y) = \chi(x) \vee \chi(y) \quad \text{and} \quad \chi(xy) = \chi(x) \wedge \chi(y).$$

This is the 'logical' arithmetic that supports, for instance, arguments used in Observation 3.1: for any real $n \times n$ matrix $X \geqslant 0$, $(XX)_{ij} = \sum_k X_{ik}X_{kj} \neq 0$ iff there is at least one $k \in [n]$ for which $X_{ik} \neq 0$ and $X_{kj} \neq 0$ iff $\bigvee_{k\in[n]}(\chi(X_{ik}) \wedge \chi(X_{kj})) = 1$. It is no surprise, therefore, that several of the considerations apparently presented for real non-negative matrices above, have immediate analogues for boolean arithmetic – in fact, one could argue, that the boolean interpretation is closer to the combinatorial essence. We briefly sum up these analogues with a view to their use in the analysis of L^k-equivalence, while the real versions are related to C^k-equivalence. The boolean analogue of a doubly stochastic matrix with non-negative real entries is a matrix without null rows or columns.

Also note that the definitions of irreducibility and X-relatedness are applicable to boolean matrices without any changes. Observations 3.1 and 3.3 go through (as just indicated), and so does Lemma 3.2. For Lemma 3.6, one may look at X-related partitions of sets $[m]$ and $[n]$, where not necessarily $n = m$, by any boolean matrix X without null rows or columns, and obtains the relationship between the characteristic vectors as stated there, now in terms of boolean arithmetic – but of course we do not get any numerical equalities between the sizes of the partition sets. Lemma 3.7, finally, applies to boolean arithmetic, exactly as stated.

▶ **Lemma 3.9.** *In the sense of boolean arithmetic for matrices with entries in $\mathbb{B} = \{0,1\}$:*

(a) *Any symmetric $Z \in \mathbb{B}^{n,n}$ induces a unique partition of $[n]$ for which the diagonal minors induced by the partition sets are irreducible and the remaining blocks null; $d, d' \in [n]$ are in the same partition set if, and only if, in the sense of boolean arithmetic $(Z^\ell)_{dd'} = 1$ for any/all $\ell \geqslant n-1$.*

(b) *If two partitions (not necessarily of the same set) with the same number of partition sets are related by some boolean matrix $X \in \mathbb{B}^{m,n}$, then the characteristic vectors $(\mathbf{d}_i)_{i\in I}$ and $(\mathbf{d}'_i)_{i\in I}$ of the partitions are related by $\mathbf{d}_i = X\mathbf{d}'_i$ and $\mathbf{d}'_i = X^t\mathbf{d}_i$ in the sense of boolean arithmetic.*

(c) *For any matrix $X \in \mathbb{B}^{m,n}$ without null rows or columns, the symmetric boolean matrices $Z = XX^t$ and $Z' = X^tX$ have diagonal entries 1 and induce partitions that are X/X^t-related, and agree with the partitions induced by higher powers of Z and Z' or on the basis of $Z^\ell X$ and $X(Z')^\ell$ for any $\ell \in \mathbb{N}$. For $\ell \geqslant m-1, n-1$, the partition blocks in Z*

and Z' have entries 1 throughout, and $Z^\ell X$ and $X(Z')^\ell$ have entries 1 in all positions relating elements from matching partition sets.

▶ **Observation 3.10.** *For a symmetric boolean matrix $Z \in \mathbb{B}^{n,n}$ with $Z_{dd} = 1$ for all $d \in [n]$, the characteristic vectors $\mathbf{d}_i$ of the partition $[n] = \dot{\bigcup}_{i \in I} D_i$ induced by Z satisfy the following 'eigenvector' equation in terms of boolean arithmetic:*

$$Z\mathbf{d}_i = \mathbf{d}_i \quad \text{(boolean)}, \quad \text{for all } i \in I.$$

3.2 Eigenvalues and -vectors

▶ **Lemma 3.11.** *If $Z \in \mathbb{R}^{n,n}$ is doubly stochastic, then it has eigenvalue 1. If Z is doubly stochastic and irreducible with strictly positive diagonal entries, then the eigenspace for eigenvalue 1 has dimension 1 and is spanned by the vector $\mathbf{d} := (1, \dots, 1)^t$.*

Proof. It is obvious that $\mathbf{d}$ is an eigenvector of Z with eigenvalue 1. The eigenspace with eigenvalue 1 is contained in that of Z^{n-1}, which has entries strictly between 0 and 1 throughout if Z is irreducible with strictly positive diagonal, by Observation 3.1. For 1-dimensionality observe that all entries of $Z^{n-1}\mathbf{v}$ are convex combinations of the entries of $\mathbf{v}$ with coefficients strictly between 0 and 1. ◀

▶ **Corollary 3.12.** **(a)** *Let $Z \in \mathbb{R}^{n,n}$ be doubly stochastic with positive diagonal, and $[n] = \dot{\bigcup}_i D_i$ a partition with $Z_{D_i D_j} = 0$ for $i \neq j$ and such that the minors $Z_{D_i D_i}$ are irreducible for all i. Then the eigenspace for eigenvalue 1 of Z is the direct sum of the 1-dimensional subspaces spanned by the characteristic vectors $\mathbf{d}_i$ of the partition sets D_i.*

(b) *If $Z = X^t X \in \mathbb{R}^{n,n}$ for some doubly stochastic matrix X, then the eigenspace for eigenvalue 1 is the direct sum of the spans of the characteristic vectors $\mathbf{d}_i$ from the unique partition $[n] = \dot{\bigcup}_i D_i$ of $[n]$ induced by Z according to Lemma 3.2.*

3.3 Stable partitions

▶ **Definition 3.13.** Let $A \in \mathbb{R}^{n,n}$ and $[n] = \dot{\bigcup}_{i \in I} D_i$ a partition. We call this partition a *stable partition for A* if there are numbers $(s_{ij})_{i,j \in I}$ and $(t_{ij})_{i,j \in I}$ such that for all $i, j \in I$:

$$d \in D_i \quad \Rightarrow \quad \sum_{d' \in D_j} A_{dd'} = s_{ij} \quad \text{and} \quad \sum_{d' \in D_j} A_{d'd} = t_{ij}.$$

If there are s_{ij} such that $\sum_{d' \in D_j} A_{dd'} = s_{ij}$ for all $d \in D_i$, we call the partition *row-stable*; similarly, for t_{ij} such that $\sum_{d' \in D_j} A_{d'd} = t_{ij}$ for all $d \in D_i$, *column-stable.*

For symmetric A, column- and row-stability are equivalent (with $t_{ij} = s_{ij}$).

Note that the row and column sums in the definition are the D_i-components of $A\mathbf{d}_j$ and of $\mathbf{d}_j^t A = (A^t \mathbf{d}_j)^t$, respectively. So, for instance, row stability precisely says that for all i the vector $A\mathbf{d}_i$ is in the span of the vectors $\mathbf{d}_j$.

▶ **Lemma 3.14.** *Let $A \in \mathbb{R}^{n,n}$ commute with some symmetric matrix of the form $Z = XX^t \in \mathbb{R}^{n,n}$ for some doubly stochastic $X \in \mathbb{R}^{n,n}$. Then the partition $[n] = \dot{\bigcup}_i D_i$ of $[n]$ induced by Z according to Lemma 3.2 is stable for A.*

Proof. Using the characteristic vectors $\mathbf{d}_i$ of the partition sets again, we have $ZA\mathbf{d}_i = AZ\mathbf{d}_i = A\mathbf{d}_i$, and thus $A\mathbf{d}_i$ is an eigenvector of Z with eigenvalue 1. Hence by Corollary 3.12, it is in the span of the vectors $\mathbf{d}_j$, and this means that the partition is row stable. Column stability is established similarly. ◀

▶ **Corollary 3.15.** *Let A commute with $Z = XX^t$ and B commute with $Z' = X^tX$, where X is doubly stochastic (cf. Lemma 3.14). Then the partitions induced by Z and Z', which are X-related by Lemma 3.7, are stable for A and B, respectively.*

Aside: boolean arithmetic

We give a separate elementary proof of the analogue of Lemma 3.14 for boolean arithmetic. Here the definition of a *boolean* stable partition is this natural analogue of Definition 3.13.

▶ **Definition 3.16.** A partition $[n] = \dot{\bigcup}_{i \in I} D_i$ is *boolean stable* for $A \in \mathbb{B}^{n,n}$ if, in the sense of boolean arithmetic, $\sum_{d' \in D_j} A_{dd'}$ and $\sum_{d' \in D_j} A_{d'd}$ only depend on the set D_i for which $d \in D_i$.

Note that boolean stability implies that, for the characteristic vectors $\mathbf{d}_i$ of the partition, $(A\mathbf{d}_j)_d = \sum_{d' \in D_j} A_{dd'}$ is the same for all $d \in D_i$, so that also here $A\mathbf{d}_j$ is a boolean linear combination of the characteristic vectors $\mathbf{d}_i$.

▶ **Lemma 3.17.** *Let $A \in \mathbb{B}^{n,n}$ commute, in the sense of boolean arithmetic, with some symmetric matrix of the form $Z = XX^t \in \mathbb{B}^{n,n}$ with entries $Z_{dd} = 1$ for all $d \in [n]$. Then the partition $[n] = \dot{\bigcup}_i D_i$ induced by Z according to Lemma 3.9 is boolean stable for A.*

4 Fractional isomorphism

4.1 C^2-equivalence and linear equations

The *adjacency matrix* of graph $\mathcal{A}$ is the square matrix A with rows and columns indexed by vertices of $\mathcal{A}$ and entries $A_{aa'} = 1$ if aa' is an edge of $\mathcal{A}$ and $A_{aa'} = 0$ otherwise. By our assumption that graphs are undirected and simple, A is a symmetric square matrix with null diagonal. It will be convenient to assume that our graphs always have an initial segment $[n]$ of the positive integers as their vertex set. Then the adjacency matrices are in $\mathbb{B}^{n,n} \subseteq \mathbb{R}^{n,n}$. Throughout this subsection, we assume that $\mathcal{A}$ and $\mathcal{B}$ are graphs with vertex set $[n]$ and with adjacency matrices A, B, respectively. It will be notationally suggestive to denote typical indices of matrices $a, a', \ldots \in [n]$ when they are to be interpreted as vertices of $\mathcal{A}$, and $b, b', \ldots \in [n]$ when they are to be interpreted as vertices of $\mathcal{B}$.

Recall (from the discussion in the introduction) that two graphs $\mathcal{A}, \mathcal{B}$ are isomorphic if, and only if, there is a permutation matrix X such that $AX = XB$. We can rewrite this as the following integer linear program in the variables X_{ab} for $a, b \in [n]$.

$$\boxed{\begin{array}{llr} \text{ISO} & \sum_{b' \in [n]} X_{ab'} = \sum_{a' \in [n]} X_{a'b} = 1, & \\ & \sum_{a' \in [n]} A_{aa'} X_{a'b} = \sum_{b' \in [n]} X_{ab'} B_{b'b}, & \\ & X_{ab} \geq 0 & \text{for all } a, b \in [n]. \end{array}}$$

Then $\mathcal{A}$ and $\mathcal{B}$ are isomorphic if, and only if, ISO has an integer solution.

▶ **Definition 4.1.** Two graphs $\mathcal{A}, \mathcal{B}$ are *fractionally isomorphic*, $\mathcal{A} \approx \mathcal{B}$, if, and only if, the system ISO has a real solution.

So graphs are fractionally isomorphic if, and only if, there is a doubly stochastic matrix X such that $AX = XA$. Note that fractionally isomorphic graphs necessarily have the same number of vertices (this will be different for the boolean analogue, which cannot count).

A *stable partition* of the vertex set of an undirected graph is a stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ for its adjacency matrix in the sense of Definition 3.13. The characteristic parameters for a stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ for A are the numbers $s_{ij} = s^A_{ij}$ such that $s_{ij} = \sum_{d'\in D_j} A_{dd'}$ for all $d \in D_i$. (As A is symmetric, the parameters t_{ij} of Definition 3.13 are equal to the s_{ij}.) We call two stable partitions $\dot{\bigcup}_{i\in I} D_i$ for a matrix A and $\dot{\bigcup}_{i\in J} D'_i$ for a matrix B *equivalent* if $I = J$ and $|D_i| = |D'_i|$ for all $i \in I$ and $s^A_{ij} = s^B_{ij}$ and for all $i, j \in I$.

▶ **Lemma 4.2.** *$\mathcal{A}$ and $\mathcal{B}$ are C^2-equivalent if, and only if, there are equivalent stable partitions $\dot{\bigcup}_{i\in I} D_i$ for A and $\dot{\bigcup}_{i\in I} D'_i$ for B.*

Proof sketch. The partition of the elements $\mathcal{A}$ and $\mathcal{B}$ according to their C^2-type yields equivalent stable partitions (two elements have the same C^2-type if they satisfy the same C^2-formulae with one free variable). For the converse, it can be shown that equivalent stable partitions give player **II** a winning strategy in the bijective 2-pebble game. ◀

▶ **Theorem 4.3** (Ramana–Scheinerman–Ullman)**.** *Two graphs are C^2-equivalent if, and only if, they are fractionally isomorphic.*

Proof. In view of Lemma 4.2, it suffices to prove that $\mathcal{A}$ and $\mathcal{B}$ have equivalent stable partitions if, and only if, they are fractionally isomorphic.

For the forward direction, suppose that we have equivalent stable partitions $\dot{\bigcup}_{i\in I} D_i$ for A and $\dot{\bigcup}_{i\in J} D'_i$ for B. For all $a \in D_i, b \in D'_j$ we let $X_{ab} := \delta(i,j)/n_i$, where $n_i := |D_i| = |D'_i|$. (Here and elsewhere we use Kronecker's δ function defined by $\delta(i,j) = 1$ if $i = j$ and $\delta(i,j) = 0$ otherwise.) An easy calculation shows that this defines a doubly stochastic matrix X with $AX = XB$, that is, a solution for ISO.

For the converse implication, suppose that X is a doubly stochastic matrix such that $AX = XB$. Since A and B are symmetric, also $X^tA = BX^t$, which implies that A commutes with $Z := XX^t$ and B with $Z' := X^tX$.

From Lemma 3.14 and Corollary 3.15, the partitions $[n] = \dot{\bigcup}_{i\in I} D_i$ and $[n] = \dot{\bigcup}_{i\in I} D'_i$ that are induced by the symmetric matrices Z and Z' are X-related and stable for A and for B, respectively. We need to show that $|D_i| = |D'_i|$ and that the partitions also agree w.r.t. the parameters s_{ij}.

By Lemma 3.6 we have $|D_i| = |D'_i|$ and $\mathbf{d}_i = X\mathbf{d}'_i$ and $\mathbf{d}'_i = X^t\mathbf{d}_i$, where $\mathbf{d}_i$ and $\mathbf{d}'_i$ for $i \in I$ are the characteristic vectors of the two partitions. Thus for all $i, j \in I$,

$$(\mathbf{d}'_i)^tB\mathbf{d}'_j = (X^t\mathbf{d}_i)^tBX^t\mathbf{d}_j = \mathbf{d}^t_iXBX^t\mathbf{d}_j = \mathbf{d}^t_iAXX^t\mathbf{d}_j = \mathbf{d}^t_iAZ\mathbf{d}_j = \mathbf{d}^t_iA\mathbf{d}_j,$$

where the last equality follows from the fact that $\mathbf{d}_j$ is an eigenvector of Z with eigenvalue 1 by Corollary 3.12. Note that $\mathbf{d}^t_iA\mathbf{d}_j$ is the number of edges of $\mathcal{A}$ from D_i to D_j. By stability of the partition, we have $s^A_{ij} = \mathbf{d}^t_iA\mathbf{d}_j/|D_i|$ and similarly $s^B_{ij} = (\mathbf{d}'_i)^tB\mathbf{d}'_j/|D'_i|$, so that $s^A_{ij} = s^B_{ij}$. ◀

4.2 L²-equivalence and boolean linear equations

W.r.t. an adjacency matrix $A \in \mathbb{B}^{n,n}$, a boolean stable partition $[n] = \dot{\bigcup}_{i\in I} D_i$ has as parameters just the boolean values ι^A_{ij} defined by $\iota^A_{ij} = 0$ if $A_{D_iD_j} = 0$ and $\iota^A_{ij} = 1$ otherwise. Boolean (row-)stability of the partition for A implies that $\iota^A_{ij} = 1$ if, and only if, for each individual $d \in D_i$ there is at least one $d' \in D_j$ such that $A_{dd'} = 1$.

To capture the situation of 2-pebble game equivalence, though, we now need to work with similar partitions that are stable both w.r.t. A and w.r.t. to the adjacency matrix A^c of the complement of the graph with adjacency matrix A. Here the complement of a graph

$\mathcal{A}$ is the graph $\mathcal{A}^c$ with the same vertex set as $\mathcal{A}$ obtained by replacing edges by non-edges and vice versa. Hence $A^c_{aa'} = 1$ if $A_{aa'} = 0$ and $a \neq a'$, and $A^c_{aa'} = 0$ otherwise. While a partition in the sense of real arithmetic is stable for A if, and only if, it is stable for A^c, this is no longer the case for boolean arithmetic. Let us call a partition that is boolean stable for both A and A^c, *boolean bi-stable* for A.

Then the following captures the situation of two graphs that are 2-pebble game equivalent. We note that 2-pebble equivalence is a very rough notion of equivalence, if we look at just simple undirected graphs – but the concepts explored here do have natural extensions to coloured, directed graphs, and form the basis for the analysis of k-pebble equivalence, which is non-trivial even for simple undirected graphs.

L^2-equivalence of two graphs does not imply that the graphs have the same size. In the following, we always assume that $\mathcal{A}, \mathcal{B}$ are graphs with vertex sets $[m], [n]$ respectively and that $A \in \mathbb{B}^{m,m}$ and $b \in \mathbb{B}^{n,n}$ are their adjacency matrices. We call two bi-stable partitions $[m] = \dot\bigcup_{i\in I} D_i$ for A (and A^c) and $[n] = \dot\bigcup_{i\in J} D'_i$ for B (and B^c) *b-equivalent* if $I = J$ and $\iota^A_{ij} = \iota^B_{ij}$ and $\iota^{A^c}_{ij} = \iota^{B^c}_{ij}$ and for all $i, j \in I$. Note that b-equivalence does not imply $|D_i| = |D'_i|$.

▶ **Lemma 4.4.** *$\mathcal{A}$ and $\mathcal{B}$ are L^2-equivalent if, and only if, there are b-equivalent bi-stable partitions $[m] = \dot\bigcup_{i\in I} D_i$ for A and $[n] = \dot\bigcup_{i\in J} D'_i$ for B.*

▶ **Definition 4.5.** $\mathcal{A}$ and $\mathcal{B}$ are *boolean isomorphic*, $\mathcal{A} \approx_{\text{bool}} \mathcal{B}$, if there is some boolean matrix X without null rows or columns such that $AX = XB$ and $A^cX = XB^c$.

▶ **Theorem 4.6.** *Two graphs are L^2-equivalent if, and only if, they are boolean isomorphic.*

5 Relaxations in the style of Sherali–Adams

In this section we refine the connection between the Sherali–Adams hierarchy of LP relaxation of the integer linear program ISO to equivalence in the finite variable counting logics. Throughout this section, our parameter $k \geqslant 2$ is the number of variables available in the logics C^k or L^k. As before, $\mathcal{A}$ and $\mathcal{B}$ are graphs with vertex sets $[m]$ and $[n]$, respectively, and A and B are their adjacency matrices.

The *level-$(k-1)$ Sherali–Adams relaxation* of the integer linear program ISO is the following linear program in the variables X_p for all $p \subseteq [m] \times [n]$ of size $|p| < k$. We write $p^\frown ab$ for the extension of p by the pair (a, b) (which need not be a proper extension).

$$\boxed{\begin{array}{lll}
\mathrm{ISO}(k-1) & \left.\begin{array}{l} X_\emptyset = 1 \quad \text{and} \\ X_p = \sum_{b'} X_{p^\frown ab'} = \sum_{a'} X_{p^\frown a'b} \\ \text{for } \ell := |p|+1 < k, a \in [m], b \in [n] \end{array}\right\} & \textsc{Cont}(\ell) \text{ for } \ell < k \\[2ex]
 & \left.\begin{array}{l} \sum_{a'} A_{aa'} X_{p^\frown a'b} = \sum_{b'} X_{p^\frown ab'} B_{b'b} \\ \text{for } \ell := |p|+1 < k, a \in [m], b \in [n] \end{array}\right\} & \textsc{Comp}(\ell) \text{ for } \ell < k \\[2ex]
 & X_p \geq 0 \text{ for } |p| \leq k-1 &
\end{array}}$$

We call the equations $\textsc{Comp}(\ell)$ for $1 \leq \ell < k$ *comaptibility equations* and the equations $\textsc{Cont}(\ell)$ for $0 \leq \ell < k$ *continuity equations*, where we let $\textsc{Cont}(0)$ be the equation $X_\emptyset = 1$. We will also consider these equations independently of $\mathrm{ISO}(k-1)$, as in the next lemma.

▶ **Lemma 5.1.** *If $\mathcal{A} \equiv^k_{\mathsf{C}} \mathcal{B}$, then there is a non-negative solution (X_p) for the combination of the continuity equations* Cont(ℓ) *of levels $\ell \leqslant k$ (!) with the compatibility equations* Comp(ℓ) *of levels $\ell < k$.*

Proof. For tuples $\mathbf{a} \in [m]^\ell$ and $\mathbf{b} \in [n]^\ell$ of length $\ell \leq k$, we write $\mathrm{tp}(\mathbf{a}) = \mathrm{tp}(\mathbf{b})$ if $\mathcal{A}, \mathbf{a}$ and $\mathcal{B}, \mathbf{b}$ satisfy the same C^k-formulae $\varphi(\mathbf{x})$ (equality of C^k-types). We write $p = \mathbf{ab}$ to indicate that p consist of the pairs $a_i b_i$ of corresponding entries in these tuples.

To define the solution, we let $X_\emptyset := 1$. For $p = \mathbf{ab}$, we let $X_p = 0$ if $\mathrm{tp}(\mathbf{a}) \neq \mathrm{tp}(\mathbf{b})$, and we let $X_p := 1/\#_{\mathbf{b}'}(\mathrm{tp}(\mathbf{a}) = \mathrm{tp}(\mathbf{b}'))$ otherwise. Tedious but straightforward calculations show that this indeed defines a solution of the desired equations. ◀

Thus in particular, if $\mathcal{A}$ and $\mathcal{B}$ are C^k-equivalent then the system $\mathrm{ISO}(k-1)$ has a solution. Unfortunately, the converse does not hold (as we will see later). The solvability of $\mathrm{ISO}(k-1)$ only implies a weaker equivalence between $\mathcal{A}$ and $\mathcal{B}$, which we call $\mathsf{C}^{<k}$-equivalence. It is defined in terms of a game, the *weak bijective k-pebble game* on $\mathcal{A}, \mathcal{B}$. The game is played by two players. Positions of the game are sets $p \subseteq [m] \times [n]$ of size $|p| \leq k-1$, and the initial position is $\emptyset$. A single round of the game, starting in position p, is played as follows.

1. If $|p| = k-1$, player **I** selects a pair $ab \in p$. If $|p| < k-1$, he omits this step.
2. Player **II** selects a bijection between $[m]$ and $[n]$. If no such bijection exists, i.e., if $m \neq n$, the game ends and player **II** loses.
3. Player **I** chooses a pair $a'b'$ from this bijection.
4. If $p^+ := p\widehat{\ }a'b'$ is a local isomorphism then the new position is

$$p' := \begin{cases} (p \backslash ab)\widehat{\ }a'b' & \text{if } |p| = k-1, \\ p\widehat{\ }a'b' & \text{if } |p| < k-1. \end{cases}$$

Otherwise, the play ends and player **II** loses.

Player **II** wins a play if it lasts forever. Structure $\mathcal{A}$ and $\mathcal{B}$ are $\mathsf{C}^{<k}$*-equivalent*, $\mathcal{A} \equiv^{<k}_{\mathsf{C}} \mathcal{B}$, if player **II** has a winning strategy for the game.

Note that the weak bijective k-pebble game requires more of the second player than the bijective $(k-1)$-pebble game, because p^+ rather than just p' is required to be a local isomorphism. On the other hand, it requires less than the bijective k-pebble game: the bijective k-pebble game precisely requires the second player to choose the bijection without prior knowledge of the pair ab that will be removed from the position. A strategy for player **II** in the weak version is good for the usual version if it is fully symmetric or uniform w.r.t. the pebble pair that is going to be removed. However, this is only relevant if $k \geq 3$. The weak bijective 2-pebble game and the bijective 2-pebble game are essentially the same.

The core of the proof of the following is analogous to that of Theorem 4.3.

▶ **Theorem 5.2.** *$\mathcal{A} \equiv^{<k}_{\mathsf{C}} \mathcal{B}$ if, and only if,* $\mathrm{ISO}(k-1)$ *has a solution.*

▶ Remark 5.3. The weak bijective k-pebble game is equivalent to a bisimulation-like game with $k-1$ pebbles where in each round the first player may slide a pebble along an edge of one of the graphs and the second player has to respond by sliding the corresponding pebble along an edge of the other graph. In this version, the game coresponds to the $(k-1)$-pebble sliding game introduced by Atserias and Maneva [1].

We will see in the next section that $\mathsf{C}^{<k}$-equivalence neither coincides with C^{k-1}-equivalence nor with C^k-equivalence. Thus it remains to give a characterisation of C^k-equivalence. By the previous theorem and the observation that C^k-equivalence is situated between $\mathsf{C}^{<k}$-equivalence and $\mathsf{C}^{<k+1}$-equivalence, we know that we need a linear program

that is "between" ISO$(k-1)$ and ISO(k). Surprisingly, we obtain such a linear program by combining the two in a very simple way: we take the continuity equations from ISO(k) and the compatibility equations from ISO$(k-1)$. Thus the resulting linear program, which we call ISO$(k-1/2)$, has variables X_p for all $p \subseteq [m] \times [n]$ of size $|p| \leq k$ and consists of the equations $\textsc{Cont}(\ell)$ for $\ell \leq k$ and the equations $\textsc{Comp}(\ell)$ for $\ell \leq k-1$, together with the non-negativity constraints $X_p \geq 0$. So Lemma 5.1 proves one implication of the theorem.

▶ **Theorem 5.4.** *$\mathcal{A} \equiv_{\mathsf{C}}^{k} \mathcal{B}$ if, and only if,* ISO$(k-1/2)$ *has a solution.*

5.1 Boolean arithmetic and L^k-equivalence

We saw in Section 4.2 that equations, which are direct consequences of the basic continuity and compatibility equations w.r.t. the adjacency matrices A and B, may carry independent weight in their boolean interpretation. This is no surprise, because the boolean reading is much weaker, especially due to the absorptive nature of $\vee$, which unlike $+$ does not allow for inversion. $AX = XB$ for doubly stochastic X and $A, B \in \mathbb{B}^{n,n}$ implies $A^c X = XB^c$.

We now augment the boolean requirements by corresponding boolean equations that express

(a) compatibility also w.r.t. A^c and B^c, as in boolean fractional isomorphism,

(b) the new constraint $X_p = 0$ whenever p is not a local bijection.

In the presence of the continuity equations, which force monotonicity, it suffices for (b) to stipulate $X_{aa'bb'} = 0$ for all $a, a' \in [m]$, $b, b' \in [n]$ such that *not* $a = a' \Leftrightarrow b = b'$. This is captured by the constraint $\textsc{Match}(2)$ below. So we now use the following boolean version of the Sherali–Adams hierarchy ISO$(k-1)$ and ISO$(k-1/2)$ for $k \geq 2$.

B-ISO$(k-1)$

$$\left.\begin{array}{l} X_\emptyset = 1 \quad \text{and} \\ X_p = \sum_{b'} X_{p^\frown ab'} = \sum_{a'} X_{p^\frown a'b} \\ \text{for } |p| < k, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Cont}(\ell) \text{ for } \ell < k$$

$$\left.\begin{array}{l} X_{ab^\frown ab'} = 0 = X_{ab^\frown a'b} \\ \text{for } a \neq a' \in [m], b \neq b' \in [n] \end{array}\right\} \quad \textsc{Match}(2)$$

$$\left.\begin{array}{l} \sum_{a'} A_{aa'} X_{p^\frown a'b} = \sum_{b'} X_{p^\frown ab'} B_{b'b} \\ \text{for } |p| < k-1, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Comp}(\ell) \text{ for } \ell < k$$

$$\left.\begin{array}{l} \sum_{a'} A^c_{aa'} X_{p^\frown a'b} = \sum_{b'} X_{p^\frown ab'} B^c_{b'b} \\ \text{for } |p| < k-1, a \in [m], b \in [n] \end{array}\right\} \quad \textsc{Comp}(\ell)^c \text{ for } \ell < k$$

For B-ISO$(k-1/2)$ we require $\textsc{Cont}(\ell)$ for all $\ell \leqslant k$, i.e., additionally for $\ell = k$.

▶ Remark 5.5. B-ISO$(k-1)$ and B-ISO$(k-1/2)$ are systems of boolean equations, and the reader may wonder whether they can be solved efficiently. At first sight, it may seem NP-complete to solve such systems (just like boolean satisfiability). However, our systems consist of "linear" equations of the forms $\sum_{i \in I} X_i = \sum_{j \in J} X_j$ and $\sum_{i \in I} X_i = 0$ (which is actually a special case of the first for $J = \emptyset$) and $\sum_{i \in I} X_i = 1$. It is an easy exercise to prove that such systems of linear boolean equations can be solved in polynomial time.

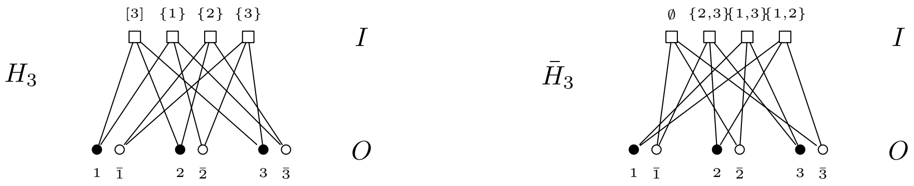

[3] {1} {2} {3}
H_3
I
O
1 $\bar{1}$ 2 $\bar{2}$ 3 $\bar{3}$
∅ {2,3}{1,3}{1,2}
$\bar{H}_3$
I
O
1 $\bar{1}$ 2 $\bar{2}$ 3 $\bar{3}$

Figure 1 The Cai–Fürer–Immerman gadgets.

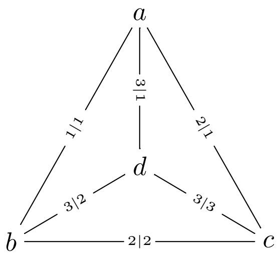


Figure 2 Structure $\mathcal{A}$.

We define a *weak k-pebble* game as a straightforward adaptation of the weak bijective k-pebble game to the setting without counting, and we denote weak k-pebble equivalence as in $\mathcal{A} \equiv_{\mathsf{L}}^{<k} \mathcal{B}$.

▶ **Theorem 5.6.** *W.r.t. boolean arithmetic:*
(a) B-ISO$(k-1)$ *has a solution if, and only if,* $\mathcal{A} \equiv_{\mathsf{L}}^{<k} \mathcal{B}$.
(b) B-ISO$(k-1/2)$ *has a solution if, and only if,* $\mathcal{A} \equiv_{\mathsf{L}}^{k} \mathcal{B}$.

6 The gap

Based on a construction due to Cai, Fürer, and Immerman [4], for $k \geqslant 3$ we construct graphs showing that $\mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B} \not\Rightarrow \mathcal{A} \equiv_{\mathsf{C}}^{k} \mathcal{B}$, and that $\mathcal{A} \equiv_{\mathsf{C}}^{k-1} \mathcal{B} \not\Rightarrow \mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B}$.

▶ **Example 6.1.** For every $k \geqslant 3$, there are graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \equiv_{\mathsf{C}}^{k-1} \mathcal{B}$ but $\mathcal{A} \not\equiv_{\mathsf{C}}^{<k} \mathcal{B}$.

We describe the graphs $\mathcal{A}$ and $\mathcal{B}$ for $k = 4$; the adaptation of the construction to other k is straightforward. The graphs are the straight and the twisted version of the Cai–Fürer–Immerman companions of the 4-clique.

We use copies of the standard degree 3 gadget H_3 and its dual $\bar{H}_3$ shown in Figure 1. We think of these as coloured graphs where the colours distinguish inner vertices (marked I) as well as outer vertices (marked O) as well as the three pairs of outer vertices. This is without loss of generality, since we may eliminate colours, e.g., by attaching simple, disjoint paths of different lengths to the members of each group of vertices. The non-trivial automorphisms of this decorated variant of H_3 and $\bar{H}_3$ precisely allow for simultaneous swaps within exactly two pairs of outer vertices.

Let $\mathcal{A}$ consist of four decorated copies of H_3, copies a, b, c, d say, that are linked by edges in corresponding outer nodes as shown in Figure 2. $\mathcal{B}$ consists of three decorated copies of H_3 (labelled a, b, c) and one of $\bar{H}_3$ (labelled d), and linked in the same manner.

It can be shown that player **I** has a winning strategy in the weak bijective 4-pebble game on $\mathcal{A}, \mathcal{B}$, whereas player **II** has a winning strategy in the bijective 3-pebble game.

▶ **Example 6.2.** For every $k \geqslant 3$, there are graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \equiv_{\mathsf{C}}^{<k} \mathcal{B}$ but $\mathcal{A} \not\equiv_{\mathsf{C}}^{k} \mathcal{B}$.

We describe the graphs for $k = 3$. We use variants of $\mathcal{A}$ and $\mathcal{B}$ as in the last example, but with one marked inner node: in both $\mathcal{A}$ and $\mathcal{B}$ we mark the inner node $(a, [3])$ by a new colour (which can be eliminated by attaching a path of some characteristic length, as observed above). We denote these modified structures as $\mathcal{A}_*$ and $\mathcal{B}_*$.

Then it can be shown that player **I** has a winning strategy in the bijective 3-pebble game on $\mathcal{A}_*, \mathcal{B}_*$, whereas player **II** has a winning strategy in the weak bijective 3-pebble game.

References

1 A. Atserias and E. Maneva. Sherali–Adams relaxations and indistinguishability in counting logics. In *Innovations in Theoretical Computer Science (ITCS)*, 2012.

2 D. Bienstock and N. Ozbay. Tree-width and the Sherali-Adams operator. *Discrete Optimization*, 1:13–21, 2004.

3 J. Buresh-Oppenheim, N. Galesi, S. Hoory, A. Magen, and T. Pitassi. Rank bounds and integrality gaps for cutting planes procedures. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 318–327, 2003.

4 J. Cai, M. Fürer, and N. Immerman. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1992.

5 M. Charikar, K. Makarychev, and Y. Makarychev. Integrality gaps for Sherali-Adams relaxations. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, pages 283–292, 2009.

6 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer-Verlag, 2nd edition, 1999.

7 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.

8 E. Grädel and M. Otto. Inductive definability with counting on finite structures. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M.M. Richter, editors, *Computer Science Logic, 6th Workshop, CSL '92, San Miniato 1992, Selected Papers*, volume 702 of *Lecture Notes in Computer Science*, pages 231–247. Springer-Verlag, 1993.

9 M. Grohe. Fixed-point definability and polynomial time on graphs with excluded minors. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, 2010.

10 M. Grohe and M. Otto. Pebble games and linear equations, 2012. Full version of this paper. Available on arXiv, arXiv:1204.1990.

11 L. Hella. Logical hierarchies in PTIME. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 360–368, 1992.

12 N. Immerman. *Descriptive Complexity.* Springer-Verlag, 1999.

13 N. Immerman and E. Lander. Describing graphs: A first-order approach to graph canonization. In A. Selman, editor, *Complexity theory retrospective*, pages 59–81. Springer-Verlag, 1990.

14 B. Laubner. Capturing polynomial time on interval graphs. In *Proceedings of the 25th IEEE Symposium on Logic in Computer Science*, pages 199–208, 2010.

15 L. Libkin. *Elements of Finite Model Theory.* Springer-Verlag, 2004.

16 C. Mathieu and A. Sinclair. Sherali-Adams relaxations of the matching polytope. In *Proceedings of the 41st ACM Symposium on Theory of Computing*, pages 293–302, 2009.

17 M. Otto. *Bounded variable logics and counting – A study in finite models*, volume 9 of *Lecture Notes in Logic.* Springer-Verlag, 1997.

18 M. Ramana, E. Scheinerman, and D. Ullman. Fractional isomorphism of graphs. *Discrete Mathematics*, 132:247–265, 1994.

19 G. Schoenebeck. Linear level Lasserre lower bounds for certain k-CSPs. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 593–602, 2008.

20 G. Schoenebeck, L. Trevisan, and M. Tulsiani. Tight integrality gaps for Lovász-Schrijver LP relaxations of vertex cover and max cut. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 302–310, 2007.

21 H.D. Sherali and W.P. Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3:411, 1990.

Herbrand-Confluence for Cut Elimination in Classical First Order Logic

Stefan Hetzl[1] and Lutz Straßburger[2]

1 **Institute of Discrete Mathematics and Geometry**
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
`hetzl@logic.at`

2 **INRIA Saclay – Île-de-France**
Ecole Polytechnique, LIX
Rue de Saclay, 91128 Palaiseau Cedex, France
`lutz@lix.polytechnique.fr`

Abstract

We consider cut-elimination in the sequent calculus for classical first-order logic. It is well known that this system, in its most general form, is neither confluent nor strongly normalizing. In this work we take a coarser (and mathematically more realistic) look at cut-free proofs. We analyze which witnesses they choose for which quantifiers, or in other words: we only consider the Herbrand-disjunction of a cut-free proof. Our main theorem is a confluence result for a natural class of proofs: all (possibly infinitely many) normal forms of the non-erasing reduction lead to the same Herbrand-disjunction.

1998 ACM Subject Classification F.4.1. Mathematical Logic, F.4.2. Grammars and Other Rewriting Systems, F.1.1. Models of Computation

Keywords and phrases proof theory, first-order logic, tree languages, term rewriting, semantics of proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.320

1 Introduction

The constructive content of proofs has always been a central topic of proof theory and it is also one of the most important influences that logic has on computer science. Classical logic is widely used and presents interesting challenges when it comes to understanding the constructive content of its proofs. These challenges have therefore attracted considerable attention, see, for example, [24, 11, 10], [6], [26, 27], [8], [21], or [5], for different investigations in this direction.

A well-known, but not yet well-understood, phenomenon is that a single classical proof usually allows several different constructive readings. From the point of view of applications this means that we have a choice among different programs that can be extracted. In [25] the authors show that two different extraction methods applied to the same proof produce two programs, one of polynomial and one of exponential average-case complexity. This phenomenon is further exemplified by case studies in [26, 3, 4] as well as the asymptotic results [2, 15]. The reason for this behavior is that classical "proofs often leave algorithmic detail underspecified" [1].

On the level of cut-elimination in the sequent calculus this phenomenon is reflected by the fact that the standard proof reduction without imposing any strategy is not confluent. In this

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 320–334
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

paper we consider cut-elimination in classical first-order logic and treat the question which cut-free proofs one can obtain (by the strategy-free rewriting system) from a single proof with cuts. As our aim is to compare cut-free proofs we need a notion of equivalence of proofs: clearly the syntactic equality makes more differences than those which are mathematically interesting. Being in a system with quantifiers, a natural and more realistic choice is to consider two cut-free proofs equivalent if they choose the same terms for the same quantifiers, in other words: if they have the same Herbrand-disjunction.

A cut-reduction relation will then be called *Herbrand-confluent* if all its normal forms have the same Herbrand-disjunction. The main result of this paper is that, for a natural class of proofs, the standard reduction without erasing of subproofs is Herbrand-confluent. This result is surprising as this reduction is neither confluent nor strongly normalizing and may produce normal forms of arbitrary size (which—as our result shows—arise only from repetitions of the same instances).

As a central proof technique we use rigid tree languages which have been introduced in [19] with applications in verification (e.g. of cryptographic protocols as in [20]) as their primary purpose. To a proof we will associate a rigid tree grammar whose language is invariant under non-erasing cut-elimination and hence equal to the only obtainable Herbrand-disjunction. This property suggests the new notion of *Herbrand-content* of a proof, which is defined as the language of the grammar of the proof, and which is a strong invariant. A side effect of this proof technique is a combinatorial description of how the structure of a cut-free proof is related to that of a proof with cut. Such descriptions are important theoretical results which underlie applications such as algorithmic cut-introduction as in [18].

In Section 2 we briefly review the sequent calculus and cut-elimination for classical first-order logic. In Section 3 we describe regular and rigid tree grammars which we relate to proofs in Section 4. Section 5 is devoted to proving the invariance of the Herbrand-content under duplication of subproofs, and finally, in Section 6, we collect all results together.

2 Sequent Calculus and Cut-Elimination

For the sake of simplicity, we consider only a one-sided sequent calculus and formulas in negation normal form, but the results can be proved for a two-sided sequent calculus in the same way.

▶ **Definition 1.** A proof is a tree of multisets of formulas. Axioms are of the form $A, \overline{A}$ for A atomic (where $\overline{A}$ denotes the De Morgan-dual of A). The inference rules are:

$$\frac{\Gamma, A[x\backslash\alpha]}{\Gamma, \forall x\, A}\,\forall \qquad \frac{\Gamma, A[x\backslash t]}{\Gamma, \exists x\, A}\,\exists \qquad \frac{\Gamma, A, A}{\Gamma, A}\,\mathsf{c} \qquad \frac{\Gamma}{\Gamma, A}\,\mathsf{w} \qquad \frac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \wedge B}\,\wedge \qquad \frac{\Gamma, A, B}{\Gamma, A \vee B}\,\vee \qquad \frac{\Gamma, A \qquad \overline{A}, \Delta}{\Gamma, \Delta}\,\mathsf{cut}$$

where α is called *eigenvariable* and does not appear in $\Gamma, \forall x\, A$ and t does not contain a bound variable. We use the notation $[x\backslash\alpha]$ for the substitution that replaces x by the eigenvariable α. Similarly, $[x\backslash t]$ is the substitution that replaces x with t.

The explicitly mentioned formula in a conclusion of an inference rule, like $A \vee B$ for $\vee$ is called *main formula*. Analogously, the explicitly mentioned formulas in the premises of an inference rule, like A and B for $\vee$, are called *auxiliary formulas*. In the context of a concrete derivation we speak about *main* and *auxiliary occurrences* of inferences.

▶ **Definition 2.** A proof is called *regular* if different $\forall$-inferences have different eigenvariables.

We use the following convention: We use lowercase Greek letters $\alpha, \beta, \gamma, \delta, \ldots$ for *eigenvariables* in proofs, and $\pi, \psi, \ldots$ for proofs. For a proof π we write $\mathrm{EV}(\pi)$ for the set of

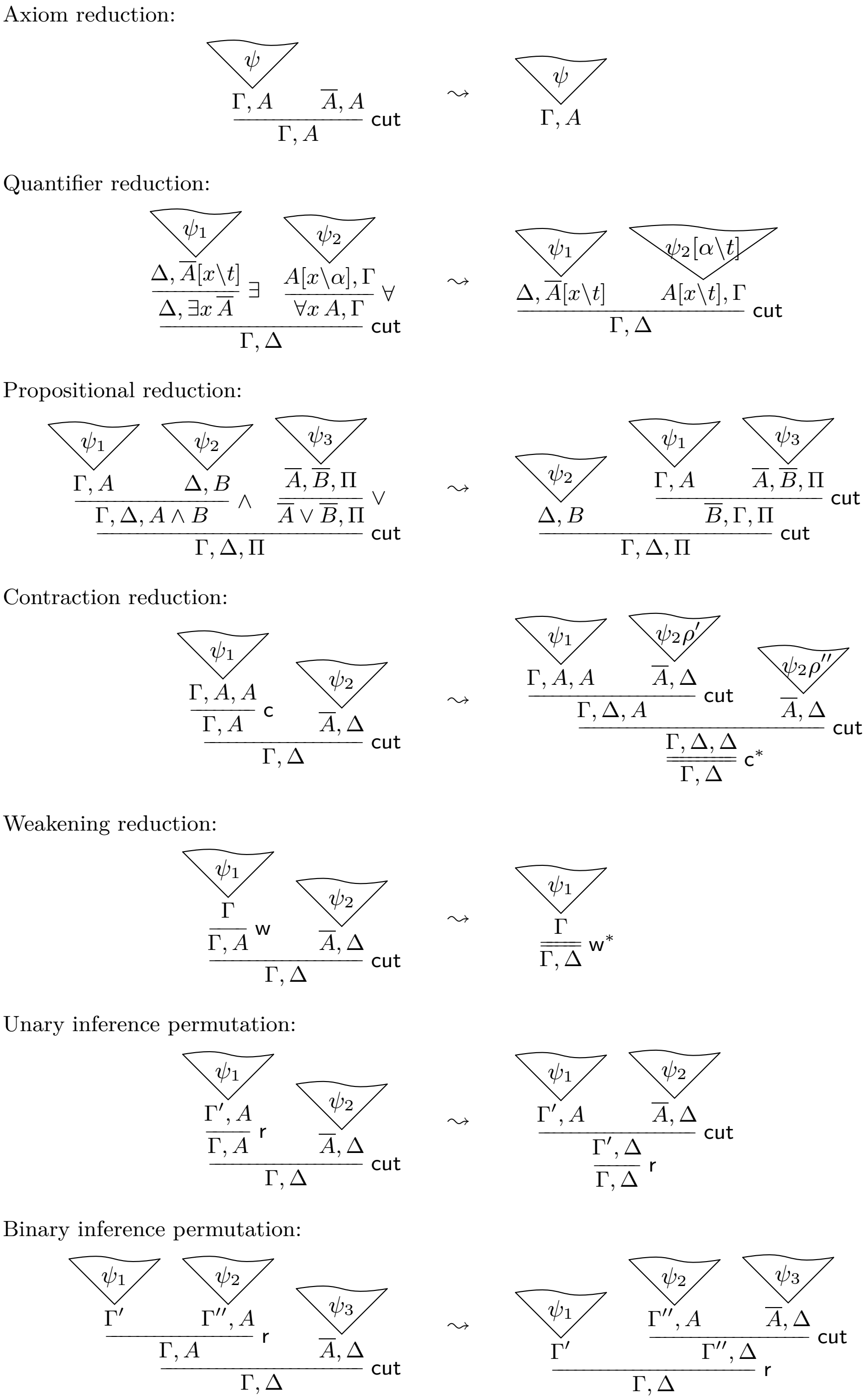


Figure 1 Cut-reduction steps.

eigenvariables of ∀-inferences of π. Furthermore, we write $|\pi|$ for the number of inferences in π. Our results do not depend on technical differences in the definition of the calculus (which in classical logic are inessential) such as the choice between multiplicative and additive rules and the differences in the cut-reduction induced by these choices. However, for the sake of precision, let us formally define the cut-reduction we use in this paper.

▶ **Definition 3.** Cut-reduction is defined on regular proofs and consists of the proof rewrite steps shown in Figure 1 (as well as all their symmetric variants), where in the contraction reduction step $\rho' = [\alpha\backslash\alpha']_{\alpha\in\mathrm{EV}(\psi_2)}$ and $\rho'' = [\alpha\backslash\alpha'']_{\alpha\in\mathrm{EV}(\psi_2)}$ are substitutions replacing each eigenvariable occurrence α in ψ_2 by fresh copies, i.e., α' and α'' are fresh for the whole proof. We write $\rightsquigarrow$ for the compatible (w.r.t. the inference rules), reflexive and transitive closure of $\rightsquigarrow$.

The above system for cut-reduction consists of purely local, minimal steps and therefore allows the simulation of many other reduction relations. We chose to work in this system in order to obtain invariance results of maximal strength. Among the systems that can be simulated literally are for example all color annotations of [11] in the multiplicative version of LK defined there. The real strength of the results in this paper lies however in the general applicability of the used proof techniques: the extraction of a grammar from a proof (that is described in the next sections) is possible in all versions of sequent calculus for classical logic and in principle also in other systems like natural deduction.

3 Regular and Rigid Tree Grammars

Formal language theory constitutes one of the main areas of theoretical computer science. Traditionally, a formal language is defined to be a set of strings but this notion can be generalized in a straightforward way to considering a language to be a set of first-order terms. Such tree languages possess a rich theory and many applications, see e.g. [13], [9]. In this section we introduce notions and results from the theory of tree languages that we will use for our proof-theoretic purposes.

A *ranked alphabet* Σ is a finite set of symbols which have an associated arity (their *rank*). We write $\mathscr{T}_\Sigma$ to denote the set of all finite trees (or terms) over Σ, and we write $\mathscr{T}_\Sigma(X)$ to denote the set of all trees over Σ and a set X of variables (seen as symbols of arity 0). We also use the notion of *position* in a tree, which is a list of natural numbers. We write ε for the empty list (the root position), and we write $p.q$ for the concatenation of lists p and q. we write $p \leq q$ if p is a prefix of q and $p < q$ if p is a proper prefix of q. Clearly, $\leq$ is a partial order and $<$ is its strict part. We write $\mathrm{Pos}(t)$ to denote the set of all position in a term $t \in \mathscr{T}_\Sigma(X)$.

▶ **Definition 4.** A *regular tree grammar* is a tuple $G = \langle N, \Sigma, \theta, P\rangle$, where N is a finite set of *non-terminal symbols*, and Σ is a ranked alphabet, such that $N \cap \Sigma = \emptyset$, θ is the *start symbol* with $\theta \in N$, and P is a finite set of production rules of the form $\beta \to t$ with $\beta \in N$ and $t \in \mathscr{T}_\Sigma(N)$.

The derivation relation $\to_G$ of a regular tree grammar $G = \langle N, \Sigma, \theta, P\rangle$ is defined as follows. We have $s \to_G r$ if there is a production rule $\beta \to t$ in P and a position $p \in \mathrm{Pos}(s)$, such that $s|_p = \beta$ and r is obtained from s by replacing β at p by t. The *language* of G is then defined as $L(G) = \{t \in \mathscr{T}_\Sigma \mid \theta \to_G^* t\}$, where $\to_G^*$ is the transitive, reflexive closure of $\to_G$. A *derivation* $\mathscr{D}$ of a term $t \in L(G)$ is a sequence $t_0 \to_G t_1 \to_G \ldots \to_G t_n$ with $t_0 = \theta$ and $t_n = t$. Note that a term t might have different derivations in G.

In [19] the class of rigid tree languages has been introduced with applications in verification (e.g. of cryptographic protocols as in [20]) as primary motivation. It will turn out that this class is appropriate for describing cut-elimination in classical first-order logic. In contrast to [19] we do not use automata but grammars—their equivalence is shown in [17].

▶ **Definition 5.** A *rigid tree grammar* is a tuple $\langle N, N_R, \Sigma, \theta, P\rangle$, where $\langle N, \Sigma, \theta, P\rangle$, is a regular tree grammar and $N_R \subseteq N$ is the set of *rigid non-terminals*. We speak of a *totally rigid tree grammar* if $N_R = N$. In this case we will just write $\langle N_R, \Sigma, \theta, P\rangle$.

A derivation $\theta = t_0 \rightarrow_G t_1 \rightarrow_G \ldots \rightarrow_G t_n = t$ of a rigid tree grammar $G = \langle N, N_R, \Sigma, \theta, P\rangle$ is a derivation in the underlying regular tree grammar satisfying the additional *rigidity condition*: If there are $i, j < n$, a non-terminal $\beta \in N_R$, and positions p and q such that $t_i|_p = \beta$ and $t_j|_q = \beta$ then $t|_p = t|_q$. The language $L(G)$ of the rigid tree grammar G is the set of all terms $t \in \mathscr{T}_\Sigma$ which can be derived under the rigidity condition. For a given derivation $\mathscr{D}\colon \theta = t_0 \rightarrow_G t_1 \rightarrow_G \ldots \rightarrow_G t_n = t$ and a non-terminal β we say that $p \in \mathrm{Pos}(t)$ is a *β-position in $\mathscr{D}$* if there is an $i \leq n$ with $t_i|_p = \beta$, i.e., either a production rule $\beta \rightarrow s$ has been applied at p in $\mathscr{D}$, or β occurs at position p in t. In the context of a given grammar G, we sometimes write $\mathscr{D}\colon \alpha \rightarrow^*_G t$ to specify that $\mathscr{D}$ is a derivation starting with α and ending with the term t.

▶ **Lemma 6.** *Let $G = \langle N, N_R, \Sigma, \theta, P\rangle$ be a rigid tree grammar and let $t \in L(G)$. Then there is a derivation $\theta \rightarrow_G \ldots \rightarrow_G t$ which uses at most one β-production for each $\beta \in N_R$.*

Proof. Given any derivation of t, suppose both $\beta \rightarrow s_1$ and $\beta \rightarrow s_2$ are used at positions p_1 and p_2 respectively. Then by the rigidity condition $t|_{p_1} = t|_{p_2}$ and we can replace the derivation at p_2 by that at p_1 (or the other way round). This transformation does not violate the rigidity condition because it only copies existing parts of the derivation. ◀

▶ **Lemma 7.** *Let $G = \langle N_R, \Sigma, \theta, P\rangle$ be a totally rigid tree grammar and $\theta \neq \beta \in N_R$, such that there is exactly one t with $\beta \rightarrow t$ in P. If $G' = \langle N_R \setminus \{\beta\}, \Sigma, \theta, (P \setminus \{\beta \rightarrow t\})[\beta\backslash t]\rangle$ then $L(G) = L(G')$.*

Proof. If a G-derivation of a term s uses β, it must replace β by t hence s is derivable using the productions of G' as well. The rigidity condition is preserved as the equality constraints of the G'-derivation are a subset of those of the G-derivation. Conversely, given a G'-derivation of a term s we obtain a derivation of s from the productions of G by replacing applications of $\delta \rightarrow r[\beta\backslash t]$ by $\delta \rightarrow r$ followed by a copy of $\beta \rightarrow t$ for each occurrence of β in r. Let $\gamma_1, \ldots, \gamma_n$ be the non-terminals that appear in t. By the rigidity condition for $i \in \{1, \ldots, n\}$ there is a unique term at all γ_i-positions in the derivation. Hence β fulfills the rigidity condition as well, and we have obtained a G-derivation of s. ◀

▶ **Lemma 8.** *If a rigid tree grammar G' is obtained from another rigid tree grammar G by deletion of production rules, then $L(G') \subseteq L(G)$.*

Proof. Every G'-derivation is a G-derivation. ◀

▶ **Notation 9.** For a given non-terminal β and a term t, we will write $\beta \in t$ or $t \ni \beta$ for denoting that β occurs in t.

▶ **Definition 10.** Let G be a tree grammar. A *path* of G is a list $\mathscr{P}$ of productions $\alpha_1 \rightarrow t_1$, $\ldots$, $\alpha_n \rightarrow t_n$ with $n \geq 1$ and $\alpha_{i+1} \in t_i$ for all $i \in \{1, \ldots, n-1\}$. The *length* of a path is $|\mathscr{P}| = n$. We will also write $\mathscr{P}\colon \alpha_1 \rightarrow t_1 \ni \alpha_2 \rightarrow \ldots \ni \alpha_n \rightarrow t_n$ to denote a path.

For a given path $\mathscr{P}\colon \alpha_1 \rightarrow t_1 \ni \alpha_2 \rightarrow \ldots \ni \alpha_n \rightarrow t_n$ we say that $\alpha_1, \ldots, \alpha_n$ are *on the path* $\mathscr{P}$ and write $\alpha_i \in \mathscr{P}$ for that. We also write $\mathscr{P}\colon \alpha_1 \dashrightarrow t_n$ and $\mathscr{P}\colon \alpha_1 \dashrightarrow \alpha_n$, if we do not want to explicitly mention the intermediate steps. For a fixed grammar G, we write $\alpha \dashrightarrow \beta$ to denote that there is a path $\mathscr{P}$ in G with $\mathscr{P}\colon \alpha \dashrightarrow \beta$.

For a set P of production rules, we write $\alpha \prec_P \beta$ (or simply $\alpha \prec \beta$, when P is clear from context) if there is a production $\alpha \rightarrow t$ in P with $\beta \in t$. We write $\prec^+$ for the transitive closure of $\prec$, and $\prec^*$ for its reflexive, transitive closure. Note that $\alpha \dashrightarrow \beta$ implies $\alpha \prec^+ \beta$, but not the other way around, since β could be a non-terminal with no production $\beta \to s$ in P.

▶ **Definition 11.** A tree grammar $\langle N, \Sigma, \theta, P\rangle$ is called *cyclic* if $\alpha \prec_P^+ \alpha$ for some $\alpha \in N$, and *acyclic* otherwise.

▶ **Lemma 12.** *If G is totally rigid and acyclic, then up to renaming of the non-terminals $G = \langle\{\alpha_1, \dots, \alpha_n\}, \Sigma, \alpha_1, P\rangle$ with $L(G) = \{\alpha_1[\alpha_1\backslash t_1]\cdots[\alpha_n\backslash t_n] \mid \alpha_i \to t_i \in P\}$.*

Proof. Acyclicity permits a renaming of non-terminals, such that $\alpha_i \prec_P^+ \alpha_j$ implies $i < j$. Then $L(G) \supseteq \{\alpha_1[\alpha_1\backslash t_1]\cdots[\alpha_n\backslash t_n] \mid \alpha_i \to t_i \in P\}$ is obvious. For the left-to-right inclusion, let $\mathscr{D}\colon \alpha_0 = s_0 \to_G \dots \to_G s_n = s \in \mathscr{T}_\Sigma$ be a derivation in G. By Lemma 6 we can assume that for each j at most one production whose left-hand side is α_j is applied, say $\alpha_j \to t_j$. By acyclicity we can rearrange the derivation so that $\alpha_j \to t_j$ is only applied after $\alpha_i \to t_i$ for all $i < j$. For those α_j which do not appear in the derivation we can insert any substitution without changing the final term so we obtain $s = \alpha_0[\alpha_0\backslash t_0]\cdots[\alpha_n\backslash t_n]$. ◀

This lemma entails that $|L(G)| \leq \prod_{i=1}^n |\{t \mid \alpha_i \to t \in P\}|$, in particular we are dealing with a finite language. The central questions in this context are (in contrast to the standard setting in formal language theory) not concerned with *representability* but with the *size of a representation*.

4 Proofs as Grammars

We will now restrict our attention to a certain class of proofs, called *simple proofs* below.

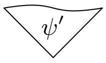
▶ **Definition 13.** A proof π is called *simple* if it is regular, the end-sequent is of the form $\exists x_1 \cdots \exists x_n\, A$ with A quantifier-free, and every cut in π whose cut-formula contains a quantifier is of the following form, where B is quantifier-free:

$$\dfrac{\Gamma, \exists x\, B \qquad \dfrac{\overline{B}[x\backslash\alpha], \Delta}{\forall x\, \overline{B}, \Delta}\,\forall}{\Gamma, \Delta}\,\mathsf{cut} \tag{1}$$

The above definition requires regularity which is a necessary assumption in the context of cut-elimination. The restriction of the end-sequent is done for expository purposes only, and can be extended to arbitrary sequents. The requirement of the $\forall$-rule being applied directly above the cut is natural as the rule is invertible. Moreover, any proof which does not fulfill this requirement can be pruned to obtain one that does, by simply permuting $\forall$-inferences down and identifying their eigenvariables when needed. The only significant restriction is that of disallowing quantifier alternations in the cut formulas. We conjecture that the central results extend to the general case. However, this will require the development of an adequate class of grammars.

▶ **Observation 14.** Simple proofs have the technically convenient property of exhibiting a 1-1 relationship between eigenvariables and cuts. For an eigenvariable α we will therefore write $\forall_\alpha$ for the inference introducing α and cut_α for the corresponding cut.

▶ **Definition 15.** Let π be a proof of $\exists x_1 \cdots \exists x_n A$ and let ψ be a subproof of π. The *Herbrand-set* $\mathrm{H}(\psi, \pi)$ *of* ψ *with respect to* π is defined as follows. If ψ is an axiom, then $\mathrm{H}(\psi, \pi) = \emptyset$. If ψ is of the form



$$\frac{\Gamma, A[x_n\backslash t]}{\Gamma, \exists x_n A}\,\exists$$

then $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi', \pi) \cup \{A[x\backslash t]\}$. If ψ ends with any other unary inference and ψ' is its immediate subproof then $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi', \pi)$. If ψ ends with a binary rule and ψ' and ψ'' are its immediate subproofs, then $\mathrm{H}(\psi, \pi) = \mathrm{H}(\psi', \pi) \cup \mathrm{H}(\psi'', \pi)$. We write $\mathrm{H}(\pi)$ for $\mathrm{H}(\pi, \pi)$.

▶ **Definition 16.** Let Q be an occurrence of a formula $\exists x\, A$ in a proof. We define the set $\mathrm{tm}(Q)$ of *terms associated with* Q as follows: if Q is introduced as the main formula of a weakening, then $\mathrm{tm}(Q) = \emptyset$. If Q is introduced by a quantifier rule $\dfrac{\Gamma, A[x\backslash t]}{\Gamma, \exists x\, A}\,\exists$ then $\mathrm{tm}(Q) = \{t\}$. If Q is the main formula in the conclusion of a contraction, and Q_1 and Q_2 are the two occurrences of the same formula in the premise that are contracted, then $\mathrm{tm}(Q) = \mathrm{tm}(Q_1) \cup \mathrm{tm}(Q_2)$. In all other cases, an inference with the occurrence Q in the conclusion has a corresponding occurrence Q' of the same formula in one of its premises, and we let $\mathrm{tm}(Q) = \mathrm{tm}(Q')$.

▶ **Definition 17.** Let π be a simple proof, let $\alpha \in \mathrm{EV}(\pi)$, and let Q be the occurrence of the existentially quantified cut-formula in the premise of cut_α. Then we write $\mathrm{B}(\alpha)$ for the set $\{\ [\alpha\backslash t] \mid t \in \mathrm{tm}(Q)\ \}$ of substitutions and we define $\mathrm{B}(\pi) = \bigcup_{\alpha \in \mathrm{EV}(\pi)} \mathrm{B}(\alpha)$.

Structures similar to the above $\mathrm{B}(\pi)$ have been investigated also in [14] and [22] where they form the basis of proof net like formalisms using local reductions for quantifiers in classical first-order logic. Our aim in this work is however quite different: we use these structures for a global analysis of the sequent calculus.

▶ **Definition 18.** The *grammar of a simple proof* π is defined to be the totally rigid grammar $\mathrm{G}(\pi) = \langle N_R, \Sigma, \theta, P\rangle$ with

$$\begin{aligned} N_R &= \mathrm{EV}(\pi) \cup \{\theta\} \\ \Sigma &= \Sigma(\pi) \cup \{\wedge, \vee\} \\ P &= \{\theta \to A \mid A \in \mathrm{H}(\pi)\} \cup \{\alpha \to t \mid [\alpha\backslash t] \in \mathrm{B}(\pi)\} \end{aligned}$$

where $\Sigma(\pi)$ is the signature of π, the rank of $\wedge$ and $\vee$ is 2, and θ does not occur in π.

▶ **Lemma 19.** *If* π *is a simple proof, then* $\mathrm{G}(\pi)$ *is acyclic.*

Proof. By induction on the number of cuts in π. The grammar of a cut-free proof is trivially acyclic. For the induction step, let r be the lowest binary inference with subproofs π_1 and π_2 s.t. either (i) r is a cut or (ii) r is not a cut but both π_1 and π_2 contain at least one cut. Let P, P_1, and P_2 be the set of productions induced by the cuts in π, π_1, π_2, respectively. In case (ii), $\prec_P = \prec_{P_1} \cup \prec_{P_2}$, which is acyclic by induction hypothesis (since $\mathrm{EV}(\pi_1) \cap \mathrm{EV}(\pi_2) = \emptyset$). In case (i), let P_r be the productions induced by the cut r, then $\prec_P = \prec_{P_1} \cup \prec_{P_2} \cup \prec_{P_\mathsf{r}}$. By induction hypothesis, $\prec_{P_1}$ and $\prec_{P_2}$ are acyclic and as the cut-formula in r contains at most one quantifier, also $\prec_{P_\mathsf{r}}$ is acyclic. Therefore, a cycle in $\prec_P^+$ must be of the form $\alpha_1 \prec_{P_1}^* \beta_1 \prec_{P_\mathsf{r}} \alpha_2 \prec_{P_2}^+ \beta_2 \prec_{P_\mathsf{r}} \alpha_1$ where $\alpha_1, \beta_1 \in \mathrm{EV}(\pi_1)$ and $\alpha_2, \beta_2 \in \mathrm{EV}(\pi_2)$. However, r contains only one quantifier and depending on its polarity all productions in P_r lead from π_1 to π_2 or from π_2 to π_1 but not both, so $\prec_P$ is acyclic. ◀

We now come to a central definition of this paper.

▶ **Definition 20.** For a simple proof π, we define its *Herbrand-content* as $[\![\pi]\!] = L(\mathrm{G}(\pi))$.

Lemma 19 together with Lemma 12 implies that the Herbrand-content of a simple proof π with n cuts can be written as

$$[\![\pi]\!] = \{A[\alpha_1\backslash t_1]\cdots[\alpha_n\backslash t_n] \mid A \in \mathrm{H}(\pi), [\alpha_i\backslash t_i] \in \mathrm{B}(\alpha_i)\}.$$

Note that for cut-free π we have $[\![\pi]\!] = \mathrm{H}(\pi)$, i.e. the Herbrand-content is nothing else but the Herbrand-disjunction induced by the proof. Furthermore, the Herbrand-content is a strong invariant: it is not changed by axiom reduction, propositional reduction and inference permutations as those transformations do not change the grammar. Furthermore, Lemma 7 shows that $[\![\pi]\!]$ is not changed by quantifier reduction and Lemma 8 shows that if $\pi \rightsquigarrow \pi'$ is a step of weakening reduction then $[\![\pi']\!] \subseteq [\![\pi]\!]$. A more difficult result is that the Herbrand-content is even invariant under the reduction of a contraction; the following section is devoted to proving this.

5 Invariance under Duplication

For simplifying the presentation, we assume in the following (without loss of generality) that the $\forall$-side is on the right of a cut and the $\exists$-side on the left. Then, a production $\beta \to t$ in $\mathrm{G}(\pi)$ corresponds to three inferences in π: a cut, an instance of the $\forall$-rule, and an instance of the $\exists$-rule, that we denote by cut_β, $\forall_\beta$, and $\exists_t$, respectively, and that are, in general, arranged in π as shown below.

$$\dfrac{\begin{array}{c}\dfrac{\Gamma', A[x\backslash t]}{\Gamma', \exists x\, A}\,\exists_t\\ \vdots\\ \Gamma, \exists x\, A\end{array} \qquad\qquad \begin{array}{c}\dfrac{\overline{A}[x\backslash\beta], \Delta'}{\forall x\,\overline{A}, \Delta'}\,\forall_\beta\\ \vdots\\ \forall x\,\overline{A}, \Delta\end{array}}{\Gamma, \Delta}\,\mathsf{cut}_\beta \tag{2}$$

The additional condition that $\forall_\beta$ is directly above cut_β, as indicated in (1) is only needed because in the following we make extensive use of Observation 14: there is a one-to-one correspondence between the cuts and the eigenvariables in π, and thus, the notation cut_β makes sense.

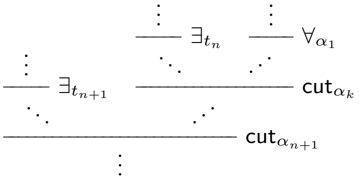
Furthermore, we say that the instances cut_β, $\forall_\beta$, and $\exists_t$ *are on a path* $\mathscr{P}$ *in* $\mathrm{G}(\pi)$ if the production $\beta \to t$ is in $\mathscr{P}$.

▶ **Definition 21.** Let π be a proof containing the configuration $\dfrac{\dfrac{\vdots}{\ddots}\,\mathsf{r}_1 \quad \dfrac{\vdots}{\iddots}\,\mathsf{r}_2}{\vdots}\,\mathsf{r}_3$, where r_1, r_2, and r_3 are arbitrary rule instances, and r_3 is a branching rule, and r_1 and r_2 might or might not be branching. Then we say that r_1 is *on the left above* r_3, denoted by $\mathsf{r}_1 \Lsh \mathsf{r}_3$, and r_2 is *on the right above* r_3, denoted by $\mathsf{r}_3 \Rsh \mathsf{r}_2$, and r_1 and r_2 are *in parallel*, denoted by $\mathsf{r}_1 \Lsh\!\Rsh \mathsf{r}_2$.

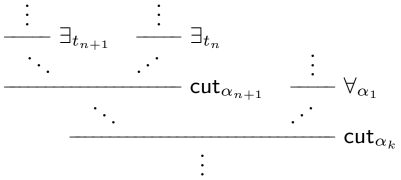
▶ **Lemma 22.** *Let π be a simple proof and $\mathscr{P}\colon \alpha_1 \to t_1 \ni \alpha_2 \ldots \to t_n$ be a path in $\mathrm{G}(\pi)$. Then there is a $k \in \{1,\ldots,n\}$ s.t. cut_{α_k} is lowermost among all inferences on $\mathscr{P}$. Furthermore, $\forall_{\alpha_1}$ is on the right above cut_{α_k} and $\exists_{t_n}$ is on the left above cut_{α_k}.*

Proof. We proceed by induction on n. If $n = 1$, then $n = k = 1$. For the induction step consider a path $\alpha_1 \to t_1 \ni \ldots \ni \alpha_n \to t_n \ni \alpha_{n+1} \to t_{n+1}$. As $\alpha_{n+1} \in t_n$ we know that $\exists_{t_n}$

must be on the right above $\mathsf{cut}_{\alpha_{n+1}}$. By induction hypothesis there is a $k \in \{1, \ldots, n\}$ such that we are in one of the following two situations



or



In the first case we let $l = n + 1$ and in the second we let $l = k$. In both cases cut_{α_l} has the desired properties. ◀

▶ **Lemma 23.** *Let π be a simple proof, $\mathrm{G}(\pi) = \langle N_R, \Sigma, \theta, P\rangle$, and $\beta, \alpha \in \mathrm{EV}(\pi)$. If $\beta \dashrightarrow \alpha$ then either $\mathsf{cut}_\alpha \Lsh \mathsf{cut}_\beta$ or $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_\beta$ or $\mathsf{cut}_\alpha \Lsh\!\Rsh \mathsf{cut}_\beta$.*

Proof. Since $\beta \dashrightarrow \alpha$, we have a path $\beta \to \ldots \ni \alpha \to t$ for some t. By Lemma 22 there is a γ, such that $\exists_t \Lsh \mathsf{cut}_\gamma$ and $\mathsf{cut}_\gamma \Rsh \forall_\beta$, and such that cut_α and cut_β are not below cut_γ. Furthermore, cut_α must be below $\exists_t$, and cut_β below $\forall_\beta$. If $\gamma = \beta$, then $\mathsf{cut}_\alpha \Lsh \mathsf{cut}_\beta$. If $\gamma = \alpha$, then $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_\beta$. And if $\gamma \neq \beta$ and $\gamma \neq \alpha$, then $\mathsf{cut}_\alpha \Lsh\!\Rsh \mathsf{cut}_\beta$. ◀

▶ **Lemma 24.** *Let $\mathrm{G}(\pi) = \langle N_R, \Sigma, \theta, P\rangle$ be the grammar of a simple proof π, such that there are two paths*

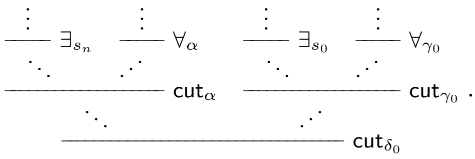
$$\beta \to t \ni \gamma_0 \to s_0 \ni \gamma_1 \to s_1 \ni \ldots \to s_{n-1} \ni \gamma_n = \alpha \to s_n$$
$$\beta \to t \ni \delta_0 \to r_0 \ni \delta_1 \to r_1 \ni \ldots \to r_{m-1} \ni \delta_m = \alpha \to r_m$$

such that γ_0 and δ_0 occur at two different positions in t. Then we have one of the following two cases:

1. *we have $\gamma_i = \delta_j$ for some $0 \leq i < n$ and $0 \leq j < m$, or*
2. *for all $0 \leq i < n$ and $0 \leq j < m$ we have $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\gamma_i}$ and $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_j}$.*

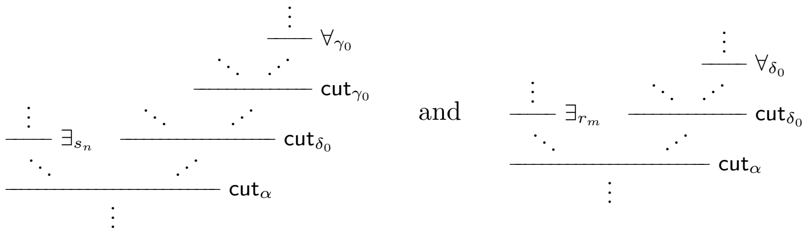
Proof. Note that because of acyclicity of $\mathrm{G}(\pi)$, we have that $\beta \neq \gamma_i$ for all $i \leq n$ and $\beta \neq \delta_j$ for all $j \leq m$, in particular $\beta \neq \alpha$. Assume, for the moment, that $m, n > 0$; the case of one of them being zero will be treated at the very end of the proof. Then $\gamma_0 \neq \alpha$ and $\delta_0 \neq \alpha$. If $\gamma_0 = \delta_0$, we have case 1. So, assume also $\gamma_0 \neq \delta_0$. As $\beta \to t$ is a production in $\mathrm{G}(\pi)$, the proof π contains a formula which contains both γ_0 and δ_0 hence $\forall_{\gamma_0}$ and $\forall_{\delta_0}$ are not parallel. Since we have $\mathsf{cut}_{\gamma_0} \Rsh \forall_{\gamma_0}$ and $\mathsf{cut}_{\delta_0} \Rsh \forall_{\delta_0}$, we also have that cut_{γ_0} and cut_{δ_0} are not parallel. Without loss of generality, assume that cut_{δ_0} is below cut_{γ_0}. Then $\mathsf{cut}_{\delta_0} \Rsh \mathsf{cut}_{\gamma_0}$ (since $\mathsf{cut}_{\gamma_0} \Lsh \mathsf{cut}_{\delta_0}$ would entail $\forall_{\gamma_0} \Lsh\!\Rsh \forall_{\delta_0}$). Since we have $\delta_0 \dashrightarrow \alpha$, we can apply Lemma 23, giving us three possibilities:

- If $\mathsf{cut}_\alpha \Lsh \mathsf{cut}_{\delta_0}$ then we have the situation



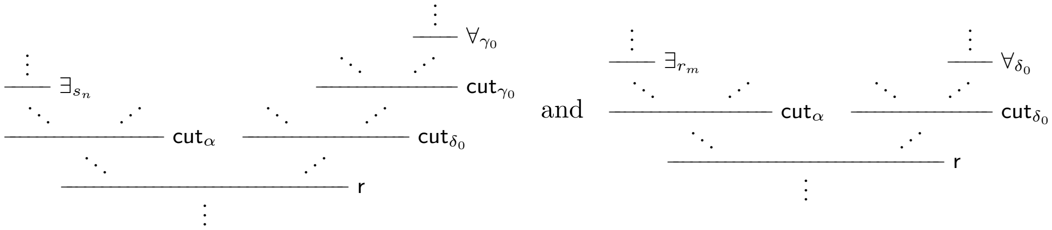
  By Lemma 22 applied to the path $\gamma_0 \dashrightarrow s_n$ we have that cut_{δ_0} must coincide with cut_{γ_i} for some $0 \leq i < n$ (since π is a tree), so $\delta_0 = \gamma_i$ (by Observation 14), and we are in case 1.

- If $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_0}$ then we are in *both* of the following two situations:



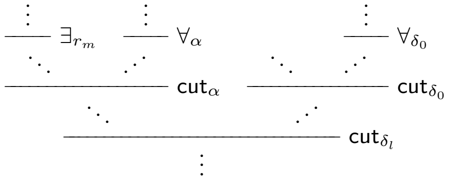
Thus, by Lemma 22 applied to the paths $\gamma_0 \dashrightarrow s_n$ and $\delta_0 \dashrightarrow r_m$ we know that $\mathsf{cut}_\alpha = \mathsf{cut}_{\gamma_k} = \mathsf{cut}_{\delta_l}$ for some $0 \leq k \leq n$ and $0 \leq l \leq m$ hence $\gamma_k = \alpha = \delta_l$. Furthermore $k = n$ and $l = m$ by acyclicity of $\mathrm{G}(\pi)$. Now consider any γ_i with $0 \leq i < n$. Since $\gamma_i \dashrightarrow \alpha$, we can apply Lemma 23 and get either $\mathsf{cut}_\alpha \Lsh \mathsf{cut}_{\gamma_i}$ or $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\gamma_i}$ or $\mathsf{cut}_\alpha \Lsh\!\Rsh \mathsf{cut}_{\gamma_i}$. Since by Lemma 22 cut_{γ_i} must be above cut_α, we conclude $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\gamma_i}$. With the same reasoning we can conclude that $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_j}$ for all $0 \leq j < m$. We are therefore in case 2.

- If $\mathsf{cut}_\alpha \Lsh\!\Rsh \mathsf{cut}_{\delta_0}$ then we are in *both* of the following two situations:



By Lemma 22 applied to the paths $\gamma_0 \to \ldots \to s_n$ and $\delta_0 \to \ldots \to r_m$, the rule r coincides with cut_{γ_i} and cut_{δ_j} for some $0 < i < n$ and $0 < j < m$, therefore $\gamma_i = \delta_j$ (by Observation 14), and we are in case 1.

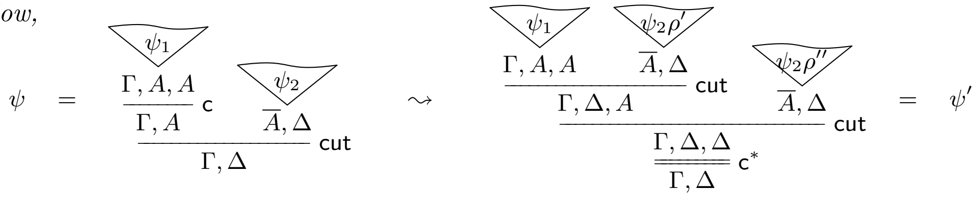
It remains to treat the case $n = 0$ or $m = 0$. If $m = n = 0$ then we are trivially in case 2 (there is no $0 \leq i < n$ or $0 \leq j < m$). If $n = 0$ and $m > 0$, we can apply Lemma 22 to the path $\delta_0 \to \ldots \to r_m$ and obtain an $l \in \{0, \ldots, m\}$ such that we are in the situation



But by the same argument as at the beginning of the proof, we also have that $\forall_\alpha$ and $\forall_{\delta_0}$ cannot be in parallel (α and δ_0 both appear in t), and therefore either $\mathsf{cut}_{\delta_0} \Rsh \mathsf{cut}_\alpha$ or $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_0}$. Since $\delta_0 \dashrightarrow \alpha$, the only possibility is $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_0}$, by Lemma 23. Thus $\mathsf{cut}_\alpha = \mathsf{cut}_{\delta_l}$, and therefore $l = m$ and we are in case 2. The case $m = 0$ and $n > 0$ is similar. ◀

The following is the main result of this section:

▶ **Proposition 25.** *Let π be a simple proof that contains a subproof ψ, shown on the left below,*



and let π' be the proof obtained from π from replacing ψ by ψ' shown on the right above, where $\rho' = [\alpha\backslash\alpha']_{\alpha\in\mathrm{EV}(\psi_2)}$ and $\rho'' = [\alpha\backslash\alpha'']_{\alpha\in\mathrm{EV}(\psi_2)}$ are substitutions that replace all eigenvariables in ψ_2 by fresh copies. Then $[\![\pi]\!] = [\![\pi']\!]$.

Proof. Let us first show $[\![\pi]\!] \subseteq [\![\pi']\!]$: write P for the productions of $\mathrm{G}(\pi)$ and P' for those of $\mathrm{G}(\pi')$. Let $F \in [\![\pi]\!] = L(\mathrm{G}(\pi))$ and $\mathscr{D}$ be its derivation. If the duplicated cut is quantifier-free, then $P' = P\rho' \cup P\rho''$ hence $\mathscr{D}\rho'$ (as well as $\mathscr{D}\rho''$) is a derivation of F in $\mathrm{G}(\pi')$. If the duplicated cut contains a quantifier, let α be its eigenvariable, let $t_1, \ldots, t_k$ be its terms coming from the left copy of A and $t_{k+1}, \ldots, t_n$ those from the right copy of A and let $Q = \{\alpha \to t_1, \ldots, \alpha \to t_n\} \subseteq P$. We then have

$$P' = (P \setminus Q)\rho' \cup \{\alpha' \to t_1, \ldots, \alpha' \to t_k\} \cup (P \setminus Q)\rho'' \cup \{\alpha'' \to t_{k+1}, \ldots, \alpha'' \to t_n\} \quad .$$

If $\mathscr{D}$ does not contain α, then $\mathscr{D}\rho'$ (as well as $\mathscr{D}\rho''$) is a derivation of F in $\mathrm{G}(\pi')$. If $\mathscr{D}$ does contain α, then by Lemma 6 we can assume that it uses only one α-production, say $\alpha \to t_i$. If $1 \leq i \leq k$, then $\mathscr{D}\rho'$ is a derivation of F in $\mathrm{G}(\pi')$ and if $k < i \leq n$, then $\mathscr{D}\rho''$ is a derivation of F in $\mathrm{G}(\pi')$.

Let us now show $[\![\pi']\!] \subseteq [\![\pi]\!]$: let F be a formula in $[\![\pi']\!] = L(\mathrm{G}(\pi'))$, and let $\mathscr{D}'$ be a derivation of F in $\mathrm{G}(\pi')$. We construct $\mathscr{D} = \mathscr{D}'(\rho')^{-1}(\rho'')^{-1}$ by “undoing” the renaming of the variables in ψ_2. Then $\mathscr{D}$ is a derivation for F, using the production rules of $\mathrm{G}(\pi)$, but possibly violating the rigidity condition.

First, observe that only non-terminals $\alpha \in \mathrm{EV}(\psi_2)$ can violate the rigidity condition in $\mathscr{D}$: if $\beta \notin \mathrm{EV}(\psi_2)$ violates the rigidity condition then there are β-positions p_1, p_2 in $\mathscr{D}$ with $F|_{p_1} \neq F|_{p_2}$ and as $\beta\rho'\rho'' = \beta$ the positions p_1, p_2 are also β-positions in $\mathscr{D}'$ and they violate the rigidity condition in $\mathscr{D}'$ which is a contradiction to $\mathscr{D}'$ being a $\mathrm{G}(\pi')$-derivation.

Now define for each $\alpha \in \mathrm{EV}(\psi_2)$ the value $\mathbf{n}(\mathscr{D}, \alpha)$ to be the number of pairs $(p_1, p_2) \in \mathrm{Pos}(F) \times \mathrm{Pos}(F)$ where p_1 and p_2 are α-positions in $\mathscr{D}$ with $p_1 \neq p_2$ and $F|_{p_1} \neq F|_{p_2}$, and define $\mathbf{n}(\mathscr{D}) = \sum_{\alpha\in\mathrm{EV}(\psi_2)} \mathbf{n}(\mathscr{D}, \alpha)$. We proceed by induction on $\mathbf{n}(\mathscr{D})$ to show that $\mathscr{D}$ can be transformed into a derivation which does no longer violate rigidity. If $\mathbf{n}(\mathscr{D}) = 0$ then $\mathscr{D}$ obeys the rigidity condition, and we are done. Otherwise there is at least one $\alpha \in \mathrm{EV}(\psi_2)$ with $\mathbf{n}(\mathscr{D}, \alpha) > 0$. We now pick one such α which is minimal with respect to $\prec^*$ (which exists since $\mathrm{G}(\pi)$ is acyclic). Let p_1 and p_2 be α-positions in $\mathscr{D}$ with $p_1 \neq p_2$ and $F|_{p_1} \neq F|_{p_2}$, let p be the maximal common prefix of p_1 and p_2 and let q be the maximal prefix of p where a production rule has been applied in $\mathscr{D}$. Due to the tree structure of F, the position q is uniquely defined, and q is a β-position for some non-terminal β, and some production rule $\beta \to t$ has been applied at position q in $\mathscr{D}$, and we have two paths:

$$\beta \to t \ni \gamma_0 \to s_0 \ni \gamma_1 \to s_1 \ni \ldots \to s_{n-1} \ni \gamma_n = \alpha \to s_n$$

$$\beta \to t \ni \delta_0 \to r_0 \ni \delta_1 \to r_1 \ni \ldots \to r_{m-1} \ni \delta_m = \alpha \to r_m$$

where γ_0 and δ_0 occur at two different positions in t. Thus, we can apply Lemma 24, giving us the following two cases:

- We have $\gamma_i = \delta_j$ for some $0 \leq i < n$ and $0 \leq j < m$. Say $\eta = \gamma_i = \delta_j$, and let p_γ and p_δ be the positions of γ_i and δ_j (respectively) in $\mathscr{D}$. Since $\eta \prec^+ \alpha$ we know that η does not violate the rigidity condition (we chose α to be minimal), and therefore $F|_{p_\gamma} = F|_{p_\delta} = F'$. Let $\mathscr{D}_\gamma \colon \gamma_i \to^*_{\mathrm{G}(\pi)} F'$ and $\mathscr{D}_\delta \colon \delta_j \to^*_{\mathrm{G}(\pi)} F'$ be the two subderivations of $\mathscr{D}$ starting in positions p_γ and p_δ, respectively. Without loss of generality, we can assume that $\mathbf{n}(\mathscr{D}_\gamma) \leq \mathbf{n}(\mathscr{D}_\delta)$. Then let $\tilde{\mathscr{D}}$ be the derivation obtained from $\mathscr{D}$ by replacing $\mathscr{D}_\delta$ by $\mathscr{D}_\gamma$. Then $\tilde{\mathscr{D}}$ is still a derivation for F, but $\mathbf{n}(\tilde{\mathscr{D}}) < \mathbf{n}(\mathscr{D})$.
- For all $0 \leq i < n$ and $0 \leq j < m$ we have $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\gamma_i}$ and $\mathsf{cut}_\alpha \Rsh \mathsf{cut}_{\delta_j}$. So all inferences of the path $\gamma_0 \to \ldots \to s_{n-1}$ as well as all inferences of $\delta_0 \to \ldots \to r_{m-1}$ are in ψ_2. Therefore all variables of of these paths are in $\mathrm{EV}(\psi_2)$. As α violates the rigidity in $\mathscr{D}$

one of p_1, p_2 must be a α'-position and the other a α''-position in $\mathscr{D}'$ because $\mathscr{D}'$ does satisfy the rigidity condition. Without loss of generality we can assume that p_1 is the α'-position and p_2 the α''-position. As the paths are contained completely in ψ_2 we have $\gamma_0 \in \mathrm{EV}(\psi_2)\rho'$ and $\delta_0 \in \mathrm{EV}(\psi_2)\rho''$ which is a contradiction as no term can contain both a variable from $\mathrm{EV}(\psi_2)\rho'$ and one from $\mathrm{EV}(\psi_2)\rho''$. ◀

6 Herbrand-Confluence

We now turn to cut reduction sequences that start with a simple proof. All the reductions shown in Figure 1 preserve simplicity, except the following:

$$\dfrac{\dfrac{\cdots \qquad \dfrac{\cdots}{\cdots}\,\forall_\alpha}{\cdots}\,\mathsf{cut}_\alpha \qquad \dfrac{\cdots}{\cdots}\,\forall_\beta}{\cdots}\,\mathsf{cut}_\beta \quad\rightsquigarrow\quad \dfrac{\cdots \qquad \dfrac{\dfrac{\cdots}{\cdots}\,\forall_\alpha \qquad \dfrac{\cdots}{\cdots}\,\forall_\beta}{\cdots}\,\mathsf{cut}_\beta}{\cdots}\,\mathsf{cut}_\alpha$$

where cut_α is permuted down under cut_β (using the bottommost reduction in Fig. 1) and the cut formula of cut_β has its ancestor on the right side of cut_α. So in the following, when we speak about a *reduction sequence of simple proofs* we require that the above reduction is immediately followed by permuting $\forall_\alpha$ down as well, in order to arrive at

$$\dfrac{\cdots \qquad \dfrac{\dfrac{\cdots \qquad \dfrac{\cdots}{\cdots}\,\forall_\beta}{\cdots}\,\mathsf{cut}_\beta}{\cdots}\,\forall_\alpha}{\cdots}\,\mathsf{cut}_\alpha$$

which is again simple. Recall that for our result this step is not strictly needed. We only add it here to simplify the presentation.

Collecting together the results proved in this paper we then obtain the following theorem.

▶ **Theorem 26.** *If $\pi \rightsquigarrow \pi'$ is a reduction sequence of simple proofs, then $[\![\pi]\!] \supseteq [\![\pi']\!]$.*

Proof. By induction on the length of the reduction $\pi \rightsquigarrow \pi'$ making a case distinction on the applied reduction step. If $\pi_i \rightsquigarrow \pi_{i+1}$ is a propositional reduction, an axiom reduction or a rule permutation, we even have $\mathrm{G}(\pi_i) = \mathrm{G}(\pi_{i+1})$. If it is a quantifier reduction, then $[\![\pi_i]\!] = [\![\pi_{i+1}]\!]$ by Lemma 7. If it is the reduction of a contraction, then $[\![\pi_i]\!] = [\![\pi_{i+1}]\!]$ by Proposition 25. If it is the reduction of a weakening, then $[\![\pi_i]\!] \supseteq [\![\pi_{i+1}]\!]$ by Lemma 8. ◀

▶ **Corollary 27.** *If $\pi \rightsquigarrow \pi'$ is a reduction sequence of simple proofs and π' is cut-free, then* $\mathrm{H}(\pi') \subseteq [\![\pi]\!]$.

This corollary shows that $[\![\pi]\!]$ is an upper bound (w.r.t. the subset relation) on the Herbrand-disjunctions obtainable by cut-elimination from π. Let us now compare this result with another upper bound that has previously been obtained in [16]. To that aim let $\mathrm{G}_0(\pi)$ denote the regular tree grammar underlying $\mathrm{G}(\pi)$ which can be obtained by setting all non-terminals to non-rigid. In this notation, a central result of [16], adapted to this paper's setting of proofs of non-prenex formulas, is

▶ **Theorem 28.** *If $\pi \rightsquigarrow \pi'$ and π' is cut-free, then* $\mathrm{H}(\pi') \subseteq L(\mathrm{G}_0(\pi))$.

While the above theorem 28 applies also to non-simple proofs, Corollary 27 is stronger in several respects:

First, the size of the Herbrand-content is by an exponential smaller than the size of the bound given by Theorem 28. Indeed, it is a straightforward consequence of Lemma 12 that the language of a totally rigid acyclic tree grammar with n production rules is bound by n^n. On the other hand, there are acyclic regular tree grammars G_n with $2n$ productions and $|L(G_n)| = n^{n^n}$ (by encoding in G_n the construction of a tree of depth n and branching degree n with an independent choice between n constant symbols at each leaf). These grammars can be obtained from appropriately constructed proofs.

Secondly, the class of totally rigid acyclic tree grammars can be shown to be in exact correspondence with the class of simple proofs in the following sense. Not only can we use a totally rigid acyclic tree grammar to simulate the process of cut-elimination, we can also—in the other direction—use cut-elimination to simulate the process of calculating the language of a grammar. It is shown in [17] how to transform an arbitrary acyclic totally rigid tree grammar G into a simple proof that has a $\rightsquigarrow$ normal form whose Herbrand-disjunction is essentially the language of G.

The third and—for the purposes of this paper—most important difference is that the bound of Corollary 27 is *tight* (in a sense that we are going to make precise now). This property of the Herbrand-content leads naturally to a confluence result for classical logic.

For tightening this bound, a first obvious observation is that there is no mechanism for deletion in the grammar but there is one in cut-elimination: the reduction of weakening. So, any cut-elimination strategy that does exactly compute $[\![\pi]\!]$ must be non-erasing. Consequently we define the *non-erasing cut-reduction* $\overset{ne}{\rightsquigarrow}$ as $\rightsquigarrow$ without the reduction rule for weakening. Note that a $\overset{ne}{\rightsquigarrow}$-normal form π is an analytic proof as well, e.g. $\mathrm{H}(\pi)$ is a (tautological!) Herbrand-disjunction. In contrast to a $\rightsquigarrow$-normal form (which might contain implicit redundancy) a $\overset{ne}{\rightsquigarrow}$-normal form might also contain explicit redundancy in the form of cuts whose cut-formulas are introduced by weakening on one or on both sides. Non-erasing reduction is also of interest in the context of the λ-calculus where it is often considered in the form of the λI-calculus and gives rise to the conservation theorem (see Theorem 13.4.12 in [7]). Our situation here is however quite different: neither $\rightsquigarrow$ nor $\overset{ne}{\rightsquigarrow}$ is confluent and neither of them is strongly normalizing. Nevertheless we obtain:

▶ **Theorem 29.** *If $\pi \overset{ne}{\rightsquigarrow} \pi'$ is a reduction sequence of simple proofs, then $[\![\pi]\!] = [\![\pi']\!]$.*

Proof. Inspection of the proof of Theorem 26 shows that the reduction of weakening is the only step that does not preserve the Herbrand-content. ◀

▶ **Definition 30** (Herbrand-confluence)**.** A relation $\longrightarrow$ on a set of proofs is called *Herbrand-confluent* iff $\pi \longrightarrow \pi_1$ and $\pi \longrightarrow \pi_2$ with π_1 and π_2 being normal forms for $\longrightarrow$ implies that $\mathrm{H}(\pi_1) = \mathrm{H}(\pi_2)$.

▶ **Corollary 31.** *The relation $\overset{ne}{\rightsquigarrow}$ is Herbrand-confluent on the set of simple proofs.*

How do these results fit together with $\overset{ne}{\rightsquigarrow}$ being neither confluent nor strongly normalizing? In fact, note that it is possible to construct a simple proof which permits an infinite $\overset{ne}{\rightsquigarrow}$ reduction sequence from which one can obtain normal forms of arbitrary size by bailing out from time to time. This can be done by building on the propositional double-contraction example found e.g. in [11, 12, 26] and in a similar form in [28]. While these infinitely many normal forms do have pairwise different Herbrand-disjunctions when regarded as *multisets*, Corollary 31 shows that as *sets* they are all the same. This observation shows that the lack of strong normalization is taken care of by using sets instead of multisets as data structure. But what about the lack of confluence? Results like [2] and [15] show that the number of $\rightsquigarrow$ normal forms with different Herbrand-disjunctions can be enormous. On the other hand we have just seen that $\overset{ne}{\rightsquigarrow}$ induces only *a single* Herbrand-disjunction: $[\![\pi]\!]$. The relation between $[\![\pi]\!]$ and the many Herbrand-disjunctions induced by $\rightsquigarrow$ is explained by Corollary 27: $[\![\pi]\!]$ contains them all as subsets.

7 Conclusion

We have shown that non-erasing cut-elimination for the class of simple proofs is Herbrand-confluent. While there are different and possibly infinitely many normal forms, they all induce the same Herbrand-disjunction. This result motivates the definition of this unique Herbrand-disjunction as Herbrand-*content* of the proof with cut.

As future work, the authors plan to extend this result to arbitrary first-order proofs. The treatment of blocks of quantifiers is straightforward: the rigidity condition must be changed to apply to vectors of non-terminals. Treating quantifier alternations is more difficult: the current results suggest to use a *stack* of totally rigid tree grammars, each layer of which corresponds to one layer of quantifiers (and is hence acyclic). Concerning further generalizations, note that the method of describing a cut-free proof by a tree language is applicable to any proof system with quantifiers that has a Herbrand-like theorem, e.g., even full higher-order logic as in [23]. The difficulty consists in finding an appropriate type of grammars.

Given the wealth of different methods for the extraction of constructive content from classical proofs, what we learn from our work is this: the first-order structure possesses (in contrast to the propositional structure) a unique and canonical unfolding. The various extraction methods hence do not differ in the choice of how to unfold the first-order structure but only in choosing *which part* of it to unfold. We therefore see that the effect of the underspecification of algorithmic detail in classical logic is redundancy.

Acknowledgments

The authors would like to thank Paul-André Melliès for helpful comments on this work. The first author was supported by a Marie Curie Intra European Fellowship within the 7th European Community Framework Programme and by the projects I-603 N18 and P22028 of the Austrian Science Fund (FWF).

References

1 Jeremy Avigad. The computational content of classical arithmetic. In Solomon Feferman and Wilfried Sieg, editors, *Proofs, Categories, and Computations: Essays in Honor of Grigori Mints*, pages 15–30. College Publications, 2010.

2 Matthias Baaz and Stefan Hetzl. On the non-confluence of cut-elimination. *Journal of Symbolic Logic*, 76(1):313–340, 2011.

3 Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Cut-Elimination: Experiments with CERES. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2005.

4 Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. CERES: An Analysis of Fürstenberg's Proof of the Infinity of Primes. *Theoretical Computer Science*, 403(2–3):160–175, 2008.

5 Matthias Baaz and Alexander Leitsch. Cut-elimination and Redundancy-elimination by Resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

6 Franco Barbanera and Stefano Berardi. A Symmetric Lambda Calculus for Classical Program Extraction. *Information and Computation*, 125(2):103–117, 1996.

7 Hendrik Pieter Barendregt. *The Lambda Calculus*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1984.

8 Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined Program Extraction from Classical Proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

9 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.

10 Pierre-Louis Curien and Hugo Herbelin. The Duality of Computation. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00)*, pages 233–243. ACM, 2000.

11 Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A New Deconstructive Logic: Linear Logic. *Journal of Symbolic Logic*, 62(3):755–807, 1997.

12 Jean Gallier. Constructive Logics. Part I: A Tutorial on Proof Systems and Typed λ-Calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

13 Ferenc Gécseg and Magnus Steinby. Tree Languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages: Volume 3: Beyond Words*, pages 1–68. Springer, 1997.

14 Willem Heijltjes. Classical proof forestry. *Annals of Pure and Applied Logic*, 161(11):1346–1366, 2010.

15 Stefan Hetzl. The Computational Content of Arithmetical Proofs. to appear in the *Notre Dame Journal of Formal Logic*.

16 Stefan Hetzl. On the form of witness terms. *Archive for Mathematical Logic*, 49(5):529–554, 2010.

17 Stefan Hetzl. Applying Tree Languages in Proof Theory. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *Language and Automata Theory and Applications (LATA) 2012*, volume 7183 of *Lecture Notes in Computer Science*. Springer, 2012.

18 Stefan Hetzl, Alexander Leitsch, and Daniel Weller. Towards Algorithmic Cut-Introduction. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR-18)*, volume 7180 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2012.

19 Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata. In Adrian Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *Third International Conference on Language and Automata Theory and Applications (LATA) 2009*, volume 5457 of *Lecture Notes in Computer Science*, pages 446–457. Springer, 2009.

20 Florent Jacquemard, Francis Klay, and Camille Vacher. Rigid tree automata and applications. *Information and Computation*, 209:486–512, 2011.

21 Ulrich Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer, 2008.

22 Richard McKinley. Herbrand expansion proofs and proof identity. In *Classical Logic and Computation (CL&C) 2008, participant's proceedings*, 2008. available at `http://wwwhomes.doc.ic.ac.uk/~svb/CLaC08/programme.html`.

23 Dale Miller. A Compact Representation of Proofs. *Studia Logica*, 46(4):347–370, 1987.

24 Michel Parigot. $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning (LPAR) 1992*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 1992.

25 Diana Ratiu and Trifon Trifonov. Exploring the Computational Content of the Infinite Pigeonhole Principle. *Journal of Logic and Computation*, 22(2):329–350, 2012.

26 Christian Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, October 2000.

27 Christian Urban and Gavin Bierman. Strong Normalization of Cut-Elimination in Classical Logic. *Fundamenta Informaticae*, 45:123–155, 2000.

28 J. Zucker. The Correspondence Between Cut-Elimination and Normalization. *Annals of Mathematical Logic*, 7:1–112, 1974.

A Computational Interpretation of the Axiom of Determinacy in Arithmetic

Takanori Hida

Research Institute for Mathematical Sciences, Kyoto University
Kitashirakawa Oiwakecho, Sakyo-ku, Kyoto 606-8502, Japan
hida@kurims.kyoto-u.ac.jp

Abstract

We investigate the computational content of the axiom of determinacy (AD) in the setting of classical arithmetic in all finite types with the principle of dependent choices (DC). By employing the notion of realizability interpretation for arithmetic given by Berardi, Bezem and Coquand (1998), we interpret the negative translation of AD. Consequently, the combination of the negative translation with this realizability semantics can be seen as a model of DC, AD and the negation of the axiom of choice at higher types. In order to understand the computational content of AD, we explain, employing Coquand's game theoretical semantics, how our realizer behaves.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases The axiom of determinacy, Gale-Stewart's theorem, Syntactic continuity, Realizability interpretation, Coquand's game semantics

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.335

1 Introduction

The theory of infinite games has proven to be very effective in the study of various fields of logic and mathematics. There are a number of related works, and lots of game concepts have been proposed. Prominent among infinite game theory is the *two person infinite game with perfect information*, in which two players collaborate to define an infinite sequence of natural numbers by choosing a natural number alternately. There are many intriguing questions over this game, e.g., which games can be shown to be *determined*, in the sense that one of the two players has a winning strategy? D. Gale and F.M. Stewart [6] proved that all games for open or closed pay-off sets are determined. As the study of determinacy has revealed several remarkable consequences to mathematics, the axiom of determinacy (AD, for short) was introduced out of theoretical interest [17]: For every subset A of the Baire space ω^ω, the game $G(A)$ is determined. A substantial amount of research has been conducted over this topic and a number of deep results have been obtained (see [8]).

The focus of this paper, however, is on a somewhat different aspect from prior set-theoretical ones: what is the computational content of AD? For this purpose, we employ the notion of realizability interpretation for arithmetic given in [2]. Realizability interpretation, which is one formalization of BHK-interpretation, assigns a term to a valid formula. A realizer of a formula provides computational evidence for that formula, and thus endows it with computational content. Although there are other techniques for program extraction from formal proofs such as Curry-Howard correspondence [19], the realizability interpretation is better suited for our purpose: This methodology enables us to give interpretations even for some *non-trivial axioms* and *proofs using axioms*, because the definition of the realizability relation proceeds by induction not on *proofs* but on *formulas*.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 335–349
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Several notions of realizability interpretation have been presented [21]. Nevertheless, in terms of the usage of realizability for the exploration of the computational content of classical proofs, existing research can be classified into two categories: direct and indirect approaches. The former studies computational meanings immediately in the setting of classical logic. The fact that such an approach is possible comes as rather a surprise; from the viewpoint of computation, classical logic is much more difficult to deal with than intuitioninstic logic. Among this line of research, Krivine's classical realizability is of great importance [12, 13, 14]. This technique is developed as a generalization of forcing and, using the orthogonal structure between terms and stacks, provides a way of examining computational content of classical logic. On the other hand, the indirect approach, which we follow in this paper, consists of two steps. The first step is to embed classical proofs into an intuitionistic system by a negative translation; then by using some notion of intuitionistic realizability, we interpret the translated proofs.

The rest of this paper is organized as follows. In the next section, we define the basic terminology and provide background information briefly. In order to realize a negative translation of AD, we formalize the statement of Gale-Stewart's theorem in arithmetic and prove it in section 3. After presenting the notion of realizability in section 4, we interpret the negative translation of the statement "every subset of ω^ω is open" and, as a consequence, obtain a required realizer in section 5. Corollaries of this result include the (relative) consistency of the principle of dependent choices and AD in arithmetic. Since our purpose is not just to give a realizer of AD but also to know its computational meaning, we explain the behavior of the realizer using Coquand's game theoretical semantics in section 6. In the final section, we discuss future work.

2 Notations and Definitions

2.1 Infinite Games

▶ **Definition 1.** (Two person, infinite game with perfect information)

For an arbitrary subset A of the set ω^ω of all infinite sequences of natural numbers, G(A) denotes the following game:

- There are two players, Player I and Player II,
- At each round i, Player I chooses an $x_i \in \omega$, then Player II chooses a $y_i \in \omega$,

I	x_0		x_1		$\cdots$		x_i		$\cdots$	
II		y_0		y_1		$\cdots$		y_i		$\cdots$

- Player I wins the game $G(A)$ if the infinite sequence $\langle x_0, y_0, x_1, y_1, \dots \rangle$ is in A.

Each choice is called a *move* of the game, and the infinite sequence $\langle x_0, y_0, x_1, y_1, \dots \rangle$ is called a *play* of the game. We refer to A as the *pay-off set* for the game $G(A)$. *Perfect information* means that both players have complete access to the way the game has been played so far.

▶ **Definition 2.** (Strategies)

A strategy for Player I is a function $\sigma : \{s \in \omega^{<\omega} \mid s \text{ is of even length}\} \to \omega$,

A strategy for Player II is a function $\tau : \{s \in \omega^{<\omega} \mid s \text{ is of odd length}\} \to \omega$,

where $\omega^{<\omega}$ is the set of all finite sequences of natural numbers.

Each player decides his or her move according to a strategy as follows:

▶ **Definition 3.** (The plays $\sigma * y$ and $x * \tau$)

Let σ be a strategy for Player I. For each $y = \langle y_0, y_1, \dots \rangle$, $\sigma * y$ denotes the play $\langle a_0, y_0, a_1, y_1, \dots \rangle$, where $a_0 = \sigma(\langle\rangle)$ and $a_{n+1} = \sigma(\langle a_0, y_0, \dots, a_n, y_n \rangle)$.

Let τ be a strategy for Player II. For each $x = \langle x_0, x_1, \dots \rangle$, $x * \tau$ denotes the play $\langle x_0, b_0, x_1, b_1, \dots \rangle$, where $b_n = \tau(\langle x_0, b_0, \dots, x_n \rangle)$.

▶ **Definition 4.** (Winning strategies)

A strategy σ is a winning strategy for Player I in $G(A)$ if $\sigma * y \in A$ for all $y \in \omega^\omega$.

A strategy τ is a winning strategy for Player II in $G(A)$ if $x * \tau \notin A$ for all $x \in \omega^\omega$.

▶ **Definition 5.** (Determined)

A game $G(A)$ is determined if either Player I or Player II has a winning strategy in this game.

A set $A \subset \omega^\omega$ is determined if the game $G(A)$ is determined.

One natural question over this property would be: How much determinacy is derivable? It is easy to see that all finite and cofinite subsets are determined. More interestingly, it has been proven that all open and closed subsets [6], and all Borel subsets [15] are determined. (Recall that the standard topology on ω^ω is induced by an open base $\{O(s) \mid s \in \omega^{<\omega}\}$, where $O(s) := \{f \in \omega^\omega \mid s \text{ is an initial segment of } f\}$. This space is called the *Baire space*).

▶ **Definition 6.** (The axiom of determinacy (AD))

The axiom of determinacy (AD) is the statement that every $A \subset \omega^\omega$ is determined.

The relationship between AD and choice principles is worth pointing out: AD contradicts the (full) axiom of choice (AC) in ZF set theory[6], but implies a restricted version of the axiom of countable choice [18]. As regards the principle of dependent choices (DC), which is an essential tool in exploring the consequences of AD, it is known that DC is independent from ZF+AD [9].

There are a number of striking results around AD, such as its role in the study of consistency strength and applications to infinite combinatorics. The investigation of determinacy extends even to the area of second order arithmetic, e.g., [16]. The reader can find more information in, e.g., [8].

2.2 Systems of Arithmetic

In order to investigate AD in arithmetic, let us fix the basic terminology of arithmetic and present fundamental results. Firstly, we describe minimal (HA^ω_-), intuitionistic (HA^ω) and classical (HA^ω_c) arithmetic in all finite types. We borrow most of our notation from [2].

▶ **Definition 7.** (Formal systems HA^ω_-, HA^ω and HA^ω_c)

Types, terms and formulas of the three systems are the same and given by the following grammars:

Types $\tau, \tau' ::= \mathtt{N} \mid \tau \to \tau'$

Terms $t, u ::= x^\tau \mid \lambda x^\tau . t^{\tau'} \mid t^{\tau \to \tau'} u^\tau \mid \mathtt{0}^\mathtt{N} \mid \mathtt{s}^{\mathtt{N}\to\mathtt{N}} \mid \mathtt{Rec}_\tau^{\tau \to ((\mathtt{N}\to\tau\to\tau)\to(\mathtt{N}\to\tau))}$

where t^τ or $t : \tau$ indicate that a term t is of type τ.

Formulas $\phi, \psi ::= \bot \mid t^\mathtt{N} = t'^\mathtt{N}$ (prime formula) $\mid \phi \wedge \psi \mid \phi \Rightarrow \psi \mid \forall x : \tau\, \phi \mid \exists x : \tau\, \phi$

For every formula ϕ, we write $\neg\, \phi$ in place of $\phi \Rightarrow \bot$ for brevity.

Higher type equations are abbreviations, e.g., $f^{\mathtt{N}\to\mathtt{N}} = g^{\mathtt{N}\to\mathtt{N}}$ stands for $\forall n : \mathtt{N}\, (fn = gn)$.

Theory of HA^ω_-

- Axioms and rules for first order many sorted minimal logic (with each sort corresponding to a type).
- Equality axioms and the induction schema:

$$t = t \qquad (eq_1)$$

$$t_1 = s_1 \Rightarrow \cdots \Rightarrow t_k = s_k \Rightarrow ft_1 \cdots t_k = fs_1 \cdots s_k \qquad (eq_2)$$

$$t_1 = s_1 \Rightarrow \cdots \Rightarrow t_k = s_k \Rightarrow \{P(t_1, \ldots, t_k) \Leftrightarrow P(s_1, \ldots, s_k)\} \qquad (eq_3)$$

$$\phi(\mathtt{0}) \wedge \forall n \{\phi(n) \Rightarrow \phi(\mathtt{s}n)\} \Rightarrow \forall n\, \phi(n) \qquad (Ind)$$

- Successor axioms:

$$\neg\ \mathtt{s}n = \mathtt{0} \quad (Suc_1) \qquad \mathtt{s}n = \mathtt{s}m \Rightarrow n = m \quad (Suc_2)$$

- The defining equations of the constant $\mathtt{Rec}_\tau$ for each type τ:

$$\mathtt{Rec}\, tu\mathtt{0} = t \quad (Rec_0) \qquad \mathtt{Rec}\, tu(\mathtt{s}v) = uv(\mathtt{Rec}\, tuv) \quad (Rec_s)$$

- λ-calculus axiom and rules:

$$(\lambda x.t)u = t[u/x] \quad (\beta)$$

$$\frac{t = t'}{tu = t'u}\ (Ap_1) \qquad \frac{u = u'}{tu = tu'}\ (Ap_2) \qquad \frac{t = t'}{\lambda x.t = \lambda x.t'}\ (ir)$$

The theory of HA^ω (resp. HA^ω_c) is obtained from that of HA^ω_- by changing the base logic from minimal to intuitionistic (resp. classical).

From now on, we assume that the variables i, j, k, l, m, n are of type $\mathtt{N}$, f, g are of $\mathtt{N} \to \mathtt{N}$ and χ is of $(\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$, and omit types whenever there is no fear of confusion.

We then consider the following schemata (parametric in ϕ):

▶ **Definition 8.** (Schemata $Comp(\tau)$, $AC(\tau, \tau')$ and $DC(\tau)$)

$Comp(\tau)$ $\quad \exists \chi : \tau \to \mathtt{N}\ \forall x : \tau\ \{\chi(x) = \mathtt{1} \Leftrightarrow \phi(x)\}$.

$AC(\tau, \tau')$ $\quad \forall x : \tau\, \exists y : \tau'\ \phi(x, y) \Rightarrow \exists f : \tau \to \tau'\ \forall x : \tau\ \phi(x, f(x))$.

$DC(\tau)$ $\quad \forall x : \tau\, \exists y : \tau\ \phi(x, y) \Rightarrow \forall a : \tau\ \exists f : \mathtt{N} \to \tau\, \{f(\mathtt{0}) = a \wedge \forall n\, \phi(f(n), f(n + \mathtt{1}))\}$.

Using this notation, CAC (*the axiom of countable choice*) and DC (*the principle of dependent choices*) are expressed as $AC(\mathtt{N}, \tau)$ for all types τ and $DC(\tau)$ for all types τ, respectively.

▶ Remark. We refer to the schema $Comp(\tau)$ as comprehension under the identification of a set $\{x : \tau \mid \phi(x)\}$ with a function $\chi : \tau \to \mathtt{N}$ satisfying $\forall x : \tau\, \{\chi(x) = \mathtt{1} \Leftrightarrow \phi(x)\}$, namely a (generalized) characteristic function for $\{x : \tau \mid \phi(x)\}$. This schema is not counted as an axiom of HA^ω_c, and this may be the reason why [2] avoids the standard notation "PA^ω", in which comprehension is usually assumed. The absence of comprehension in our systems of arithmetic will be crucial in section 6.

▶ **Proposition 9.** For any type τ and τ', we have

1. $HA^\omega_- \vdash AC(\tau, \tau) \Rightarrow DC(\tau)$.
2. $HA^\omega_c \vdash DC(\mathtt{N} \to \tau) \Rightarrow AC(\mathtt{N}, \tau)$, and hence DC implies CAC.
3. $HA^\omega_- \vdash AC(\tau, \tau') \Rightarrow AC(\tau, \mathtt{N})$.
4. $HA^\omega_c \vdash AC(\tau, \mathtt{N}) \Rightarrow Comp(\tau)$. In particular, CAC implies $Comp(\mathtt{N})$.

For each formula ϕ of HA^ω_c, let ϕ^K denote the negative translation of ϕ obtained by prefixing all prime formulas and existentially quantified formulas by double negations. For instance, $\{\forall n\, \exists m\, (n + \mathtt{1} = m)\}^K$ is $\forall n \neg\neg \exists m \neg\neg (n + \mathtt{1} = m)$.

Let us point out a fact, which will be crucial in section 5. This translation enables us to embed classical arithmetic further into minimal arithmetic:

▶ **Proposition 10.** [2] $HA^\omega_c + DC \vdash \phi$ implies $HA^\omega_- + DC^K \vdash \phi^K$.

3 Gale-Stewart's Theorem in Classical Arithmetic

D. Gale and F.M. Stewart [6] proved in ZF set theory that all open subsets of the Baire space are determined (Open Determinacy). In this section, we formalize that statement in HA^ω_c and show informally that it is also provable in classical arithmetic.

Before proceeding any further, it would be better to introduce several abbreviations in order to enhance the readability of the following discussion:

- "n is odd" is the prime formula $\mathtt{odd?}(n) = \mathtt{1}$, where the term $\mathtt{odd?}(n)$ of HA^ω_c is equal to $\mathtt{1}$ when n is odd, and $\mathtt{0}$ otherwise.
- "$k \leq m$" is the prime formula $k \dot{-} m = \mathtt{0}$, where $\dot{-}$ is the term for the *truncated subtraction*: $k \dot{-} m$ is $k - m$ when $k > m$, and $\mathtt{0}$ otherwise.
- "$OP(\chi)$" is the formula $\forall f \ \{\chi(f) = \mathtt{1} \Rightarrow \exists m \ \forall g \ (\mathtt{eq}_{\leq m}(f,g) = \mathtt{1} \Rightarrow \chi(g) = \mathtt{1})\}$, where the term "$\mathtt{eq}_{\leq m}(f,g)$" of HA^ω_c is equal to $\mathtt{1}$ when $f(k) = g(k)$ for all $k \in \{0, \ldots, m\}$, and $\mathtt{0}$ otherwise.
- "OP" is the formula $\forall \chi \ OP(\chi)$.

▶ Remark. It will be easy to confirm that functions like $\mathtt{odd?}$, $\dot{-}$ and $\mathtt{eq}_{\leq m}(f,g)$ can be implemented as terms of HA^ω. Notice also that these defined symbols do not add any power, for HA^ω_{-} proves the equivalence between the prime formula $\mathtt{odd?}(n) = \mathtt{1}$ (resp. $\mathtt{eq}_{\leq m}(f,g) = \mathtt{1}$) and the formula $\exists k \ (k \leq n \ \wedge \ n = \mathtt{2}k + \mathtt{1})$ (resp. $\forall k \ \{k \leq m \Rightarrow f(k) = g(k)\}$). Henceforth, we introduce defined symbols in this way, i.e., without presenting the implementation as terms of HA^ω.

▶ Remark. $\chi : (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$ can be seen as a (generalized) characteristic function for some $A \subset \omega^\omega$: $\forall f \, \{\chi(f) = \mathtt{1} \Leftrightarrow f \in A\}$. With this in mind, the formula $OP(\chi)$ is read as "χ represents an open subset of the Baire space ω^ω".

In the sequel, we need an encoding of $\omega^{<\omega}$ into ω in order to formalize the theory of infinite games within arithmetic; fix a primitive recursive bijection $\langle\langle \cdot, \ldots, \cdot \rangle\rangle : \omega^{<\omega} \to \omega$. We also write $(n)_j := a_j \ (0 \leq j < k)$ and $lh(n) := k$ if $n = \langle\langle a_0, \ldots, a_{k-1} \rangle\rangle$.

By employing this encoding, the plays in Definition 3 can be expressed by the following terms, where σ, y, τ and x are of type $\mathtt{N} \to \mathtt{N}$:

$$\sigma * y\,(i) := \begin{cases} y((i \dot{-} \mathtt{1})/\mathtt{2}) & (i : \text{odd}) \\ \sigma(\langle\langle \sigma * y\,(\mathtt{0}), \ldots, \sigma * y\,(i \dot{-} \mathtt{1}) \rangle\rangle) & (i : \text{even}) \end{cases},$$

$$x * \tau\,(i) := \begin{cases} \tau(\langle\langle x * \tau\,(\mathtt{0}), \ldots, x * \tau\,(i \dot{-} \mathtt{1}) \rangle\rangle) & (i : \text{odd}) \\ x(i/\mathtt{2}) & (i : \text{even}) \end{cases}.$$

Strictly speaking, we should use different symbols for $*$ in $\sigma * y : \mathtt{N} \to \mathtt{N}$ and $x * \tau : \mathtt{N} \to \mathtt{N}$, since now all of σ, y, τ and x are of the same type. However, no confusion may be caused by this, as it is clear from the context.

For convenience, we also adopt the following abbreviations:

- "*I has a w.s. in* $G(\chi)$" is the formula $\exists \sigma : \mathtt{N} \to \mathtt{N} \ \forall y : \mathtt{N} \to \mathtt{N} \ \chi(\sigma * y) = \mathtt{1}$.
- "*II has a w.s. in* $G(\chi)$" is the formula $\exists \tau : \mathtt{N} \to \mathtt{N} \ \forall x : \mathtt{N} \to \mathtt{N} \ \neg \ \chi(x * \tau) = \mathtt{1}$.
- "$Det(\chi)$" is the formula $\neg\,$(*I has a w.s. in* $G(\chi)$) $\Rightarrow$ (*II has a w.s. in* $G(\chi)$).
- "AD" is the formula $\forall \chi \ Det(\chi)$.

Now, let us formalize open determinacy in the language of HA^ω_c and prove it within arithmetic. Although the proof is presented informally, it can easily be formalized in $HA^\omega_c + CAC$.

▶ **Theorem 11.** (Gale-Stewart [6]) $HA^\omega_c + CAC \vdash \forall \chi \ \{OP(\chi) \Rightarrow Det(\chi)\}$.

Proof. For each $x : \mathtt{N}$ with $x = \langle\langle n_0, \dots, n_{k-1}\rangle\rangle$ and $f : \mathtt{N} \to \mathtt{N}$, let us define $x \mathbin{@} f : \mathtt{N} \to \mathtt{N}$ by

$$x \mathbin{@} f := \langle n_0, \dots, n_{k-1}, f(0), f(1), f(2), \dots \rangle.$$

Using this notation, for each $\chi : (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$ and $x : \mathtt{N}$, we introduce $\chi/_{x} : (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$ by

$$\chi/_{x}\,(f) = 1 \Leftrightarrow \chi(x \mathbin{@} f) = 1.$$

▶ **Lemma 12.** For every $x : \mathtt{N}$ with $lh(x)$ odd, if $\neg\,(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}2}\rangle\rangle}))$, then there exists a y such that $\neg\,(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}2}, (x)_{lh(x)\dot{-}1}, y\rangle\rangle}))$.

Proof. We show the contraposition of the above statement. If there exists an x such that $lh(x)$ is odd, and $(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}1}, y\rangle\rangle}))$ holds for all y, then CAC yields a $\varphi : \mathtt{N} \to (\mathtt{N} \to \mathtt{N})$ such that $\varphi(y)$ is a winning strategy for Player I in the game $G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}1}, y\rangle\rangle})$. Define a strategy $\rho : \mathtt{N} \to \mathtt{N}$ for Player I by:

$$\rho(n) = \begin{cases} (x)_{lh(x)\dot{-}1} & (n = \langle\langle\rangle\rangle) \\ \varphi(y)\langle\langle p_0, \dots, p_{2l\dot{-}1}\rangle\rangle & (n = \langle\langle (x)_{lh(x)\dot{-}1}, y, p_0, \dots, p_{2l\dot{-}1}\rangle\rangle) \\ 0 & (\text{else}) \end{cases}.$$

Then, $\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}2}\rangle\rangle}(\rho * z) = \chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}1}, z(0)\rangle\rangle}(\varphi\,(z(0)) * \mathtt{shift}(z)) = 1$ holds for all $z : \mathtt{N} \to \mathtt{N}$, where $\mathtt{shift}(z)$ is $\lambda n.\, z(n+1)$. This means that ρ is a winning strategy for Player I in the game $G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}2}\rangle\rangle})$. ◀

▶ **Lemma 13.** There exists a $\tau : \mathtt{N} \to \mathtt{N}$ such that for all $x : \mathtt{N}$, we have

$$\{lh(x)\ is\ odd \wedge \neg\,(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}2}\rangle\rangle})) \Rightarrow \\ \neg\,(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle (x)_0, \dots, (x)_{lh(x)\dot{-}1}, \tau(x)\rangle\rangle}))\}.$$

Proof. Apply CAC to the statement of the previous lemma. ◀

Assume $OP(\chi)$ and $\neg\,(I\ has\ a\ w.s.\ in\ G(\chi))$. We show that the above τ is indeed a winning strategy for Player II. Assume for contradiction that there were an $x : \mathtt{N} \to \mathtt{N}$ such that $\chi(x * \tau) = 1$; then there exists an m such that, for all g, $\mathtt{eq}_{\leq m}(x * \tau, g) = 1$ implies $\chi(x * \tau) = \chi(g)$. In particular, if $\mathtt{eq}_{\leq 2m+1}(x * \tau, g) = 1$ holds, then $\chi(x * \tau)$ is equal to $\chi(g)$. Therefore, for all $y : \mathtt{N} \to \mathtt{N}$, it follows that $\chi/_{\langle\langle x * \tau(0), \dots, x * \tau(2m+1))\rangle\rangle}((\lambda n.0) * y) = \chi(\langle\langle x * \tau\,(0), \dots, x * \tau\,(2m+1)\rangle\rangle \mathbin{@} ((\lambda n.0) * y)) = 1$.

On the other hand, $(m+1)$-times applications of Lemma 13 to the hypothesis $\neg(I\ has\ a\ w.s.\ in\ G(\chi))$ yields $\neg\,(I\ has\ a\ w.s.\ in\ G(\chi/_{\langle\langle x * \tau(0), \dots, x * \tau(2m+1))\rangle\rangle}))$. This means that for the strategy $\lambda n.0$, there exists a $y : \mathtt{N} \to \mathtt{N}$ with $\chi/_{\langle\langle x * \tau(0), \dots, x * \tau(2m+1))\rangle\rangle}((\lambda n.0) * y) \neq 1$. A contradiction. ◀

▶ Remark. By Proposition 9 and Theorem 11, we immediately see that $HA^{\omega}_{c} + DC$ proves $\forall\chi\ \{OP(\chi) \Rightarrow Det(\chi)\}$.

4 Realizability Interpretation

This section is a recapitulation of the notion of the realizability interpretation given in [2]. Since we would like to interpret HA^{ω} in a programming language with this methodology, we need first to present the (infinitary) programming language $\mathscr{P}$. Roughly speaking, $\mathscr{P}$ is an extension of Gödel's system T with list operators, the fixed-point combinator and some auxiliary constructs (needed for realizing DC^{K}). The types and terms of $\mathscr{P}$ are extensions of that of HA^{ω}.

▶ **Definition 14.** (The programming language $\mathscr{P}$)

Types Given by the following grammar: $\tau, \tau' ::= \mathtt{N} \mid \mathtt{Unit} \mid \mathtt{Abs} \mid \tau \to \tau' \mid \tau \times \tau' \mid [\tau]$

Here, $[\tau]$ is the type for lists of objects of type τ.

Seen as a type of $\mathscr{P}$, a type of HA^{ω} is called an $\mathtt{N}$-*type*.

Terms Given by the following grammar:

$$\begin{aligned}
t, u ::= {}& x^{\tau} \mid \lambda x^{\tau}.t^{\tau'} \mid t^{\tau \to \tau'} u^{\tau} && \text{(lambda terms)}\\
&\mid \mathtt{0}^{\mathtt{N}} \mid \mathtt{s}^{\mathtt{N}\to\mathtt{N}} \mid \mathtt{Rec}_{\tau}^{\tau\to((\mathtt{N}\to\tau\to\tau)\to(\mathtt{N}\to\tau))} && \text{(system T constants)}\\
&\mid Y_{\tau}^{(\tau\to\tau)\to\tau} && \text{(the fixed-point combinator)}\\
&\mid \langle,\rangle^{\tau_1\to\tau_2\to(\tau_1\times\tau_2)} \mid \pi_i^{(\tau_1\times\tau_2)\to\tau_i}\ (i=1,2) && \text{(pairing and projection)}\\
&\mid \mathtt{nil}^{[\tau]} \mid \mathtt{cons}^{\tau\to[\tau]\to[\tau]} \mid \mathtt{Lrec}^{(\tau\to[\tau]\to\sigma\to\sigma)\to\sigma\to[\tau]\to\sigma} && \text{(list operators)}\\
&\mid \mathtt{Dummy}^{\mathtt{Abs}} \mid \mathtt{Axiom}_{\mathtt{i}}^{\mathtt{N}\to\mathtt{Abs}}(i=1,2) \mid ()^{\mathtt{Unit}} && \text{(technical constants)}\\
&\mid (\lambda\!\!\!\lambda n.t_n^{\tau})^{\mathtt{N}\to\tau} && \text{(an infinite term)}
\end{aligned}$$

Infinite operator $\lambda\!\!\!\lambda$ allows us to built a single term $\lambda\!\!\!\lambda n.t_n^{\tau}$ out of an arbitrary sequence $t_0^{\tau}, t_1^{\tau}, \ldots$ of terms of type τ.

We abbreviate $\mathtt{s0}$ as $\mathtt{1}$, $\mathtt{ss0}$ as $\mathtt{2}$ and so on, and refer to them as *numerals* hereafter.

Formulas For each type τ, $t_1^{\tau} = t_2^{\tau}$ is a formula.

Theory

- The defining equations of the constant $\mathtt{Rec}_{\tau}$ for each type τ (see Definition 7).
- λ-calculus axiom and rules (see Definition 7).
- The axiom for the fixed-point combinator $\mathtt{Y}_{\tau}$: $\mathtt{Y}t = t(\mathtt{Y}t)$ $\quad (Y)$
- Pairing axioms and list axioms:

$$\begin{aligned}
&\pi_i\langle t_1, t_2\rangle = t_i\ (i=1,2) && (pr_i)\\
&\mathtt{Lrec}(f, u, \mathtt{nil}) = u && (Lrec_0)\\
&\mathtt{Lrec}(f, u, \mathtt{cons}(t, L)) = f(t, L, \mathtt{Lrec}(f, u, L)) && (Lrec_1)
\end{aligned}$$

- The axiom for infinite terms: $(\lambda\!\!\!\lambda n.t_n)\mathtt{k} = t_k$ $\quad (\beta\!\!\!\beta)$

▶ Remark. Although infinite terms and unfamiliar constants appear to be ad hoc, such terms are *not* included for computational purposes. In fact, every theorem of $HA^{\omega}_{-} + DC^{K}$ can be realized without them [2]; moreover, these technical terms are not so important for the rest of this paper. However, the infinite operator $\lambda\!\!\!\lambda$ and *two* constants $\mathtt{Axiom}_1$ and $\mathtt{Axiom}_2$ are necessary for testing termination of realizers of CAC^{K} and DC^{K} [2].

Let us list several known facts about $\mathscr{P}$ [2, Section 3.4]:

- There exists a reduction relation $\rightsquigarrow$ such that its reflexive, symmetric and transitive closure coincides with $=$.

  Moreover, this reduction relation enjoys the following:
- The Church-Rosser property,
- Every closed normal form of type $\mathtt{N}$ (resp. $\mathtt{Unit}$) is a numeral (resp. $()$),
- Every closed normal form of type $\mathtt{Abs}$ is either $\mathtt{Dummy}$ or of the form $\mathtt{Axiom}_{\mathtt{i}}\mathtt{k}$.

We present two preparatory definitions in advance of the main definition of the realizability relation. With each formula ϕ of HA^{ω}, we associate a type $|\phi|$ of $\mathscr{P}$ as follows:

▶ **Definition 15.** (Associated type $|\phi|$ of $\mathscr{P}$)

- $|t = t'|$ is $\mathtt{Unit}$,
- $|\bot|$ is $\mathtt{Abs}$,

- $|\phi \Rightarrow \psi|$ is $|\phi| \to |\psi|$,
- $|\phi \wedge \psi|$ is $|\phi| \times |\psi|$,
- $|\forall x : \tau\, \phi|$ is $\tau \to |\phi|$,
- $|\exists x : \tau\, \phi|$ is $\tau \times |\phi|$.

For every closed term t of $\mathscr{P}$ of the $\mathtt{N}$-type, the technical notion of *reducibility* is given by induction on the $\mathtt{N}$-type:

▶ **Definition 16.** (Reducible terms of $\mathtt{N}$-type)

- $t : \mathtt{N}$ is reducible if t reduces to $\mathtt{k}$ for some $k \in \omega$,
- $t : \tau \to \tau'$ is reducible if $tu : \tau'$ is reducible for all reducible $u : \tau$.

All terms of HA^ω are clearly reducible.

We cite the following property of $\mathscr{P}$ from [2, Section 3.4] without proof. Note that this so-called syntactic continuity[1] can also be taken as a topological continuity: χ is a continuous function from the Baire space ω^ω to the discrete space ω.

▶ **Proposition 17.** For every reducible terms $\chi : (\mathtt{N} \to \mathtt{N}) \to \mathtt{N}$ and $f : \mathtt{N} \to \mathtt{N}$, there exists an $m \in \omega$ such that for all reducible $g : \mathtt{N} \to \mathtt{N}$ with $f(\mathtt{i}) = g(\mathtt{i})\ (\forall\, i \leq m)$, we have $\chi f = \chi g$.

This proposition says that the closed normal form of χf is determined only from finitely many outputs of f. This property does not come as a surprise, for f is used (essentially) finitely many times during the reduction of χf to a numeral.

We are now in a position to define the realizability relation $t : |\phi| ® \phi$ for a closed formula ϕ of HA^ω possibly including closed reducible terms of $\mathscr{P}$, and for a closed term t of $\mathscr{P}$ of type $|\phi|$. This notion of realizability is essentially the so-called modified realizability [11], except the restriction to the reducible terms when interpreting quantifiers, and the existence of a term t satisfying that $t ® \bot$.

▶ **Definition 18.** (Realizability relation)

- $t : \mathtt{Abs} ® \bot$ if $t = \mathtt{Axiom_i k}$ for some $k \in \omega$ and $i = 1, 2$,
- $t : \mathtt{Unit} ® t_1 = t_2$ if $t = ()$ and both t_1 and t_2 reduce to the same numeral in $\mathscr{P}$,
- $t : |\phi_1| \to |\phi_2| ® \phi_1 \Rightarrow \phi_2$ if $tu ® \phi_2$ whenever $u ® \phi_1$,
- $t : |\phi_1| \times |\phi_2| ® \phi_1 \wedge \phi_2$ if $t = \langle t_1, t_2 \rangle$ and $t_i ® \phi_i$ $(i = 1, 2)$,
- $t : \tau \to |\phi| ® \forall x : \tau\ \phi$ if $tu ® \phi\ [u/x]$ for all reducible $u : \tau$,
- $t : \tau \times |\phi| ® \exists x : \tau\ \phi$ if $t = \langle p, u \rangle$ with $p : \tau$ reducible and $u ® \phi\ [p/x]$.

▶ Remark. Since $\mathscr{P}$ contains the fixed-point combinator $\mathtt{Y}$, the non-termination problem arises. For exactly this reason, quantification should be restricted to reducible terms, in other words, to hereditarily normalizing terms. Otherwise, there could be problems such as, (i) a realizer of an existential formula may fail to give a witness, and (ii) the identity axiom "$\forall x\ (x = x)$" cannot be realized.

[1] It has been proved by Čeitin (independently by Kreisel, Lacombe and Shoenfield) that for every effective operation e and total recursive index y, a modulus of continuity for e at y can be computed by a partial recursive function under the assumption of Markov principle (see [1, Chapter IV, Theorem 3.1]). However, it would be impossible here to adopt this theorem to show the existence of a modulus-of-continuity functional, i.e., a partial recursive function which compute a modulus of continuity. This is because our setting is far from intensional. Furthermore, f and g can be non-recursive here due to the existence of the infinite operator. In fact, it is known that there is *no* extensional modulus-of-continuity functional [1, Chapter IV, Section 3.3].

▶ Remark. If, for every term t and formula ϕ, $t \circledR \phi$ holds if and only if $t \circledR \phi^K$ holds, then the negative translation K plays no effective role in realizing a formula. The first clause of the above definition is demanded to break this equivalence, at the price of the non-existence of a realizer for the ex falso axiom: $\bot \Rightarrow \phi$. Moreover, this definition allows us to use $\mathtt{Axiom_i}$ for computing witnesses [2, Section 4.4]: These terms catch a witness $\mathtt{n}$ of an existential formula $\exists n\, \phi(n)$ during reduction and freeze that datum as in the form of $\mathtt{Axiom_i n}$. When the execution of a program stops, one can pick up that $\mathtt{n}$ out of the residue of the calculation.

We say a formula ϕ is *realizable* when there exists a $\{\lambda\!\!\lambda, \mathtt{Axiom_i}(i = 1, 2)\}$-free term t satisfying that $t : |\phi| \circledR \phi$. The main theorem of [2] reads in this notation as follows:

▶ **Theorem 19.** [2] Every theorem of $HA^{\omega}_{=} + DC^K$ is realizable.

▶ Remark. In [2], the most difficult cases, the realizations of $CAC^K$and DC^K, are managed with bar recursion and continuity. The problem of the termination of realizers, which boils down to the problem of the termination of bar recursion used in them, is proved non-constructively. The difficulty is to be attributed to the fact that the *negatively translated* choice principles are much more powerful than the choice principles themselves in HA^{ω}. In fact, [7] shows that $HA^{\omega} + AC(\mathtt{N}, \mathtt{N}) + AC(\mathtt{N}, \mathtt{N} \to \mathtt{N})$ is conservative over Heyting arithmetic.

▶ **Corollary 20.** $\forall \chi\ \{OP(\chi) \Rightarrow \mathit{Det}(\chi)\}^K$ is realizable.

By closely following the proof of Theorem 11, we obtain the following realizer of the formula $\forall \chi\ \{OP(\chi) \Rightarrow \mathit{Det}(\chi)\}^K$ (see Appendix for detail): $\lambda \chi\theta\zeta\eta.\Phi P_2(\zeta, \eta, \theta) H_2(\chi)[\,]$.

5 A Realizer of the Negative Translation of AD

In view of Corollary 20, it suffices to realize OP^K for realizing AD^K.

Let us consider again the formula OP:

$$\forall \chi\ \forall f\ \{\chi(f) = \mathtt{1} \Rightarrow \exists m\ \forall g\ (\mathtt{eq}_{\leq m}(f, g) = \mathtt{1} \Rightarrow \chi(g) = \mathtt{1})\},$$

which expresses that "every χ represents an open subset of ω^{ω}". More precisely, OP states that for every element f of (the set represented by) χ, there exists an m such that a basic open neighborhood $\{g \mid \mathtt{eq}_{\leq m}(f, g) = \mathtt{1}\}$ at f is contained in χ. Following [1], we call this m a *modulus* for χ at f.

Let us recall Proposition 17 here: for every reducible χ and f, the existence of a modulus m for χ at f is assured there from the *external* viewpoint. This, however, does not imply the existence of an *internally definable* term t such that $t(\chi, f)$ is a modulus for χ at f. It would be impossible to build such a term t in our setting (see footnote 1).

To realize OP itself will also be impossible, for if there were a realizer s of OP, a modulus could be computed internally as follows: Take any reducible χ and f with $\chi(f) = \mathtt{1}$. Then, from $s \circledR OP$ and $() \circledR \chi(f) = \mathtt{1}$, we see that $s\chi f()$ witnesses $\exists m\ \forall g\ (\mathtt{eq}_{\leq m}(f, g) = \mathtt{1} \Rightarrow \chi(g) = \mathtt{1})$, and hence, $\pi_1(s\chi f())$ reduces to a modulus.

The thing is quite different when it comes to realizing the negative translation OP^K of OP. In contrast to the realization of OP, where we do have to calculate a modulus only from χ and f internally, it suffices to indicate the existence of a modulus m externally when realizing OP^K. This point—internal or external—is to be noted as an essential difference between intuitionistic and classical logic.

▶ Remark. The reader may still have some doubt if it is really possible to realize even the negative translation of such a strange statement. This happens by virtue of the absence of the comprehension schema $\mathit{Comp}(\mathtt{N} \to \mathtt{N})$ at type $\mathtt{N} \to \mathtt{N}$. In fact, $\mathit{Comp}(\mathtt{N} \to \mathtt{N})$ implies the existence of a "set" χ satisfying $\forall f\ \{\chi(f) = \mathtt{1} \Leftrightarrow \forall n\, (f(n) = \mathtt{0})\}$; but such χ is *not* open.

Now we prove:

▶ **Lemma 21.** OP^K is realizable.

Proof. First of all, let us recall the formula OP^K:

$$\forall\chi\,\forall f\,\{\neg\neg\,\chi(f) = \mathtt{1} \Rightarrow \neg\neg\,\exists m\,\forall g\,(\neg\neg\,\mathtt{eq}_{\le m}(f,g) = \mathtt{1} \Rightarrow \neg\neg\,\chi(g) = \mathtt{1})\}.$$

To realize this, we introduce a term Θ by

$$\begin{aligned} &\Theta\,\chi f u v n := v\langle n, \lambda gpq.\, t(n,g,p,q)\rangle, \text{ with}\\ &t(n,g,p,q) := \mathtt{if}\ \mathtt{eq}_{\le n}(f,g) = \mathtt{0}\ \mathtt{then}\ p(\lambda r.\mathtt{Dummy})\ \mathtt{else}\\ &\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{if}\ \chi(f) = \chi(g)\ \mathtt{then}\ uq\ \mathtt{else}\ \Theta\,\chi f u v(\mathtt{s}n), \end{aligned}$$

where $\mathtt{if}\ \mathtt{n} = \mathtt{m}\ \mathtt{then} \cdots \mathtt{else} \cdots$ is a syntactic sugar.

In the following, we show that $\lambda\chi f u v.\Theta\,\chi f u v\mathtt{0} \circledR OP^K$. Take arbitrary reducible terms χ and f. We need to prove $\Theta\,\chi f U V \mathtt{0} \circledR \perp$ for every term U and V with $U \circledR \neg\neg\,\chi\,(f) = \mathtt{1}$ and $V \circledR \neg\,\exists m\ \forall g\ (\neg\neg\,\mathtt{eq}_{\le m}(f,g) = \mathtt{1} \Rightarrow \neg\neg\,\chi(g) = \mathtt{1})$.

We first claim that, for every n, $\Theta\,\chi f U V\mathtt{sn} \circledR \perp$ implies $\Theta\,\chi f U V\mathtt{n} \circledR \perp$. Assume that $\Theta\chi f U V\mathtt{sn} \circledR \perp$. Since we have $V \circledR \{\exists m\,\forall g\,(\neg\neg\,\mathtt{eq}_{\le m}(f,g) = \mathtt{1} \Rightarrow \neg\neg\,\chi(g) = \mathtt{1}) \Rightarrow \perp\}$ and $\Theta\,\chi f U V\mathtt{n} = V\langle \mathtt{n}, \lambda gpq.\, t(\mathtt{n},g,p,q)\rangle$, in order to show our first claim, it suffices to prove $\lambda gpq.\ t(\mathtt{n},g,p,q) \circledR \forall g\ (\neg\neg\,\mathtt{eq}_{\le \mathtt{n}}(f,g) = \mathtt{1} \Rightarrow \neg\neg\,\chi(g) = \mathtt{1})$. Take an arbitrary reducible term $g : \mathtt{N} \to \mathtt{N}$ and terms P and Q satisfying $P \circledR \neg\neg\,\mathtt{eq}_{\le \mathtt{n}}(f,g) = \mathtt{1}$ and $Q \circledR \neg\,\chi\,(g) = \mathtt{1}$. We have to examine the following three cases to verify $t(\mathtt{n},g,P,Q) \circledR \perp$:

Case 1: $\exists i \le n\ \ f(i) \ne g(i)$ — $t(\mathtt{n},g,P,Q)$ reduces to $P(\lambda r.\mathtt{Dummy})$. Since we have $\mathtt{eq}_{\le \mathtt{n}}(f,g) = \mathtt{0}$ and $P \circledR \neg\neg\,\mathtt{eq}_{\le \mathtt{n}}(f,g) = \mathtt{1}$, we conclude that $P(\lambda r.\mathtt{Dummy}) \circledR \perp$.

Case 2: $\forall i \le n\ f(i) = g(i)$ and $\chi(f) = \chi(g)$ — $t(\mathtt{n},g,P,Q)$ reduces to UQ. Since we have $U \circledR \neg\neg\,\chi\,(f) = \mathtt{1}$ and $Q \circledR \neg\,\chi\,(g) = \mathtt{1}$, it follows that $UQ \circledR \perp$.

Case 3: $\forall i \le n\ \ f(i) = g(i)$ and $\chi(f) \ne \chi(g)$ — In this case, $t(\mathtt{n},g,P,Q)$ reduces to $\Theta\chi f U V\mathtt{sn}$. Hence we have $t(\mathtt{n},g,P,Q) \circledR \perp$ by the hypothesis.

Next, we claim that if m is a modulus for χ at f, then $\Theta\,\chi f U V\mathtt{m} \circledR \perp$ holds. The proof proceeds along the same line as above except the last case, which no longer happen by the fact that m is a modulus.

Since Proposition 17 assures the existence of a modulus m for χ at f, though we know the existence only externally, $\Theta\,\chi f U V\mathtt{0} \circledR \perp$ follows from the foregoing arguments. ◀

▶ Remark. Inspired by [2], U. Berger and P. Oliva presented a similar result axiomatically in [3]. Instead of implementing a bar recursion as a term of $\mathscr{P}$ using the fixed-point combinator $\mathtt{Y}$, they extended the calculus by *directly adding* the so-called *modified bar recursion* (MBR), which allows us to approximate a choice function and to realize DC^K. We have an impression that OP^K is not realizable in their framework. If an unbounded search used as in Θ were primitive recursively definable (p.r.d.) in MBR, the functional $\hat{\mu}$ would also be p.r.d. in MBR, where $\hat{\mu}(\chi,f) := \min\{\, k \mid \chi(f{\restriction}k \mathbin{@} \lambda n.\mathtt{0}) = \chi(f{\restriction}k \mathbin{@} \lambda n.\mathtt{1})\}$. If so, Kohlenbach's bar recursion (KBR) is p.r.d. in MBR, because KBR is p.r.d. in $\hat{\mu}$ plus Spector's bar recursion (SBR) [10], and SBR is p.r.d. in MBR [4]. However, KBR is *not* p.r.d. in MBR [4].

▶ **Theorem 22.** AD^K is realizable.

Proof. Follows easily from Corollary 20 and Lemma 21. ◀

▶ **Corollary 23.** $\neg\,AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})^K$ is realizable.

Proof. $HA_c^\omega + AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$ proves an instance $\exists\chi\forall f\,\{\chi(f) = \mathtt{1} \Leftrightarrow \forall n\,(f(n) = \mathtt{0})\}$ of $Comp(\mathtt{N} \to \mathtt{N})$ by Proposition 9 . In view of the remark above Lemma 21, we find that $HA_c^\omega + AC(\mathtt{N} \to \mathtt{N}, \mathtt{N}) + OP$ is inconsistent. Thus, HA_c^ω proves $OP \Rightarrow \neg\, AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$, and hence, $OP^K \Rightarrow \neg\, AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})^K$ is realizable. The assertion follows from Lemma 21. ◀

▶ Remark. Corollary 23 shows that it is impossible to realize the axiom of choice at higher order in the framework of [2]. But even further is indicated by the above discussion: To realize such an axiom would be hopeless in any reasonable setting—at least if one sticks to the usual indirect approach. If we assume continuity, we will fail to realize $AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})^K$. This is because we may interpret OP^K, which contradicts the axiom of choice at higher type as we saw. On the other hand, if we drop the assumption of continuity, to realize even CAC becomes difficult. It is only a novel idea, if any, that can open the possibility for the higher order.

From these results, we find that the combination of the negative translation K and the realizability semantics à la [2] can be seen as a model of $HA_c^\omega + DC + AD + \neg\, AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$. Therefore, we can reduce the consistency of this system to that of $\mathscr{P}$:

▶ **Corollary 24.** $HA_c^\omega + DC + AD + \neg\, AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$ is consistent.

The next is a straightforward consequence of the previous corollary and Proposition 9.

▶ **Corollary 25.** $HA_c^\omega + DC \vdash AC(\mathtt{N}, \tau)$, but $HA_c^\omega + DC \nvdash AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$.

As far as the author knows, Corollaries 24 and 25 do not follow trivially from known results in set theory.[2] One future work is to investigate whether or not Corollary 24 remains true in the presence of $Comp(\mathtt{N} \to \mathtt{N})$.

Note that the foregoing three corollaries are still valid even if we replace $AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$ by $AC(\mathtt{N} \to \mathtt{N}, \tau)$ for an arbitrary type τ (see Proposition 9).

6 How Does the Realizer Behave?

By combining the realizer of OP^K given in the proof of Lemma 21 and the realizer of $\forall\chi\,\{OP(\chi) \Rightarrow Det(\chi)\}^K$ given after Corollary 20 (see also Appendix), we obtain a realizer of AD^K. This section is devoted to the explanation of its behavior.

Our realizer takes the following two steps:

Step 1: construct a strategy τ for Player II

In order to facilitate understanding of this step, let us employ Coquand's game theoretical semantics [5]. Firstly we give a recapitulation of that semantics below. The semantics is defined for an infinitary propositional calculus. The formulas of this calculus are defined inductively as: (i) 0 and 1 are (atomic) formulas, and (ii) if ϕ_i $(i \in I)$ are formulas, where I is a countable set, then both $\bigwedge_{i\in I} \phi_i$ and $\bigvee_{i\in I} \phi_i$ are formulas. Note that each arithmetical formula can be represented as a formula of this infinitary propositional calculus in a natural way. For instance, AD is expressed as:

$$\bigvee_{\sigma} \bigwedge_{y} \chi(\sigma * y) = \mathtt{1} \;\vee\; \bigvee_{\tau} \bigwedge_{x} \overline{\chi(x * \tau) = \mathtt{1}}. \qquad (*)$$

[2] As regards Corollary 24, one may think that at least the consistency of $HA_c^\omega + DC + AD$ follow trivially from that of $ZF + DC + AD$. This is certainly so, but the consistency of $ZF + DC + AD$ itself is *much stronger* than that of ZFC [8].

Henceforth, some formulas of HA^{ω}_{c} will be considered as formulas of this calculus without further explanation.

We then introduce the notion of *classical validity* by specifying the set $\mathcal{V}$ of classically valid formulas. $\mathcal{V}$ is the smallest set of formulas satisfying: (i) $1 \in \mathcal{V}$, (ii) $\bigwedge_{i \in I} \phi_i \in \mathcal{V}$ if $\phi_i \in \mathcal{V}$ for all $i \in I$, and (iii) $\bigvee_{i \in I} \phi_i \in \mathcal{V}$ if there exists an $i_0 \in I$ such that either ϕ_{i_0} is 1, or ϕ_{i_0} is of the form $\bigwedge_{j \in J} \phi_{i_0 j}$ with $\phi_{i_0 j} \vee \bigvee_{i \in I} \phi_i \in \mathcal{V}$ for all $j \in J$.[3]

Game theoretical semantics for this calculus is given as a perfect information game over a formula between two players: ∃loise, who plays for existential formulas, and ∀belard, who plays for universal formulas. Here, we regard atomic formulas as both universal and existential. The game for a formula ϕ is played as follows: If ∃loise (resp. ∀belard) has to play and ϕ is atomic, then ∃loise (resp. ∀belard) wins if ϕ is 1 (resp. 0). If ϕ is universal of the form $\bigwedge_{i \in I} \phi_i$, then ∀belard has to choose an $i \in I$ and ∃loise starts the game for ϕ_i. If ϕ is existential of the form $\bigvee_{i \in I} \phi_i$, then ∃loise chooses an $i \in I$ and wins if ϕ_i is 1, looses if ϕ_i is 0. When ϕ_i is universal of the form $\bigwedge_{j \in J} \phi_{ij}$, ∃loise can start the game not for ϕ_{ij} but for $\phi_{ij} \vee \bigvee_{i \in I} \phi_i$ after ∀belard returns a $j \in J$. The rule of this game is rather unfair to ∀belard; it is only ∃loise who is allowed to change her mind and backtrack in her choice. It will be easy to verify that

▶ **Proposition 26.** [5] ∃loise has a winning strategy for $\phi \;\Leftrightarrow\; \phi$ is classically valid.

Then, in order to describe this step, let us consider the following instance of the axiom of countable choice used in the proof of Lemma 13:

$$CAC_{\phi} := \bigvee_{x} \bigwedge_{y} \overline{\phi_{xy}} \vee \bigvee_{\tau} \bigwedge_{x} \phi_{x\tau(x)}, \text{ with}$$

$$\phi_{xy} := \overline{(lh\,(x)\ \ is\ odd)}^{K} \vee (I\ has\ a\ w.s.\ in\ G(\mathcal{X}/_{\langle\langle (x)_0, \ldots, (x)_{lh(x) \dot{-} 2}\rangle\rangle}))^{K} \vee \overline{(I\ has\ a\ w.s.\ in\ G(\mathcal{X}/_{\langle\langle (x)_0, \ldots, (x)_{lh(x) \dot{-} 1}, y\rangle\rangle}))^{K}},$$

where the formula $\overline{\phi}$ is the complement of a formula ϕ obtained by interchanging 0 and 1, $\bigvee$ and $\bigwedge$. Observe that ϕ_{xy} is the direct translation of the statement of Lemma 12.

How our realizer constructs a strategy τ is illustrated by the following dialog.[4] ∀belard's answers should be read as values provided by arguments of our realizer, in other words, the environment. ∃loise's way of answering should be compared to the way our realizer returns values to the environment:

∃loise: Let me kick off the game by choosing, say $\tau_0 = \lambda n.0$.

∀belard: Then, my choice is $x = x_0$. By this, I can win in the game for $\phi_{x_0\,\tau_0(x_0)} (= \phi_{x_0 0})$.
▶ Now the formula is $CAC_{\phi} \vee \phi_{x_0 0}$.

∃loise: What is your answer when I play $x = x_0$ in the game for $\bigvee_x \bigwedge_y \overline{\phi_{xy}}$?

∀belard: In that case, I choose $y = y_0$. This can make you lose in the game for $\overline{\phi_{x_0 y_0}}$.
▶ Now the formula is $CAC_{\phi} \vee \phi_{x_0 0} \vee \overline{\phi_{x_0 y_0}}$.

∃loise: Since it is you who said that $\overline{\phi_{x_0 y_0}}$ is false, $\phi_{x_0 y_0}$ should be true, right?
(∀belard: Oops!) Then I backtrack my previous choice τ_0 and select
$\tau_1 := \lambda x.$ `if` $x = x_0$ `then` y_0 `else` $\tau_0(x)$.

∀belard: Well, $x = x_1$ is fine. This time, I can defeat you in the game for $\phi_{x_1 \tau_1(x_1)}$.

[3] Since only the original paper [5] employs $\phi_{i_0 j} \vee \bigvee_{i \in I - \{i_0\}} \phi_i$ instead of $\phi_{i_0 j} \vee \bigvee_{i \in I} \phi_i$ in the formulation of the classical validity, we shall adopt the formulation given in the subsequent papers.

[4] Strictly speaking, the following "game" is different from the concept defined so far; now that τ ranges over the (uncountable) set of all functions from `N` to `N`. In fact, by introducing conjunctions and disjunctions over `N` $\to$ `N`, cut-elimination theorem, which holds in the original version, is no longer true [2, Section 2.3]. It serves only as an explanation of the behavior.

▶ Now the formula is $CAC_\phi \vee \phi_{x_0 0} \vee \overline{\phi_{x_0 y_0}} \vee \phi_{x_1 0}$.

∃loise: If I choose $x = x_1$ in the game for $\bigvee_x \bigwedge_y \overline{\phi_{xy}}$, what is your choice?

∀belard: Again that question!? $\cdots$ (sigh). $y = y_1$ is the best option; I will win in the game for $\overline{\phi_{x_1 y_1}}$.

▶ Now the formula is $CAC_\phi \vee \phi_{x_0 0} \vee \overline{\phi_{x_0 y_0}} \vee \phi_{x_1 0} \vee \overline{\phi_{x_1 y_1}}$.

∃loise: Wait a minute. It means $\phi_{x_1 y_1}$ should be true, doesn't it. (∀belard: Oh no!) I do not have to stick to my previus move any more; let me choose $\tau_2 := \lambda x.\, \texttt{if } x = x_1 \texttt{ then } y_1 \texttt{ else } \tau_1(x)$.

$\vdots$

Both players continue playing in this way and, at each round, ∃loise updates a strategy τ using ∀belard's previous answers x_i and y_i. If ∀belard decides his move on *finitely much information* from the move of ∃loise, then, for some n and m with $n < m$, his choice for x in the n-th and m-th round will be the same one: $x_n = x_m$. (This assumption on ∀belard comes true in $\mathscr{P}$ due to continuity). Observe that, at that point, the formula is $CAC_\phi \vee \cdots \vee \overline{\phi_{x_n y_n}} \vee \cdots \vee \phi_{x_m y_n}$.

Step 2: derive the determinacy

Our realizer at last witnesses the determinacy. Let us continue employing the terminology of Coquand's game.

∃loise plays the game for the formula $CAC_\phi \vee \cdots \vee \overline{\phi_{x_n y_n}} \vee \cdots \vee \phi_{x_m y_n}$. Since either $\phi_{x_m y_n}$ or $\overline{\phi_{x_n y_n}}$ is true, ∃loise can certainly win by playing the games for $\phi_{x_m y_n}$ and $\overline{\phi_{x_n y_n}}$ alternately. If it turns out that $\phi_{x_m y_n}$ is true, the realizer concludes that τ satisfies $\bigwedge_x \phi_{x\tau(x)}$. This is because τ has been constructed so that $\phi_{x\,\tau(x)}$ holds for all possible moves x of ∀belard. With the help of the property OP^K, which is verified by just a simple unbounded search for a modulus, τ is understood as a winning strategy for Player II, in other words, $\bigwedge_x \overline{\chi(x * \tau) = \mathtt{1}}$ is verified. (Recall the proof of Theorem 11 here: the construction of a winning strategy τ for Player II is conducted without appealing to OP. In other words, OP has *nothing* to do with the construction of τ—the role of OP is to confirm that if a $\tau : \mathtt{N} \to \mathtt{N}$ satisfies $\bigwedge_x \phi_{x\tau(x)}$, then it is indeed a required one). On the other hand, when $\overline{\phi_{x_n y_n}}$ is true, our realizer tries to disprove the non-existence of a winning strategy for Player I by actually constructing a winning strategy for Player I as in the proof of Lemma 12 and, as a consequence, $\bigvee_\sigma \bigwedge_y \chi(\sigma * y) = \mathtt{1}$ is verified. Neither ∀belard nor ∃loise knows which player—Player I or Player II—is shown to have a winning strategy. This is because we cannot calculate the modulus m internally (cf. the previous section).

In summary, our realizer behaves as follows: It first constructs a strategy τ for Player II *not* by choosing values herself *but* by making use of ∀belard's returns $x_0, y_0, x_1, y_1, \ldots$ against her attempt at exposing falsehood. When a good approximation is made, it either verifies that τ is indeed a winning strategy for Player II with the help of OP, or shifts the blame to the assumption that Player I has no winning strategy.

The behavior of our realizer reflects the proof of Theorem 11: Since we proved that theorem by constructing a winning strategy for Player II under the assumption that Player I does not have a winning strategy, the resulting realizer constructs a winning strategy for Player II. If we change the proof so that it constructs a winning strategy for Player I assuming that there is no winning strategy for Player II, the corresponding realizer will try to construct a winning strategy for Player I. One future work is, based on a game theoretical intuition, to build a realizer of AD^K that works *symmetrically*, in other words, a realizer which behaves in such a way that winning strategies for Player I and Player II are constructed alternately by backtracking. (see $(*)$—the formula itself is symmetric).

7 Future Work

As emphasized previously, our focus is on the computational content of the AD rather than on the set theoretical applications. Since, insofar as the author knows, there are not so much research on the computational aspect of AD, the author wishes more work would be conducted in this area. This paper will conclude with suggestions for future research:

Indirect approach to AD: Over the property determinacy, several axioms have been proposed and explored [8]. It would be interesting to see whether these variants are realizable. The following intriguing problem should also be addressed: To realize AD^K not in arithmetic but in stronger systems, e.g., in ZF set theory.

Direct approach to AD: Krivine's classical realizability is a machinery which enables us to extract the computational content directly from second order classical logic. All axioms of ZF set theory are realizable in that framework [12]. Moreover, by adding the quote (or clock) instruction to the calculus, both CAC and DC become realizable [13]. It would be interesting to ask, for realizing AD, what kind of instruction we should add to the calculus? What instruction is indispensable? If we can realize AD with some instructions, this technology will attract more attention of set theorists. This is because Krivine's classical realizability yields a new model of ZF+DC+AD, if AD is realizable.

Consistency in arithmetic: In ZF set theory, (full) AC contradicts AD [6]. This is because a well-ordering of the set of all strategies, the existence of which follows from an equivalent of AC, enables us to build a non-determined pay-off set by means of transfinite induction. It seems hard to adjust that proof to the setting of arithmetic. Does full AC and AD contradict in HA^ω_c? Or, does $AC(\mathtt{N} \to \mathtt{N}, \mathtt{N})$ refute AD in HA^ω_c?

Acknowledgements I thank my master thesis supervisors Kazushige Terui and Masahito Hasegawa for lots of helpful comments, Stefano Berardi for explaining the underlying ideas of his joint work [2], and Makoto Tatsuta for drawing my attention to [1]. Finally, I would like to thank the anonymous referee who made a lot of useful suggestions.

A The Realizer of the Negative Translation of Open Determinacy

$\lambda\chi\theta\zeta\eta.\Phi P_2(\zeta,\eta,\theta)H_2(\chi)[\,]$ ® $\forall\chi\ \{OP(\chi) \Rightarrow Det(\chi)\}^K$, where

$$\Phi PHL := P\,(\mathtt{fun}\ L,\ \lambda x.\ \mathtt{rea}\ L\ x\ \ (\lambda x'x''.\ Hx\lambda\langle y,z\rangle.\Phi PH\,((x,y,z):L)))\,;$$

$$\mathtt{fun}\ [(x_0,y_0,z_0),\dots,(x_n,y_n,z_n)]\ x\ \ := y_i\ (\text{when } \exists\ i \le n\ (x = x_i)),\ \ := \mathtt{0}\ (\text{otherwise});$$

$$\mathtt{rea}\ [(x_0,y_0,z_0),\dots,(x_n,y_n,z_n)]\ x\ a := z_i\ (\text{when } \exists\ i \le n\ (x = x_i)), := a\ (\text{otherwise});$$

$$H_2(\chi) := \lambda sh.h\langle \mathtt{0},\ \lambda\langle w_1,w_2\rangle v.\ \mathtt{if}\ (lh(s)\text{ is odd})$$
$$\mathtt{then}\ c_1\chi s(\lambda q.q())H_1(h)\ P_1(s,w_2)\ \mathtt{else}\ w_1(\lambda e.\ \mathtt{Dummy})\rangle;$$

$$P_2(\zeta,\eta,\theta) := \lambda\langle\tau,q\rangle.\eta\langle\tau,\lambda x\xi.\theta(x*\tau)\xi(\lambda\langle m,u\rangle.Q_1(\zeta,q,x*\tau,m)Q_2(u,x*\tau))\,\rangle;$$

$$H_1(h) := \lambda m'z'.h\langle m',\ \lambda zx.xz'\rangle;$$

$$P_1(s,w_2) := \lambda\langle\sigma,p\rangle.w_2\,(\lambda r.\ r\langle F'(\sigma,s),\ \lambda y'.\ p\ y'(\mathtt{0})\ \mathtt{shift}(y')\rangle)\,;$$

$$c_1 := \lambda\chi suh'p'.\mathtt{if}\ (lh(s)\text{ is odd})\ \mathtt{then}\ \Phi p'h'[\,]\ \mathtt{else}\ u(\lambda r.\mathtt{Dummy});$$

$$Q_1(\zeta,q,x*\tau,m) := \mathtt{Rec}\ \zeta\,(\lambda nz.\ q\ \langle\langle x*\tau\,(\mathtt{0}),\dots,x*\tau\,(\mathtt{2}n)\rangle\rangle\langle\lambda d.\ d(),\ z\rangle)\,m+\mathtt{1};$$

$$Q_2(u,x*\tau,m) := \lambda l.\ l\langle\lambda j.\ \mathtt{0},\lambda y.\ u\ F(x*\tau,m,y)\ (\lambda k.k())\rangle;$$

$$F(x*\tau,m,y)(n) := \lambda n.\mathtt{if}\ n \le \mathtt{2}m+\mathtt{1}\ \mathtt{then}\ x*\tau\ (n)\ \mathtt{else}\ ((\lambda n.\mathtt{0})*y)\ (n \mathbin{\dot{-}} \mathtt{2}m \mathbin{\dot{-}} \mathtt{2});$$

$$F'(\sigma,s)(n) := \begin{cases} (s)_{lh(s)\dot{-}1} & (n = \langle\langle\rangle\rangle) \\ \sigma(m)\langle\langle k_0,\dots,k_{2j+1}\rangle\rangle & (n = \langle\langle (s)_{lh(s)\dot{-}1}, m, k_0,\dots,k_{2j+1}\rangle\rangle) \\ \mathtt{0} & (\text{else}) \end{cases}.$$

References

1 M.J. Beeson: Foundations of constructive mathematics. A series of modern surveys in mathematics, Springer Verlag, pp. xxiii+466, 1985

2 S. Berardi, M. Bezem and Th. Coquand: On the computational content of the axiom of choice. J. Symbolic Logic vol. 63, no. 2, pp. 600–622, 1998

3 U. Berger and P. Oliva: Modified bar recursion and classical dependent choice. Lecture Notes in Logic vol. 20, pp. 89–107, 2005

4 U. Berger and P. Oliva: Modified bar recursion. Math. Structures Comput. Sci. vol. 16, issue 2, pp. 163–183, 2006

5 Th. Coquand: A semantics of evidence for classical arithmetic. J. Symbolic Logic vol. 60, no. 1, pp. 325–337, 1995

6 D. Gale and F.M. Stewart: Infinite games with perfect information. In: H.W. Kuhn and A.W. Tucker (eds.) Contributions to the theory of games, vol 2. Ann. Math. Studies 28. Princeton Univ. Press, pp. 245–266, 1953

7 N. Goodman: Intuitionistic arithmetic as a theory of constructions. PhD thesis, Stanford University, pp. 222, 1968

8 A. Kanamori: The higher infinite, second edition. Springer monographs in mathematics, pp. xxii+536, 2003

9 A.S. Kechris: The axiom of determinacy implies dependent choices in $\mathbf{L}(\mathbb{R})$. J. Symbolic Logic vol. 49, no. 1, pp. 161–173, 1984

10 U. Kohlenbach: Theory of majorizable and continuous functionals and their use for the extraction of bounds from non-constructive proofs: effective moduli of uniqueness for best approximations from ineffective proofs of uniqueness (German). PhD thesis, Goethe-Universität Frankfurt, pp. xxii+278, 1990

11 U. Kohlenbach: Applied proof theory: Proof interpretations and their use in mathematics. Springer monographs in mathematics, pp. xix+532, 2008

12 J.-L. Krivine: Typed lambda-calculus in classical Zermelo-Frænkel set theory. Arch. Math. Logic 40, no. 3, pp. 189–205, 2001

13 J.-L. Krivine: Dependent choice, 'quote' and the clock. Theoret. Comput. Sci. vol. 308, pp. 259–276, 2003

14 J.-L. Krivine: Realizability algebras II: new models of ZF+DC. http://www.pps.jussieu.fr/~krivine/articles/R_ZF.pdf

15 D.A. Martin: Borel determinacy. Ann. Math. vol. 102, no. 2, pp. 363–371, 1975

16 A. Montalbán and R.A. Shore: The limits of determinacy in second order arithmetic. Proc. London Math. Soc. vol. 104, part 2, pp.223–252, 2012

17 J. Mycielski and H. Steinhaus: A mathematical axiom contradicting the axiom of choice. Bulletin de l'Académie Polonaise des Sciences, Série des Sciences Mathématiques, Astronomiques et Physiques 10, pp.1–3, 1962

18 J. Mycielski and S. Swierczkowski: On the Lebesgue measurability and the axiom of determinateness. Fundamenta Mathematicae vol. 54, pp. 67–71, 1964

19 M.H. Sørensen and P. Urzyczyn: Lectures on the Curry-Howard isomorphism. Stud. Logic Found. Math. 149, Elsevier, pp. xiv+442, 2006

20 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 (ed.) Recursive function theory: Proc. Symposia in Pure Mathematics, vol. 5, American Mathematical Society, Providence, Rhode Island, pp. 1–27, 1962

21 A.S. Troelstra: Realizability. In: S.R. Buss (ed.) Handbook of proof theory. Stud. Logic Found. Math. 137, Elsevier, pp. 407–473, 1998

Church-Rosser Properties of Normal Rewriting*

Jean-Pierre Jouannaud[1,2] and Jianqi Li[3,4]

1 **INRIA-LIAMA, Beijing, China**
2 **Software Chair, Tsinghua University, Beijing, China**
3 **School of Software, Tsinghua University, Beijing, China**
4 **Tsinghua National Laboratory for Information Science and Technology, Beijing, China**

Abstract

We prove a general purpose abstract Church-Rosser result that captures most existing such results that rely on termination of computations. This is achieved by studying *abstract normal rewriting* in a way that allows to incorporate positions at the abstract level. New concrete Church-Rosser results are obtained, in particular for higher-order rewriting at higher types.

1998 ACM Subject Classification F4.2. Grammars and Other Rewriting Systems

Keywords and phrases abstract normal rewriting, Church-Rosser property

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.350

1 Introduction

Background. Rewrite rules have been used in mathematics and computer science for ages. Orienting an equation into a rewrite rule is most convenient when the obtained set of rules is terminating, since many other properties, such as "unique normal form", become then decidable. Orienting all equations at hand into rewrite rules is not always possible, forcing us to distinguish a subset of rules from the subset of remaining equations. Consider the set: $Inv : x + x^{-1} = 0, \quad Z : x + 0 = x, \quad A : (x + y) + z = x + (y + z), \quad C : x + y = y + x.$ Termination forbids orienting C, and orienting A contradicts termination in C-congruence classes: it becomes necessary to distinguish rules from equations. Having A, C, Z as equations allows for cheaper pattern-matching and unification than A, C alone, but ACZ-congruence classes are infinite, raising problems. A winning schema is to restrict computations to terms in Z normal form modulo AC [4, 11, 16]. Rewriting with Inv then operates modulo ACZ, but on terms in Z modulo AC normal form.

Goal. In this paper, we investigate rewriting with a set of rules R (Inv in our example) modulo a set itself made of a set of rules S (Z) and a set of equations E (AC). Another example is Nipkow's higher-order rewriting, for which R is the user's set of higher-order rules, S corresponds to β-reduction and η-expansions which are built, together with α-conversion (E), in the rewriting mechanism via higher-order pattern-matching. Higher-order rewriting is indeed our main target, and our interest is in checking its confluence under some termination assumption.

State of the art. Rewriting with rules only (when E and S are both empty) is called *plain rewriting*. Confluence of plain rewriting reduces, under a termination assumption, to the

* INRIA Project FORMES, Laboratory of Formal Methods, Institute of Software Theory and System, Tsinghua University. This work was supported by the Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation 2011-9, Chinese National 973 Plan grant 2010CB328003, the NSFC-ANR grant 60811130468 and NSFC grant 60903030.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 350–365
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

joinability of its *critical pairs*, which are minimal divergent computations obtained by unifying lefthand sides of rules at non-variable subterms [14]. When rewriting terms with mixed sets of rules and equations, confluence must be replaced by the more general *Church-Rosser* property [10]. There are several approaches corresponding to different definitions of rewriting in presence of equations. In the first three, S is empty. Huet uses plain rewriting with R and joinability modulo E, but must assume strong linearity conditions on E and R [9]. Lankford uses plain rewriting with R in equivalence classes of terms modulo E [15]. His *class rewriting* may require searching the entire class of a term to be rewritten. The de facto standard introduced by Peterson and Stickel, *rewriting modulo*, uses instead plain rewriting with all possible E-variants of the instances of rules in R, which is implemented via E-pattern matching [21]. All three approaches require finiteness of E-equivalence classes [10], see also [2]. To remove this assumption, Marché defined *normalized rewriting*, a complex schema where the rules in S are confluent modulo E in Stickel's sense and a term is first normalized with S modulo E before being rewritten with R modulo E [16]. Normalized rewriting assumes termination of $R \cup S$ in E-equivalence classes. Nipkow uses a subtle variation of the latter requiring termination of R modulo $E \cup S$, in which simply typed higher-order terms *in normal form* for S modulo E are rewritten modulo $S \cup E$ (using higher-order pattern-matching) with some set of higher-order rules which lefthand sides are patterns *à la* Miller [17].

In all these approaches, the Church-Rosser property is reduced, via a termination assumption, to the joinability modulo E of some critical pairs. This reduction is doable provided rewriting a term does not change its structure before it is pattern-matched. This is the case of all definitions but that of class rewriting and of normalized rewriting. In the latter case, the analysis of divergent computations leads to a notion of critical pair which is quite complex. On the other hand, Nipkow's variant, which we call *normal rewriting*, has not lead yet to a general Church-Rosser test.

Very early on, the rewriting community has developed an abstract approach to the analysis of the confluence and Church-Rosser properties [23], or to similar results such as the finite development theorem in lambda calculus [8, 22]. This trend has been very successful for orthogonal systems and extensions thereof, but has not yet delivered all its promises for the case of terminating systems for which the presence of critical pairs requires a slightly different analysis. A tentative to capture both cases within a unique framework, decreasing diagrams, has been carried out by van Oostrom [20]. If this work has been very successful at the abstract level of relations, it has not yet bared fruits in the more difficult case of concrete relations over terms.

Contributions. Our main contribution is a careful investigation of an abstract rewriting relation, *normal rewriting*, with a set of rules R modulo a pair (S, E), which set of rules S is itself convergent modulo the set of equations E. Normal rewriting is then defined as a compositional paradigm: normalization in the $S \cup E$-structure is *modulo* E, while normalization in the $(R \cup (S \cup E))$-structure is modulo $S \cup E$ on S modulo E normal forms. We then reduce the abstract Church-Rosser property to properties of *abstract critical rewriting patterns.* Our abstract treatment departs from the usual practice by introducing a setting of ternary relations on an abstract set of terms and an abstract set of positions, which we call *abstract positional rewriting.* This setting allows us to develop our notion of abstract normal rewriting, and then to study the reduction from its Church-Rosser property to the critical rewriting patterns quite smoothly, therefore solving the aforementioned problem.

Our second contribution, an application of the above results, is a careful investigation of the Church-Rosser properties of first-order normal rewriting first, then of various variants of Nipkow's higher-order normal rewriting. These applications are direct reductions from abstract critical patterns to concrete critical pairs, which exploit the term structure explicitly. A major strength of our result is that it allows to capture the existing notions of rewriting in both the first-order and higher-order cases. While its application to the first-order case yields limited new results, it allows us in the higher-order case to overcome all typing restrictions imposed in Nipkow's work.

While it may seem that rewriting is a subject beaten to death, recent work has shown that confluence of normal rewriting is indeed at the heart of important problems such as [1], hence making the present contributions very timely. An early attempt appeared in [12].

Surveys are [7, 23, 22] for first and higher-order rewriting and [5] for typed lambda calculus.

Organization. Section 2 investigates the Church-Rosser properties of abstract normal rewriting. First-order rewriting is developed in Section 3 which ends with a study of our introductory example. After a brief introduction of simply-typed lambda-calculi, higher-order rewriting at simple types is studied in section 4.1 and at higher-types in Section 4.2. Various definitions of formal derivation serve illustrating the results. Concluding remarks come in Section 5.

2 Normal rewriting

We introduce the notion of normal rewriting on an abstract set, investigating then its Church-Rosser properties. Our treatment differs from similar attempts by introducing abstract positions from the start. This allows us to carry out the investigation of the abstract Church-Rosser property much further, and reduce it to the joinability of abstract critical patterns via clean, technically simpler proofs than in the current literature. The ability to analyze the Church-Rosser property at the abstract level up to the analysis of critical pairs is the key to the obtention of our main result, Theorem 2.5, which proof is quite delicate.

2.1 Abstract positional rewriting

2.1.1 Abstract terms and positions

In the entire Section 2, we assume given two abstract sets:

- $\mathcal{T}$ which elements are called *terms* ;
- $\mathcal{P}$ which elements are called *positions*, equipped with a partial order $>_{\mathcal{P}}$ and a minimum Λ.

A *domain* P_p is any set of positions $p' \geq_{\mathcal{P}} p$ such that $p' \in P_p$ and $p' \geq_{\mathcal{P}} q \geq_{\mathcal{P}} p$ implies $q \in P_p$. A domain is meant to be the set of non-variable positions of some lefthand side of rule in a term.

Lexicography: we shall use the letters s, t, u, v, w for terms and p, q for positions, the notations P_p and Q_q for *domains*, and the notation $\mathcal{D}_{\mathcal{P}}$ for the set of domains over $\mathcal{P}$. We write $p\#q$ for incomparable positions p, q, and $q >_{\mathcal{P}} P_p$ for $q \geq_{\mathcal{P}} p$ and $\forall p' \in P_p \,.\, p' \not\geq_{\mathcal{P}} q$. We will freely use the following key property of domains, which first 3 cases are called respectively *"disjoint case"*, *"critical case"* and *"ancestor case"* in the litterature:

$$\forall p \in \mathcal{P}\ \forall P_p \in \mathcal{P}.(q\#p \vee q \in P_p \vee q >_{\mathcal{P}} P_p \vee p >_{\mathcal{P}} q).$$

2.1.2 Relations

We use the notation $u\xrightarrow{P_p}v$ with $u,v \in \mathcal{T}$, $P_p \in \mathcal{D}_{\mathcal{P}}$ for an arbitrary ternary relation in $\mathcal{T}{\times}\mathcal{D}_{\mathcal{P}}{\times}\mathcal{T}$. We may omit any of u, v, P_p, in which case they are existentially quantified. We also write for short $u\xrightarrow{p}v$, where p is understood as the minimum of some domain P_p, or $u\xrightarrow{p\in P}v$ to indicate the property P that the position p should satisfy, or even $u\xrightarrow{P}v$, with the same meaning. In practice, P will usually be the property $(\geq_{\mathcal{P}}\ q)$ for some given q, characterizing the set of positions $\{p \mid p{\geq_{\mathcal{P}}}\ q\}$.

The relation $\longrightarrow$ is *reflexive* if $\forall u \in \mathcal{T}\,.\, u\longrightarrow u$, *symmetric* if for all s,t,p, $s\xrightarrow{p}t$ iff $t\xrightarrow{p}s$, and *transitive* if for all u,v,w s.t. $u\longrightarrow v\longrightarrow w$ then $u\longrightarrow w$. Given $\longrightarrow$, we write $\longleftarrow$ for its inverse, $\longleftrightarrow$ for its symmetric closure, $\xrightarrow{+}$ for its transitive closure and $\xrightarrow{*}$ for its reflexive, transitive closure (domains become lists thereof in the latter two closures).

The term t is a *successor below* $p \in \mathcal{P}$ of s for $\longrightarrow$ if $s\xrightarrow{\geq_{\mathcal{P}}\ p}t$, and s is in *normal form below* p if it has no successor below p. We denote by $s{\downarrow}^p$ the term in normal form below p such that $s\xrightarrow{(\geq_{\mathcal{P}}\ p)^*}s{\downarrow}^p$. We omit the mention *below* p and write $s{\downarrow}$ whenever $p = \Lambda$. A term s is *strongly normalizing* below p for $\longrightarrow$ iff it is in normal form below p, or if otherwise all its successors are themselves strongly normalizing below p. The relation $\longrightarrow$ is *terminating* if all terms are strongly normalizing below Λ.

2.1.3 Rewriting modulo

▶ **Definition 2.1** (Rewriting modulo). Given two relations $\xrightarrow{p}_X$ and $\underset{Z}{\overset{q}{\longleftrightarrow}}$ (assumed symmetric) on $\mathcal{T} \times \mathcal{D}_{\mathcal{P}} \times \mathcal{T}$, *rewriting with* X *modulo* Z *at* p is defined as:

$$\xrightarrow{p}_{X_Z} \quad := \quad \underset{Z}{\overset{(\geq_{\mathcal{P}}\ p)^*}{\longleftrightarrow}}\ \xrightarrow{p}_X$$

Z and *modulo* Z are omitted if $Z = \emptyset$. The symmetric closure of Z should be understood in case Z is not symmetric.

The beauty of rewriting modulo a theory lies in the assumption that equality steps can only occur below the rewriting position: at the term level, rules can be fired by pattern matching lefthand sides modulo the theory, avoiding searching the equivalence class of the term to be rewritten.

Rewriting modulo is assumed to satisfy the *commutation* (*) and *joinability* (**) properties:

$$(*) \qquad {}_X\xleftarrow{p}\ \xrightarrow{q}_Y \subseteq \xrightarrow{p}_Y\ {}_X\xleftarrow{q} \qquad \text{if} \quad p\#q$$

$$(**) \qquad {}_X\xleftarrow{P_p}\ \xrightarrow{q}_Y \subseteq \xrightarrow{(>_{\mathcal{P}}\ p)^*}_Y\ {}_X\xleftarrow{q}\ \ {}_Y\xleftarrow{(>_{\mathcal{P}}\ P_p)^*} \qquad \text{if} \quad q{>_{\mathcal{P}}}\ P_p$$

Note that (*) is a particular case of (**), apart from the condition which could be generalized. We prefer to distinguish these two axioms because they lead to different calculations in the proof of our main result. Note also the absence of a “monotonicity” axiom allowing to move rewrites up, which is achieved by changing the position incorporated to the rewrite.

2.1.4 Normal rewriting

▶ **Definition 2.2.** A *normal abstract rewriting system* (NARS) is a tuple $(\mathcal{T}, \mathcal{P}, R, S, E)$, where $\mathcal{T}$ and $\mathcal{P}$ are (possibly omitted) abstract sets of terms and positions, while $\longrightarrow_R$, $\longrightarrow_S$

and $\xleftrightarrow[E]{}$ are ternary relations on $\mathcal{T} \times \mathcal{D}_{\mathcal{P}} \times \mathcal{T}$ called respectively *rewriting* R, *simplification* S and *equality* E, the latter being supposed symmetric, such that:

(i) the relation S_E, called E*-simplification*, is *Church-Rosser modulo* E *below* any position p:

$$s \xleftrightarrow[S \cup E]{(\geq_{\mathcal{P}}\ p)^*} t \quad \text{iff} \quad s \xrightarrow{(\geq_{\mathcal{P}}\ p)^*}{}_{S_E} \xleftrightarrow[E]{(\geq_{\mathcal{P}}\ p)^*} {}_{S_E}\xleftarrow{(\geq_{\mathcal{P}}\ p)^*} t$$

(ii) $\longrightarrow_{R_{SE}} \cup \longrightarrow_{S_E}$ is E*-terminating*: the relation $s \succ t$ iff $s =_E (\longrightarrow_{R_{SE}} \cup \longrightarrow_{S_E}) =_E t$ is well-founded, where R_{SE} stands for *rewriting* with R *modulo* $S \cup E$

(iii) operating on terms in $\downarrow^p_{S_E}$ normal form, *normal rewriting at q below p with R modulo* (S, E) is defined as $\quad s \xrightarrow{p}_{R_{S_E\downarrow}} t \; = \; s \xrightarrow{q \geq_{\mathcal{P}}\ p}_{R_{SE}} u\downarrow^p_{S_E} t.$

The main reason for orienting simplifiers is indeed to bypass two fundamental assumptions of rewriting modulo: *termination* of $\longrightarrow_R$ in *finite* $S \cup E$-equivalence classes. The finiteness assumption will disappear, while the termination assumption will be weakened.

Normal rewriting maintains S_E normal forms below p, and satisfies (*, **). Nipkow's definition assumes $p = \Lambda$. Our definition is relative to a given position p, as are all our rewriting notions. This definition is actually flexible. We might have chosen to operate on terms in$\downarrow^q_{S_E}$-normal form or to normalize the result up to position Λ. These changes would not impact our result for which p is Λ.

Notations. We use, possibly omitting the upper-index p: $\quad s \xrightarrow{\downarrow} t$ for $s \xrightarrow{*} t$ with $t = t\downarrow$, $s\downarrow^p$ for $s\downarrow^p_{S_E}$, $\quad s\Downarrow^p$ for $(s\downarrow^p_{S_E})\downarrow^p_{R_{S_E\downarrow}}$, $\quad =_E$ for the equivalence $\xleftrightarrow[E]{*}$, $\xleftrightarrow[SE]{}$ (SE) for $\xleftrightarrow[S]{} \cup \xleftrightarrow[E]{}$, and $\xleftrightarrow[RSE]{}$ (RSE) for $\xleftrightarrow[R]{} \cup \xleftrightarrow[S]{} \cup \xleftrightarrow[E]{}$.

2.2 Church-Rosser properties of NARS

Our goal here is to reduce the Church-Rosser property of a terminating NARS to local properties of the rewrite relations involved that can be checked for concrete rewrite relations on terms. After introducing key notations, we recall some Church-Rosser notions and introduce our local properties which correspond *exactly* to critical patterns.

▶ **Definition 2.3.** We define:
convertibility of a pair (s, t) below p as $s \xleftrightarrow[RSE]{(\geq_{\mathcal{P}}\ p)^*} t$;
normal joinability of a convertible pair (s, t) below p as $s\Downarrow^p \xleftrightarrow[E]{(\geq_{\mathcal{P}}\ )^*} t\Downarrow^p$
normal Church-Rosser as the normal-joinability of all convertible pairs ;
local peaks (resp. *cliffs*) as triples (s, u, t) s.t. $s \xleftarrow{p} u$ (resp. $s \xleftrightarrow[E]{p} u$) and $u \xrightarrow{q} t$;
joinability below p of a triple (s,u,t) or a pair (s,t) as $s \xrightarrow{(\geq_{\mathcal{P}}\ p)^*}{}_{S_E \cup R_{SE}} \xleftrightarrow[E]{(\geq_{\mathcal{P}}\ p)^*} {}_{S_E \cup R_{SE}}\xleftarrow{(\geq_{\mathcal{P}}\ p)^*} t.$

▶ **Definition 2.4** (Critical local peaks and cliffs).

(i) *critical rewrite peak* $\quad v\ {}_R\xleftarrow{P_p} u \xrightarrow{q}_{R_{SE}} w \quad$ s.t. $q \in P_p$ and $u = u\downarrow^p_{S_E}$

(ii) *critical rewrite cliff* $\quad s \xleftrightarrow[E]{p} u \xrightarrow{q}_{R_{SE}} t \quad$ s.t. $q \in P_p \setminus \{p\}$

(iii) *critical simplification peak* $\quad v\ {}_R\xleftarrow{P_p} u \xrightarrow{q}_{S_E} w \quad$ s.t. $q \in P_p$

(iv) *critical simplification cliff* $\quad v\ {}_S\xleftarrow{p} u \xrightarrow{q}_{R_{SE}} w \quad$ s.t. $q \in P_p \setminus \{p\}$ and $u = u\downarrow^q_{S_E}$

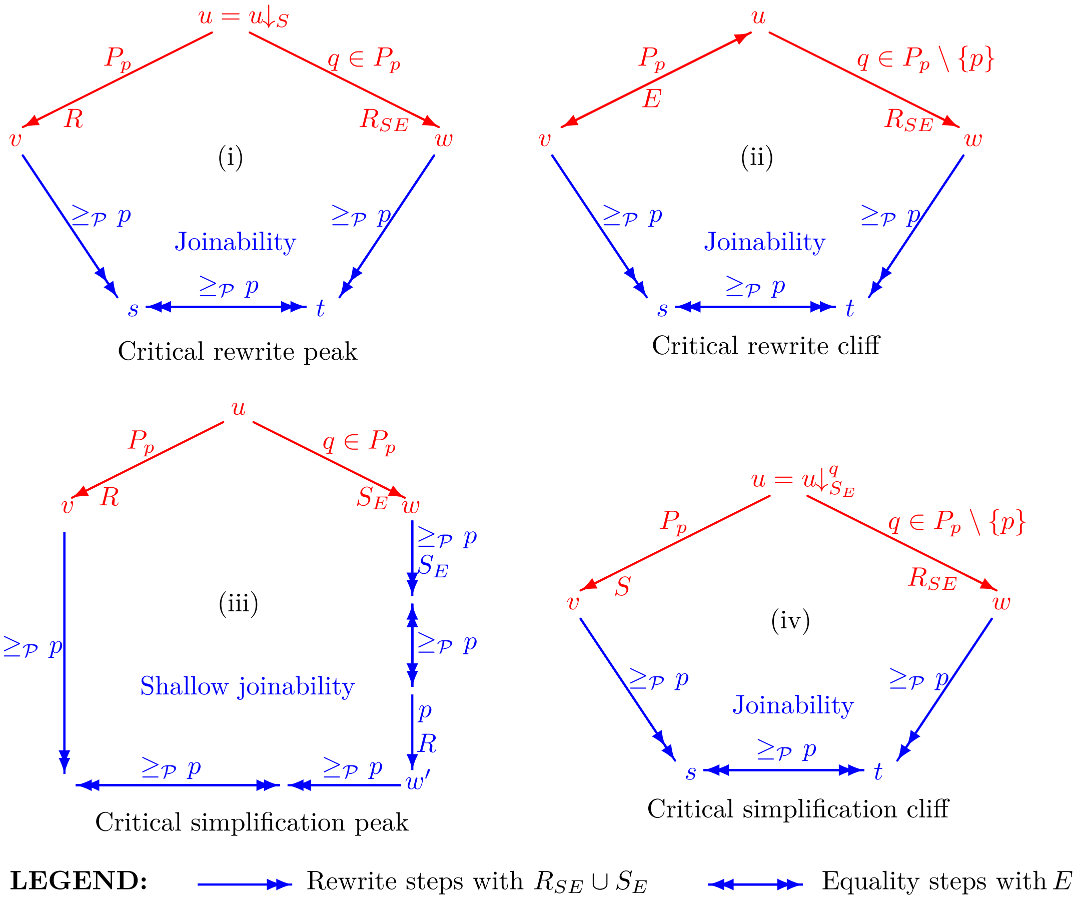


Figure 1 Abstract critical peaks.

Unlike standard practice, our local properties are *critical* in that they specialize as the usual notions of critical peaks at the concrete level. Criticality follows from two observations: (i) each peak must satisfy the condition $q \in P_p$ (or $q \in P_p \setminus \{p\}$), which implies the existence of an overlap; (ii) all local properties use a plain step from u at p, this is crucial to compute a critical pair by equating two different calculations of $u|_q$, requiring the absence of equational steps above q.

Our figures also show the properties expected from the critical peaks: joinability below p for all except critical simplification peaks, for which *shallow joinability* is required.

We can now state and prove our key abstract Church-Rosser result, without help of any intermediate step involving non-local peaks corresponding to local confluence, local coherence and the like: despite the complex hierarchical structure of a NARS, our abstract approach allows us to reduce directly the Church-Rosser property to the joinability of the critical peaks introduced above.

▶ **Theorem 2.5.** *A NARS is normal Church-Rosser if its critical (i) rewrite peaks, (ii) rewrite cliff, (iii) simplification cliffs are joinable, and its (iv) critical simplification peaks are shallow joinable.*

Proof. By definition of rewriting modulo, $\xleftrightarrow[RSE]{*} = \xleftrightarrow[R_{SE} \cup S_E \cup E]{*}$. The proof is by induction on conversions $\xleftrightarrow[R_{SE} \cup S_E \cup E]{*}$. Conversions are interpreted by multisets which elements are pairs of terms (u, v) written as $u\,v$, and are therefore compared in the well-founded order

$\succ\!\!\succ := ((\succ)_{lex})_{mul}$. Each step in a conversion contributes with one or two pairs: a step $s \longrightarrow_{R_{SE}} t$ with $s\,t$; a step $s \longrightarrow_{S_E} t$ with $t\,s$; a step $s \underset{E}{\longleftrightarrow} t$ with both $s\,t$ and $t\,s$.

By definition, the shape of a normal joinable conversion $s \overset{*}{\underset{R_{SE} \cup S_E \cup E}{\longleftrightarrow}} t$ must be of the form $s \xrightarrow{\downarrow}_{S_E} (\longrightarrow_{R_{SE}} \xrightarrow{\downarrow}_{S_E})^* \overset{*}{\underset{E}{\longleftrightarrow}} ({}_{R_{SE}}\!\longleftarrow {}_{S_E}\!\xleftarrow{\downarrow})^* {}_{S_E}\!\xleftarrow{\downarrow} t$. It follows that conversions which are not already normal joinable must contain one of the six (up to symmetry) following patterns: $u \xrightarrow{p}_{R_{SE}} v$ with $u \neq u{\downarrow}^p$, $v \; {}_{R_{SE}}\!\xleftarrow{p} u \xrightarrow{q}_{R_{SE}} w$ with $u = u{\downarrow}^p$ and $u = u{\downarrow}^q$, $v \underset{E}{\longleftrightarrow} u \xrightarrow{p}_{R_{SE}} w$ with $u = u{\downarrow}^p$, $v \; {}_{S_E}\!\longleftarrow u \xrightarrow{p}_{R_{SE}} w$ with $u = u{\downarrow}^p$, $v \; {}_{S_E}\!\longleftarrow u \longrightarrow_{S_E} w$, and $v \underset{E}{\longleftrightarrow} u \longrightarrow_{S_E} w$. In each case, we provide a smaller conversion for (v, w) yielding a smaller conversion for (s, t), hence allowing us to conclude by induction. We are indeed rewriting conversions in the style of [3].

1. $v \; {}_{S_E}\!\longleftarrow u \longrightarrow_{S_E} w$, a local peak interpreted by $\{u\,v, w\,u\}$. By definition of a NARS, we have $v \xrightarrow{*}_{S_E} \overset{*}{\underset{E}{\longleftrightarrow}} {}_{S_E}\!\xleftarrow{*} w$, which interpretation contains pairs all (strictly) smaller than $u\,v$ (or $w\,u$).

2. $v \underset{E}{\longleftrightarrow} u \longrightarrow_{S_E} w$, a cliff interpreted by $\{v\,u, u\,v, w\,u\}$. By definition of a NARS, we have $v \longrightarrow_{S_E} v' \xrightarrow{*}_{S_E} \overset{*}{\underset{E}{\longleftrightarrow}} w' \; {}_{S_E}\!\xleftarrow{*} w$, which interpretation contains pairs all smaller than $v\,u$.

3. $u \xrightarrow{p}_{R_{SE}} v$ with $u \neq u{\downarrow}^p$, a step which interpretation is $\{u\,v\}$. By definition of normal form below p, $u \xrightarrow{q}_{S_E} w$ (hence $u \succ w$) for some w and $q \geq_{\mathcal{P}} p$. By definition of rewriting modulo, $w \longrightarrow_{R_{SE}} v$. The resulting conversion is interpreted by the smaller multiset $\{w\,u, w\,v\}$.

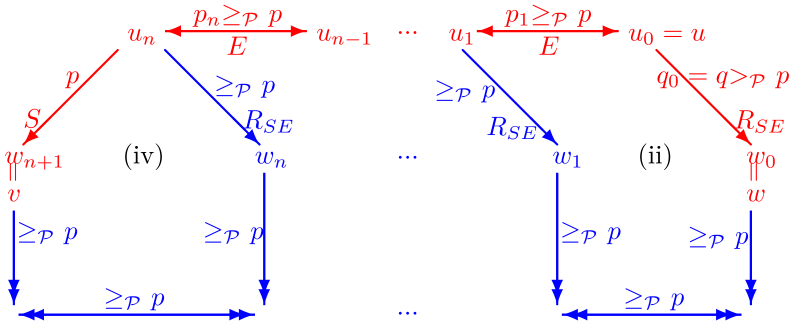
4. $v \overset{p}{\underset{E}{\longleftrightarrow}} u \xrightarrow{Q_q}_{R_{SE}} w$, a cliff interpreted by $\{v\,u, u\,v, u\,w\}$. We conclude by (*) if $p \# q$, definition of R_{SE} if $p \geq_{\mathcal{P}} q$, (**) if $p >_{\mathcal{P}} Q_q$ and assumption (ii) if $p \in Q_q \setminus \{q\}$. In all cases, the obtained proof is interpreted by pairs which are strictly smaller than $v\,u$ or $u\,w$.

5. $v \; {}_{S_E}\!\xleftarrow{P_p} u \xrightarrow{q}_{R_{SE}} w$, a peak interpreted by $\{v\,u, u\,w\}$. Thanks to the first case, we can assume that $u = u{\downarrow}^q$. There are therefore three possible cases:

– $p \# q$. Then $v \xrightarrow{q}_{R_{SE}} t \; {}_{S_E}\!\xleftarrow{p} w$ by (*), interpreted by $\{v\,t, t\,w\}$ smaller than $\{v\,u, u\,w\}$.

– $q >_{\mathcal{P}} P_p$. By (**), we get the (smaller) joinability proof $v \xrightarrow{(\geq_{\mathcal{P}}\ p)^*}_{R_{SE}} v' \; {}_{S_E}\!\xleftarrow{p} w' \; {}_{R_{SE}}\!\xleftarrow{(\geq_{\mathcal{P}}\ p)^*} w$.

– $q \in P_p \setminus \{p\}$. By definition of rewriting modulo, $v \; {}_{S}\!\xleftarrow{p} u_n \overset{p_n}{\underset{E}{\longleftrightarrow}} \cdots \overset{p_1}{\underset{E}{\longleftrightarrow}} u_0 = u$, with $\forall i \in [1..n]\ p_i \geq_{\mathcal{P}} p$. Note that all u_i are in S_E-normal form below q since this is true of u. We now show the existence of terms w_i such that $u_i \xrightarrow{q_i \geq_{\mathcal{P}}\ p}_{R_{SE}} w_i$, hence $u \succ w_i$ (requiring here an order on E-equivalence classes), and $\forall i \in [1..n]$ the pair (w_{i-1}, w_i) is joinable with steps which interpretation is made of pairs smaller than $\{u\,w\}$. Letting $w_0 = w$ and $q = q_0$, we use an induction on i: the case $i = 0$ is by assumption. Assuming the property up to $i - 1$, we proceed as follows: if $q_i \# p_{i+1}$, by (*) ; if $p_{i+1} \geq_{\mathcal{P}} q_i$, by definition of rewriting modulo ; if $q_i >_{\mathcal{P}} P_{p_{i+1}}$, by (**) ; and if $q_i \in P_{p_{i+1}}$, by assumption (ii). We close the diagram by definition of rewriting modulo if $q_n = p$, by assumption (iv) applied to the peak $v \; {}_{S}\!\xleftarrow{p} u_n \xrightarrow{q_n}_{R_{SE}} w_n$ if $q_n \in P_p$ (see the coming picture) and by property (**) if $q_n >_{\mathcal{P}} P_p$, yielding a smaller conversion each time.

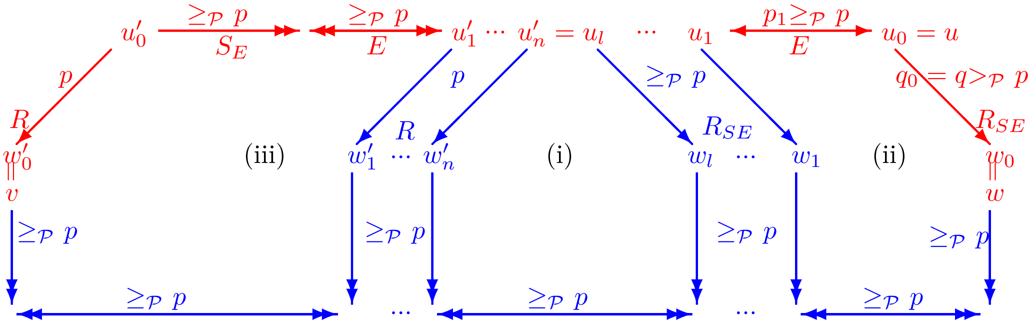


6. $v\ {}_{R_{SE}}\!\xleftarrow{P_p} u \xrightarrow{q}_{R_{SE}} w$. Thanks to the first case, we can assume wlog that u is in normal form below p and q, which comparison yields three cases up to symmetry:
– $p\#q$. Then $v \xrightarrow{q}_{R_{SE}} u'\ {}_{R_{SE}}\!\xleftarrow{p} w$ by (*), yielding a smaller conversion.
– $q{>_\mathcal{P}}\ P_p$. Property (**) yields a smaller conversion.
– $q \in P_p$. Then $v\ {}_R\!\xleftarrow{p} u'_0 \xleftrightarrow[SE]{(\geq_\mathcal{P}\ p)^*} u \xrightarrow{q}_{R_{SE}} w$ by definition of rewriting modulo, and u is in S_E-normal form below p. As shown on the picture below, we proceed in three steps.

First: we move $u'_0 \xrightarrow{p}_R$ from u'_0 to some u'_n in S_E-normal form s.t. $u'_n \xleftrightarrow[E]{(\geq_\mathcal{P}\ p)^*} u$ and $\forall i \in [1..n]$, $u'_{i-1} \xrightarrow{(\geq_\mathcal{P}\ p)^*}_{S_E} \xleftrightarrow[E]{(\geq_\mathcal{P}\ p)^*} u'_i \xrightarrow{p}_R w'_i$ and the pair (w'_{i-1}, w'_i) is joinable. Hence $u \longrightarrow_{R_{SE}} w'_i$ and therefore $u \succ w'_i$. The proof of the claim is by induction on i. If $u'_0 = u'_0{\downarrow}_{S_E}$, we are done with $n = 0$ and $w'_0 = w$. Otherwise $u'_0 \xrightarrow{\geq_\mathcal{P}\ p}_S u'_1$. By assumption (iii) (case shown on the picture) or property (**), $u'_0 \xrightarrow{(\geq_\mathcal{P}\ p)^*}_{S_E} \xleftrightarrow[E]{(\geq_\mathcal{P}\ p)^*} u'_1 \xrightarrow{p}_R w'_1$ and the pair (w'_0, w'_1) is joinable. S_E being Church-Rosser below p, $u'_1 \xrightarrow{(\geq_\mathcal{P}\ p)^*}_{S_E} \xleftrightarrow[E]{(\geq_\mathcal{P}\ p)^*} u$. Induction hypothesis applied to u'_1 concludes.

Second: let $u = u_0$, $\forall i \in [1..l]$, $u_{i-1} \xleftrightarrow[E]{\geq_\mathcal{P}\ p} u_i$ and $u_l = u'_n$. We proceed moving $u_0 \xrightarrow{\geq p}_{R_{SE}}$ from u_0 to u_l showing the existence of terms w_i such that: $\forall i \in [0..l]$, $u_i \xrightarrow{q_i \geq_\mathcal{P}\ p}_{R_{SE}} w_i$ and the pair (w_{i-1}, w_i) is joinable below p. This is done by induction on i. Case $i = 0$ is clear. The step case uses: if $p_1\#q_0$, assumption (*); if $p_1{\geq_\mathcal{P}}\ q_0$, the definition of rewriting modulo, hence $w_1 = w_0$; if $q_0{>_\mathcal{P}}\ p_1$, property (**) or assumption (ii) (the case shown on the picture). For all $i \in [0..l]$, the property $u \succ w_i$ follows from the fact that $u =_E u_i \longrightarrow_{S_E \cup R_{SE}} w_i$.

Third: we close the obtained local peak $w'_n\ {}_R\!\xleftarrow{p} u'_n = u_l \xrightarrow{q_l}_{R_{SE}} w_l$ thanks to property (**) or assumption (i) (case on the picture), using the fact that $u'_n = u_l$ is in S_E-normal form below p.



All steps in the resulting conversion $w'_0 \ldots w'_n \ldots w_l \ldots w_0$ are interpreted by pairs which we have shown to be all strictly smaller than $u\,w$, hence we are done. ◀

We are ready for applying our main result to a first-order, and then higher-order, concrete setting.

3 First-order rewriting systems

There are many versions of first-order rewriting, all covered by normal rewriting except class and normalized rewriting as discussed in introduction.

3.1 Terms and rules.

We denote by $\mathcal{T}(\mathcal{F}, \mathcal{X})$ the free algebra of *terms* generated from a signature $\mathcal{F}$ of function symbols and a denumerable set $\mathcal{X}$ of variables. We assume the usual definitions of terms, positions, and substitutions [7, 23], adopting notations from the former. We use $\mathcal{V}ar(t)$ for the set of variables of the term t, $\mathcal{P}os(t)$ for the set of positions in t, and $\mathcal{FP}os(t)$ for its set of non-variable positions, and $\cdot$ for concatenation of positions. The *subterm* of t at position p is denoted by $t|_p$, and we write $t[u]_p$ for the result of replacing $t|_p$ at position p in t by u. Positions are compared in the prefix ordering. Substitutions are homomorphic extensions of a map from variables to terms, to a map from terms to terms. t is an instance of s by the substitution σ if $t = s\sigma$, using postfix notation for the substitution operation also called *instantiation.* Computing σ is called *(plain) pattern matching.* Substitution τ *subsumes* substitution σ if $\sigma = \tau\gamma$ for some substitution γ. Two terms s, t are *unifiable* if $s\sigma = t\sigma$, and σ is called a (unique up to renaming of variables) most general (plain) unifier (mgu for short) when it is minimal wrt the (well-founded) subsumption partial order.

A rewrite system is a set of pairs called rewrite rules, written as $R = \{l_i \to r_i\}_i$, where l_i is a non-variable term called its *lefthand side* and r_i is a term called its *righthand side* such that $\mathcal{V}ar(r_i) \subseteq \mathcal{V}ar(l_i)$. A term u (*plain*) *rewrites* to a term v with the rule $l_i \to r_i$ at position $p \in \mathcal{P}os(u)$, if $u|_p = l_i\sigma$ for some substitution σ and $v = [r_i\sigma]_p$. $l_i\sigma$ is called a *redex* and $r_i\sigma$ its *reduct.* We write $u \xrightarrow{p\cdot\mathcal{FP}os(l_i)}_{l_i \to r_i} v$ or $u \xrightarrow{p}_R v$, or simply $u \xrightarrow{p} v$.

A set of equations E is a symmetric rewrite system, in which case rewriting with E at position p is written $s \overset{p}{\underset{E}{\longleftrightarrow}} t$. The conversion relation $=_E$ is called the equational theory of E. t is an E-instance of s with the substitution σ if $t =_E s\sigma$. Computing σ is called *E-pattern matching.* A substitution σ is an *E-unifier* of the terms s, t if $s\sigma =_E t\sigma$. We are interested in theories, like AC, having a finite complete set of unifiers $CSU(s, t)$ for an arbitrary pair (s, t) of terms: any unifier of $s = t$ is then an E-instance of a unifier in $CSU(s, t)$.

3.2 Plain rewriting [14]

Plain rewriting corresponds to empty sets S and E. Plain rewriting satisfies the properties (*,**). The termination assumption of the NARS implies termination of the relation $\longrightarrow_R$. Then, the Church-Rosser property of R reduces to the joinability of local rewrite peaks $s \;{}_{l\to r\in R}\!\xleftarrow{p} u \xrightarrow{q}_R t$ with $q \in \mathcal{FP}os(l)$.

▶ **Definition 3.1.** Given two rules $g \to d$ and $l \to r$ in R s.t. $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$, and a position $p \in \mathcal{FD}om(g)$ such that l and $g|_p$ unify with most general unifier σ, then $\langle d\sigma, (g[r]_p)\sigma\rangle$ is called a *plain critical pair* of $l \to r$ onto $g \to d$ at p, of which $l\sigma$ is the *overlap.*

The proof of the Knuth and Bendix theorem now reduces to the sole classical critical pair case:

▶ **Theorem 3.2** (Knuth and Bendix). *Assume R is terminating. Then R is Church-Rosser iff its plain critical pairs are joinable with $\longrightarrow_R$.*

3.3 Rewriting modulo [21, 10]

We consider here the case where the set R is Church-Rosser modulo a theory E such as associativity and commutativity, S being empty. Rewriting uses then pattern matching modulo E. Our assumption that $=_E \longrightarrow_{R_E} =_E$ is terminating is called E-termination of R. Again, rewriting modulo enjoys the properties (*,**), which is not true of plain rewriting in E-congruence classes of terms.

We need to define critical pairs modulo:

▶ **Definition 3.3.** Given two rules $g \to d$ and $l \to r$ in R s.t. $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$, and a position $p \in \mathcal{FD}om(g)$ s.t. l and $g|_p$ unify with a complete set of most general unifiers Σ, then $\{\langle d\sigma, (g[r]_p)\sigma\rangle \mid \sigma \in \Sigma\}$ is called a complete set of *E-critical pairs* of $l \to r$ onto $g \to d$ at p, of which $l\sigma$ is the *E-overlap*.

We then recall the notion of extension introduced by Peterson and Stickel in the AC-case, and by Jouannaud and Kirchner in the general case:

▶ **Definition 3.4.** Given a rule $g \to d \in E$ and a rule $l \to r \in R$ such that $\mathcal{V}ar(g) \cap \mathcal{V}ar(l) = \emptyset$, and a position $p \in \mathcal{FD}om(g) \setminus \{\Lambda\}$ such that l unifies with $g|_p$ modulo E, then the rule $g[l]_p \to g[r]_p$ is called an *E-extension* of R.

▶ **Theorem 3.5** (Jouannaud and Kirchner). *Assume R is E-terminating and closed under E-extensions. Then R is Church-Rosser modulo E provided all its E-critical pairs are joinable.*

Proof. We omit the proof of joinability of critical rewrite peaks under the assumption that E-critical pairs are joinable, and concentrate on the critical rewrite cliffs. Let $s \underset{E}{\overset{P_p}{\longleftrightarrow}} u \xrightarrow{q}_{R_{SE}} t$ with $g = d \in E$, $P_p = \mathcal{FP}os(g)$ and $l \to r \in R$. By monotonicity of rewriting, we can assume $p = \Lambda$ and $q \in \mathcal{FP}os(g) \setminus \{\Lambda\}$. Then $s = d\sigma$ and $t = g\sigma$ while $t|_q = l\sigma$ and $u = l[d\sigma]_p$. We get $(g|_q)\sigma =_E l\sigma$, hence $g[l] \to g[r] \in R$ by closure assumption. Since $s \underset{E}{\longleftrightarrow} (g[l]_q)\sigma$, then $s \longrightarrow_{R_E} t = (g[r]_p)\sigma$. ◀

As shown in [21], extensions are finitely many for AC, and more generally when E is a set of permutative axioms, since extensions of extensions are then useless.

As a simple illustrating example, let $E = AC$ and $R = \{x + 0 \to x\}$. E-termination is obvious since AC-equivalence classes are size-preserving while the rule is size decreasing. There are no E-critical pairs since $x + 0$ does not E-unify with 0. Finally, R happens to be closed under extensions by pure luck: since $(x + 0) + y =_{AC} (x + y) + 0$, the extension $(x + 0) + y \to x + y$ is indeed an AC-instance of the original rule.

3.4 Normal rewriting

We now come to the general case, where R, S, E are sets of rules and equations satisfying our termination assumption. Note that our version of normal rewriting below p satisfies our assumptions (*,**), which is not the case of Nipkow's variant for which terms are normalized above p, destroying both. We actually only need that R_{SE}, S_E and E satisfy (*,**) to show that $R_{SE} \cup S_E$ is Church-Rosser under our assumptions that they satisfy the local properties, which indeed implies the desired property for both variants.

We now need to characterize the joinability properties of the critical patterns (i, ii, iii, iv).

For rewrite peaks, we need to check for joinability *complete sets of critical pairs of R modulo $S \cup E$*. This assumes that such complete sets exist. Note that it is possible to filter out the pairs which overlap is simplifiable by S_E.

For rewrite cliffs, we generate a normalized *E-extension* $g[l]_p{\downarrow} \to g[r]_p{\downarrow}$ for each rule $l \to r$ with respect to the equation $g = d \in E$ at $p \in \mathcal{FP}os(g) \setminus \{\Lambda\}$ provided l and $g|_p$ unify modulo $E \cup S$ (and symmetrically with d).

For simplification cliffs, we generate a normalized *oriented S-extension* $g[l]_p{\downarrow} \to g[r]_p{\downarrow}$ for each rule $l \to r$ with respect to the rule $g \to d \in S$ at $p \in \mathcal{FP}os(g) \setminus \{\Lambda\}$ provided l and $g|_p$ unify modulo $E \cup S$.

We are left with simplification peaks, which require a new kind of extension:

▶ **Definition 3.6.** Given rules $l \to r \in R$ and $g \to d \in S$, and a position $q \in \mathcal{FP}os(l)$ such that $l|_q$ and g are E-unifiable with a complete set of unifiers Σ, then the rules in $\{(l[d]_q)\sigma{\downarrow} \to (r\sigma){\downarrow} \mid \sigma \in \Sigma$ are called *simplification pairs* of $g \to d$ onto $l \to r$ at position q.

▶ **Lemma 3.7.** *Assume (R, S, E) is closed under simplification pairs. Then critical simplification peaks are shallow-joinable.*

Proof. Note that $s \longrightarrow_{R_{S_E}} t$ for any simplification pair $s \to t$, hence R can be closed by these extensions without compromising soundness nor termination.

By monotonicity of rewriting, we can assume wlog that $p = \Lambda$. Let $u \xrightarrow{\Lambda}_R v$ with $l \to r \in R$ and $u \xrightarrow{q \in \mathcal{FP}os(l) \setminus \{\Lambda\}}_{S_E} w$ with $g \to d \in S$, and $w = l\tau[d]_q\tau$ (assuming $\mathcal{V}ar(l) \cap \mathcal{V}ar(g) = \emptyset$). Then, $u|_q = (l|_q)\tau =_E g\tau$. By closure assumption, some rule $l[g]\sigma{\downarrow} \to r\sigma{\downarrow}$ belongs to R for some σ such that $\tau =_E \sigma\theta$. Therefore, we get $w = (l[d]_q)\tau \xleftrightarrow[E]{(\geq_{\mathcal{P}}\ q)^*} (l[d]_q)\sigma\theta \xrightarrow{*}_{S_E} (l[d]_q)\sigma{\downarrow}\,\theta \longrightarrow_R r\sigma{\downarrow}\,\theta \;{}_{S_E}\!\xleftarrow{*} r\tau = v$ and we are done. ◀

Note that we need to generate an extension for each substitution in $CSU(l|_q, g)$ rather than the single extension $l[d]_q \to r$ as for the other cases, since the latter would not yield shallow-joinability. On the other hand, we could require that simplification pairs satisfy shallow joinability, and indeed adding them as rules ensures that property.

We are now ready for the main result of this section:

▶ **Theorem 3.8.** *Let (R, S, E) be a NARS s.t.*
(i) *SE-critical pairs of R are joinable,*
(ii) *R is closed under normalized E-extensions,*
(iii) *R is closed under normalized simplification pairs, and*
(iv) *R is closed under normalized oriented S-extensions.*
Then normal rewriting is Church-Rosser.

Normal rewriting has many advantages: first, it allows rewriting with R modulo SE, despite the fact that congruence classes modulo SE may be infinite; second, compared to rewriting modulo SE, it allows to narrow down the sets of critical pairs and extensions; third, normal rewriting has a stronger rewriting power than normalized rewriting, and less critical pairs need be computed.

3.5 Example

We consider the example of the introduction: $R = \{x + x^{-1} \to 0\}$, $S = \{x + 0 \to x\}$ and $E = AC$. First, the termination assumption is satisfied. This is classically shown by a polynomial interpretation. Define $[\![x + y]\!] = [\![x]\!] + [\![y]\!]; [\![x^{-1}]\!] = 1 + [\![x]\!]$ and $[\![0]\!] = 0$. We then verify that equations in $S \cup E$ are invariant under the interpretation, while the rule in S decreases strictly. A lexicographic argument yields termination of $R_{SE} \cup S_E$.

We know that S is Church-Rosser modulo AC, which we did as an application of Theorem 3.5.

Finally, we need to show that normal rewriting with R is Church-Rosser. $x + x^{-1}$ unifies with the strict subterm $y + z$ of A, hence we need to add the extension $x + x^{-1} + y \to y$ (simplifying the righthand side $(0 + y)$ into y) to R. To ensure local confluence with S_E, we need to add $0^{-1} \to 0$ to R which has become $R = \{x + x^{-1} \to 0, (x + x^{-1}) + y \to y, 0^{-1} \to 0\}$. We can then verify that the resulting system satisfies our result, hence is Church-Rosser.

4 Higher-order rewriting systems

Our interest here is in higher-order rewriting as introduced by Nipkow [19, 17]. Nipkow and Mayr assume that rules in R are simply typed, of basic type, in η-long β-normal form and that their lefthand sides are patterns. Higher-order rules are fired via higher-order pattern matching. Other, related approaches to higher-order rewriting are considered and compared in [22].

We now consider a simply typed lambda calculus $\lambda^{\to}$ generated by sets of function symbols $\mathcal{F}_n$ with arity $n \geq 0$ and a set of variables $\mathcal{X}$, the type structure being itself generated by a set $\mathcal{S}$ of type constants. We use a, b for types, and s, t for (raw) terms:

$$\begin{array}{rcl} a, b & := & \mathcal{S} \mid a \to b \\ s, t & := & x \mid f(s_1, \ldots, s_n) \mid \lambda x : a \,.\, s \mid (s\, t) \end{array}$$

Typing judgements are of the form $\Gamma \vdash s : a$, where Γ is a set of declarations of the form $\{x_1 : a_1, \ldots, x_n : a_n\}$ such that $x_i \neq x_j$ if $i \neq j$. A term s is *typable* of type a in the environment Γ if the judgement $\Gamma \vdash s : a$ can be proved from the following rules:

var: $$\frac{}{\Gamma \cup x : a \vdash x : a}$$

fun: $$\frac{\Gamma \vdash s_1 : a_1, \ldots, \Gamma \vdash s_n : a_n}{\Gamma \vdash f(s_1, \ldots, s_n) : a}$$

$\texttt{if}\ f : a_1 \to \ldots \to a_n \to a \in \mathcal{F}_n$

abs: $$\frac{\Gamma \cup x : a \vdash s : b}{\Gamma \vdash (\lambda x : a \,.\, s) : a \to b}$$

app: $$\frac{\Gamma \vdash s : a \to b \quad \Gamma \vdash t : a}{\Gamma \vdash (s\, t) : b}$$

As usual, substitutions are *capture*-avoiding "morphism" written $\{x_1 \mapsto s_1, \ldots, x_n \mapsto s_n\}$ when finitely variables are involved, such that x_i and s_i have the same type in the environment Γ.

$\lambda^{\to}$ comes equipped with three equations:

alpha: $\lambda x : a \,.\, s = \lambda y : a \,.\, s\{x \mapsto y\}$ if $y \notin \mathcal{V}ar(s)$

beta: $((\lambda x : a \,.\, s)\, t) = s\{x \mapsto t\}$

eta: $\lambda x : a \,.\, (s\, x) = s$ if $x \notin \mathcal{V}ar(s)$

beta is oriented as a rule from left to right, while eta can be oriented as a reduction from left to right or as an expansion from right to left. The set S of simplifiers is made of beta and a choice of orientation for eta.

Let us now recall that the presence of binders forces to rewrite modulo α-conversion, even when rewriting with the beta rule alone. The simplest example is due to Barendregt and Klop:

$(\lambda x.(x\ x)\ \lambda s.\lambda z.(s\ z)) \longrightarrow (\lambda s.\lambda z.(s\ z)\ \lambda s.\lambda z.(s\ z)) \longrightarrow \lambda z.(\lambda s.\lambda z.(s\ z)\ z) \longrightarrow \lambda z.\lambda z.(z\ z)$.

which last step has resulted in the variable z being captured by λz. We should instead rename the *inside* binder $\lambda z.$, showing once more that rewriting modulo (here, α-conversion) surfaces everywhere:

$\lambda z.(\lambda s.\lambda z.(s\ z)\ z) \xleftrightarrow[\alpha]{(\geq 1)^*} \lambda z.(\lambda s'.\lambda z'.(s'\ z')\ z) \xrightarrow[\beta]{1} \lambda z.\lambda z'.(z\ z')$.

We now move to our higher-order rewrite rule format, which definition is the following:

▶ **Definition 4.1.** A *rewrite rule* is a tuple (Γ, l, r, σ) s.t.

(i) l and r are in S_E-normal form,

(ii) $\Gamma \vdash l : \sigma$ and $\Gamma \vdash r : \sigma$,

(iii) $l = f(l_1, \ldots, l_n)$ for some $f \in \mathcal{F}_n$, and is a pattern [18],

(iv) $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$.

We write $\Gamma \vdash l \to r : \sigma$ or simply $l \to r$ if no ambiguity.

It is actually not necessary to assume that lefthand sides of rules are headed by a function symbol, but, besides being a natural assumption, it simplifies the critical pairs analysis. We shall however explain what are the additional computations needed when this assumption is not met.

On the other hand, the pattern assumption cannot be dispensed with as pointed out to us by Vincent van Oostrom:
Given type constants o, i, let $\mathcal{F} = \{f : (o \to o) \to o, a : o, h : o \to i, g : i \to o\}$, $\mathcal{X} = \{o \to o, X : i\}$, and $R = \{f(\lambda x : o\,.\,F(F(x)) \to a, h(g(X)) \to X\}$.
Then $a \longleftarrow f(\lambda x.g(h(g(h(x))))) \longrightarrow f(\lambda x\,.\,g(h(x)))$ with both terms in normal form despite the fact that there are no critical pairs in the usual sense since $f(\lambda x\,.\,F(F(x)))$ has no subterm of type i. The role of the pattern restriction is indeed to rule out these non-intuitive higher-order phenomena.

The assumption that rules are in S_E-normal form is important as well to ensure the absence of simplification peaks in most cases.

We will use variations of a single example, differentiation, writing $u(v)$ instead of $(u\,v)$:

$$\begin{array}{lll}
\mathsf{R} & : & * \qquad\qquad\qquad \%\ \text{type of reals} \\
\times^2 & : & \mathsf{R} \to \mathsf{R} \to \mathsf{R} \qquad\qquad \%\ \text{arity } 2 \\
\mathsf{diff}^1 & : & (\mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \quad \%\ \text{arity } 1 \\
\mathsf{sin}^1, \mathsf{cos}^1 & : & \mathsf{R} \to \mathsf{R} \qquad\qquad\qquad \%\ \text{arity } 1 \\
\times^2_{\to} & : & (\mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \\
F & : & \mathsf{R} \to \mathsf{R} \\
\multicolumn{3}{l}{\mathsf{diff}(\lambda x.\,\mathsf{sin}(F(x))) \to \lambda x.\,\mathsf{cos}(F(x)) \times_{\to} \mathsf{diff}(F)}
\end{array}$$

The idea here is to embed composition into the definition, the usual rule for differentiating a sinus being recovered thanks to higher-order matching by instantiating F by the identity. Note that patterns occur naturally in examples.

Our termination assumption can be verified easily here by using the Normal Higher-Order Recursive path Ordering introduced in [13]. All the coming variants can be dealt with as well.

Although $\beta\eta$-congruence classes are infinite, termination of higher-order rules like the above one is usually easy to show because the top function symbols in the rules are different from those of the λ-calculus. Normal rewriting therefore appears to be a very good fit with higher-order computations.

4.1 Higher-order rewriting at simple types

4.1.1 η as an expansion [19, 17]

E is α-conversion and S is made of β-reduction and η-expansion saturating arrow types:

$$v[u]_p \longrightarrow_\eta v[\lambda x : a\,.\,(u\,x)]_p$$
$$\text{if } \begin{cases} u : a \to b \\ x \notin \mathcal{V}ar(u) \\ \text{if } p = q \cdot 1, \text{ then } v|_q \text{ is not an application} \end{cases}$$

To comply with the so-called Nipkow's format for which rules must operate at base types, the example becomes:

$$\mathsf{diff}(\lambda x.\,\mathsf{sin}(F(x)))(y) \to \mathsf{cos}(F(y)) \times \mathsf{diff}(\lambda x.F(x))(y)$$

There are of course no extensions associated with α-conversion.

Since η is used as an expansion, and has therefore a variable as its lefthand side, there are no oriented extensions associated with the η-rule, and no simplification extensions either, since rules in R are in normal form for the simplification rules.

There are no oriented extension for the β-rule since the rules in R being of base type, their lefthand side cannot unify with an abstraction, which is the sole strict non-variable subterm of the lefthand side of the beta rule. Note that the argument still holds for lefthand sides which are not headed by a function symbol. There are no simplification extensions either: since rules are in η-long β-normal form, no subterm of a rule can unify with the lefthand side of β except for subterms of the form $(X\ x)$, where X is a higher-order variable and x a variable bound above. Instantiating X by $\lambda y.u$ for some term u yields a term which is a higher-order instance of l, hence rewrites to the corresponding instance of the righthand side. It appears therefore that rules in R are their own simplification pairs. Therefore,

▶ **Theorem 4.2** ([17]). *Higher-order rewriting with R is Church-Rosser provided*
(i) $\longrightarrow_{R_{\beta\eta}} \cup \longrightarrow_{\beta\eta^{-1}}$ *is α-terminating;*
(ii) *irreducible higher-order critical pairs are joinable.*

where higher-order critical pairs are defined a usual by solving equations of the form $l|_p =_{\beta\eta\alpha} g$ for some rules $l \to r$ and $g \to d$. Note the strong analogy with Theorem 3.2. This is due to the choice of orienting η as an expansion, and to rule out user's rules at higher-type. Note also that the presence of α in our termination assumption, which is usually omitted (that is, left implicit).

This version of Nipkow's result requires to prove that the relation $(\longrightarrow_{R_{\beta\eta}} \cup \longrightarrow_{\beta\eta^{-1}}$ is α-terminating, instead of $\longrightarrow_{R_{\beta\eta}}$ and $\longrightarrow_{\beta\eta^{-1}}$ being separately α-terminating as in Nipkow's original result. On the other hand, we conclude for the stronger Church-Rosser property instead of confluence as does Nipkow. Note also the little improvement, compared with Nipkow's result, obtained by eliminating the reducible higher-order critical pairs from the joinability test.

4.1.2 η as a reduction:

S is now made of β and η reductions. We will nevertheless recover the advantages of η-expansions by having arities for the variables as in Klop's framework: they can be η-expanded up to the saturation of their arity, as in $\lambda xy.X(x,y)$ for X of arity two and x, y of arity zero. This η-expanded term is indeed η-reduced, since $X(x)$ and X are not terms in this setting.

The only difference with the previous case is therefore the orientation of η. Since simplification pairs require the unification (modulo α) of the lefthand side of the η-rule with a subterm of a lefthand side of R, the only potential case is that of a subterm of a lefthand side of rule in R being of the form $\lambda x.X(x)$ where X is a free higher-order variable. Rewriting this subterm with η is not possible, though, since it would violate the arity of X. We are therefore left with oriented extensions, for which a lefthand side of rule would unify the only non-variable strict-subterm of the lefthand side of η, which is impossible with our rule format (which could actually be relaxed).

▶ **Theorem 4.3.** *Higher-order rewriting with R is Church-Rosser provided*
(i) $\longrightarrow_{R_{\beta\eta}} \cup \longrightarrow_{\beta\eta}$ *is α-terminating;*
(ii) *irreducible higher-order critical pairs are joinable.*

This shows that the choice of orienting eta as a reduction or an expansion has no impact on confluence when rewriting is only possible at simple types.

4.2 Higher-order rewriting at higher types

To understand the importance of the type assumption in Nipkow's format, let us consider his motivating example $R = \{\lambda x.a \to \lambda x.b\}$, where a and b are constants of a given base type. a and b are convertible terms in η-long β-normal form since $a \xleftrightarrow[\beta]{\Lambda} (\lambda x.a\ u) \xleftrightarrow[R]{1} (\lambda x.b\ u) \xleftrightarrow[\beta]{\Lambda} b$, but not joinable.

This has motivated Nipkow's restriction that the lefthand side of a higher-order rule is of base type, and therefore, is not an abstraction. Of course, it would be easy to change the rule into $a \to b$ (making it satisfy our definition of rule), therefore avoiding the problem, but this cannot be done in general. Consider for instance the rewrite rule $\lambda x.\, f(x, Z(x)) \to \lambda x.\, g(x, Z(x))$. Removing the abstraction yields $f(X, Z(X)) \to g(X, Z(X))$, a rule which left-hand side is no pattern.

Nipkow's example shows a case of critical simplification peak: adding the normalized β-extension of the original rule $\lambda x.a \to \lambda x.b$, that is $a \to b$ solves the problem. In general, the normalized β-extension of a rule $\lambda x.l(x) \to r$ is the rule $l(x) \to @(r, x)\downarrow$. Since one abstraction is pulled out, a rule can have only finitely many such extensions which are easily computable.

Our format rules out the need for these extensions, since lefthand sides cannot be abstractions. The previous argument that simplification pairs are not needed remains valid. Therefore,

▶ **Theorem 4.4.** *Higher-order rewriting with R is Church-Rosser provided*
(i) $\longrightarrow_{R_{\beta\eta}} \cup \longrightarrow_{\beta\eta^{-1}}$ *is α-terminating,*
(ii) *irreducible higher-order critical pairs are joinable.*

We can therefore reformulate our example as follows:

$$\begin{array}{lll}
\mathsf{diff}^1 & : & (\mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \\
\mathsf{sin}^0, \mathsf{cos}^0 & : & \to (\mathsf{R} \to \mathsf{R}) \\
F^0 & : & \to (\mathsf{R} \to \mathsf{R}) \\
\times^1_{\to} & : & (\mathsf{R} \to \mathsf{R}) \to (\mathsf{R} \to \mathsf{R}) \\
\multicolumn{3}{l}{\mathsf{diff}(\mathsf{sin} \circ F) \to \mathsf{diff}(\mathsf{sin}) \times^1_{\to} \mathsf{diff}(F)}
\end{array}$$

Orienting eta as a reduction would yield the same result again for our rule format.

4.3 Adding algebraic equations in E

Our main abstract result allows us to also consider equations like AC in E which would then contain both AC and α, see [6] for examples. Of course the presence of AC requires checking new pairs for the corresponding local properties. Higher-order unification modulo AC of higher-order patterns yields finite complete sets of unifiers [6], hence the calculations which require higher-order unification yield decidable tests when lefthand sides of rules are patterns.

5 Conclusion

We have given a framework for normal rewriting terms that covers a wide variety of rewriting applications, whether first or higher-order. These results are very economic thanks to an abstract framework which incorporates a lightweight axiomatization of positions. Besides, they solve a long-standing open problem regarding how to check the Church-Rosser property of higher-order rewriting at any type, whether basic or functional.

We believe that the application of our main abstract result to higher-order rewriting can be pushed further, by allowing for polymorphic or dependent types.

A referee strongly suggested many potential extensions of the framework, to a fully hierarchical setting, to non-terminating normal rewriting (using decreasing diagrams), to graph rewriting, and more. Although we are not interested ourselves in those extensions at this point, we would of course welcome efforts in these directions. Such extensions would provide the appropriate theoretical basis for several recent applications, the most surprising to us being [1].

Acknowledgements: to Delia Kesner, Femke van Raamsdonk and Albert Rubio for discussing actively this topic in the past.

References

1 O. Al-Hassani, Q.-A. Mahesar, C. Sacerdoti Coen, and V. Sorge. A term rewriting system for Kuratowski's closure-complement problem. In Tiwari A., editor, *RTA*, volume 15 of *LIPIcs*, pages 38–52. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

2 L. Bachmair. *Canonical Equational Proofs*. Birkhäuser, Boston, 1991.

3 L. Bachmair, N. Dershowitz, and J. Hsiang. Orderings for equational proofs. In *Proc. 1st LICS*, 1986.

4 T. Baird, Peterson G. E., and Wilkerson R. Complete sets of reductions modulo Associativity, Commutativity and Identity. In *Proc. 3rd RTA, LNCS 355*, pages 29–44. Springer, 1989.

5 H. Barendregt. *Handbook of Theoretical Computer Science*, volume B, chapter Functional Programming and Lambda Calculus. North-Holland, 1990.

6 A. Boudet and E. Contejean. AC-unification of higher-order patterns. In *Int. Conf. on Constraint Programming, 1997*, pages 267–281, 1997.

7 N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter Rewrite Systems. North Holland, 1990.

8 G. Gonthier, J.-J. Lévy, and P.-A. Mellies. An abstract standardisation theorem. In *LICS*, pages 72–81. IEEE Computer Society, 1992.

9 G. Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, October 1980.

10 J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.

11 J.-P. Jouannaud and C. Marché. Termination and completion modulo associativity, commutativity and identity. *Theoretical Computer Science*, 104:29–51, 1992.

12 J.-P. Jouannaud and F. van Raamsdonk. Confluence properties of terminating higher-order rewrite relations. In *Mathematical Theories of Abstraction, substitution and naming in Computer Science*, ICMS, Edinburgh, may 2007.

13 J.-P. Jouannaud and A. Rubio. Higher-order orderings for normal rewriting. In F. Pfenning, editor, *RTA*, volume 4098 of *LNCS*, pages 387–399. Springer, 2006.

14 D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*. Elsevier, 1970.

15 D. S. Lankford and Ballantyne A. M. Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions. Memo ATP-39, University of Texas, Austin, August 1977.

16 C. Marché. Normalised rewriting and normalised completion. In *Proc. 9th LICS*, 1994.

17 R. Mayr and T. Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192:3–29, 1998.

18 D. Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

19 T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.

20 V. van Oostrom. Confluence by decreasing diagrams converted. In Voronkov A., editor, *RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2008.

21 G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, April 1981.

22 F. van Raamsdonk. Higher-order rewriting. In P. Narendran and M. Rusinowitch, editors, *Proc. RTA*, LNCS 1631, pages 220–239. Springer, 1999.

23 Terese. Term rewriting systems. In *Cambridge Tracts in Theoretical Computer Science, M. Bezem, J. W. Klop, and R. de Vrijer editors*. Cambridge University Press, 2003.

A Counting Logic for Structure Transition Systems

Łukasz Kaiser[1] **and Simon Leßenich***[2]

1 **LIAFA, CNRS & Université Paris Diderot – Paris 7, France**
2 **Mathematische Grundlagen der Informatik, RWTH Aachen**

Abstract

Quantitative questions such as "what is the maximum number of tokens in a place of a Petri net?" or "what is the maximal reachable height of the stack of a pushdown automaton?" play a significant role in understanding models of computation. To study such problems in a systematic way, we introduce structure transition systems on which one can define logics that mix temporal expressions (e.g. reachability) with properties of a state (e.g. the height of the stack). We propose a counting logic $\mathsf{Q}\mu[\#\mathsf{MSO}]$ which allows to express questions like the ones above, and also many boundedness problems studied so far. We show that $\mathsf{Q}\mu[\#\mathsf{MSO}]$ has good algorithmic properties, in particular we generalize two standard methods in model checking, decomposition on trees and model checking through parity games, to this quantitative logic. These properties are used to prove decidability of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ on tree-producing pushdown systems, a generalization of both pushdown systems and regular tree grammars.

1998 ACM Subject Classification F.4.1, I.2.4

Keywords and phrases Logic in Computer Science, Quantitative Logics, Model Checking

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.366

1 Introduction

Models of computation describe temporal changes of a state of a system or a machine. For example, pushdown automata describe transformations of the stack, term rewriting systems capture modifications of terms, and Turing machines specify changes of the tape. Questions about computations of systems often ask about temporal events intertwined with properties of the state, e.g. reachability (temporal) of an empty stack (state property) or never encountering a tree of height less than one. We propose an abstract definition which distinguishes temporal transitions from the state of the system and allows to investigate properties of such systems and the corresponding logics in a uniform and systematic way.

▶ **Definition 1.** A *structure transition system* (STS) is a labeled transition system (Kripke structure) with an additional assignment $\mathfrak{m}$ of finite relational structures to the nodes of the system. Formally, given a set of transition labels R and a relational signature τ, an STS is a tuple $\mathfrak{S} = (S, \Delta, \mathfrak{m})$ where S is a set of states, $\Delta \subseteq S \times R \times S$ is a set of transitions, and $\mathfrak{m} : S \to \mathsf{FinStr}(\tau)$ assigns a finite τ-structure to each state.

Computing machines of various kinds can be viewed as finite objects which represent and generate infinite structure transition systems. Let us consider a few prominent examples.

- *Pushdown automata* induce structure transition systems in which the relational structures assigned to the nodes represent the current stack of the automaton, i.e. are words over the stack alphabet. The transition relation in the STS copies the one in the automaton.

[*] This author was supported by the ESF Research Networking Programme GAMES and the DFG Research Training Group 1298 (AlgoSyn).

- *Turing machines* generate STSs in a similar way to pushdown automata: the structure assigned to each node of the STS is again a word, and it represents the tape of the machine at that time. Transitions are induced from the ones of the machine.
- *Petri nets* give rise to STSs in which elements of the relational structures correspond to tokens in the net. Elements labeled by a predicate P_l correspond to tokens in place l in the net and firing of the net results in transitions of the STS.
- *Term rewriting systems* (TRSs) produce STSs in which one assigns a term to each node, i.e. the relational structure $\mathfrak{m}(s)$ is always a labeled tree, representing the term in s. The transition relation Δ is derived from the application of rewriting rules.
- *Graph rewriting systems* induce STSs in a similar way as TRSs, but the structures assigned to nodes are arbitrary graphs, or even hypergraphs in case of hypergraph rewriting.

Interesting properties of structure transition systems mix temporal events with state attributes. To specify them, we thus compose a temporal logic with another logic for the states. In such composition, the predicates of the temporal logic are replaced by *sentences* of the state logic, which in turn are evaluated on the relational structure assigned by $\mathfrak{m}$.

Consider the logic LTL[MSO], which allows to use MSO sentences in place of predicates in LTL. For example, the formula $G(\exists x\, a(x))$ expresses that an a-labeled element exists in the relational structure assigned to each reachable state of an STS. Since MSO sentences define regular languages of words and trees, one can express in LTL[MSO] over an STS the reachability of a state in which the stack belongs to a regular language (for pushdown systems) or in which a regular configuration appears on the tape (for Turing machines) or in which the term belongs to a regular tree language (for TRSs). Note that a LTL[MSO] formula defines a property in a uniform way, for pushdown systems, Turing machines and TRSs at the same time. Moreover, we can systematically inspect which temporal logic and which state logic can be combined to an efficient formalism on which classes of STS.

- If the state logic is trivial, i.e. consists only of the two formulas true and false, one obtains classical action-based temporal logics like LTL or the μ-calculus Lμ. Model-checking these logics is decidable on pushdown systems [13] and for linear-time logics (e.g. LTL) it is also decidable on Petri nets [5]. On the other hand, already model-checking action-based branching-time logics, e.g. Lμ, is undecidable on Petri nets [5].
- One can consider the logic Reach[MSO] in which formulas have the form Reach(ψ) for some $\psi \in$ MSO and express that an MSO-definable configuration is reachable. Model-checking this logic is decidable on pushdown systems [13] and also on Petri nets [14].
- Model-checking LTL[MSO] is decidable on STSs generated by pushdown systems [6], but the use of MSO makes it undecidable on STSs corresponding to Petri nets [5].
- For Lμ[MSO], the modal μ-calculus with MSO sentences as predicates, the model checking problem is decidable on context-free and prefix-recognizable rewrite systems, and thus also on pushdown systems [13]. It is also decidable on some classes of graph or structure rewriting systems, e.g. for separated structure rewriting [11].

Combinations of temporal and state logics, as the ones above, allow to express interesting properties of structure transition systems, but, since the formulas of these logics are Boolean, they are limited to yes-or-no answers. For example, it is not possible to ask “how high will the stack get on all runs?” of a pushdown automaton or “how many tokens will there maximally be in a place?” of a Petri net. Such questions are often very important for understanding the behavior of the system. Note also that the answer to a question of this kind might be either an integer or $\pm\infty$, in case the stack size or the number of tokens is unbounded. Such boundedness problems have been studied extensively for many models.

- The question whether the maximal stack size on runs of a pushdown system is bounded or not, intertwined with temporal properties, has been studied in [2, 9], and is a special case of the problem solved in this work.
- The boundedness problem for Petri nets was shown to be decidable in [12] and became one of the most important tools in Petri net analysis.
- On Turing machines, establishing the bound on the size of the tape during a computation is the same as determining its space complexity.
- Graph rewriting systems are used to model e.g. biochemical processes and one often asks for the number of particles of certain kind produced in the process.

In the next section, we introduce a counting logic $\mathsf{Q}\mu[\#\mathsf{MSO}]$ which allows to express queries like the ones discussed above. Fundamental algorithmic techniques from model checking generalize to this logic, as we show in Section 3. We apply these methods in Section 4 to compute the value of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formulas on tree-producing pushdown systems. In the proof, we use two key lemmas, proved in Section 5 and Section 6.

2 Counting μ-Calculus on Structure Transition Systems

To express questions of the above form, we propose the *counting μ-calculus*, a *quantitative* logic in which each formula has not just a Boolean value, but it evaluates to a number in $\mathbb{Z}_\infty := \mathbb{Z} \cup \{-\infty, \infty\}$. This logic, denoted $\mathsf{Q}\mu[\#\mathsf{MSO}]$, allows to use counting terms on state structures and maximum, minimum and fixed-point operations in the temporal part. This suffices to express all the example questions presented above and to query for boundedness. Note that it is not a probabilistic logic, and we do not introduce operators for sums over different paths. $\mathsf{Q}\mu[\#\mathsf{MSO}]$ is composed of the quantitative μ-calculus [7], and for quantitative predicates uses counting terms on top of MSO formulas, defined as follows.

▶ **Definition 2.** An *MSO counting term* has the form $\#_{x_1\cdots x_n}\varphi(x_1,\cdots,x_n)$, where $\{x_1,\cdots,x_n\}$ is the set of *all* free variables of the MSO formula φ. On a finite structure $\mathfrak{A}$, the term represents the number of tuples $a_1,\cdots,a_n$ such that $\mathfrak{A} \models \varphi(a_1,\cdots,a_n)$,

$$[\![\#_{\overline{x}}\,\varphi(\overline{x})]\!]^{\mathfrak{A}} := |\{\overline{a} \mid \mathfrak{A} \models \varphi(\overline{a})\}|.$$

For a formula φ without free variables, we set $\#\varphi = 1$ if φ holds and $\#\varphi = 0$ in the other case.

Using the above counting terms as predicate symbols, formulas of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ are built according to the following grammar, analogous to [7].

$$\psi \;::=\; \#_{\overline{x}}\varphi(\overline{x}) \mid X \mid \neg\psi \mid \psi \wedge \psi \mid \psi \vee \psi \mid \Box_r\,\psi \mid \Diamond_r\,\psi \mid \mu X.\psi \mid \nu X.\psi,$$

where $r \in R$ are labels of the transitions and each $X \in \mathrm{Var}$ is a fixed point variable and must appear positively in φ, i.e. under an even number of negations. We will often write $\Diamond$ for the disjunction of $\Diamond_r$ over all $r \in R$ for a finite R, and $\Box$ analogously. The semantics of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ combines the quantitative μ-calculus [7] with MSO counting terms.

▶ **Definition 3.** Let $\mathfrak{S} = (S, \Delta, \mathfrak{m})$ be a structure transition system and $\mathcal{F} := \{f : S \to \mathbb{Z}_\infty\}$ the set of quantitative assignments to the states of $\mathfrak{S}$. Given an evaluation of fixed point variables $\varepsilon : \mathrm{Var} \to \mathcal{F}$ we define the evaluation of a $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formula ψ, denoted $[\![\psi]\!]^{\mathfrak{S}}_{\varepsilon} : S \to \mathbb{Z}_\infty$, in the following inductive way.

- $[\![X]\!]^{\mathfrak{S}}_{\varepsilon} = \varepsilon(X)$
- $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{\mathfrak{S}}_{\varepsilon}(s) = [\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{\mathfrak{m}(s)} = |\{\overline{a} \mid \mathfrak{m}(s) \models \varphi(\overline{a})\}|$

- $[\![\neg\psi]\!]^{\mathfrak{S}}_{\varepsilon} = -1 \cdot [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}$
- $[\![\psi_1 \wedge \psi_2]\!]^{\mathfrak{S}}_{\varepsilon} = \min([\![\psi_1]\!]^{\mathfrak{S}}_{\varepsilon}, [\![\psi_2]\!]^{\mathfrak{S}}_{\varepsilon})$, $[\![\psi_1 \vee \psi_2]\!]^{\mathfrak{S}}_{\varepsilon} = \max([\![\psi_1]\!]^{\mathfrak{S}}_{\varepsilon}, [\![\psi_2]\!]^{\mathfrak{S}}_{\varepsilon})$
- $[\![\Diamond_r\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s) = \sup_{\{s' \mid (s,r,s') \in \Delta\}} [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s')$, $[\![\Box_r\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s) = \inf_{\{s' \mid (s,r,s') \in \Delta\}} [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon}(s')$
- $[\![\mu X.\psi]\!]^{\mathfrak{S}}_{\varepsilon}$ is the least and $[\![\nu X.\psi]\!]^{\mathfrak{S}}_{\varepsilon}$ the greatest fixed point of the operator $f \mapsto [\![\psi]\!]^{\mathfrak{S}}_{\varepsilon[X \leftarrow f]}$

To compute the fixed point, we consider $\mathcal{F}$ as a complete lattice with pointwise order, i.e. $f \leq g$ if and only if $f(s) \leq g(s)$ for all states $s \in S$.

Note that the above definition is very similar to the classical, Boolean μ-calculus. As in the standard case, one can evaluate fixed points inductively, e.g. the least fixed point starting from a function which assigns $-\infty$ to all states. The Boolean logic Lμ[MSO] can in fact be embedded in Qμ[#MSO] as follows: take a formula ψ of Lμ[MSO] in negation normal form and replace each literal $\neg\varphi$ by $\#\neg\varphi$ and each φ by $\#\varphi$. These terms have now value 1 if φ holds and 0 in the other case, and since the semantics coincide, the Qμ[#MSO] formula obtained in this way will evaluate to ∞ or 1 if ψ holds and to 0 or $-\infty$ otherwise. In this sense the logic Qμ[#MSO] subsumes Lμ[MSO], and therefore also several other Boolean logics, e.g. LTL[MSO] and CTL[MSO]. But, of course, in addition to Boolean ones, Qμ[#MSO] allows to express quantitative properties, e.g. the following.

- The formula $\psi_\# = \mu X.(\#_x(x = x) \vee \Diamond X)$ calculates the bound on the size of structures appearing on all runs of the STS from where it is evaluated. Therefore $\varphi_\#(s) \neq \infty$ if and only if there is a bound on the size of the structures on all runs from s, and checking if $\varphi_\#(s) \neq \infty$ answers the boundedness problems mentioned before.
- The formula $\psi_\times = \nu X.(\#_{x,y}(a(x) \wedge b(y)) \wedge \Box X)$ computes the minimal *product* of the number of a-labeled elements and b-labeled ones on all paths from the node in which it is evaluated.
- For two atoms φ_1 and φ_2, we will denote the formula $\mu X.(\varphi_2 \vee (\varphi_1 \wedge \Diamond X))$ by $\varphi_1 \,\mathsf{until}\, \varphi_2$, as it is the standard Lμ formula expressing the LTL until modality. In the quantitative setting, this formula calculates the maximal value of φ_1 at the last node in which $\varphi_1 > \varphi_2$ on all paths. If we set $\varphi_1 = \#_x a(x)$ and $\varphi_2 = \#_x b(x)$ then $\varphi_1 \,\mathsf{until}\, \varphi_2$, on words, computes the maximal number of as reached on prefixes of runs on which there are more as than bs.

3 Model Checking Games and Decomposition

The logic Qμ[#MSO] is a composition of a quantitative extension of the μ-calculus Lμ and a counting extension of MSO. Good algorithmic properties of Lμ stem from its connection to parity games, and decidability of MSO on linear orders and trees has its roots in the decomposition property. It is therefore natural to ask whether these basic methods generalize to Qμ[#MSO]. We give a positive answer, showing both model-checking games for Qμ on structure transition systems and a decomposition theorem for #MSO. These two tools will be crucial in the decidability proof in the next section.

3.1 Model-Checking Games

To model-check Qμ one can use quantitative parity games, as shown in [7]. We use an almost identical notion (except for discounts) for Qμ[#MSO] on structure transition systems.

▶ **Definition 4.** A *quantitative parity game* (QPG) $\mathcal{G}$ is a tuple $\mathcal{G} = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ such that (V, E) is a directed graph whose vertices V are partitioned into positions $V_{\max}$ of Maximizer and positions $V_{\min}$ of Minimizer. Every vertex is assigned a color by $\Omega \colon V \to \{0, \cdots, d\}$ and terminal vertices $T = \{v \in V \mid vE = \emptyset\}$ are labeled by the payoff function $\lambda \colon T \to \mathbb{Z}$.

How to play. Every play starts at some vertex $v \in V$. For every vertex in $V_{\max}$, Maximizer chooses a successor vertex and the play proceeds from that vertex (analogously for Minimizer). If the play reaches a terminal vertex, it ends. We denote by $\pi = v_0 v_1 \ldots$ the (possibly infinite) play through vertices $v_0 v_1 \ldots$, given that $(v_n, v_{n+1}) \in E$ for every n. The outcome $p(\pi)$ of a finite play $\pi = v_0 \ldots v_k$ is given by $\lambda(v_k)$. The outcome of an infinite play depends only on the lowest priority seen infinitely often. We will assign the value $-\infty$ to every infinite play where the lowest priority seen infinitely often is odd, and ∞ to those where it is even.

Goals. The two players have opposing objectives regarding the outcome of the play. Maximizer wants to maximize the outcome, while Minimizer wants to minimize it.

Strategies. A strategy of Maximizer (Minimizer) is a function $s : V^* V_{\max} \to V$ ($s : V^* V_{\min} \to V$) with $(v, s(hv)) \in E$ for each h, v. A play $\pi = v_0 v_1 \ldots$ is *consistent with a strategy* s of Maximizer if $v_{n+1} = s(v_0 \ldots v_n)$ for every n such that $v_n \in V_{\max}$, and dually for strategies of Minimizer. For strategies f, g of the two players, we denote by $\alpha_{f,g}(v)$ the unique play starting at v which is consistent with both f and g.

Determinacy. A game is *determined* if, for each position v, the highest outcome Maximizer can assure from this position and the lowest outcome Minimizer can assure coincide,

$$\sup_{f \in \Sigma_{\max}} \inf_{g \in \Sigma_{\min}} p(\alpha_{f,g}(v)) \quad = \inf_{g \in \Sigma_{\min}} \sup_{f \in \Sigma_{\max}} p(\alpha_{f,g}(v)) =: \mathsf{val}\,\mathcal{G}(v),$$

where $\Sigma_{\min}, \Sigma_{\max}$ are the sets of all possible strategies for Minimizer and Maximizer and the achieved outcome is called the *value of* $\mathcal{G}$ *at* v.

As shown in [7], quantitative parity games are *determined* and can be used for model-checking. We adapt the construction from [7] and construct a model-checking game satisfying the following properties.

▶ **Theorem 5** (c.f. [7]). *For every $\psi \in$ Qμ[#MSO] and every STS $\mathfrak{S}$ one can construct the quantitative parity game* $\mathsf{MC}(\mathfrak{S}, \psi) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ *which is a* model-checking game *for ψ and $\mathfrak{S}$, i.e. it satisfies the following properties:*

(1) $V = (S \times \mathsf{Sub}(\psi)) \cup \{(\infty), (-\infty)\}$, *where* $\mathsf{Sub}(\psi)$ *is the set of subformulas of* ψ.
(2) *In terminal positions* (s, ψ) *the formula* ψ *has the form* $\#_{\overline{x}}\varphi(\overline{x})$ *or* $\neg\#_{\overline{x}}\varphi(\overline{x})$.
(3) *If* $((s, \psi), (s', \psi')) \in E$ *then either* $s = s'$ *or* $(s, s') \in \Delta$.
(4) $\Omega(s, \psi)$ *depends only on* ψ, *not on* s.
(5) *val*$\mathsf{MC}(\mathfrak{S}, \psi)(s, \psi) = [\![\psi]\!]^{\mathfrak{S}}(s)$.

The construction of the model-checking game corresponds to the one in [7], only that in the setting of STS, discounts are not needed and have thus been removed.

In this work, we will be especially interested in *pushdown quantitative parity games*. For a finite stack alphabet Γ, bottom symbol $\bot \notin \Gamma$ and a finite state set Q, we define a pushdown process $\mathcal{A} = (Q, \hookrightarrow)$ in the standard way (see Definition 1 in [16]) and we denote by $E_{\hookrightarrow}$ the corresponding one-step transition relation. We say that a QPG is a pushdown game over $\mathcal{A}$ if it has positions from $\mathcal{A}$, i.e. of the form (q, s) for $s \in \Gamma^*$ and $q \in Q$, moves given by $E_{\hookrightarrow}$, and the partition into $V_{\max}$ and $V_{\min}$ and the color Ω of a position (q, s) depend only on q and not on s. We say that a pushdown process or QPG is *pop-free* if $\hookrightarrow$ contains no pop-rules, equivalently if for each edge in $E = E_{\hookrightarrow}$ from (q, s) to (q', s') it holds that $|s'| \geq |s|$. In Section 5 we adapt the ideas from [19] to prove the following reduction.

▶ **Theorem 6.** *For every pushdown QPG $\mathcal{G}$ over $\mathcal{A} = (Q, \hookrightarrow)$ one can compute a finite set Q', $q'_0 \in Q'$, a pop-free pushdown process $\mathcal{A}' = (Q \times Q', \hookrightarrow')$ and a QPG $\mathcal{G}'$ over $\mathcal{A}'$ such that* val$\mathcal{G}(q, \varepsilon) =$ val$\mathcal{G}'((q, q'_0), \varepsilon)$ *for each* $q \in Q$.

3.2 #MSO Decomposition on Trees

The technique above allows us to reduce model-checking of the temporal part of a Qμ[#MSO] formula to solving a QPG. But to provide algorithms for Qμ[#MSO] we also need a method to handle the counting terms, at least on structures such as words and trees. For MSO, e.g. on trees, this can be done by *decomposing* a formula: instead of checking φ on the whole tree, one can compute tuples of formulas to check on the subtrees. Here we show that this method can be extended to MSO counting terms.

Trees. We consider at most k-branching finite trees with nodes labeled by symbols from a finite alphabet Γ. We accordingly represent them by relational structures over the signature $\tau = \{S_1, \cdots, S_k\} \cup \{P : P \in \Gamma\}$, where each S_i is a binary relation representing the i-th successor. We use P_i and Q_j for the symbols from Γ and write $P - t_1 \cdots t_l$ for the tree with P-labeled root and subtrees t_i. We write t_Q for the tree of height 1 consisting of only the root labeled by Q.

Types. Recall that an m-type in n variables $\tau_{m,n} \subseteq$ MSO is a maximal satisfiable set of formulas with quantifier rank at most m and free variables among $x_1, \ldots, x_n$. We will use Hintikka formulas to finitely represent types, as in the following lemma.

▶ **Lemma 7** (Hintikka Lemma [10]). *Given $m \in \mathbb{N}$ and variables $\overline{x} = x_1, \cdots, x_n$, one can compute a* finite *set $H_{m,n}$ of formulas with quantifier rank m and free variables $\overline{x}$ such that:*

- *For every tree t and vertices $v_1, \cdots, v_n \in t$ there is a* unique *$\tau \in H_{m,n}$ such that $t \vDash \tau(\overline{v})$.*
- *If $\tau_1, \tau_2 \in H_{m,n}$ and $\tau_1 \neq \tau_2$ then $\tau_1 \wedge \tau_2$ is unsatisfiable.*
- *If $\tau \in H_{m,n}$ and $\varphi(\overline{x})$ is a formula with $qr(\varphi) \leq m$, then either $\tau \vDash \varphi$ or $\tau \vDash \neg\varphi$.*

*Furthermore, given such τ and φ, it is computable which of these two possibilities holds. Elements of $H_{m,n}$ are called (m,n)-*Hintikka formulas *and $H_{m,\leq n} = \bigcup_{i \leq n} H_{m,i}$.*

Let us fix a counting term $\#_{\overline{x}}\varphi(\overline{x})$, which we will decompose. In this section, if we omit the quantifier rank m, we mean $m = \mathrm{qr}(\varphi)$. For a tuple $\overline{x} = (x_1, \cdots, x_n)$ of variables we write $[\overline{x}]_l = (\overline{x}_0, \cdots, \overline{x}_l)$ for a partition of $\overline{x}$ into $l+1$ disjoint sets, and $\{[\overline{x}]_l\}$ for the set of all such partitions. Let us first recall the standard MSO decomposition theorem on trees.

▶ **Theorem 8** ([18, 8]). *Let $t = Q - t_1 \cdots t_l$ be a tree and $\varphi(\overline{x})$ an MSO formula. One can compute a finite set $\Phi = \{(\varphi_0(\overline{x}_0), \varphi_1(\overline{x}_1), \cdots, \varphi_l(\overline{x}_l)) \mid (\overline{x}_0, \cdots, \overline{x}_l) \in \{[\overline{x}]_l\}\}$, where all φ_i have a quantifier rank not exceeding that of φ, such that*

$$t \vDash \varphi(\overline{x}) \iff \text{there ex. a } \overline{\varphi} \in \Phi \text{ with } t_Q \vDash \varphi_0(\overline{x}_0) \text{ and } t_i \vDash \varphi_i(\overline{x}_i) \text{ for all } i \in \{1, \ldots, l\}.$$

Observe that the condition above is a disjunction over the $(l+1)$-tuples in Φ of conjunctions over each tuple. We prove a similar decomposition theorem for #MSO in which, intuitively, the disjunctions are replaced by sums and the conjunctions by products. Note that counting terms can count, e.g., the product of the number of as in the left subtree and the number of bs in the right one. But in such case, the free variables in the decomposition are split, and thus the number of terms in a product is limited by the number of free variables. To ensure that assignments are not counted twice, we decompose to Hintikka formulas.

▶ **Theorem 9.** *Let $t = Q - t_1 \cdots t_l$ be a tree and let $\#_{\overline{x}}\varphi(\overline{x})$ be a counting term. Then, for every partition $[\overline{x}]_l = (\overline{x}_0, \cdots, \overline{x}_l)$ of $\overline{x}$, one can compute a finite set $\Psi_{[\overline{x}]_l}$ of $(l+1)$-tuples of Hintikka formulas from $H_{qr(\varphi), \leq |\overline{x}|}$ such that*

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^t = \sum_{[\overline{x}]_l \in \{[\overline{x}]_l\}} \sum_{\overline{\tau} \in \Psi_{[\overline{x}]}} [\![\#_{\overline{x}_0}\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot [\![\#_{\overline{x}_1}\tau_1(\overline{x}_1)]\!]^{t_1} \cdot \ldots \cdot [\![\#_{\overline{x}_l}\tau_l(\overline{x}_l)]\!]^{t_l}.$$

4 Qμ[#MSO] on Tree-Producing Pushdown Systems

In this section we show our main result, namely that Qμ[#MSO] can be effectively evaluated on STSs generated by tree-producing pushdown systems, which generalize both pushdown processes and regular tree grammars with control states and universal application. This is therefore an extension of the classical decidability results for MSO and Lμ on pushdown systems and regular trees to the quantitative setting.

▶ **Definition 10.** An *increasing tree-rewriting rule* for $P \in \Gamma$ has the form $P \leftarrow t$, where t is a Γ-labeled tree of height ≤ 2, i.e. $t = Q - Q_1 \cdots Q_k$ or $t = t_Q$.

Note that increasing tree-rewriting rules are exactly the same as productions in a normalized regular tree grammar. But we apply these rules *universally*, i.e. always to *all* leaves to which a rule can be applied. Formally, for two Γ-labeled trees t_1, t_2 and a rule $r : P \leftarrow t$, we write $t_1 \xrightarrow{r} t_2$, or $r(t_1) = t_2$, if t_2 is obtained from t_1 by replacing every P-labeled leaf by t. We denote the set of all increasing tree-rewriting rules for Γ-labeled trees by $\mathcal{R}_\Gamma$, or just $\mathcal{R}$.

Let us take a starting tree, say t_Q, and apply a sequence of rules $\overline{r} = r_1, \dots, r_n$ resulting in the tree $t = r_n \cdots r_1(t_Q)$. In the qualitative setting, given an MSO sentence φ, one asks which sequences of rules lead to a tree t such that $t \vDash \varphi$. The set of such sequences of rules turns out to be regular (c.f. [11] Theorem 2) and one can effectively construct an automaton to recognize it. Our main technical result, stated below, generalizes this to the quantitative setting of MSO counting terms using integer counters and affine update functions. Recall that a function $f : \mathbb{N}^k \to \mathbb{N}^k$ is affine if $f(\overline{c}) = A\overline{c} + B$ for some matrix $A \in \mathbb{N}^{k \times k}$ and $B \in \mathbb{N}^k$.

▶ **Theorem 11.** *For all $Q \in \Gamma$, $\varphi \in$ MSO, one can compute $k \in \mathbb{N}$, an initial value $I \in \mathbb{N}^k$, an evaluation vector $E \in \mathbb{N}^{1 \times k}$ and an affine update function $\mathrm{up}_r : \mathbb{N}^k \to \mathbb{N}^k$ for each $r \in \mathcal{R}$ such that, for all finite sequences $r_1, \cdots, r_n \in \mathcal{R}^*$,*

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_Q)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I) = E \cdot (\mathrm{up}_{r_n}(\dots(\mathrm{up}_{r_1}(I)))\dots).$$

Also, k, E and the functions up_r depend only on the quantifier rank and free variables of φ.

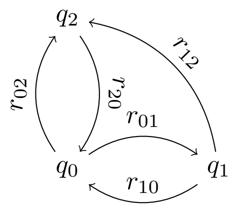
The proof of this theorem is given in Section 6, but we will first show how it can be applied to evaluate Qμ[#MSO] on tree-producing pushdown systems.

▶ **Definition 12.** A *tree-producing pushdown system* (TPPDS) $\mathfrak{T} = (Q, E)$ is a directed graph with $(\mathcal{R} \cup \{\bot\}) \times (\mathcal{R} \cup \{\mathsf{pop}, \varepsilon\})$-labeled edges, i.e. $E \subseteq Q \times (\mathcal{R} \cup \{\bot\}) \times (\mathcal{R} \cup \{\mathsf{pop}, \varepsilon\}) \times Q$.

A TPPDS $\mathfrak{T} = (Q, E)$ with initial tree t_P gives rise to the infinite structure transition system $\mathfrak{S}(\mathfrak{T}) = (S, \Delta, \mathfrak{m})$ with states $S = Q \times \mathcal{R}^*$, the structure assignment $\mathfrak{m}(q, \overline{r}) = \overline{r}(t_P)$ and transitions as in a pushdown process:

$$\begin{aligned}
\Delta = \quad & \big\{\big((q,\varepsilon), r', (q', r')\big) \mid q, q' \in Q,\ (q, \bot, r', q') \in E\big\} \\
\cup & \big\{\big((q,\overline{r}r), r', (q', \overline{r}rr')\big) \mid q, q' \in Q,\ (q, r, r', q') \in E\big\} \\
\cup & \big\{\big((q,\overline{r}r), \varepsilon, (q', \overline{r}r)\big) \mid q, q' \in Q,\ (q, r, \varepsilon, q') \in E\big\} \\
\cup & \big\{\big((q,\overline{r}r), \mathsf{pop}, (q', \overline{r})\big) \mid q, q' \in Q,\ (q, r, \mathsf{pop}, q') \in E\big\}.
\end{aligned}$$

Observe that standard pushdown systems are subsumed by TPPDS. To obtain the stack in the corresponding STS one uses rules where the right-hand side is a tree of branching degree 1, i.e. a word. Properties like stack unboundedness can be formulated in Qμ[#MSO] as was shown in Section 2.



$$\begin{aligned} r_{01} &= P \leftarrow P - PPP \\ r_{02} &= P \leftarrow Q - PP \\ r_{10} &= \mathsf{pop} \\ r_{12} &= P \leftarrow P \\ r_{20} &= P \leftarrow Q - PP \end{aligned}$$

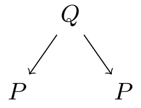
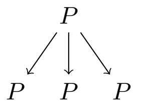
run	$q_0q_1q_0q_2$	$q_0q_1q_2$
tree		

Figure 1 Example of a TPPDS with 2 initial runs from (q_0, t_P).

▶ **Example 13.** Consider the TPPDS in Figure 1. The resulting trees of two initial runs, starting in state q_0 with initial tree t_P, are given in the table on the right.

Let us remark that the underlying transition graph of the STS generated by a TPPDS is always a pushdown graph, no matter whether one uses rules that generate trees or only words. But there is a substantial difference when quantitative questions are asked, e.g. about the size of the structure generated by a run. Trees allow to model richer classes of systems (c.f. Corollary 16), but their size can grow exponentially in the number of rewriting steps and thus they require more refined counting techniques than words.

Our main decidability result, stated below, is a consequence of Theorem 11, the existence of model-checking games (Theorem 5), pop elimination (Theorem 6), and an algorithm to solve counter parity games [1].

▶ **Theorem 14.** *Given a formula $\psi \in Q\mu[\#MSO]$, a TPPDS $\mathfrak{T} = (Q, E)$, a state $q \in Q$ and a starting symbol $P \in \Gamma$, one can compute $[\![\psi]\!]^{\mathfrak{S}(\mathfrak{T})}(q, t_P)$.*

Proof. First, we transform ψ to negation normal form. By Theorem 5, to compute $[\![\psi]\!]^{\mathfrak{S}(\mathfrak{T})}(q, t_P)$ it suffices to determine the value of the game $\mathsf{MC} = \mathsf{MC}(\mathfrak{S}(\mathfrak{T}), (q, t_P)) = (V, V_{\max}, V_{\min}, E, \lambda, \Omega)$ from $((q, \varepsilon), \psi)$. By Theorem 5 (5), the vertices in V have the form $((q, \overline{r}), \psi)$ or $(\pm\infty)$, and V can be infinite. Observe that MC is a pushdown QPG with states $Q \times \mathsf{Sub}(\psi)$. By Theorem 6, MC can be transformed into an equivalent pop-free pushdown QPG MC' with states Q'. We define a finite game $\widehat{\mathsf{MC}}$ with labeled edges which removes the dependence of moves on the top stack symbol and still represents MC' in the sense that the payoff of a play in MC' can be computed from the sequence of visited labels in $\widehat{\mathsf{MC}} := (\widehat{V}, \widehat{V}_{\max}, \widehat{V}_{\min}, \widehat{E}, \widehat{\Omega})$. The positions $\widehat{V} := (Q' \times (\mathcal{R} \cup \{\varepsilon\})) \cup \{(\infty), (-\infty)\}$ have generally the form (q, r) and simply omit all but the last rule r from the stack in the position in MC'. The $(\mathcal{R} \cup \{\varepsilon\})$-labeled edge relation:

$$\begin{aligned} \widehat{E} \;:=\; & \Big\{\big((q, r), r', (q', r')\big) \mid \big((q, \overline{r}r), (q', \overline{r}rr')\big) \in E'\Big\} \\ & \cup \Big\{\big((q, r), \varepsilon, (q', r)\big) \mid \big((q, \overline{r}r), (q', \overline{r}r)\big) \in E'\Big\} \\ & \cup \Big\{\big((q, r), r, (\pm\infty)\big) \mid \big((q, \overline{r}r), (\pm\infty)\big) \in E'\Big\}. \end{aligned}$$

Note that from each position (q, r) in $\widehat{\mathsf{MC}}$ there are exactly the same possible moves as from each position $(q, \overline{r}r)$ in MC'. Since the stack in a position in MC' is a function of the starting tree t_P and the observed rules $\overline{r}$, there is a one-to-one correspondence between plays and strategies in both games.

We define the coloring function $\widehat{\Omega}(q, r) = \Omega'(q, \overline{r}r)$ for any tree $\overline{r}$, and it is well defined since in a QPG the value $\Omega(q, \overline{r}r)$ depends only on q, not on the stack. The payoff of an infinite play in $\widehat{\mathsf{MC}}$ depends on the minimal color seen infinitely often, exactly as in MC', and is thus equal to the payoff of the corresponding play in the original model-checking game. For a finite play $\pi = v_0 \cdot r_1 \cdot v_1 \cdot r_2 \cdot \dots \cdot r_n \cdot v_n$, the last position, by Theorem 5 (5), either corresponds to a formula $\#_{\overline{x}}\varphi(\overline{x})$, or to a formula $\neg\#_{\overline{x}}\varphi(\overline{x})$, or to $(\pm\infty)$. In the first case we set the payoff in $\widehat{\mathsf{MC}}$ to $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}$, in the second one to $-1\cdot[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}$ and for $(\pm\infty)$ to $\pm\infty$. This definition of payoffs clearly ensures that corresponding plays in MC' and $\widehat{\mathsf{MC}}$ result in the same outcome, and due to the one-to-one correspondence mentioned above, the value of MC' from $((q,\varepsilon),\psi)$ is the same as the value of $\widehat{\mathsf{MC}}$ from $((q,\varepsilon),\psi)$, which we compute below, thus also solving MC.

Let $\{\varphi_1,\dots,\varphi_l\}$ be an enumeration of the MSO formulas in ψ which are counted, i.e. of $\{\varphi \mid \#_{\overline{x}}\,\varphi(\overline{x}) \in \mathsf{Sub}(\psi)\}$. By Theorem 11, for each such φ_i there is the corresponding dimension k^i, initial values I^i, evaluation vector E^i and update functions up^i_r. We combine the initial values to an aggregate initial $I = \langle I^1,\dots,I^l\rangle$, which is a vector of dimension $k = k^1 + \dots + k^l$. The aggregate update functions are also composed component-wise:

$$\mathrm{up}_r(\langle\overline{c}^1,\dots,\overline{c}^l\rangle) = \langle\mathrm{up}^1_r(\overline{c}^1),\dots\mathrm{up}^l_r(\overline{c}^l)\rangle,$$

and we extend each evaluation vector E^i to a vector $\widehat{E}^i$ of dimension k by filling it with 0s on all dimensions except for the k^i ones. By Theorem 11, we have that

$$[\![\#_{\overline{x}}\varphi_i(\overline{x})]\!]^{r_n\cdots r_1(t_P)} = \widehat{E}^i\cdot(\mathrm{up}_{r_1}\circ\cdots\circ\mathrm{up}_{r_s})(I). \tag{1}$$

Let us thus use the aggregate functions to transform $\widehat{\mathsf{MC}}$ into a game $\widetilde{\mathsf{MC}}$, played on the same arena, in which in each move an affine function is applied to a vector of k integers. To construct $\widetilde{\mathsf{MC}}$, we replace every edge label r in $\widehat{\mathsf{MC}}$ by the function up_r and the payoff function λ in $\widetilde{\mathsf{MC}}$, depending on the current value of the vector $\overline{c}$, is given by

$$\lambda(s,\overline{c}) = \begin{cases} \pm\infty & \text{if } s = (\pm\infty),\\ E^i\cdot\overline{c} & \text{if } s = (p,\ \#_{\overline{x}}\varphi_i(\overline{x})),\\ -E^i\cdot\overline{c} & \text{if } s = (p,\ \neg\#_{\overline{x}}\varphi_i(\overline{x})). \end{cases}$$

By (1), the payoffs of corresponding plays in $\widehat{\mathsf{MC}}$ and in $\widetilde{\mathsf{MC}}$ are the same, and due to the one-to-one correspondence of moves, plays and strategies we also get that the value of $\widetilde{\mathsf{MC}}$ from $((q,\varepsilon),\psi)$ starting with vector I is the same as the value of $\widehat{\mathsf{MC}}$ from this position, and thus equal to $[\![\psi]\!]^{\mathfrak{S}(\mathfrak{T})}(q,t_P)$. But the game $\widetilde{\mathsf{MC}}$ is a special case of a *counter parity game* with k counters [1], and, as proved in [1], its value can be computed. ◀

The above theorem demonstrates that $\mathsf{Q}\mu[\#\mathsf{MSO}]$ indeed allows to apply the methods known for qualitative logics to the quantitative case. As TPPDS subsume pushdown systems, we obtain the following corollary.

▶ **Corollary 15.** *Given a formula $\psi \in$ Qμ[#MSO], a pushdown process $\mathcal{A} = (Q,\hookrightarrow)$ generating an STS $\mathfrak{P}$, and a state $q_0 \in Q$, one can compute $[\![\psi]\!]^{\mathfrak{P}}(q_0,\bot)$.*

Furthermore, the class of tree-producing pushdown systems includes finite systems and, as shown in [11], MSO formulas on separated structure rewriting systems can also be reduced to formulas to be checked on a TPPDS.

▶ **Corollary 16.** *Given a formula $\psi \in$ Qμ[#MSO], a separated structure rewriting system (c.f. [11]) generating an STS $\mathfrak{S}$, and an initial state $s \in \mathfrak{S}$, one can compute $[\![\psi]\!]^{\mathfrak{S}}(s)$.*

Before, we reduced model-checking $\mathsf{L}\mu[\mathsf{MSO}]$ to computing the values of $\mathsf{Q}\mu[\#\mathsf{MSO}]$ formulas. Thus, we can say that Corollary 15 strictly generalizes previous results on model-checking $\mathsf{LTL}[\mathsf{MSO}]$ and $\mathsf{L}\mu[\mathsf{MSO}]$ on pushdown systems [13, 6] and Corollary 16 subsumes the result from [11] for $\mathsf{L}\mu[\mathsf{MSO}]$ on separated structure rewriting systems. On the quantitative side, these corollaries also subsume the result from [7] for $\mathsf{Q}\mu$ on finite systems and model-checking parity conditions with unboundedness on pushdown systems [2, 9].

5 Eliminating pop from Pushdown QPGs

In this section, we prove Theorem 6 using methods similar to the ones developed in [19] and later in [3] and [15], but with a construction symmetric with respect to the players.

▶ **Theorem 6.** *For every pushdown QPG $\mathcal{G}$ over $\mathcal{A} = (Q, \hookrightarrow)$ one can compute a finite set Q', $q'_0 \in Q'$, a pop-free pushdown process $\mathcal{A}' = (Q \times Q', \hookrightarrow')$ and a QPG $\mathcal{G}'$ over $\mathcal{A}'$ such that $\mathsf{val}\mathcal{G}(q, \varepsilon) = \mathsf{val}\mathcal{G}'((q, q'_0), \varepsilon)$ for each $q \in Q$.*

Construction of the game $\mathcal{G}'$. Intuitively, $\mathcal{G}'$ simulates $\mathcal{G}$, but in each push-move the player makes claims about minimal colors that will be seen in $\mathcal{G}$ until the stack pops back to the same content, if it does. The opponent is then asked to either proceed as if the claim happened – moving to a claimed state with one of the claimed colors – or to allow the push and accept to lose if a pop satisfying the claim occurs later.

To construct the set Q', let d be the maximal color assigned by Ω in $\mathcal{G}$ and $[d] = \{0, \ldots, d\}$. We consider the set of *claims* defined as $\mathcal{C} = \{C : Q \to \mathcal{P}([d])\}$, i.e. a claim C assigns to each state a set of colors. We define Q' as the disjoint union of three kinds of states, $Q' = \{\bot\} \cup Q'_0 \cup Q'_1$: the state $\bot$ for the empty stack, $Q'_0 = [d] \times \mathcal{C} \times \{\max, \min\}$ for positions where players make claims, and the set $Q'_1 = \Gamma \times \mathcal{C} \times \{\max, \min\} \times (\{\bot\} \cup Q'_0)$ for positions where players answer to claims.

We construct the relation $\hookrightarrow'$ and the game $\mathcal{G}'$ in the following way. Positions $(q, \bot, \bot)$ and (q, m, C, p, s) belong to the same player to whom q belongs in $\mathcal{G}$. The epsilon-moves from $\hookrightarrow$, i.e. the ones which do not change the stack, are preserved in $\mathcal{G}'$ and lead to $(q', \bot, \bot)$ or, respectively, to $(q', \min(m, \Omega(q')), C, p, s)$ updating the minimal color for C. The pop-moves do not occur in $\mathcal{G}'$, but, if a pop-move to q' was allowed in $\mathcal{G}$ from (q, s), then in $\mathcal{G}'$ there is a move from (q, m, C, p, s) to a sink position. This sink position is winning for player p who made the claim if the claim was true, and winning for the opponent otherwise. Formally, it has payoff $+\infty$ if $p = \max$ and $\min(m, \Omega(q')) \in C(q')$ (true claim) or if $p = \min$ and $\min(m, \Omega(q')) \notin C(q')$ (false claim), and $-\infty$ otherwise. The payoff at a terminal position (q, x, s) in $\mathcal{G}'$ is the same as the payoff in (q, s) in $\mathcal{G}$.

The push-moves from $\mathcal{G}$ translate to claims made in $\mathcal{G}'$ as depicted in Figure 2. If $\hookrightarrow$ allowed to push $a \in \Gamma$ in $\mathcal{G}$ from (q, s) leading to a state q', then in $\mathcal{G}'$ we add the moves from (q, x, s), for $x = \bot$ or $x \in Q'_0$, to (q', a, C, p, x, s), where C is any claim and p is the player to whom q belongs in $\mathcal{G}$, i.e. the one who makes the claim.

Positions where claims are answered, i.e. of the form (q', a, C, p, x, s), belong to the opponent of the player p who just made the claim. From each such position there is exactly one possible push-move leading to $(q', \Omega(q'), C, p, sa)$, i.e. the push is made as intended. Additionally, for each color d_c and state $q_c \in Q$ such that $d_c \in C(q_c)$, there is a move to the position (q_c, x', s) which corresponds to playing as in the claim. In this case, if $x = \bot$ then $x' = \bot$ as well and we set $\Omega(q_c, \bot, \bot) = \min(\Omega(q_c), d_c)$. If $x = (m, C_0, p_0)$, then $x' = (\min(m, d_c), C_0, p_0)$, i.e. the minimal color is updated, and we again set $\Omega(q_c, x', s) = \min(\Omega(q_c), d_c)$.

Correctness of the construction. Let σ be a strategy of one of the players in $\mathcal{G}$. We define the corresponding truthful strategy σ' in $\mathcal{G}'$ by induction on the length of play prefixes. Also, to each play prefix π' consistent with σ' we assign a corresponding play prefix π in $\mathcal{G}$ consistent with σ. Intuitively, when the player is supposed to make a claim in $\mathcal{G}'$, he considers all plays extending π in $\mathcal{G}$ and consistent with σ and, in σ', chooses a claim with the colors that really occur. When the opponent makes a claim in $\mathcal{G}'$ and indeed there is an extension consistent with σ which returns to the same stack in the claimed state and

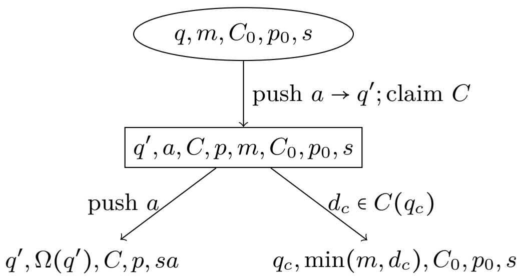

q, m, C_0, p_0, s
push $a \to q'$; claim C
$q', a, C, p, m, C_0, p_0, s$
push a
$d_c \in C(q_c)$
$q', \Omega(q'), C, p, sa$
$q_c, \min(m, d_c), C_0, p_0, s$

Figure 2 Claims and moves corresponding to push operations.

color, then σ' accepts this claim, and the corresponding play in $\mathcal{G}$ is prolonged to the point in which the stack pops back to the claimed state. If no play is consistent with the claim of the opponent, i.e. it is false, then the push-move is made. A formal proof of correctness is similar to the one in Chapter 5 of [17].

6 #MSO Evaluation Using Counters

In this section we prove Theorem 11. Let us therefore fix a symbol $P \in \Gamma$ and an MSO formula φ. This also fixes the quantifier rank $m = \mathrm{qr}(\varphi)$ and $\overline{x}$ as the free variables of φ. We will prove, for an arbitrary sequence $r_1, \ldots, r_n \in \mathcal{R}^*$, that $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I)$.

The proof will be by induction on the length of the sequence $r_1, \ldots, r_n$; the appropriate I, E and up_r will be constructed in the process. For clarity, we omit the easy case of rules $P \leftarrow t_Q$ and we first assume that the types of the subtrees $r_n \cdots r_{i+1}(t_P)$ are known. In subsection 6.1 we show how these types can be guessed and checked afterwards, and in 6.2 we provide the final construction and proof of Theorem 11.

Notation. We consider the sequence of rules $r_1, \ldots, r_n$ and denote the tree after i rewritings by t_i, i.e. the rewriting sequence can be written as $t_P \xrightarrow{r_1} t_1 \xrightarrow{r_2} \cdots \xrightarrow{r_n} t_n$. We will also evaluate terms on the subtrees which result from rewriting a symbol $P \in \Gamma$ from the i-th step on, i.e. on $r_n \cdots r_{i+1}(t_P)$. For a Hintikka formula $\tau \in H_{m,\le|\overline{x}|}$ and a symbol $P \in \Gamma$ we write $[\![\tau(\overline{y}), P]\!]^i := [\![\#_{\overline{y}}\tau(\overline{y})]\!]^{r_n\cdots r_{i+1}(t_P)}$, i.e. $[\![\tau, P]\!]^i$ is the number of tuples $\overline{a}$ such that $\tau(\overline{a})$ holds on the subtree generated from P in the last $n - i$ rewriting steps.

By Λ we denote the set of all unordered sequences $\left(\overline{\tau, P}\right) = \{(\tau_1(\overline{y}_1), P_1), \cdots, (\tau_k(\overline{y}_k), P_k)\}$, where $P_i \in \Gamma$, $(\overline{y}_1, \cdots, \overline{y}_k)$ is a partition into *nonempty* sets $\overline{y}_i \ne \varnothing$ of a subset $\overline{y} \subseteq \overline{x}$ of free variables of φ, and each $\tau_i \in H_{m,\le|\overline{x}|}$ is a Hintikka formula with free variables $\overline{y}_i$. We will use elements of Λ as *indices*, i.e. we will operate on a vector $c \in \mathbb{N}^{|\Lambda|}$ and we will write $c\left[\left(\overline{\tau, P}\right)\right]$ for the number in c corresponding to position $\left(\overline{\tau, P}\right)$. Let us prove the first induction step.

▶ **Lemma 17** (First step). *There exists a $c \in \mathbb{N}^{|\Lambda|}$ such that*

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)} = \sum_{\left(\overline{\tau,P}\right)\in\Lambda} c\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau,P)\in\left(\overline{\tau,P}\right)} [\![\tau, P]\!]^1.$$

Proof. Assume that $r_1 = P \leftarrow Q - Q_1 \cdots Q_l$. By Lemma 9, $[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}$ equals

$$\sum_{[\overline{x}]_l \in \{[\overline{x}]_l\}} \sum_{\overline{\tau}\in\Psi_{[\overline{x}]_l}} [\![\#_{\overline{x}_0}\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot [\![\#_{\overline{x}_1}\tau_1(\overline{x}_1)]\!]^{r_n\cdots r_2(Q_1)} \cdot \ldots \cdot [\![\#_{\overline{x}_l}\tau_l(\overline{x}_l)]\!]^{r_n\cdots r_2(Q_l)}.$$

Using the notation introduced above, each product in the inner sum can be written as $[\![\tau_0(\overline{x}_0)]\!]^{t_Q} \cdot \prod_{i=1}^{l}[\![\tau_i(\overline{x}_i), Q_i]\!]^1$.

In each inner product $\prod_{i=1}^{l}[\![\tau_i(\overline{x}_i),Q_i]\!]^1$, the factors $[\![\tau_i(\overline{x}_i),Q_i]\!]^1$ in which τ_i is a sentence are known to be either 0 or 1. If they are 1, they can safely be omitted, otherwise the whole product becomes 0. Furthermore, if, after the removal, two inner products coincide for two different τ_0 and τ_0', then the sum of the two outer products can be written as

$$[\![\tau_0(\overline{x}_0)]\!]^{t_Q}\cdot\prod_{i=1}^{l}[\![\tau_i(\overline{x}_i),Q_i]\!]^1+[\![\tau_0'(\overline{x}_0)]\!]^{t_Q}\cdot\prod_{i=1}^{l}[\![\tau_i(\overline{x}_i),Q_i]\!]^1$$
$$=\left([\![\tau_0]\!]^{t_Q}+[\![\tau_0']\!]^{t_Q}\right)\cdot\prod_{i=1}^{l}[\![\tau_i(\overline{x}_i),Q_i]\!]^1.$$

We set $c\left[\left(\overline{\tau,P}\right)\right]$ to the sum of the $[\![\tau_0(\overline{x}_0)]\!]^{t_Q}$ over all products where the inner products coincide with $\left(\overline{\tau,P}\right)$, i.e. to the sum of those where $\{(\tau_i,Q_i)\mid 1\leq i\leq l,\overline{x}_i\neq\varnothing\}=\left(\overline{\tau,P}\right)$. ◀

Note that the vector c constructed above depends only on the first rule r_1 and which *sentences* τ_i hold in $r_n\cdots r_2(t_P)$, for each P. We will remove the dependence on the types τ_i in subsection 6.1. First, we show that the sum of the above form can be maintained further into the rewriting sequence and that the vector c only needs to be updated in a linear way.

▶ **Lemma 18** (Induction step)**.** *Let $c\in\mathbb{N}^{|\Lambda|}$ be a vector such that*

$$(S):=[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n\cdots r_1(t_P)}=\sum_{\left(\overline{\tau,P}\right)\in\Lambda}c\left[\left(\overline{\tau,P}\right)\right]\cdot\prod_{(\tau,P)\in\left(\overline{\tau,P}\right)}[\![\tau,P]\!]^{i-1}.$$

Then there exists a vector $\widetilde{c}\in\mathbb{N}^{|\Lambda|}$ such that

$$(S)=\sum_{\left(\overline{\tau,P}\right)\in\Lambda}\widetilde{c}\left[\left(\overline{\tau,P}\right)\right]\cdot\prod_{(\tau,P)\in\left(\overline{\tau,P}\right)}[\![\tau,P]\!]^{i},$$

and each $\widetilde{c}\left[\left(\overline{\tau,P}\right)\right]$ can be computed as a linear combination of the numbers $c\left[\left(\overline{\tau,P}\right)\right]$.

Proof. Let $r_i=R\leftarrow Q-Q_1\cdots Q_l$ be the i-th rule. By applying Lemma 17 to the sequence $r_n\cdots r_i(t_R)$, we get that for all τ there exists c' such that:

$$[\![\tau(\overline{y}),R]\!]^{i-1}=\sum_{\left(\overline{\tau,P}\right)\in\Lambda}c'\left[\left(\overline{\tau,P}\right)\right]\prod_{(\tau,P)\in\left(\overline{\tau,P}\right)}[\![\tau,P]\!]^{i}.\tag{2}$$

For symbols $P\neq R$, the trees $r_n\cdots r_i(t_P)$ and $r_n\cdots r_{i+1}(t_P)$ coincide, thus $[\![\tau,P]\!]^{i-1}=[\![\tau,P]\!]^{i}$.

Recall the sum for position $i-1$. This sum can be split into two parts, namely over those $\left(\overline{\tau,P}\right)$ where R does not occur, and those where it does:

$$(S)=\underbrace{\sum_{\left(\overline{\tau,P}\right)\in\Lambda,R\notin\overline{P}}c\left[\left(\overline{\tau,P}\right)\right]\cdot\prod_{(\tau,P)\in\left(\overline{\tau,P}\right)}[\![\tau,P]\!]^{i-1}}_{(A)}$$
$$+\underbrace{\sum_{\left(\overline{\tau,P}\right)\in\Lambda,R\in\overline{P}}c\left[\left(\overline{\tau,P}\right)\right]\cdot\prod_{(\tau,P)\in\left(\overline{\tau,P}\right)}[\![\tau,P]\!]^{i-1}}_{(B)}.$$

For (A), it holds that all $[\![\tau,P]\!]^{i-1}$ are equal to their corresponding $[\![\tau,P]\!]^{i}$. For (B), the sum is updated: when considering a summand $c\left[\left(\overline{\tau,P}\right)\right]\prod_{\tau,P}[\![\tau,P]\!]^{i-1}$ with $R\in\overline{P}$, each factor $[\![\tau,R]\!]^{i-1}$ is replaced by the corresponding sum from (2), resulting in:

$$c\left[\left(\overline{\tau,P}\right)\right]\left(\prod_{\tau,P,P\neq R}[\![\tau,P]\!]^{i}\right)\cdot\left(\prod_{\tau,R}\underbrace{\left(\sum_{\left(\overline{\tau,P}\right)'}c'\left[\left(\overline{\tau,P}\right)'\right]\prod_{\tau',P'}[\![\tau',P']\!]^{i}\right)\Bigg)}_{\text{right-hand side of (2) for }[\![\tau,R]\!]^{i-1}}.$$

As in the decomposition for a $[\![\tau, R]\!]^{i-1}$ the only free variables $\overline{z}$ are those of τ, it follows that $c'\left[\left(\overline{\tau, P}\right)\right] = 0$ for all $\left(\overline{\tau, P}\right)$ where some $(\tau, P) \in \left(\overline{\tau, P}\right)$ has free variables other than $\overline{z}$. Note that, for all $[\![\tau, P]\!]^{i}$ with $R \neq P$ that already were there before inserting the right-hand sides of (2), the τ have free variables disjoint from $\overline{z}$. Thus, when multiplied out after omitting all summands with $c'[\left(\overline{\tau, P}\right)] = 0$, we obtain a sum over Λ of c-weighted products. This can be joined with (A) to obtain $(S) = \sum_{\left(\overline{\tau,P}\right)\in\Lambda} \widetilde{c}\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau,P)\in\left(\overline{\tau,P}\right)} [\![\tau, P]\!]^{i}$, where $\widetilde{c}$ is as follows.

Calculating $\widetilde{c}$. We show now how to linearly combine different entries of c to get the entries of $\widetilde{c}$. To this end, we create, for all $\sigma \in H_{m,\leq|\overline{x}|}$, a set $\mathcal{D}_\sigma$, collecting all $\left(\overline{\tau, P}\right)$ from the right-hand side of (2) for $[\![\sigma, R]\!]^{i-1}$ along with the respective $c'\left[\left(\overline{\tau, P}\right)\right]$.

$$\mathcal{D}_\sigma = \left\{ \left(\left(\overline{\tau, P}\right), c'\left[\left(\overline{\tau, P}\right)\right]\right) \mid \left(\overline{\tau, P}\right) \text{occurs in right-hand side of (2) for } [\![\sigma, R]\!]^{i-1} \right\}.$$

Since only positive values of $c'\left[\left(\overline{\tau, P}\right)\right]$ are of interest, we collect all $\mathcal{D}_\sigma$ in the following set:

$$\mathcal{D} = \left\{ \left(\sigma, \left(\overline{\tau, P}\right), d\right) \mid \sigma \in H_{m,\leq|\overline{x}|},\ \left(\left(\overline{\tau, P}\right), d\right) \in \mathcal{D}_\sigma,\ d > 0 \right\}.$$

To determine the value of $\widetilde{c}\left[\left(\overline{\tau, P}\right)\right]$, all those $c\left[\left(\overline{\tau, P}\right)'\right]$ have to be considered in which a factor $[\![\sigma, R]\!]^{i-1}$ is replaced, and, when multiplying out, a product over $\left(\overline{\tau, P}\right)$ results. We do this by adding — for every subset $\left(\overline{\tau, P}\right)'$ of $\left(\overline{\tau, P}\right)$ and every τ such that $c'\left[\left(\overline{\tau, P}\right)'\right]$ appears on the right-hand side of (2) for $[\![\sigma, R]\!]^{i-1}$ — the entries in c for the set obtained from $\left(\overline{\tau, P}\right)$ by replacing $\left(\overline{\tau, P}\right)'$ with (σ, R), weighted with the respective $c'\left[\left(\overline{\tau, P}\right)'\right]$. This amounts to a sum over the set $\mathcal{D}$ defined above. In addition, if $R \notin \overline{P}$, we have to copy the old counter value, as in the respective subtrees nothing was changed. Thus, let $\text{old}\left[\left(\overline{\tau, P}\right)\right] = c\left[\left(\overline{\tau, P}\right)\right]$ if $R \notin \overline{P}$ and $\text{old}\left[\left(\overline{\tau, P}\right)\right] = 0$ otherwise. We obtain the following equation.

$$\widetilde{c}\left[\left(\overline{\tau, P}\right)\right] = \text{old}\left[\left(\overline{\tau, P}\right)\right] + \sum_{\left(\overline{\tau,P}\right)' \subseteq \left(\overline{\tau,P}\right)} \sum_{\left(\sigma, \left(\overline{\tau,P}\right)', d\right) \in \mathcal{D}} d \cdot c\left[\left(\overline{\tau, P}\right) \setminus \left(\overline{\tau, P}\right)' + (\sigma, R)\right]. \tag{3}$$

◀

Note that the construction of $\widetilde{c}$ above depends only on the vectors c' from the right-hand sides of (2), which are in turn obtained from Lemma 17 and depend only on the rule r_i and which Hintikka sentences hold in $r_n \cdots r_{i+1}(t_P)$, for each P.

6.1 Guessing Types of Subtrees

In this subsection, we remove the dependency on the Hintikka sentences described above by *correctly guessing* the types of the subtrees. To this end, we will extend the vector c we operate on by additional entries, corresponding to *type functions* which guess the types.

We represent types by Hintikka sentences: for any tree t, the m-type $\text{tp}^m(t)$ is the unique $\tau \in H_{m,0}$ such that $t \models \tau$.

▶ **Definition 19.** For a set of symbols Γ and a natural number m, a *type function* $L \colon \Gamma \to H_{m,0}$ assigns an m-type to every symbol from Γ.

Notice that, for every finite Γ and every m, there are only finitely many type functions, thus the set $\mathcal{L} = \{L \colon \Gamma \to H_{m,0}\}$ (for our fixed Γ and $m = \text{qr}(\varphi)$) is finite. Denote by L_F the unique type function that assigns to every P the type of t_P. For a sequence $r_1, \ldots, r_n$ of rules from $\mathcal{R}$ we say that $L_0, \ldots, L_n$ is the *compatible* sequence of type functions if $L_i(P) = \text{tp}^m(r_n \cdots r_{i+1}(t_P))$ for all P. Note that $L_n = L_F$. We show that compatible type functions can be computed *backwards* and in a *uniform way*.

▶ **Lemma 20.** *There exists a computable function* $\mathrm{pre}\colon \mathcal{L} \times \mathcal{R} \to \mathcal{L}$ *such that for all sequences* $r_1, \dots, r_n$ *of rules,* pre *together with* L_F *induces a compatible sequence, i.e.* $L_0, \cdots, L_n$ *with* $L_n = L_F$ *and* $L_{i-1} = \mathrm{pre}(L_i, r_i)$ *is compatible.*

Instead of updating the vector c depending on the correct types, we will now maintain a separate vector c_L for every type function L. Each vector c_L will be updated as if the types given by L were the correct ones, and the types will be updated by pre, i.e. we define $\widetilde{c_L}\left[\left(\overline{\tau, P}\right)\right]$ exactly as in (3) but using $c_{\mathrm{pre}(L,r_i)}\left[\left(\overline{\tau, P}\right)\right]$ for $c\left[\left(\overline{\tau, P}\right)\right]$. At the end, the correct $L = L_F$ will be chosen and the above lemma guarantees correctness, as proved below.

6.2 Proof of Theorem 11

▶ **Theorem 11.** *For all* $Q \in \Gamma$, $\varphi \in$ *MSO, one can compute* $k \in \mathbb{N}$*, an initial vector* $I \in \mathbb{N}^k$*, an evaluation vector* $E \in \mathbb{N}^{1 \times k}$ *and an affine update function* $\mathrm{up}_r : \mathbb{N}^k \to \mathbb{N}^k$ *for each* $r \in \mathcal{R}$ *such that, for all finite sequences* $r_1, \cdots, r_n \in \mathcal{R}^*$,

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_Q)} = E \cdot (\mathrm{up}_{r_1} \circ \cdots \circ \mathrm{up}_{r_n})(I).$$

Also, E, k *and the functions* up_r *depend only on the quantifier rank and free variables of* φ*.*

Proof. Let m be the quantifier rank of φ and Λ and $\mathcal{L}$ as defined above. We set $k := |\mathcal{L}| \cdot |\Lambda|$. To define the initial vector I, we compute the Hintikka disjunction $\tau_1 \vee \cdots \vee \tau_s \equiv \varphi$. For every $L \in \mathcal{L}$, we define the vector $I_L \in \mathbb{N}^{|\Lambda|}$ in which all positions are set to 0, except for those where $\left(\overline{\tau, P}\right) = (\tau_i, Q)$, for $1 \le i \le s$, which are set to 1. We fix an enumeration $L_1, \cdots, L_{|\mathcal{L}|}$ of $\mathcal{L}$, and set $I := \langle I_{L_1}, \cdots, I_{L_{|\mathcal{L}|}} \rangle \in \mathbb{N}^k$.

Let pre be the function from Lemma 20. For every $r \in \mathcal{R}$, we define the update function $\mathrm{up}_r(\langle c_1, \cdots, c_{|L|} \rangle) := \langle \widetilde{c_1}, \cdots, \widetilde{c_{|L|}} \rangle$, where $\widetilde{c_L}\left[\left(\overline{\tau, P}\right)\right]$ is set as in (3) but with $c_{\mathrm{pre}(L,r_i)}$ used in place of c. By Lemmas 18 and 20, after the last step, the vector c_{L_F} contains the entries evaluated for the compatible type sequence, and thus the value of the counting term

$$[\![\#_{\overline{x}}\varphi(\overline{x})]\!]^{r_n \cdots r_1(t_Q)} = \sum_{\left(\overline{\tau, P}\right) \in \Lambda} c_{L_F}\left[\left(\overline{\tau, P}\right)\right] \cdot \prod_{(\tau, P) \in \left(\overline{\tau, P}\right)} [\![\tau]\!]^{t_P}.$$

Note that every $[\![\tau]\!]^{t_P}$ is a constant and so is $\pi_{\left(\overline{\tau, P}\right)} := \prod_{(\tau, P) \in \left(\overline{\tau, P}\right)} [\![\tau]\!]^{t_P}$ for every $\left(\overline{\tau, P}\right)$. Thus, the above sum is equal to the product of the updated vector c with the row vector E, which has $\pi_{\left(\overline{\tau, P}\right)}$ in the component corresponding to $c_{L_F}\left[\left(\overline{\tau, P}\right)\right]$ and 0 everywhere else. ◀

7 Conclusion

The question how well-known methods used for regular languages can be extended to the quantitative setting has been approached recently from several angles. From the side of automata, this problem has been investigated in [4] and related papers with focus on counting, and from the logic side, progress has been made in the study of quantitative temporal logics [7]. The counting logic Qμ[#MSO] introduced in this work is the first formalism which exhibits the desirable properties of a well-behaved quantitative temporal logic and allows at the same time to apply decomposition techniques, the logical counterpart of automata. As we show, the value of a Qμ[#MSO] formula is computable on an important class of infinite structure transition systems which generalize both pushdown systems and regular tree grammars. This raises interesting new questions whether this result can be extended, e.g. if the value of a Qμ[#MSO] formula is computable on higher-order pushdown systems or recursion schemes, and in general which results for Boolean logics can be generalized to the counting case. The decomposition theorem we proved for Qμ[#MSO] allows us to conjecture that many results which rely on automata techniques can indeed be extended to Qμ[#MSO], and that it may lead to a canonical quantitative counting logic for regular languages.

Acknowledgment. We are very grateful to Igor Walukiewicz for pointing out that the methods from [19] can also be applied in the quantitative setting, as is done in Section 5.

References

1 D. Berwanger, Ł. Kaiser, and S. Lessenich. Solving counter parity games. In *Proc. of MFCS'12*, LNCS. Springer, 2012. To appear.

2 A.-J. Bouquet, O. Serre, and I. Walukiewicz. Pushdown games with unboundedness and regular conditions. In *Proc. of FSTTCS'03*, vol. 2914 of *LNCS*, pp. 88–99. Springer, 2003.

3 A. Carayol, M. Hague, A. Meyer, C.-H. L. Ong, and O. Serre. Winning regions of higher-order pushdown games. In *Proc. of LICS'08*, pp. 193–204. IEEE Computer Society, 2008.

4 T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proc. of ICALP'09, Part II*, vol. 5556 of *LNCS*, pp. 139–150. Springer, 2009.

5 J. Esparza. On the decidability of model checking for several μ-calculi and Petri nets. In *Proc. of CAAP'94*, vol. 787 of *LNCS*, pp. 115–129. Springer, 1994.

6 J. Esparza, A. Kučera, and S. Schwoon. Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.*, 186(2):355 – 376, 2003.

7 D. Fischer, E. Grädel, and Ł. Kaiser. Model checking games for the quantitative μ-calculus. *Theory Comput. Syst.*, 47(3):696–719, 2010.

8 T. Ganzow and Ł. Kaiser. New algorithm for weak monadic second-order logic on inductive structures. In *Proc. of CSL'10*, vol. 6247 of *LNCS*, pp. 366–380. Springer, 2010.

9 H. Gimbert. Parity and exploration games on infinite graphs. In *Proc. of CSL'04*, vol. 3210 of *LNCS*, pp. 56–70. Springer, 2004.

10 J. Hintikka. Distributive normal forms in the calculus of predicates. *Acta Philosophica Fennica*, 6, 1953.

11 Ł. Kaiser. Synthesis for structure rewriting systems. In *Proc. of MFCS'09*, vol. 5734 of *LNCS*, pp. 415–427. Springer, 2009.

12 R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

13 O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proc. of CAV'00*, vol. 1855 of *LNCS*, pp. 36–52. Springer, 2000.

14 E. W. Mayr. An algorithm for the general Petri net reachability problem. In *Proc. of STOC'81*, pp. 238–246. ACM, 1981.

15 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In *Proc. of ICALP'11 (2)*, vol. 6756 of *LNCS*, pp. 162–173. Springer, 2011.

16 O. Serre. Note on winning positions on pushdown games with ω-regular conditions. *Inf. Process. Lett.*, 85(6):285–291, 2003.

17 O. Serre. *Contribution à l'étude des jeux sur des graphes de processus à pile.* PhD thesis, Université Paris 7, 2004.

18 S. Shelah. The monadic theory of order. *Ann. Math.*, 102:379–419, 1975.

19 I. Walukiewicz. Pushdown processes: Games and model checking. In *Proc. of CAV '96*, vol. 1102 of *LNCS*, pp. 62–74. Springer, 1996.

Parametricity in an Impredicative Sort

Chantal Keller[1] and Marc Lasson[2]

1 INRIA Saclay–Île-de-France at École Polytechnique
Chantal.Keller@inria.fr

2 École Normale Supérieure de Lyon, Université de Lyon, LIP *
marc.lasson@ens-lyon.org

Abstract

Reynold's abstraction theorem is now a well-established result for a large class of type systems. We propose here a definition of relational parametricity and a proof of the abstraction theorem in the Calculus of Inductive Constructions (CIC), the underlying formal language of Coq, in which parametricity relations' codomain is the impredicative sort of propositions. To proceed, we need to refine this calculus by splitting the sort hierarchy to separate informative terms from non-informative terms. This refinement is very close to CIC, but with the property that typing judgments can distinguish informative terms. Among many applications, this natural encoding of parametricity inside CIC serves both theoretical purposes (proving the independence of propositions with respect to the logical system) as well as practical aspirations (proving properties of finite algebraic structures). We finally discuss how we can simply build, on top of our calculus, a new reflexive Coq tactic that constructs proof terms by parametricity.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Calculus of Inductive Constructions; parametricity; impredicativity; Coq; universes.

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.381

1 Introduction

The Coq system [24] is a proof assistant based on the Curry-Howard correspondence: propositions are represented as types and their proofs are their inhabitants. The underlying type system is called the Calculus of Inductive Constructions (CIC in short). In this type system, types and their inhabitants are expressions built from the same grammar and every well-formed expression has a type.

One specificity of Coq among other interactive theorem provers based on Type Theory is the presence of an impredicative sort to represent the set of propositions: Prop. Impredicativity means that propositions may be built by quantification over objects which types inhabit any sort, including the sort of propositions (for instance the Agda language has a similar type system except that propositions live in predicative universes [18]). This sort plays a decisive role in the Coq system: in addition to guaranteeing the compositionality of the propositional world, it contains the non-computational content, i.e., expressions meant to be erased by the program extraction process. In particular, it allows the user to add axioms (like the law of excluded middle, axiom of choice, proof irrelevance, etc...) without jeopardizing program extraction.

* UMR 5668 CNRS ENS Lyon UCBL INRIA

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 381–395
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The other sorts are a predicative hierarchy of universes called $\texttt{Type}_0, \texttt{Type}_1, \ldots$. Contrary to Prop, it is stratified: one is not allowed to form a type of a given universe by quantifying over objects of types of higher universes (stratification has been introduced in order to overcome Girard's paradox, see [7] for details). The sort $\texttt{Type}_0$ (also called Set) contains data-types and basic informative types. And $\texttt{Type}_1$ contains types that are quantified over elements of $\texttt{Type}_0$, and so on.

One major component of Coq is its extraction mechanism [15], which produces an untyped term of a ML-like language from any well typed term of Coq. One obvious interest is to obtain certified ML code. Roughly speaking, it proceeds by replacing type annotations and propositional subterms by a dummy constant. A difficulty of program extraction is to decide which terms are informative and which may be erased. The presence of the sort Prop only partially solves this problem in Coq since the system has to distinguish computations over data types from computations over types, although they all live in Type.

In this paper, we propose a new calculus which refines the Calculus of Inductive Constructions, called $\mathsf{CIC_r}$. By adding a new predicative hierarchy of sorts $\texttt{Set}_0$, $\texttt{Set}_1$, ..., it confines the types of all informative expressions and purges the hierarchy $\texttt{Type}_0$, $\texttt{Type}_1$, ... of all computational content. In other words, it guarantees that inhabitants of types in Set are the only expressions which do not disappear during the extraction process.

In spite of that, this new calculus may be naturally embedded into CIC by a very simple forgetful operation. Moreover it remains very close to CIC and in practice only few terms are not representable in $\mathsf{CIC_r}$. That is why it represents a big step towards an implementation in the Coq system.

Being able to identify expressions with computational content – or in other words *programs* – was essential to achieve our initial goal: formalizing the parametricity theory for the Calculus of Inductive Constructions.

Parametricity is a concept introduced by Reynolds [22] to study the type abstraction of System F. It expresses the fact that well-typed programs must behave uniformly over their arguments with an abstract type: if the type is abstract, then the functions do not have access to its implementation. Wadler [26] explained how this could be used to deduce interesting properties shared by all programs of the same type. Later, Plotkin and Abadi [21] introduced a logic in which these uniformity properties can be expressed and proved. This logic may be generalized into a second-order logic with higher-order individuals [23, 27].

The main tools of parametricity theory are logical relations defined inductively over the structure of types together with the so-called *abstraction theorem*, which builds a proof that any closed program is related to itself for the relation induced by its type. For instance the relation induced by the type $\forall\alpha, (\alpha \to \alpha) \to \alpha \to \alpha$ of Church numerals is given by the following definition (represented here in Coq using the standard encoding of relations):

$$\begin{aligned}\lambda(f\,f' : \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha).\forall\alpha\,\alpha'\,(R : \alpha \to \alpha' \to \texttt{Prop})\,(g : \alpha \to \alpha)(g' : \alpha' \to \alpha').\\ (\forall x\,x'.R\,x\,x' \to R\,(g\,x)\,(g'\,x')) \to \forall z\,z'.R\,z\,z' \to R\,(f\,\alpha\,g\,z)\,(f'\,\alpha'\,g'\,z')\end{aligned}$$

The abstraction theorem tells that any closed term F of type $\forall\alpha, (\alpha \to \alpha) \to \alpha \to \alpha$ is related to itself according to this relation.

Recently, the work from Bernardy *et al.* [5] generalized these constructions up to a large class of Pure Type Systems and showed that parametricity theory accommodates well with dependent types. But this cannot be straightforwardly adapted to CIC, because parametricity relations live in higher universes instead of using the standard encoding of relations in Prop. Besides, it is difficult to make parametricity relations live inside Prop while conserving abstraction.

But parametricity in a system like Coq would be profitable: it could lead to more automation, for instance for developing mathematical theories: we give here an example in finite group theory (Section 5.3). Basing on our refined calculus, we started an implementation of a Coq tactic that can build closed instances of the abstraction theorem [1].

The paper is organized as follows. After explaining in details why we need to refine the Calculus of Inductive Constructions, we present $\mathsf{CIC_r}$ in Sections 2 and 3. Section 4 is devoted to the definition of relational parametricity, and the proof of the abstraction theorem, without and with inductive types. In Section 5, we present different kinds of "theorems for free" that are derived from the general abstraction theorem, like independence of the law of excluded middle with respect to $\mathsf{CIC_r}$ or standard properties of finite groups. We finally explain the algorithm behind the implementation of the Coq tactic (Section 6) before discussing related works and concluding.

2 $\mathsf{CIC_r}$: a refined calculus of constructions with universes

2.1 The need for a refinement

In older versions of CIC, Set was not a synonym for $\mathtt{Type}_0$ but a special impredicative sort containing data-types and basic informative types. However, there is a smaller demand from the users for the impredicativity of Set rather than the possibility to add classical axioms to CIC, and having both may lead to the inconsistency of the system (the conjunction of excluded middle and description conflicts with the impredicativity of Set [10]). As a result, nowadays Set is predicative and behaves in CIC as the first level of the hierarchy of universes.

In $\mathsf{CIC_r}$, we want to reintroduce the sort Set of informative types in order to mark the distinction between expressions with computational content and expressions which are erased during the extraction process. To stay close to CIC, we want Set to be predicative, so we introduce a hierarchy of sorts $\mathtt{Set}_0$, $\mathtt{Set}_1$, ...

In the refinement, the CIC hierarchy of sorts Type is thus divided into two classes : a hierarchy of sorts Set, whose inhabitants have a computational content, and a hierarchy of sorts Type, whose inhabitants are uninformative. There is a difference of level between inhabitants of Set and inhabitants of Type: the inhabitants of Set are inhabited only by non-habitable expressions whereas Type contains the signatures of predicates and type constructors which are themselves, when fully applied, inhabited respectively by proofs and programs.

In Coq, deciding to which of this two classes an expression of type Type belongs is essential in the extraction mechanism. In the original two-sorted calculus of constructions (i.e. without universes), the top-sort contains only arities and therefore the level of terms can be easily obtained by looking at the type derivation and the extraction procedure is simple [19]. However, in Coq, to extract the computational content of an inhabitant of sort Type, the extraction algorithm decides if a type is informative by inspecting the shape of its normal form [16, 17]. Therefore termination of extraction relies on the normalization of CIC. It makes the correction of the extraction difficult to formally certify.

2.2 Presentation of the calculus

The syntax of $\mathsf{CIC_r}$ is the same as the standard calculus of constructions except that we extend the set of sorts. Terms are generated by the following grammar:

$$A, B \quad := \quad x \quad | \quad s \quad | \quad \forall x : A.B \quad | \quad \lambda x : A.B \quad | \quad (A\, B)$$

$$i < j \; \frac{}{\mathtt{Set}_i <: \mathtt{Set}_j} \; (\textsc{Sub}_{1-1}) \qquad i < j \; \frac{}{\mathtt{Type}_i <: \mathtt{Type}_j} \; (\textsc{Sub}_{1-2})$$

$$\frac{A <: B}{\forall x : C.A <: \forall x : C.B} \; (\textsc{Sub}_2)$$

Figure 1 Subtyping rules.

where s ranges over the set $\{\mathtt{Prop}\} \cup \{\mathtt{Set}_i, \mathtt{Type}_{i+1} | i \in \mathbb{N}\}$ of *sorts* and x ranges over the set of *variables*. In the remaining of the paper, when no confusion is possible, Set stands for "$\mathtt{Set}_i$ for some i", and Type stands for "$\mathtt{Type}_i$ for some i". The notation $i \vee j$ represents the maximum of i and j.

As usual, we will consider terms up to α-conversion and we denote by $A[B/x]$ the term built by substituting the term B to each free occurrence of x in A. The β-reduction $\rhd$ is defined as in CIC, and we write $A \equiv B$ to denote the β-conversion.

A context Γ is a list of couples $x : A$ where x is a variable and A is term. The empty context is written $\langle\rangle$. The system has subtyping, given by the rules in **Figure 1**. The typing rules of $\mathsf{CIC_r}$ are given in **Figure 2**.

The word *type* is a synonymous for a term that can be typed by a sort following those rules. We call *informative types* inhabitants of Set, *programs* inhabitants of informative types, *propositions* inhabitants of Prop and *proofs* inhabitants of propositions. The sort $\mathtt{Type}_i$ adds a shallow level to the system; it is populated with two kinds of terms: *arities*, which are terms whose head normal forms have the form $\forall(x_1 : A_1) \dots (x_n : A_n).s$ where s is either Prop, $\mathtt{Set}_j$ or $\mathtt{Type}_j$ with $j < i$; and higher-order functions that manipulate arities, and whose types are arities with $\mathtt{Type}_{i+1}$ as a conclusion. We say that a term has some sort s if s is the type of its type.

3 Inductive types

The calculus is extended with inductive definitions and fixpoints. The presentation is very similar to Chapter 4.5 of the Reference Manual of Coq [24], and one can report to it to have further details.

3.1 Inductive types and fixpoints

The grammar of $\mathsf{CIC_r}$ terms is extended with:

$$A, B, P, Q, F \cdots := \cdots \quad | \quad I \quad | \quad c \quad | \quad \mathtt{case}_I(A, \overrightarrow{Q}, P, \overrightarrow{F}) \quad | \quad \mathtt{fix}\,(x : A).B$$

We write $\mathtt{Ind}^p(I : A, c_1 : C_1, \dots, c_k : C_k)$ to state that I is a well-formed inductive definition typed with p parameters, of arity A, with k constructors $c_1, \dots, c_k$ of respective types $C_1, \dots, C_k$. It requires that:

1. the names I and c_j are fresh;
2. A is a well-typed arity of conclusion Prop of Set: it is convertible to $\forall\overrightarrow{(x : P)}^p\overrightarrow{(y : B)}^n.r$ where $r \in \{\mathtt{Prop}, \mathtt{Set}\}$;
3. for any j, C_j has the form $\forall\overrightarrow{(x : P)}^p\overrightarrow{(z : E_j)}^{n_j}.I\,\overrightarrow{x}^p\,\overrightarrow{D_j}^n$ where I may appear inside the E_js only as a conclusion. This is called the *strict positivity* condition, and is mandatory for the system to be coherent [8];

$$\frac{}{\vdash \texttt{Prop} : \texttt{Type}_1}\ (\textsc{Ax}_1) \qquad \frac{}{\vdash \texttt{Set}_i : \texttt{Type}_{i+1}}\ (\textsc{Ax}_2) \qquad \frac{}{\vdash \texttt{Type}_i : \texttt{Type}_{i+1}}\ (\textsc{Ax}_3)$$

$$x \notin \Gamma, s \in \mathcal{S}\ \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\ (\textsc{Var}) \qquad x \notin \Gamma, s \in \mathcal{S}\ \frac{\Gamma \vdash B : C \qquad \Gamma \vdash A : s}{\Gamma, x : A \vdash B : C}\ (\textsc{Weak})$$

$$B \equiv C, s \in \mathcal{S}\ \frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : s}{\Gamma \vdash A : B}\ (\textsc{Conv}) \qquad B <: C\ \frac{\Gamma \vdash A : B}{\Gamma \vdash A : C}\ (\textsc{Cum})$$

$$r \in \{\texttt{Prop}, \texttt{Set}_i, \texttt{Type}_i\}\ \frac{\Gamma \vdash A : r \qquad \Gamma, x : A \vdash B : \texttt{Set}_i}{\Gamma \vdash \forall x : A.B : \texttt{Set}_i}\ (\forall_1)$$

$$r \in \{\texttt{Prop}, \texttt{Set}_i, \texttt{Type}_i\}\ \frac{\Gamma \vdash A : r \qquad \Gamma, x : A \vdash B : \texttt{Type}_i}{\Gamma \vdash \forall x : A.B : \texttt{Type}_i}\ (\forall_2)$$

$$s \in \mathcal{S}\ \frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : \texttt{Prop}}{\Gamma \vdash \forall x : A.B : \texttt{Prop}}\ (\forall_3)$$

$$\frac{\Gamma \vdash M : \forall x : A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B[N/x]}\ (\textsc{App}) \qquad \frac{\Gamma, x : A \vdash B : C}{\Gamma \vdash \lambda x : A.B : \forall x : A.C}\ (\textsc{Abs})$$

Figure 2 The refined calculus of construction with universes : $\mathsf{CIC_r}$.

4. for any j, $\forall \overrightarrow{(z : E_j)}^{n_j} .I\, \overrightarrow{x}^{p}\, \overrightarrow{D_j}^{n}$ is a well-typed expression of sort r under the context $(\overrightarrow{(x : P)}^{p}, I : A)$.

Notice that we do not allow inductive definitions in a nonempty context, but this is only for a matter of clarity.

Declaring a new inductive definition adds new constants I and c_j to the system, together with the top left two typing rules presented in **Figure 3**.

The bottom rule of **Figure 3** is the typing rule for the `case` construction which is used to implement elimination schemes. Two sorts are involved in eliminations: s, the sort of the inductive type we eliminate, which may be `Prop` or `Set`; and r, the type of the type of the term we construct, which may be `Prop`, `Set` or `Type`. The four cases of elimination that do

$$\frac{}{\vdash I : A}\ (\textsc{Ind}) \qquad \frac{}{\vdash c_j : C_j}\ (\textsc{Constr}) \qquad f \text{ is guarded}\ \frac{\Gamma, f : A \vdash M : A}{\Gamma \vdash \texttt{fix}(f : A).M : A}\ (\textsc{Fix})$$

$$\text{(under restr.)}\ \frac{\begin{array}{c}\Gamma \vdash M : I\, \overrightarrow{Q}^{p}\, \overrightarrow{G}^{n} \qquad \Gamma \vdash T : \forall \overrightarrow{y : B[\overrightarrow{Q}^{p}/\overrightarrow{x}^{p}]}^{n} .I\, \overrightarrow{Q}^{p}\, \overrightarrow{y}^{n} \to r \\ \left(\Gamma \vdash F_j : \forall \overrightarrow{(z : E_j[\overrightarrow{Q}^{p}/\overrightarrow{x}^{p}])}^{n_j} .T\, \overrightarrow{D_j[\overrightarrow{Q}^{p}/\overrightarrow{x}^{p}]}^{n}\, (c_j\, \overrightarrow{Q}^{p}\, \overrightarrow{z}^{n_j})\right)_{j=1\ldots k}\end{array}}{\Gamma \vdash \texttt{case}_I(M, \overrightarrow{Q}^{p}, T, \overrightarrow{F}^{k}) : T\, \overrightarrow{G}^{n}\, M}\ (\textsc{Case})$$

Figure 3 The rules for an inductive type $\texttt{Ind}^p(I : A, c_1 : C_1, \ldots, c_k : C_k)$.

not involve Type are called *small eliminations.* They are used to build:

- proofs and programs by inspecting programs;
- proofs by inspecting proofs;
- programs by inspecting proofs under restriction (1) (see below).

The other two cases are called *large eliminations.* Strong elimination is mainly used to build propositions by case analysis and to internally prove the minimality of informative inductive definitions. For instance, using large elimination over nat, one may build a predicate P of type $\texttt{nat} \to \texttt{Prop}$ such that $P\,0 \equiv \top$ and $P\,(\texttt{S}\ 0) \equiv \bot$ and thus proves that $0 \neq (\texttt{S}\ 0)$. Similarly, large elimination may be used to build informative types (for instance, to build a "type constructor" $T_{\alpha,\beta} : \texttt{nat} \to \texttt{Set}$ parametrized by a type α and an informative type β such that $T_{\alpha,\beta}\, n \equiv \alpha \to \cdots \to \alpha \to \beta$ with n occurrences of α) or to build arities (for instance, to build an "arity constructor" $A_\alpha : \texttt{nat} \to \texttt{Type}$ parametrized by a type α such that $A_\alpha\, n \equiv \alpha \to \cdots \to \alpha \to \texttt{Prop}$ with n occurrences of α).

Eliminations from Prop to other sorts are restricted to inductive definitions that have at most one constructor, and such that all the arguments (which are not parameters) of this constructor are of sort Prop:

$$k = 0 \text{ or } \left(k = 1 \text{ and } \vdash E : \texttt{Prop} \text{ for any } E \in \overrightarrow{E_1}^{n_1}\right) \tag{1}$$

This is essential for coherence [7] and has a computational interpretation: it is natural that computing an informative type should not rely on any proof structure, that would disappear during program extraction [15, 16].

▶ Example 1. Here are a few examples of inductive definitions:

- $\texttt{Ind}^0(\texttt{nat} : \texttt{Set}_0,\quad 0 : \texttt{nat},\quad \texttt{S} : \texttt{nat} \to \texttt{nat})$
- $\texttt{Ind}^1(\texttt{list}_i : \texttt{Set}_i \to \texttt{Set}_i,\quad \texttt{nil}_i : \forall A : \texttt{Set}_i\,.\,\texttt{list}_i\ A,\quad \texttt{cons}_i : \forall A : \texttt{Set}_i\,.A \to \texttt{list}_i\ A \to \texttt{list}_i\ A)$
- $\texttt{Ind}^0(\texttt{True} : \texttt{Prop},\quad \texttt{I} : \texttt{True})$
- $\texttt{Ind}^0(\texttt{False} : \texttt{Prop})$
- $\texttt{Ind}^2(\texttt{eq}_i : \forall(A : \texttt{Set}_i).A \to A \to \texttt{Prop},\quad \texttt{refl}_i : \forall(A : \texttt{Set}_i)(x : A).\,\texttt{eq}_i\ x\,x)$
- $\texttt{Ind}^2(\texttt{eqP} : \forall(A : \texttt{Prop}).A \to A \to \texttt{Prop},\quad \texttt{reflP} : \forall(A : \texttt{Prop})(x : A).\,\texttt{eqP}\ x\,x)$
- $\texttt{Ind}^2(\texttt{eqT}_i : \forall(A : \texttt{Type}_i).A \to A \to \texttt{Prop},\quad \texttt{reflT}_i : \forall(A : \texttt{Type}_i)(x : A).\,\texttt{eqT}_i\ x\,x)$

Note that we have three levels of Leibniz equality: $\texttt{eq}_i$ for comparing programs, eqP for comparing proofs and $\texttt{eqT}_i$ for comparing everything else (we find the same kind of triplication for other standard encodings like cartesian product, disjoint sum and the existential quantifier).

The second operator to deal with inductive definitions is fixpoint definition. The typing rule for the fixpoint is defined on the top right of **Figure 3**. It is also restricted to avoid non-terminating terms, which would lead to absurdity. The restriction is called the *guard condition*: one argument should have an inductive type, and must structurally decrease on each recursive call. One may refer to [11] for further details.

We extend the reduction with the ι-reduction rules:

$$\begin{array}{rcl} \texttt{case}_I(c_j\,\overrightarrow{Q}^p\,\overrightarrow{M}^{n_j}, \overrightarrow{Q}^p, T, \overrightarrow{F}^k) & \rhd & F_j\,\overrightarrow{M}^{n_j} \\ (\texttt{fix}(f : A).M)\,(c_j\,\overrightarrow{Q}^p\,\overrightarrow{M}^{n_j}) & \rhd & M[\texttt{fix}(f : A).M/f]\,(c_j\,\overrightarrow{Q}^p\,\overrightarrow{M}^{n_j}) \end{array}$$

and $\equiv$ denotes the $\beta\iota$-equivalence.

3.2 Embedding CIC$_\mathsf{r}$ into CIC and coherence

This calculus embeds easily into CIC by mapping $\texttt{Set}_i$ and $\texttt{Type}_i$ onto the sort $\texttt{Type}_i$ of CIC:

▶ Lemma 1. Let $|\bullet|$ be the context-closed function from terms of CIC$_\mathsf{r}$ to terms of CIC such that $|\texttt{Prop}| = \texttt{Prop}$ and $|\texttt{Type}_i| = |\texttt{Set}_i| = \texttt{Type}_i$, then we have :

$$\Gamma \vdash A : B \Rightarrow |\Gamma| \vdash_{\mathsf{CIC}} |A| : |B|$$

Since $|\forall X : \texttt{Prop}\,.X| = \forall X : \texttt{Prop}\,.X$, the logical coherence (the existence of an unprovable proposition) of CIC ensures the coherence of CIC$_\mathsf{r}$.

Conversely, some terms of CIC do not have a counterpart in the refinement: we cannot mix informative and uninformative types. An example is the following Coq definition:

```
fun (b:bool) ⇒ if b then nat else Set
```

4 Relational parametricity

In this setting, we have a natural notion of parametricity: we can define a translation that maps types to relations and other terms to proofs that they belong to those relations. What is new is that relations over objects of type Prop or Set have Prop as a codomain, which is more natural in a calculus with an impredicative sort for propositions.

We go step by step. First, we define parametricity for the calculus without inductive types, and show the abstraction theorem for this restriction. Subsequently, we add inductive types with large eliminations forbidden, and finally see how large eliminations behave with parametricity.

4.1 Parametricity for the calculus without inductive types

▶ Definition 1 (Parametricity relation). The parametricity translation $[\![\bullet]\!]$ is defined by induction on the structure of terms:

$$
\begin{aligned}
[\![\langle\rangle]\!] &= \langle\rangle && (1)\\
[\![\Gamma, x : A]\!] &= [\![\Gamma]\!], x : A, x' : A', x_R : [\![A]\!]\,x\,x' && (2)\\
\\
[\![s]\!] &= \lambda(x : s)(x' : s).x \to x' \to \hat{s} && (3)\\
[\![x]\!] &= x_R && (4)\\
[\![\forall x{:}A.B]\!] &= \lambda(f : \forall x : A.B)(f' : \forall x' : A'.B').\quad \forall(x : A)(x' : A')(x_R : [\![A]\!]\,x\,x').\\
&\qquad\qquad [\![B]\!]\,(f\,x)\,(f'\,x') && (5)\\
[\![\lambda x : A.B]\!] &= \lambda(x : A)(x' : A')(x_R : [\![A]\!]\,x\,x').[\![B]\!] && (6)\\
[\![(A\,B)]\!] &= ([\![A]\!]\,B\,B'\,[\![B]\!]) && (7)
\end{aligned}
$$

with $\hat{\texttt{Prop}} = \hat{\texttt{Set}_i} = \texttt{Prop}$ and $\hat{\texttt{Type}_i} = \texttt{Type}_i$ and where A' denotes the term A in which we have replaced each variable x by a fresh variable x'.

It is easy to prove by induction that the previous definition is well-behaved with respect to substitution and conversion:

▶ Lemma 2 (Substitution lemmas). **1.** $(A[B/x])' = A'[B'/x']$

2. $[\![A[B/x]]\!] = [\![A]\!][B/x][B'/x'][[\![B]\!]/x_R]$

3. $A_1 \equiv_\beta A_2 \Rightarrow [\![A_1]\!] \equiv_\beta [\![A_2]\!]$

$$\begin{aligned}\Theta_I(\overrightarrow{Q}^p, T, \overrightarrow{F}^n) = \lambda \overrightarrow{(x : A)(x' : A')(x_R : [\![A]\!]\, x\, x')}^n\, (a : I\, \overrightarrow{Q}^p\, \overrightarrow{x}^n)(a' : I\, \overrightarrow{Q'}^p\, \overrightarrow{x'}^n) \\ (a_R : [\![I]\!]\, \overrightarrow{Q\, Q'\, [\![Q]\!]}^p\, \overrightarrow{x\, x'\, x_R}^n\, a\, a'). \\ [\![T]\!]\, \overrightarrow{x\, x'\, x_R}^n\, a\, a'\, a_R\, (\mathtt{case}_I\, (a, \overrightarrow{Q}^p, T, \overrightarrow{F}^n))\, (\mathtt{case}_I\, (a', \overrightarrow{Q'}^p, T', \overrightarrow{F'}^n))\end{aligned}$$

Figure 4 The definition of Θ_I.

The abstraction theorem states that the parametricity transformation preserves typing.

▶ Theorem 1 (Abstraction without inductive definitions). If $\Gamma \vdash A : B$, then $[\![\Gamma]\!] \vdash A : B$, $[\![\Gamma]\!] \vdash A' : B'$, and $[\![\Gamma]\!] \vdash [\![A]\!] : [\![B]\!]\, A\, A'$.

Proof. The proof is a straightforward induction on the derivation of $\Gamma \vdash A : B$. The first item is essentially proved by invoking structural rules and by propagating induction hypothesis. The key steps of the second items are the rule (Ax_2), which requires cumulativity, and the rules ($\forall_1$-$\forall_3$), which involve many abstraction and product rules. ◀

4.2 Why does not it work directly in CIC?

In the syntactic theory of parametricity for dependent types presented in [5], relations over a type of some universe are implemented as predicates ranging in the same universe. This can be read in the following piece of definition : $[\![\mathtt{Type}_i]\!] = \lambda(x\, x' : \mathtt{Type}_i).x \to x' \to \mathtt{Type}_i$. We cannot simply replace the conclusion with $\mathtt{Prop}$, because in CIC one has $\vdash \mathtt{Type}_i : \mathtt{Type}_{i+1}$, and the abstraction theorem would require that $\vdash [\![\mathtt{Type}_i]\!] : [\![\mathtt{Type}_{i+1}]\!]\ \mathtt{Type}_i\ \mathtt{Type}_i$ which is equivalent to $\vdash \lambda(x\, x' : \mathtt{Type}_i).x \to x' \to \mathtt{Prop} : \mathtt{Type}_i \to \mathtt{Type}_i \to \mathtt{Prop}$ but this last sequent is not derivable. In our refinement, $[\![\mathtt{Prop}]\!]$ and $[\![\mathtt{Set}]\!]$ have $\mathtt{Prop}$ as a conclusion, but this is not a problem since we do not have $\vdash \mathtt{Set}_i : \mathtt{Set}_{i+1}$.

This refined calculus is very convenient to set the basis for parametricity. As we argued, it has also nice properties regarding realizability and extraction: as an example, the correctness of extraction in this calculus would not rely on the termination of the β-reduction. Even if possible, obtaining the same result directly in CIC would have required a complete reworking of parametricity relations.

The calculus is very close to CIC, though. In Section 6, we discuss if it is possible to write a tactic in Coq that would exploit this work, without changing Coq's calculus.

4.3 Adding inductive types

As a first step, we restrict ourselves to small eliminations: we do not allow large eliminations. We will see in Subsection 4.4 that we are actually able to handle large eliminations over a big class of inductive definitions.

We write $\Gamma \vdash_{\mathrm{SE}} A : B$ to denote sequents typable in $\mathsf{CIC_r}$ where large eliminations are forbidden. Let us suppose that $\mathtt{Ind}^p(I : A, \overrightarrow{c : C}^k)$, we will define a fresh inductive symbol $[\![I]\!]$ and a family $([\![c_i]\!])_{i=1\ldots k}$ of fresh constructor names. Then we extend Definition 1 with

$$\begin{aligned}[\![\mathtt{fix}(x : A).B]\!] = (\mathtt{fix}(x_R : [\![A]\!]\, x\, x').[\![B]\!])\, [\mathtt{fix}(x : A).B/x][\mathtt{fix}(x' : A').B'/x'] \\ [\![\mathtt{case}_I(M, \overrightarrow{Q}^p, T, \overrightarrow{F}^n)]\!] = \mathtt{case}_{[\![I]\!]}([\![M]\!], \overrightarrow{Q, Q', [\![Q]\!]}^p, \Theta_I(\overrightarrow{Q}^p, T, \overrightarrow{F}^n), \overrightarrow{[\![F]\!]}^n)\end{aligned}$$

where Θ_I is defined in **Figure 4**.

We want to extend Theorem 1 with inductive definitions. We prove the following theorem:

▶ Theorem 2 (Abstraction with inductive definitions). **1.** If $\mathtt{Ind}^p(I : A, \overrightarrow{c : C}^k)$ is a valid inductive definition then so is $\mathtt{Ind}^{3p}([\![I]\!] : [\![A]\!]\, I\, I, \overrightarrow{[\![c]\!] : [\![C]\!]\, c\, c'}^k)$.

2. If $\Gamma \vdash_{\mathrm{SE}} A : B$ then $[\![\Gamma]\!] \vdash_{\mathrm{SE}} A : B$, $[\![\Gamma]\!] \vdash_{\mathrm{SE}} A' : B'$, and $[\![\Gamma]\!] \vdash_{\mathrm{SE}} [\![A]\!] : [\![B]\!]\, A\, A'$.

Proof. The first item requires to check the constraints to build inductive types: the typing and the strict positivity. As for Theorem 1, the second item is proved by induction on the structure of the proof of $\Gamma \vdash A : B$. One needs to check that the guard condition is preserved in the (Fix) rule and that the (Case) rule is well-formed. The key idea here is that the translation of terms containing only small eliminations also contains only small eliminations. ◀

4.4 Overcoming the restriction over large elimination

Suppose we now authorize the whole large elimination (with restriction (1)). The definition generated by the following inductive definition $\mathtt{Ind}^0(\mathtt{box}_i : \mathtt{Set}_{i+1}, \mathtt{close}_i : \mathtt{Set}_i \to \mathtt{box}_i)$ is

$$\begin{aligned}&\mathtt{Ind}^0\big([\![\mathtt{box}_i]\!] : \mathtt{box}_i \to \mathtt{box}_i \to \mathtt{Prop},\\ &\qquad [\![\mathtt{close}_i]\!] : \forall (A\, A' : \mathtt{Set}_i).(A \to A' \to \mathtt{Prop}) \to [\![\mathtt{box}_i]\!]\,(\mathtt{close}_i\ A)\,(\mathtt{close}_i\ A')\big)\end{aligned}$$

If we want to prove parametricity for the (Case) rule when we build a $\mathtt{Type}$, one should provide an inhabitant of: $\forall (A\, A' : \mathtt{Set}_i).[\![\mathtt{box}_i]\!]\,(\mathtt{close}_i\ A)\,(\mathtt{close}_i\ A') \to (A \to A' \to \mathtt{Prop})$. But since $[\![\mathtt{box}_i]\!]\,(\mathtt{close}_i\ A)\,(\mathtt{close}_i\ A')$ has type $\mathtt{Prop}$ and $A \to A' \to \mathtt{Prop}$ has type $\mathtt{Type}$, we cannot build the expected relation by deconstructing a proof of $[\![\mathtt{box}_i]\!]\,(\mathtt{close}_i\ A)\,(\mathtt{close}_i\ A')$: this is forbidden by restriction (1).

However, let us consider the following example:

$$\mathtt{Ind}^0(I : \mathtt{Set}, \mathrm{N} : \mathtt{nat} \to I, \mathrm{B} : \mathtt{bool} \to I)$$

Let say we need to translate the following large elimination (for the sake of readability, we present it with the Coq syntax):

```
Definition f (x : I) := match x with
  | N n ⇒ vector n
  | B b ⇒ nat
end.
```

We can swap the destruction of x_R for two nested destructions of x and x' which produces k^2 branches (where k in the number of constructors). But only k of them are actually possible (we use here the `Program` keyword in order to let the system infer dependent type annotations for each `match`):

```
Program Definition f_R (x x’ : I) (x_R : ⟦I⟧ x x’) :=
  match x with
  | N n ⇒ match x’ with
    | N n’ ⇒ let n_R := inv n n’ x_R in ⟦vector n⟧
    | B b’ ⇒ absurd (vector n→nat→Prop) (abs12 n b’ x_R)
    end
  | B b ⇒ match x’ with
    | N n’ ⇒ absurd (nat→vector n’→Prop) (abs21 b n’ x_R)
    | B b ⇒ ⟦nat⟧
    end
  end.
```

where the following terms are all implemented with an authorized large elimination:

$$\begin{array}{rcl}
\texttt{inv} & : & \forall (n\, n' : \texttt{nat}). [\![I]\!]\, (\mathrm{N}\, n)\, (\mathrm{N}\, n') \to [\![\texttt{nat}]\!]\, n\, n' \\
\texttt{abs}_{12} & : & \forall (n : \texttt{nat})(b' : \texttt{bool}). [\![I]\!]\, (\mathrm{N}\, n)\, (\mathrm{B}\, b') \to \texttt{False} \\
\texttt{abs}_{21} & : & \forall (b : \texttt{bool})(n' : \texttt{nat}). [\![I]\!]\, (\mathrm{B}\, b)\, (\mathrm{N}\, n') \to \texttt{False} \\
\texttt{absurd} & : & \forall (\alpha : \texttt{Type}). \texttt{False} \to \alpha
\end{array}$$

We notice that this example runs smoothly because all the arguments of all the constructors have type `Prop` or `Set`, which avoids the pitfall of the `box` example.

That is why we propose to restrict large elimination from `Set` to `Type` to the class of small inductive definitions (this class was introduced by Paulin in [20] to restrict the large elimination in vanilla Coq where the sort `Set` of informative types is impredicative):

▶ Definition 2 (Small inductive definitions). We say that $\texttt{Ind}^p(I : A, \overrightarrow{c : C}^k)$ is a *small inductive definition* if all the arguments of each constructor are of sort `Prop` or $\texttt{Set}_m$ for some m. More formally, if for all $1 \le i \le k$, $\vdash c_i : \forall \overrightarrow{(x : P)}^p \overrightarrow{(y : B)}^{n_i} . I\, \overrightarrow{x}^p\, \overrightarrow{D_j}^n$ then $\overrightarrow{x : P}^p, \overrightarrow{y : B}^{j-1} \vdash B_j : r$ with $r = \texttt{Prop}$ or $r = \texttt{Set}_m$ for some m.

With this restriction, the abstraction theorem holds in presence of large elimination:

▶ Theorem 3. Theorem 2 holds when $\vdash_{\mathrm{SE}}$ stands for derivability where large elimination is authorized over small inductive definitions and forbidden otherwise.

5 Examples of "free theorems"

In this section we give a few examples of consequences of the abstraction theorem. Most examples that can be found in the literature (see for instance [5,26]) may be easily implemented in our framework. To improve readability, we use "$=_\alpha$" and "$\exists x : \alpha$" to denote respectively standard inductive encodings of the Leibniz equality and existential quantifier.

5.1 The type of Church numerals

Let $\texttt{church}_i$ be $\forall \alpha : \texttt{Set}_i, (\alpha \to \alpha) \to \alpha \to \alpha$, the type of Church numerals. Let $\texttt{iter}_i$ be the following expression

$$\begin{array}{l}
\texttt{fix iter}_i : \texttt{nat} \to \texttt{church}_i\, .\lambda (n : \texttt{nat})(\alpha : \texttt{Set}_i)(f : \alpha \to \alpha)(z : \alpha). \\
\qquad\qquad\qquad\qquad \texttt{case}(n, \lambda k : \texttt{nat}\, .\alpha, z, \lambda p : \texttt{nat}\, .f\, (\texttt{iter}_i\; p\, \alpha\, f\, z))
\end{array}$$

which is the primitive recursive operator which composes a function n times with itself.

The relation $[\![\texttt{church}_i]\!] : \texttt{church}_i \to \texttt{church}_i \to \texttt{Prop}$ is the relation unfolded in the introduction. One can prove easily the following property on any $f : \texttt{church}_i$:

$$[\![\texttt{church}_i]\!]\, f\, f \to\; \exists n : \texttt{nat}\, .\forall (\alpha : \texttt{Set}_i)(g : \alpha \to \alpha)(z : \alpha).\, \texttt{iter}_i\; n\, \alpha\, g\, z =_\alpha f\, \alpha\, g\, z$$

which states that, if f is in relation with itself by $[\![\texttt{church}_i]\!]$, then there exists an integer n such that f is extensionally equal to $\texttt{iter}_i\; n$. Now suppose we have a closed term F such that $\vdash F : \texttt{church}_i$. By the abstraction theorem we obtain a proof $[\![F]\!]$ that $[\![\texttt{church}_i]\!]\, F\, F$ and therefore that F is extensionally equal to $\texttt{iter}_i\; n$ for some n.

5.2 The tree monad

Binary trees carrying information of type α on their leaves may be implemented by the following inductive definition :

$$\mathtt{Ind}^1(\mathtt{tree}_i : \mathtt{Set}_i \to \mathtt{Set}_{i+1}, \mathtt{leaf}_i : \forall \alpha : \mathtt{Set}_i .\alpha \to T\,\alpha, \mathtt{node}_i : \forall \alpha : \mathtt{Set}_i .T\,\alpha \to T\,\alpha \to T\,\alpha)$$

and it is possible to represent in CIC the function $\mathtt{map}_i$ of type $\forall(\alpha\,\beta : \mathtt{Set}_i).(\alpha \to \beta) \to \mathtt{tree}_i\ \alpha \to \mathtt{tree}_i\ \beta$ which maps a function to all the leaves of a tree.

The generated relation $[\![\mathtt{tree}_i]\!]$ tells that two trees are related if they have the same shape and elements at the same position in each tree are related. It is then not difficult to prove for any function $f : \alpha \to \alpha'$ that $[\![\mathtt{tree}_i]\!]\,\alpha\,\alpha'\,R_f$ is a relation representing the graph of the $\mathtt{map}$ function where R_f is $\lambda(x : \alpha)(x' : \alpha').fx =_{\alpha'} x'$ and represents the graph of f.

We can also define in the system the multiplication of the monad by programming an expression μ_i of type $\forall\alpha.\,\mathtt{tree}_i\,(\mathtt{tree}_i\ \alpha) \to \mathtt{tree}_i\,\alpha$ with the following computational behavior:

$$\mu_i\,\alpha\,(\mathtt{leaf}_i\ \alpha\,x) \equiv x \qquad \text{and} \qquad \mu_i\,\alpha\,(\mathtt{node}_i\ \alpha\,x\,y) \equiv \mathtt{node}_i\ \alpha\,(\mu_i\,\alpha\,x)\,(\mu_i\,\alpha\,y)$$

As μ_i is closed, an application of the abstraction theorem which instantiates the relation to the graph of f proves the naturality of μ_i.

5.3 Parametricity and algebra

Obtaining "free theorems" by parametricity can be extended to data types with structure. In this section, we take the example of finite groups, which is directly related to the Ssreflect library [12] developed in Coq; but our reasoning applies to a large variety of algebraic structures.

In Chapter 3.4 of his PhD. thesis [9], François Garillot observed that algebraic developments require lots of proofs by isomorphism, which often look similar. Intuitively, a polymorphic function operating on groups can only compose elements using the laws given by the group's structure, and thus cannot create new elements.

More formally, we take an arbitrary group $\mathcal{H}$ defined by a carrier $\alpha : \mathtt{Set}_0$, a unit element $\mathtt{e} : \alpha$, a composition law $\cdot : \alpha \to \alpha \to \alpha$, an inverse function $\mathtt{inv} : \alpha \to \alpha$, and the standard axioms stating that $\cdot$ is associative, $\mathtt{e}$ is neutral on the left and composing with the inverse on the left produces the unit. On top of this, we define the type of all the finite subgroups of $\mathcal{H}$ with the following one-constructor inductive definition:

$$\begin{aligned}\mathtt{Ind}^0\Big(&\mathtt{fingrp} : \mathtt{Set}_0, \mathtt{Fingrp} : \forall\,\mathtt{elements} : \mathtt{list}\,\alpha.\\ &\quad \mathtt{e} \in \mathtt{elements} \to\\ &\quad (\forall x\,y.x \in \mathtt{elements} \to y \in \mathtt{elements} \to x \cdot y \in \mathtt{elements}) \to\\ &\quad (\forall x.x \in \mathtt{elements} \to \mathtt{inv}\,x \in \mathtt{elements}) \to \mathtt{fingrp}\Big)\end{aligned}$$

where $\in : \alpha \to \mathtt{list}\,\alpha \to \mathtt{Prop}$ is the standard inductive predicate stating if an element appears in a list.

Suppose we have a closed term $Z : \mathtt{fingrp} \to \mathtt{fingrp}$ (examples of such terms abound: eg. the center, the normalizer, the derived subgroup...). The abstraction theorem states that for any $R : \alpha \to \alpha \to \mathtt{Prop}$ compatible with the laws of $\mathcal{H}$ and for any $G\,G' : \mathtt{fingrp}$, $[\![\mathtt{fingrp}]\!]_R\,G\,G' \to [\![\mathtt{fingrp}]\!]_R\,(Z\,G)\,(Z\,G')$ where $[\![\mathtt{fingrp}]\!]_R$ is the relation on subgroups induced by R. Given this, we can prove the following properties:

- for any G, $Z\,G \subset G$ (if we take $R : x\,y \mapsto x \in G$);

- for any G, for any ϕ a morphism of $\mathcal{H}$, $\phi(Z\,G) = Z\,\phi(G)$ (if we take $R : x\,y \mapsto y = \phi(x)$). It entails that $Z\,G$ is a *characteristic subgroup* of $\mathcal{H}$.

To prove this, we use the axiom of proof irrelevance (that can be safely added to the system as we will show in the next subsection). The proof is straightforward by unfolding the definitions. A complete Coq script can be found online [1].

5.4 Classical axioms

One interesting feature of Coq is the ability to add axioms in the system. However when the parametricity transformation $[\![\cdot]\!]$ will encounter the axiom, it will ask for a proof that it is related to itself. Let consider an axiom P such that $\vdash P : s$ where s is Prop or Set. Here three situations are possible:

- Either P is what we call *provably parametric*: the user can provide a proof of $\forall h : P.[\![P]\!]\,h\,h$ and this proof may be used by the abstraction theorem to prove parametricity for terms involving the axiom.
- Or P is *provably not parametric*: there exists a proof that $\forall (h\,h' : P).\neg([\![P]\!]\,h\,h')$. It means that the axiom would break the parametricity of the system: there is no way to invoke the abstraction theorem on a term which uses that axiom.
- Or it is neither provably parametric nor provably not parametric or the user does not know. In this case, the parametricity of the axiom may be added as a new axiom at the user's risk.

Note that if $\neg P$ is provable then P is both provably parametric and provably not parametric and by the abstraction theorem, if P is provable then it is of course provably parametric. It is also easy to deduce from the abstraction theorem that if $P \to Q$ is provable then P provably parametric implies Q provably parametric, and Q provably not parametric implies P provably not parametric. Hence these notions do not depend on the formulation of your axioms.

5.4.1 Proof irrelevance

The axiom of *proof irrelevance* $\texttt{PI} = \forall (X : \texttt{Prop})(p\,q : X), p =_X q$ states that there is at most one proof of any proposition. It is provably parametric since

$$\begin{aligned}[\![\texttt{PI}]\!]\,h\,h' = \forall (X\,X' : \texttt{Prop})\,(X_R : X \to X' \to \texttt{Prop})\\ (p : X)(p' : X')(p_R : X_R\,p\,p')(q : X)(q' : X')(q_R : X_R\,q\,q').[\![\texttt{eqP}]\!]\,X\,X'\,p\,p'\,p_R\,q\,q'\,q_R\end{aligned}$$

may be proved (with PI) equivalent to

$$\begin{aligned}\forall (X\,X' : \texttt{Prop})\,(X_R : X \to X' \to \texttt{Prop})(p : X)(p' : X')(p_R : X_R\,p\,p').\\ [\![\texttt{eqP}]\!]\,X\,X'\,p\,p'\,p_R\,p\,p'\,p_R\end{aligned}$$

which is directly provable by $[\![\texttt{reflP}]\!]$. Therefore PI may be safely added to the system.

5.4.2 Independence of the law of excluded middle

From a user perspective provably not parametric axioms are bad news, but it provides meta-theoreticians a very simple way to prove independence results. Indeed, if a formula is provably not parametric then the abstraction theorem tells you this formula is not provable without large elimination over not small inductive definitions.

▶ Lemma 3. If P is provably not parametric, there is no closed term A of type P (in the restriction of large elimination to small inductive definitions).

For instance, Peirce's law $\mathtt{Peirce} = \forall (X\,Y : \mathtt{Prop}).((X \to Y) \to X) \to X$ (which is known to be equivalent to the excluded middle) is provably not parametric.

6 Towards a Coq implementation

This paper sets the theoretical foundation for an implementation of a reflexive Coq tactic generating the consequences of parametricity for definitions in the Calculus of Constructions. Two approaches are possible:

- modify Coq's calculus to implement CIC_r. The implementation of the translation becomes straightforward;
- do not modify Coq's calculus, but let the translation distinguish informative terms.

The first approach would require to transform Coq radically. We followed the second approach, and started the implementation of a prototype for Coq commands and tactics for parametricity, called CoqParam [1].

In a system like Coq, *reflection* establishes a correspondence between:

- a subset of the Coq terms: this is called the *shallow embedding*;
- a Coq inductive data type representing these terms: this is called the *deep embedding*;
- the OCaml internal representation of those terms.

The deep embedding and the OCaml representation give access to the structure of the terms (whereas the shallow embedding does not), which is very useful to build properties and proofs by computing over this structure. This process, called *computational reflection*, is a well-known way to design powerful automatic tactics in Coq [2, 13, 14].

Parametricity comes well within the spirit of computational reflection: the abstraction theorem is a way to build proofs of terms by inspecting their structures. Our tactic is based on this remark: given a well-typed closed term $\vdash A : B$, it builds the well-typed proof $\vdash [\![A]\!] : [\![B]\!]\,A\,A$, going from the shallow embedding to the OCaml internal representation (this step is called *reification*), and the other way round. The difficulty is to decide, during reification, whether objects of type $\mathtt{Type}$ in Coq should have type $\mathtt{Set}$ or $\mathtt{Type}$ in CIC_r. The tactic does not handle this yet (as well as full inductive types).

Notice that, with this method, we do not have to generally prove the abstraction theorem in Coq: Coq's type checker will prove it on each instance. One may also be interested in a formal proof of the abstraction theorem. It means that the deep embedding should be defined. As the refinement is very close to Coq, this would thus require a large effort.

7 Related works and discussion

Since the introduction of parametricity for system F [22, 26], it has been extended to many logical systems based on Type Theory. Among others, we can cite system $\mathcal{F}_\omega$ by Vytiniotis and Weirich [25] and a large subset of PTSs by Bernardy *et al.* [5,6]. In all these presentations, no sort is impredicative, and parametricity relations live either in a meta-logic or in a different sort than propositions. To our knowledge, this is the first time parametricity relations live in an impredicative sort representing propositions, making them more usable in a system like Coq.

Bernardy *et al.* [5] also explain two possible ways to handle inductive definitions: one by translating induction principles, and one by defining a new inductive data-type as the translation of the initial data-type. Our approach is close to the second method proposed by [5]. We also show how to translate fixpoint definitions, which are more common than inductive principles.

Parametricity and parts of the abstraction theorem have been formalized for deep embeddings of logical systems in Agda [5] and in Coq [3, 4]. Our approach is different: we do not want to have a formal proof of the abstraction theorem (in a first step), but we want to have a practical tool that actually computes results produced by the abstraction theorem. This does not compromise soundness anyway, since the terms produced by this tool are type-checked by Coq's kernel.

8 Conclusion

As we argue throughout the article, the system presented here distinguishes clearly via typing which expressions will be computationally meaningful after extraction. It allows us to define a notion of parametricity for which relations lie in the sort of propositions. This opens up a new way to define automatic tactics in interactive theorem provers based on Type Theory.

Moreover it is known that parametricity and realizability seen as syntactic constructions are closely related [6]. That is why it seems possible to build an internal realizability theory inside our framework. It would permit to develop a similar tactic to prove automatically that program extracted from any closed term will realize its own type. The user would then be able to use this proof to show the correctness of his programs without relying on the implementation of the extraction function.

Finally, it remains to understand why parametric relations do not fit in the sort of proposition in presence of large elimination on non-small data types. We conjecture that parametric relations for large inductive definitions are not proof-irrelevant (in particular, they cannot be interpreted as set-theoretical relations).

Acknowledgments The authors are particularly grateful to François Garillot and Georges Gonthier who suggested the use of parametricity to obtain theorems from free in the setting of algebra, and provided the stimulus for this work. We also thank Assia Mahboubi for providing useful help about the spirit of the Ssreflect library. We finally thank the anonymous reviewers for their encouragements and constructive remarks.

References

1 Preliminary implementation of a Coq tactic. `http://www.lix.polytechnique.fr/~keller/Recherche/coqparam.html`.

2 Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *CPP*, volume 7086 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2011.

3 Robert Atkey. A deep embedding of parametric polymorphism in Coq. In *Workshop on Mechanizing Metatheory*, 2009.

4 Robert Atkey. Syntax for Free: Representing Syntax with Binding Using Parametricity. In Pierre-Louis Curien, editor, *TLCA*, volume 5608 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2009.

5 Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Parametricity and dependent types. In Paul Hudak and Stephanie Weirich, editors, *ICFP*, pages 345–356. ACM, 2010.

6 Jean-Philippe Bernardy and Marc Lasson. Realizability and Parametricity in Pure Type Systems. In Martin Hofmann, editor, *FOSSACS*, volume 6604 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2011.

7 Thierry Coquand. An Analysis of Girard's Paradox. In *LICS*, pages 227–236. IEEE Computer Society, 1986.

8 Thierry Coquand and Christine Paulin. Inductively defined types. In Per Martin-Löf and Grigori Mints, editors, *Conference on Computer Logic*, volume 417 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 1988.

9 François Garillot. *Generic Proof Tools and Finite Group Theory.* PhD thesis, École Polytechnique, 2011.

10 Herman Geuvers. Inconsistency of classical logic in type theory. Unpublished notes, 2001.

11 Eduardo Giménez. Codifying Guarded Definitions with Recursive Schemes. *Types for proofs and Programs*, pages 39–59, 1995.

12 Georges Gonthier, Assia Mahboubi, Laurence Rideau, Enrico Tassi, and Laurent Théry. A Modular Formalisation of Finite Group Theory. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *Lecture Notes in Computer Science*. Springer, 2007.

13 Benjamin Grégoire and Assia Mahboubi. Proving Equalities in a Commutative Ring Done Right in Coq. In Joe Hurd and Thomas F. Melham, editors, *TPHOLs*, volume 3603 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2005.

14 Benjamin Grégoire, Laurent Théry, and Benjamin Werner. A Computational Approach to Pocklington Certificates in Type Theory. *Functional and Logic Programming*, 2006.

15 Pierre Letouzey. A New Extraction for Coq. In Herman Geuvers and Freek Wiedijk, editors, *TYPES*, volume 2646 of *Lecture Notes in Computer Science*, pages 200–219. Springer, 2002.

16 Pierre Letouzey. *Programmation fonctionnelle certifiée: L'extraction de programmes dans l'assistant Coq.* PhD thesis, Université Paris-Sud, July 2004.

17 Pierre Letouzey. Extraction in Coq: An Overview. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *CiE*, volume 5028 of *Lecture Notes in Computer Science*, pages 359–369. Springer, 2008.

18 Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory.* PhD thesis, Chalmers Univ. of Tech, 2007.

19 Christine Paulin-Mohring. Extracting F(omega)'s Programs from Proofs in the Calculus of Constructions. In *POPL*, pages 89–104, 1989.

20 Christine Paulin-Mohring. Inductive definitions in the system coq rules and properties. In Marc Bezem and Jan Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer Berlin / Heidelberg, 1993. 10.1007/BFb0037116.

21 Gordon D. Plotkin and Martín Abadi. A Logic for Parametric Polymorphism. In Marc Bezem and Jan Friso Groote, editors, *TLCA*, volume 664 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 1993.

22 John C. Reynolds. Types, Abstraction and Parametric Polymorphism. In *IFIP Congress*, pages 513–523, 1983.

23 Izumi Takeuti. An Axiomatic System of Parametricity. *Fundam. Inform.*, 33(4), 1998.

24 The Coq Development Team. *The Coq Proof Assistant: Reference Manual.* INRIA, 2012.

25 Dimitrios Vytiniotis and Stephanie Weirich. Parametricity, Type Equality, and Higher-Order Polymorphism. *Journal of Functional Programming*, 20(02):175–210, 2010.

26 Philip Wadler. Theorems for free! In *Proceedings of the fourth international conference on Functional programming languages and computer architecture*, FPCA '89, pages 347–359, New York, NY, USA, 1989. ACM.

27 Philip Wadler. The Girard-Reynolds isomorphism (second edition). *Theor. Comput. Sci.*, 375(1-3):201–226, 2007.

Two-Variable Universal Logic with Transitive Closure*

Emanuel Kieroński and Jakub Michaliszyn

Institute of Computer Science, University of Wrocław, Poland
`{kiero,jmi}@cs.uni.wroc.pl`

Abstract

We prove that the satisfiability problem for the two-variable, universal fragment of first-order logic with constants (or, alternatively phrased, for the Bernays-Schönfinkel class with two universally quantified variables) remains decidable after augmenting the fragment by the transitive closure of a single binary relation. We give a 2-NExpTime-upper bound and a 2-ExpTime-lower bound for the complexity of the problem. We also study the cases in which the number of constants is restricted. It appears that with two constants the considered fragment has the finite model property and NExpTime-complete satisfiability problem. Adding a third constant does not change the complexity but allows to construct infinity axioms. A fourth constant lifts the lower complexity bound to 2-ExpTime. Finally, we observe that we are close to the border between decidability and undecidability: adding a third variable or the transitive closure of a second binary relation lead to undecidability.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases two-variable logic, transitive closure, decidability

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.396

1 Introduction

Classical papers from the 1930s showed that the satisfiability problem for first-order logic, FO, is undecidable. This raised the question which natural fragments of FO are decidable. A large research program led to a complete characterization, with respect to the decidability, of the so-called quantifier prefix classes. In particular, the Bernays-Schönfinkel class, i.e. the class of all formulas starting from a quantifier prefix of the form $\exists^*\forall^*$ followed by a quantifier free formula, appeared to be decidable. Note that, as existential quantifiers can be simulated by constants, the Bernays-Schönfinkel class may be alternatively viewed as the universal fragment of FO (i.e. the class of universal prenex-normal form FO formulas) with constants.

Another interesting decidable fragment of FO is the two-variable fragment, FO^2. With respect to the number of variables it appears to be the maximal fragment whose satisfiability problem is decidable, as undecidability of FO^3 follows from [8]. Decidability of FO^2 was shown in [15] 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 [5] to singly exponential, which implied a NExpTime-upper bound on the complexity of the satisfiability problem. A corresponding lower bound follows from [4, 13], so the satisfiability problem for FO^2 is NExpTime-complete.

The importance of FO^2 can be justified by the fact that it or its natural extensions and variants embed many formalisms used in computer science, such as modal, temporal

* Partially supported by Polish Ministry of Science and Higher Education grant N N206 371339.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 396–410
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

or description logics. Unfortunately, FO^2 has a drawback, which becomes significant when one thinks about practical applications: it cannot express transitivity of a binary relation. Moreover, in contrast to modal logic or to some variants of the guarded fragment [16, 12], extending FO^2 by transitivity statements leads to undecidability [6, 10].

Actually, in applications for program verification or knowledge representation it would be even more desirable to have a transitive closure operator. While in the world of modal logics there exist decidable variants equipped with transitive closure operators, with a notable example of propositional dynamic logic, PDL [3], not too many natural decidable fragments of first-order logic with transitive closure are known. One exception is an extension of the two-variable guarded fragment with a transitive closure operator applied to binary symbols appearing only in guards. This is shown to be decidable and 2-ExpTime-complete in [14]. In a recent paper [11], FO^2 with the *equivalence* closure (i.e. reflexive, symmetric and transitive closure) operator is show to be decidable, and 2-NExpTime-complete, if the closure operator is applied to two distinguished binary symbols.

In [7] the universal fragment of first-order logic with constants is shown to be decidable when extended with the *deterministic* transitive closure operator, DTC, applied to a single, distinguished binary symbol, provided that only positive occurrences of DTC are allowed (thus we cannot say, e.g. that an element satisfying P is forbidden to be connected by a deterministic path to an element satisfying Q).

Some related results are obtained also in [2] where a logic motivated by the two-variable Bernays-Schönfinkel class extended with datalog is considered. This logic allows to state that some paths exist among constants, however, as it is actually a fragment of first-order logic, it is not able to express transitive closures.

In this paper we consider the universal, two-variable fragment of first-order logic with constants, and extend it with the transitive closure of a single, distinguished binary relation. In contrast to the mentioned fragment with DTC, we allow also for negative occurrences of transitive closures.

In [7] it is shown that if we allow to use the deterministic transitive closure or the transitive closure of a single binary relation both positively and negatively, then the universal fragment of FO becomes undecidable. The proof uses four universally quantified variables. Actually, Corollary 10 from [7] suggests that also the fragment with just two variables, two constants, and the transitive closure of one relation is undecidable. However, the statement of that corollary is not precise and there is no detailed proof. In this paper we clarify this issue by showing that in the case of two variables the satisfiability problem is decidable.

We also find quite intriguing that hardness of the investigated fragment depends on the number of constants (or, alternatively phrased, on the number of existential quantifiers in $\exists^*\forall^2$ formulas).

Our results and outline of the paper. To present our results precisely we introduce the following notation. We denote by $\forall^n_{TC}[m,k]$ the set of first-order formulas of the form $\forall x_1 \ldots x_n \varphi$, with quantifier free φ, over signatures containing m pairs of distinguished binary relation symbols: $R_1, R_1^+, \ldots, R_m, R_m^+$, k constant symbols $c_1, \ldots, c_k$, and no function symbols of arity greater than 0; the equality symbol is also allowed. We consider satisfiability of such formulas over structures in which for all $1 \leq i \leq m$ the interpretation of R_i^+ is the transitive closure of the interpretation of R_i. We define also the classes of formulas in which the number of constants is unbounded as $\forall^n_{TC}[m] = \bigcup_{i=0}^{\infty} \forall^n_{TC}[m,i]$.

We prove that the satisfiability problem for $\forall^2_{TC}[1]$ is decidable in 2-NExpTime (Section 6). In the case of $\forall^2_{TC}[1,2]$ we show even an exponential model property, so it can be decided in NExpTime (Section 4). Slightly surprisingly, $\forall^2_{TC}[1,3]$ lacks the finite model property

(Section 3), but we still are able to show a NExpTime-upper complexity bound (Section 7). The satisfiability problem for $\forall^2_{TC}[1,4]$ becomes 2-ExpTime-hard (Section 5). We also note some contrasting undecidability results, namely for $\forall^3_{TC}[1]$ and $\forall^2_{TC}[2]$ (Section 7).

2 Preliminaries

2.1 Conventions

We mostly work with $\forall^2_{TC}[1]$ and its fragments with bounded number of constants. In this case, we suppose without loss of generality that signatures contain only unary and binary relation symbols (cf. [5]), we denote by R the distinguished binary relation whose transitive closure is available, and use R^+ for this transitive closure. To simplify the presentation we assume that constants are not explicitly present in the signature, but rather they are simulated by means of special unary predicates $K_1, \ldots, K_k$. In this case we require that in a model of a given formula there exists exactly one element satisfying K_i, for all $1 \leq i \leq k$; we simply denote this element by c_i. We do not obey this assumption when presenting example formulas and proving lower bounds. Eliminating constants in favor of such special unary predicates can be done in a standard way.

We use a standard convention and if $\mathfrak{A}$ is a structure then we denote its universe by A. Similarly, if $V \subseteq A$ then we denote by $\mathfrak{V}$ the substructure of $\mathfrak{A}$ induced by V, i.e. $\mathfrak{A}{\restriction}V$.

2.2 Atomic types

An (atomic) 1-*type* (over a given signature) is a maximal satisfiable set of atoms or negated atoms with free variable x. Similarly, an (atomic) 2-*type* is a maximal satisfiable set of atoms and negated atoms with free variables x, y. We assume that literals built using our special symbol R^+ are also members of atomic types. Note that the numbers of 1-types and 2-types are bounded exponentially in the size of the signature. We often identify a type with the conjunction of all its elements.

Observe that in the case of signatures restricted to unary and binary symbols, to completely describe a structure it is enough to list the 2-types of all pairs of elements. However, we usually start our constructions by defining 1-types.

For a given σ-structure $\mathfrak{A}$, and $a \in A$ we say that *a realizes* a 1-type α if α is the unique 1-type such that $\mathfrak{A} \models \alpha[a]$. We denote by $\mathrm{tp}_{\mathfrak{A}}(a)$ the 1-type realized by a. Similarly, for distinct $a, b \in A$, we denote by $\mathrm{tp}_{\mathfrak{A}}(a, b)$ the unique 2-type *realized* by the pair a, b, i.e. the type β such that $\mathfrak{A} \models \beta[a, b]$. We denote by $\boldsymbol{\alpha}[\mathfrak{A}]$ the set of all 1-types, and by $\boldsymbol{\beta}[\mathfrak{A}]$ the set of all 2-types realized in $\mathfrak{A}$. For $S_1, S_2 \subseteq A$, we denote by $\boldsymbol{\alpha}_{\mathfrak{A}}[S_1]$ the set of all 1-types realized in S_1, by $\boldsymbol{\beta}_{\mathfrak{A}}[S_1, S_2]$ the set of all 2-types $\mathrm{tp}_{\mathfrak{A}}(a_1, a_2)$ with $a_i \in S_i$. We sometimes skip subscripts if the structure is clear from the context.

2.3 Small cliques

Let $\mathfrak{A}$ be a structure. We say that $C \subseteq A$ is an R^+-*clique*, or simply a *clique*, if C is a maximal set of elements such that for all distinct $a, b \in C$ we have $\mathfrak{A} \models aR^+b \wedge bR^+a$. In the other words an R^+-clique is a maximal strongly R-connected component in $\mathfrak{A}$. We show that we can restrict our attention to structures with cliques of a bounded size.

▶ **Lemma 1.** *Let φ be a formula in $\forall^2_{TC}[1]$ and let $\mathfrak{A} \models \varphi$. Then there exists a model of φ such that the size of every R^+-clique in this model is bounded exponentially in $|\varphi|$.*

Towards a proof of this lemma we first show how to replace a single R^+-clique C in $\mathfrak{A}$ by its small counterpart C'. In [14] the following lemma is proved.

▶ **Lemma 2.** *Let φ be an FO^2 formula and $\mathfrak{M} \models \varphi$ its strongly R-connected model (an R^+-clique). Then there exists a strongly R-connected model $\mathfrak{M}' \models \varphi$ of size bounded exponentially in $|\varphi|$ such that $\boldsymbol{\alpha}[\mathfrak{M}] = \boldsymbol{\alpha}[\mathfrak{M}']$.*

We apply the above lemma to $\mathfrak{C}$ and $\psi = \varphi \wedge \psi^c$, where $\psi^c = \forall xy \bigwedge_i (K_i(x) \wedge K_i(y) \rightarrow x = y)$, obtaining a structure $\mathfrak{C}'$. In particular $\mathfrak{C}'$ contains realizations of the same special predicates K_i as $\mathfrak{C}$, and each of them is realized at most once. It remains to connect $\mathfrak{C}'$ with $\mathfrak{A}{\restriction}A \setminus C$. For any $a \in A \setminus C$ and any $\alpha \in \boldsymbol{\alpha}[C]$, if there exists $b \in C$, of type α, such that $\mathfrak{A} \models aRb \vee bRa$ then we set $b' = b$. Otherwise we choose an arbitrary element of type α in C as b'. For every element $b'' \in C'$ of type α we set $\mathrm{tp}_{\mathfrak{A}'}(a, b'') = \mathrm{tp}_{\mathfrak{A}}(a, b')$. Let us denote by $\mathfrak{A}'$ the structure so obtained. The proof of the following claim is omitted due to page limit.

▶ **Claim 3.** $\mathfrak{A}'$ is indeed a model of φ.

In the case of a finite model we apply the above step successively to all R^+-cliques, obtaining finally a model with small cliques. For the case of an infinite model, note that $\forall^2_{TC}[1]$ satisfies downward Löwenheim-Skolem property, so we may assume that the initial model is countable, and apply our procedure to all R^+-cliques in countably many steps. The desired model with small R^+-cliques is the natural limit of the described process. This finishes our proof of Lemma 1.

For a pair of distinct elements a, b we say that they are in *free position* in $\mathfrak{A}$ if $\mathfrak{A} \models \neg aR^+b \wedge \neg bR^+a$. A clique C_1 is in free position with C_2 if every element from C_1 is in free position with every element of C_2.

2.4 Saturations

In our constructions it is sometimes convenient to have structures with many R-edges. Let $\mathfrak{A}$ be a structure and let us build $\mathfrak{A}'$ by adding to $\mathfrak{A}$ a number of R-edges, in the following way. If there is a pair of elements $a_1, a_2 \in A$ such that $\mathfrak{A} \models a_1Ra_2 \wedge \neg a_2R^+a_1$ and a pair of elements $b_1, b_2 \in A$, such that $\mathrm{tp}_{\mathfrak{A}}(a_1) = \mathrm{tp}_{\mathfrak{A}}(b_1)$, $\mathrm{tp}_{\mathfrak{A}}(a_2) = \mathrm{tp}_{\mathfrak{A}}(b_2)$ and $\mathfrak{A} \models b_1R^+b_2 \wedge \neg b_1Rb_2 \wedge \neg b_2R^+b_1$, then we modify the 2-type of b_1, b_2 by setting $\mathrm{tp}_{\mathfrak{A}'}(b_1, b_2) = \mathrm{tp}_{\mathfrak{A}}(a_1, a_2)$. We repeat this step until no further modifications are possible. We call the obtained structure an *R-saturation* of $\mathfrak{A}$. A structure which is its own R-saturation is called *R-saturated*.

Note that the R-edges added in the above process do not change the R^+-relations among the elements. As all the modified 2-types are realized in $\mathfrak{A}$, we have the following proposition.

▶ **Proposition 4.** Let φ be a $\forall^2_{TC}[1]$ formula and $\mathfrak{A}$ its model. Then an R-saturation of $\mathfrak{A}$ is an R-saturated model of φ.

3 An infinity axiom

To demonstrate the strength of the considered fragment we show in this section that there exists a $\forall^2_{TC}[1, 3]$-formula $\eta = \forall xy \eta_0$ which is satisfiable but has only infinite models.

We define η_0 as the conjunction of formulas (1)-(3) below.

(1) there exists a path from c_1 to c_2 and there are no R^+-loops.

$$c_1R^+c_2 \wedge \neg xR^+x,$$

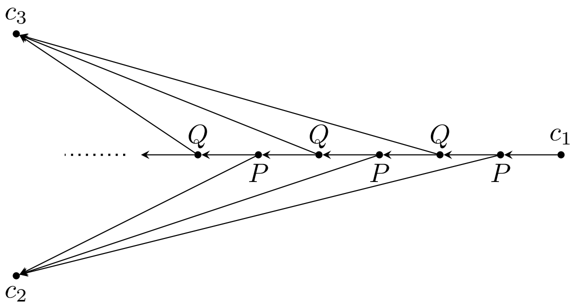


Figure 1 An infinite model of η.

(2) P and Q are disjoint, every element in P has an R^+ path to c_3, and every element in Q has an R^+-path to c_2.

$$(Px \wedge Qx \rightarrow \bot) \wedge (Px \rightarrow xR^+c_3) \wedge (Qx \rightarrow xR^+c_2),$$

(3) R-edges are allowed only between elements of specific types.

$$xRy \rightarrow ((x = c_1 \wedge Py) \vee (Px \wedge Qy) \vee (Qx \wedge Py) \vee (Px \wedge y = c_2) \vee (Qx \wedge y = c_3)).$$

It is not hard to see that η is satisfied in the infinite model depicted in Fig.1. Also any model of η must embed an infinite chain of elements, on which predicates P and Q alternate.

4 A finite model property for formulas with two constants

Now we show that the presence of three constants in the previous section was essential.

▶ **Lemma 5.** *Every satisfiable $\forall^2_{TC}[1,2]$-formula φ has a finite model of size bounded exponentially in $|\varphi|$.*

Let $\mathfrak{A} \models \varphi$ be a model with cliques bounded exponentially in $|\varphi|$, as guaranteed by Lemma 1. By Proposition 4 we may assume that $\mathfrak{A}$ is R-saturated. Let C_1 be the clique containing c_1, and C_2 be the clique containing c_2.

Note that if $C_1{=}C_2$ then $\mathfrak{A}\restriction C_1 \models \varphi$, and that if $\mathfrak{A} \models \neg c_1R^+c_2 \wedge \neg c_2R^+c_1$ then $\mathfrak{A}\restriction C_1 \cup C_2 \models \varphi$. In both cases we have finite models of φ of exponentially bounded size.

Consider the case when $\mathfrak{A} \models c_1R^+c_2 \wedge \neg c_2R^+c_1$ (the symmetric case can be treated analogously). Let us take a shortest path π from c_1 to c_2. Let us write π as $c_1 = a_{11}, a_{12}, \ldots, a_{1k_1},\ a_{21}, a_{22}, \ldots, a_{2k_2}, \ldots, a_{l1}, a_{l2}, \ldots, a_{lk_l} = c_2$, where for each i the path $a_{i1}, \ldots, a_{ik_i}$ is the maximal fragment of π containing elements from the same clique. We denote by C_i the clique containing the elements a_{ij}. Observe that if π leaves a clique C_i then it never enters it again, i.e. if $1 \leq i < j \leq l$ then $C_i \neq C_j$.

We claim that $\mathfrak{A}' = \mathfrak{A}\restriction C_1 \cup \ldots \cup C_l$ is a model of φ. Indeed, if two elements belong to the same clique in $\mathfrak{A}'$ then they also belong the same clique in $\mathfrak{A}$; if a pair of elements is connected non-symmetrically by R^+ in $\mathfrak{A}'$ then they are also connected non-symmetrically by R^+ in $\mathfrak{A}$; finally, there are no elements in free position in $\mathfrak{A}'$. Thus all atomic 2-types realized in $\mathfrak{A}'$ are also realized in $\mathfrak{A}$, which implies that $\mathfrak{A}' \models \varphi$. Note that taking whole cliques of elements from π to $\mathfrak{A}'$, instead of considering just $\mathfrak{A}\restriction\pi$, is important, as φ may require some elements to lie on an R-cycle.

We claim that the size of $\mathfrak{A}'$ is bounded exponentially in $|\varphi|$. This follows from the fact that for $1 \leq i < j \leq l$ we have $\mathrm{tp}(a_{i1}) \neq \mathrm{tp}(a_{j1})$. Indeed, assume to the contrary that for some i, j we have that $\mathrm{tp}(a_{i1}) = \mathrm{tp}(a_{j1})$. Then the path π' obtained from π by removing the fragment $a_{i1}, \ldots, a_{j-1,k_{j-1}}$ is a path from c_1 to c_2, which is shorter than π. Note that π' is indeed an R-path, since $\mathfrak{A} \models a_{i-1,k_{i-1}} R a_{i1}$, and thus, by R-saturation of $\mathfrak{A}$, we have also $\mathfrak{A} \models a_{i-1,k_{i-1}} R a_{j1}$. Thus the number of cliques in $\mathfrak{A}'$ is not greater than $|\boldsymbol{\alpha}|$, the size of every clique is bounded exponentially in $|\varphi|$, and thus also $|A'|$ is bounded exponentially in $|\varphi|$.

This finishes the proof of Lemma 5. It naturally leads to the following complexity result.

▶ **Theorem 6.** *The satisfiability problem for* $\forall^2_{TC}[1, 2]$ *is decidable in* NExpTime.

A corresponding lower bound can be obtained even in the absence of constants (assuming that we consider satisfiability in non-empty structures). The idea is similar to the proof of Theorem 5 from [7]. We construct a formula whose models are grids of exponential size. Instead of using two constants to distinguish the left-upper and the right-lower corners of the grid we say that every element is R-reachable from itself but not by a direct R-edge: $xR^+x \wedge \neg xRx$. We allow edges only between elements which are neighbors on a snake-like path through the whole grid. We allow also for an R-edge from the right-lower corner to the left-upper corner. Thus models are R-cycles which have to contain all elements of the grid.

▶ **Theorem 7.** *The satisfiability problem for* $\forall^2_{TC}[1, 0]$ *is* NExpTime*-hard.*

5 Lower bound for formulas with four constants

Now we show that in the presence of four constants the lower bound for the satisfiability problem can be lifted to 2-ExpTime. To simplify the presentation we assume first that there are nine constants available, and then we present a trick which allows to get rid of five of them.

5.1 A construction involving nine constants

The proof goes by a reduction from alternating Turing machines with exponentially bounded space. The general idea of the proof and the shape of intended models are similar to the ones used in [9]. However, the lack of existential quantifiers makes the tasks of enforcing desired shapes of models and then simulating Turing machines more tricky.

Tree-like structures. To simulate a run of an alternating Turing machine it is convenient to have a structure which resembles an infinite binary tree, with each node being able to encode a single configuration, and identify its successor nodes. Let us describe how to enforce a desired structure.

We use unary predicates $P_0, \ldots, P_{n-1}$ and assume that for any element a they encode a value $0 \leq \bar{P}(a) < 2^n$ in a natural way, i.e. $P_i(a)$ is true exactly if the ith bit of the binary representation of $\bar{P}(a)$ is equal to 1. Let us abbreviate by $\bar{P}(x) = \bar{P}(y)$, $\bar{P}(x) = \bar{P}(y) + 1$, $\bar{P}(x) = k$ (for $0 \leq k < 2^n$) quantifier-free formulas with an obvious meaning. Such formulas can be constructed of size polynomial in n in a standard fashion.

We say that elements $a_0, \ldots, a_{2^n-1}$ form a *node* in a structure $\mathfrak{A}$ if $\bar{P}(a_i) = i$ and $\mathfrak{A} \models a_{i-1} R a_i$ for $0 < i < 2^n$. The purpose of a node will be to encode information about a single configuration of a Turing machine. We use unary predicates H^d_i for $0 \leq i < 4$, $d \in \{L, R\}$ to distinguish eight types of nodes. An additional predicate H^I serves for distinguishing an initial node.

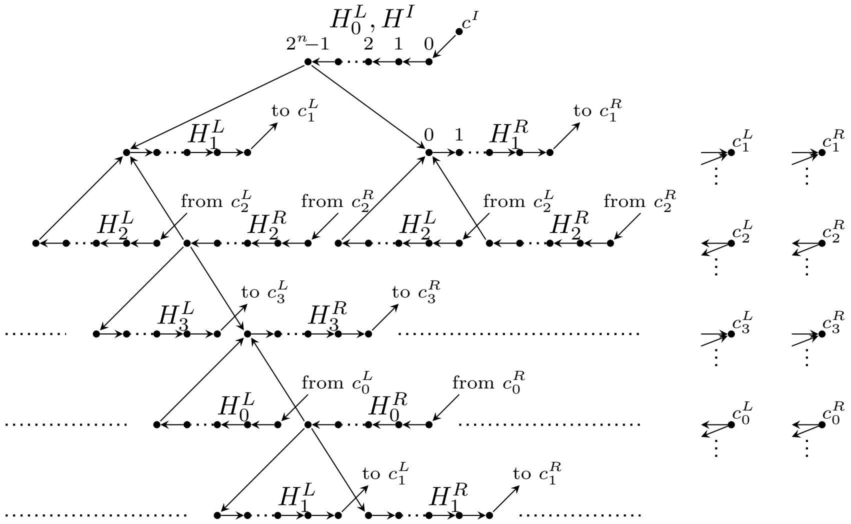


Figure 2 An initial fragment of the structure $\mathfrak{T}$ from the proof of the lower bound.

Let $\mathfrak{T}$ be the structure depicted in Fig. 2. It is drawn in a way suggesting its similarity to a binary tree, note however that actually this structure is shallow: every R-path has length not greater than $2^n + 2$.

▶ **Claim 8.** There exists a formula λ such that:

(a) $\mathfrak{T} \models \lambda$

(b) any model $\mathfrak{A} \models \lambda$ locally resembles $\mathfrak{T}$, i.e. there exists a node of type H_0^L satisfying H^I, and for every node $a_0, \ldots, a_{2^n-1}$ of type H_i^d there exists a *left successor node* $a_0^L, \ldots, a_{2^n-1}^L$ of type $H_{i+1 \mod 4}^L$ and a *right successor node* $a_0^R, \ldots, a_{2^n-1}^R$ of type $H_{i+1 \mod 4}^R$ such that if i is even then $\mathfrak{A} \models a_{2^n-1}Ra_0^L \wedge a_{2^n-1}Ra_0^R$ and if i is odd then $\mathfrak{A} \models a_{2^n-1}^L Ra_0 \wedge a_{2^n-1}^R Ra_0$.

We construct λ from five conjuncts. Conjuncts (1) and (2) say that for some elements there are paths from or to some constants. Conjuncts (3)-(5) say that R-edges are allowed only between elements of specific 1-types (actually only such types whose realizations are connected by an R-edge in $\mathfrak{T}$). Below we describe these conjuncts in more details.

(1) there is an R-path from c^I to c_1^L.

(2) every element satisfying H_0^L or H_0^R can reach (by some R^+-paths) elements c_1^L and c_1^R; every element satisfying H_2^L or H_2^R can reach elements c_3^L and c_3^R; every element satisfying H_1^L or H_1^R can be reached from elements c_2^L and c_2^R; every element satisfying H_3^L or H_3^R can be reached from elements c_0^L and c_0^R.

(3) (edges incident to constants) for $i \in \{0, 2\}$ and $d \in \{L, R\}$ element c_i^d has no incoming R-edges, and has outgoing R-edges only to elements a such that $\bar{P}(a) = 0$ and $H_i^d(a)$ holds; for $i \in \{1, 3\}$ and $d \in \{L, R\}$ element c_i^d has no outgoing R-edges, and has incoming R-edges only from elements a such that $\bar{P}(a) = 2^n - 1$ and $H_i^d(a)$ holds; c^I has no incoming R-edges and has outgoing R-edges only to elements a such that $\bar{P}(a) = 0$ and $H_0^L(a) \wedge H^I(a)$ holds.

(4) (edges inside nodes) if an element a satisfies $\bar{P}(a) < 2^n - 1 \wedge H_i^d(a)$ than it has outgoing edges only to elements b satisfying $H_i^d(b)$ such that $\bar{P}(b) = \bar{P}(a) + 1$ and $H^I(a) \leftrightarrow H^I(b)$;

if an element a satisfies $\bar{P}(a) > 0 \wedge H_i^d(a)$ than it has incoming edges only from elements b satisfying $H_i^d(b)$ such that $\bar{P}(a) = \bar{P}(b) + 1$ and $H^I(a) \leftrightarrow H^I(b)$;

(5) (edges among nodes) an element a such that $\bar{P}(a) = 2^n - 1$ and $H_i^d(a)$ for $i \in \{0, 2\}$ hold has incoming edges only from elements in H_i^d, and has outgoing edges only to elements b such that $\bar{P}(b) = 0$ and $H_{i+1}^L(b) \vee H_{i+1}^R(b) \vee H_{i-1 \mod 4}^L(b) \vee H_{i-1 \mod 4}^R(b)$; an element a such that $\bar{P}(a) = 0$ and $H_i^d(a)$ for $i \in \{1, 3\}$ hold has an outgoing edges only from elements in H_i^d, and has incoming edges only from elements b such that $\bar{P}(b) = 2^n - 1$ and $H_{i+1 \mod 4}^L(b) \vee H_{i+1 \mod 4}^R(b) \vee H_{i-1}^L(b) \vee H_{i-1}^R(b)$.

Clearly $\mathfrak{T} \models \lambda$. Consider an arbitrary model $\mathfrak{A} \models \lambda$. By (1) there must be a path from C^I to c_1^L. By (3) this path must begin with an edge to an element a_0 such that $\mathfrak{A} \models \bar{P}(a_0) = 0 \wedge H^I(a_0) \wedge H_0^L(a_0)$. Then, by (4) this path must go through a whole node of type H_0^L, satisfying also H^I. The last element of this node must have by (2) a path to c_1^L and a path to c_1^R. By (5) the first of this paths must go through an element a_1^L satisfying $\mathfrak{A} \models \bar{P}(a_1^L) = 0 \wedge H_1^L(a_1^L)$, and the other through an element a_1^R satisfying $\mathfrak{A} \models \bar{P}(a_1^R) = 0 \wedge H_1^R(a_1^R)$. Both paths must then go through whole nodes of appropriate types. Elements a_1^L and a_1^R must have by (2) paths from c_2^L and c_2^R, which again have to go through whole nodes of types H_2^L and H_2^R. This reasoning can be generalized to an inductive argument that part (b) of Claim 8 holds.

Simulating alternating Turing machines. A well-known theorem from [1] says that 2-ExpTime is equal to AExpSpace, the class of problems solvable by *alternating* Turing machines in exponentially bounded space.

For a given alternating machine M and its input w we can construct a formula κ_w^M which is satisfiable iff M accepts w. We define κ_w^M as the conjunction of λ and some formulas encoding computations of M. Every element of a model of λ corresponds to single tape cell of M, and stores information about this cell, as well as about the two neighboring cells. Thus, formulas of the form $(xR^+y \wedge \bar{P}(x) = \bar{P}(y) \wedge H_i^d(x) \wedge H_{i+1 \mod 4}^d(y)) \rightarrow \dots$ can be used to say that two consecutive nodes of a model describe two consecutive configurations of M. Details are omitted due to space limit.

5.2 Four constants suffice

The following lemma will be used to reduce the number of constants required in the proof of 2-ExpTime-hardness from nine to four. Actually, it has a stronger statement and allows to reduce satisfiability of $\forall_{TC}^2[1, n]$ and even $\forall_{TC}^2[1]$ in polynomial time to satisfiability of $\forall_{TC}^2[1, 4]$, assuming that we consider only structures in which relation R^+ restricted to constants is a partial order.

▶ **Lemma 9.** *For each n and each $\forall_{TC}^2[1, n]$ sentence φ there is a polynomially computable $\forall_{TC}^2[1, 4]$ formula φ' such that φ' has a model if and only if φ has a model in which for all $i < j$ there is no R-path from c_j to c_i.*

We sketch the main idea of the proof. Assume that constants $c_1, \dots, c_4$ are available. We simulate n additional constants by n fresh unary predicates $S_1, \dots, S_n$. We use also auxiliary unary predicates $P_1, \dots, P_n$, $Q_1, \dots, Q_n$. We say that each of the predicates S_i, P_i, Q_i is satisfied in at most one element. We want to enforce that each of S_i is satisfied at least once, and for $i < j$, each pair of realizations of S_i, S_j may appear either in free position or may be connected by an R^+ path from the one satisfying S_i to the one satisfying S_j. To do so we enforce first the upper and the lower horizontal chains of elements from Fig. 3. Then we say that the element satisfying P_i has an R-path to the element satisfying

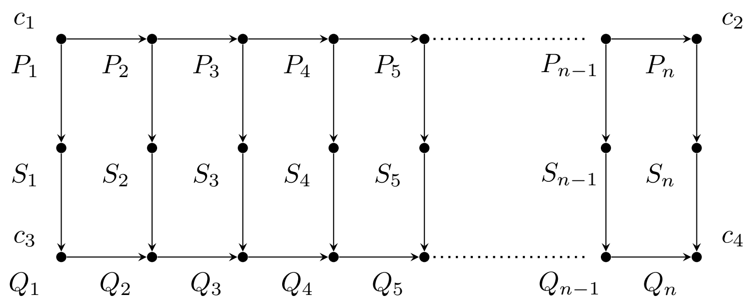


Figure 3 A model of ψ.

Q_i. By an appropriate restriction of 2-types containing R we can enforce that these paths go through elements satisfying S_i. We guarantee that all S_i are realized, by saying that there are no R-paths from P_i to Q_j for $i > j$. Here the assumption from the statement of the lemma, about admissible R^+-connections among constants is relevant. Details of the proof of Lemma 9 are omitted due to space limit.

We are now ready to formulate the following theorem.

▶ **Theorem 10.** *The satisfiability problem for* $\forall^2_{TC}[1, 4]$ *is 2-*ExpTime*-hard.*

Proof. We define λ^* by renaming the constants in λ in the following way: $c^I \to c_1$, $c_2^L \to c_2$, $c_2^R \to c_3$, $c_4^L \to c_4$, $c_4^R \to c_5$, $c_1^L \to c_6$, $c_1^R \to c_7$, $c_3^L \to c_8$, $c_3^R \to c_9$. Clearly, renaming the constants does not change the properties of formulas. Moreover, λ^* guarantees that $c_1 - c_5$ have no incoming edges and $c_6 - c_9$ have no outgoing edges, and therefore in any model of λ^* there are no paths from c_j to c_i for any $i < j$. We apply Lemma 9 to λ^* obtaining λ'. We can now replace λ by λ' when constructing κ_w^M from the previous subsection. ◀

6 Decidability of formulas with an unbounded number of constants

In this section we show that the satisfiability problem for $\forall^2_{TC}[1]$ is decidable. We use a standard approach which consists in an analysis of arbitrary models and rebuilding them to obtain a shape which admits descriptions of a bounded size. In this case we show that every formula has a model which can be divided into at most doubly exponentially many fragments, called *zones*, each of which is either a clique or an infinite, regular chain of cliques.

6.1 Clique types

Let $\mathfrak{A}$ be a structure. We say that a clique C has a *clique type* $\delta = (\mathcal{C}, \mathcal{A}, \mathcal{B})$ in $\mathfrak{A}$, if $\mathcal{C}$ is the set of atomic 1-types realized in C, $\mathcal{A}$ is the set of atomic 1-types of the elements located *above* C, i.e. the elements b such that for all $a \in C$ we have $\mathfrak{A} \models bR^+a \wedge \neg aR^+a$, and $\mathcal{B}$ is the set of atomic 1-types of the elements located *below* C, i.e. the elements b such that for all $a \in C$ we have $\mathfrak{A} \models aR^+b \wedge \neg bR^+a$. We denote by $\Delta[\mathfrak{A}]$ the set of all clique types realized in $\mathfrak{A}$. Note that $|\Delta[\mathfrak{A}]|$ is bounded doubly exponentially in the signature.

6.2 Zones

For a pair of cliques C_1, C_2 we write $C_1 \leq_c C_2$ if $C_1 = C_2$ or for all $a_1 \in C_1, a_2 \in C_2$ we have $a_1R^+a_2$. Relation $\leq_c$ naturally induces a relation $\leq_\delta$ on clique types. We define:

$\delta_1 \leq_\delta \delta_2$ iff there exist cliques $C_1, C_2 \subseteq A$, C_i of type δ_i, such that $C_1 \leq_c C_2$. Let $\leq_\delta^*$ be the transitive closure of $\leq_\delta$. Let $\delta_1 \approx \delta_2$ iff $\delta_1 \leq^* \delta_2$ and $\delta_2 \leq^* \delta_1$. Clearly, $\approx$ is an equivalence relation over $\Delta[\mathfrak{A}]$. The set of elements of $\mathfrak{A}$, belonging to the cliques realizing the extended types from the same equivalence class of $\approx$, is called a *zone*. Note that the number of zones of $\mathfrak{A}$ is bounded doubly exponentially in the signature.

We say that a zone V is *singular* if every R^+-connection inside V is symmetric. A few simple properties of zones, having straightforward proofs, are collected below.

▶ **Proposition 11.** **(i)** Let $\delta_1 = (\mathcal{C}_1, \mathcal{A}_1, \mathcal{B}_1)$ and $\delta_2 = (\mathcal{C}_2, \mathcal{A}_2, \mathcal{B}_2)$ be two clique types realized in a zone V. Then $\mathcal{A}_1 = \mathcal{A}_2$ and $\mathcal{B}_1 = \mathcal{B}_2$.

(ii) If a zone V is singular then V contains only realizations of a single clique type.

(iii) Let $\delta = (\mathcal{C}, \mathcal{A}, \mathcal{B})$ be a clique type realized in a non-singular zone V. Then for every $\alpha \in \mathcal{C}$ we have $\alpha \in \mathcal{A}$ and $\alpha \in \mathcal{B}$.

(iv) Let α_1 and α_2 be atomic types realized in a non-singular zone V. Then there exists a pair of elements a_1, a_2 in $\mathfrak{A}$ (but not necessarily in $\mathfrak{V}$) such that $\mathrm{tp}(a_1) = \alpha_1$, $\mathrm{tp}(a_2) = \alpha_2$, and $\mathfrak{A} \models a_1 R^+ a_2 \wedge \neg a_2 R^+ a_1$.

(v) Let π be a path connecting two elements belonging to a non-singular zone V. Then every element a on π belongs to V.

6.3 Making zones regular

Let V be a zone in a structure $\mathfrak{A}$. We show how to replace $\mathfrak{V}$ by a zone $\mathfrak{V}'$ being either a single clique or an infinite, regular chain of cliques, in such a way that the resulting structure $\mathfrak{A}'$ satisfies all $\forall^2_{TC}[1]$ formulas satisfied in $\mathfrak{A}$.

Building a singular zone. If V is singular then it consists of some number of cliques in free position, and, by Proposition 11 (ii), all of them have the same clique type δ. In this case $\mathfrak{V}'$ is a single realization of δ.

Building a non-singular zone. Consider a non-singular zone V. By Proposition 11 (i) there are $\mathcal{A}$, $\mathcal{B}$ such that every clique type realized in $\mathfrak{V}$ has the form $(\mathcal{C}, \mathcal{A}, \mathcal{B})$ for some set $\mathcal{C}$. The construction of a regular version of a $\mathfrak{V}$ relies on the following proposition.

▶ **Proposition 12.** There exists a sequence of (not necessarily distinct) clique types $\delta_0, \ldots, \delta_{l-1}$, where $\delta_i = (\mathcal{C}_i, \mathcal{A}, \mathcal{B})$, and atomic types $\alpha_0^{in}, \alpha_0^{out}, \ldots, \alpha_{l-1}^{in}, \alpha_{l-1}^{out}$ such that:

(a) l is bounded exponentially in the size of the signature,

(b) for every $\alpha \in \boldsymbol{\alpha}[V]$ there exists i such that $\alpha \in \mathcal{C}_i$,

(c) for every i, δ_i is a clique type of a clique in $\mathfrak{V}$,

(d) for every i we have $\alpha_i^{in}, \alpha_i^{out} \in \mathcal{C}_i$,

(e) for every i there exists in $\mathfrak{A}$ a realization a of α_i^{out} and a realization b of $\alpha_{i+1 \bmod l}^{in}$ such that $\mathfrak{A} \models aRb \wedge \neg bR^+a$.

Proof. Let $\delta'_0, \ldots, \delta'_{s-1}$ be an enumeration of all clique types from $\Delta[\mathfrak{V}]$. By the definition of a zone and the relation $\leq_\delta$ there is a $\leq_\delta$-path from δ'_i to $\delta'_{i+1 \bmod s}$, for every $0 \leq i < s$. By concatenating such paths we obtain a sequence $\delta_0, \ldots, \delta_{t-1}$ meeting conditions (b)-(e) (assuming a natural choice of α_i^{in} and α_i^{out}). In this path we choose for every $\alpha \in \boldsymbol{\alpha}[\mathfrak{V}]$ a clique type $\delta_\alpha = (\mathcal{C}_\alpha, \mathcal{A}_\alpha, \mathcal{B}_\alpha)$ such that $\alpha \in \mathcal{C}_\alpha$. Observe that if $\alpha_i^{in} = \alpha_j^{in}$ for some $0 \leq i < j < t$ such that $\delta_i, \ldots, \delta_{j-1}$ does not contain any δ_α then we can remove $\delta_i, \ldots, \delta_{j-1}$ from the sequence without violating conditions (b)-(e). This observation allows to easily shorten the sequence to a required length. ◀

Let $\delta_0, \ldots, \delta_{l-1}$ be a sequence of clique types guaranteed by Proposition 12. We construct $\mathfrak{V}'$ as an infinite chain of cliques $\ldots C_{-2}, C_{-1}, C_0, C_1, C_2, \ldots$ such that the clique C_i has type $\delta_{i \bmod l}$. For every pair $\alpha_1, \alpha_2 \in \boldsymbol{\alpha}[V]$ we choose a 2-type $\beta_{1\to2} \models xR^+y \wedge \neg yR^+x \wedge \alpha_1(x) \wedge \alpha_2(y)$ realized in $\mathfrak{A}$. An appropriate $\beta_{1\to2}$ exists in $\boldsymbol{\beta}[\mathfrak{A}]$ by Proposition 11 (iv). If it is possible we choose $\beta_{1\to2}$ containing xRy. For all $a_1 \in C_i, a_2 \in C_j$, $i < j$, such that $\mathrm{tp}(a_1) = \alpha_1$, $\mathrm{tp}(a_2) = \alpha_2$ we set $\mathrm{tp}(a_1, a_2) := \beta_{1\to2}$. This finishes the construction of $\mathfrak{V}'$. Note that by our choice of atomic 2-types and condition (e) from Proposition 12, we have that for all $i < j$ there exists an R-path from each element of C_i to each element of C_j.

Connecting a rebuilt zone to the remaining part of the model. Consider an element $a \in A \setminus V$. Let $\alpha = \mathrm{tp}_{\mathfrak{A}}(a)$. We distinguish three cases.

Case 1: In $\mathfrak{A}$ element a is in free position with all elements in V. For any 1-type $\alpha' \in \boldsymbol{\alpha}[\mathfrak{V}']$ we find an element $b \in V$ of type α' (such an element exists as our construction ensures that $\boldsymbol{\alpha}[\mathfrak{V}'] = \boldsymbol{\alpha}[\mathfrak{V}]$), and for any $b' \in V'$ of type α' we set $\mathrm{tp}_{\mathfrak{A}'}(a, b') = \mathrm{tp}_{\mathfrak{A}}(a, b)$. Clearly this ensures that a is in free position with all elements from V'.

Case 2: In $\mathfrak{A}$ there is an R-path from a to an element of V. For any 1-type $\alpha' \in \boldsymbol{\alpha}[V]$:

- if there exists a realization $b \in V$ of α' such that $\mathfrak{A} \models aRb$ then for all $b' \in V'$ of type α' we set $\mathrm{tp}_{\mathfrak{A}'}(a, b') = \mathrm{tp}_{\mathfrak{A}}(a, b)$.
- otherwise find a realization $b \in A$ of α' such that $\mathfrak{A} \models aR^+b$ and for all $b' \in V'$ of type α' we set $\mathrm{tp}_{\mathfrak{A}'}(a, b') = \mathrm{tp}_{\mathfrak{A}}(a, b)$. Note that in this subcase the existence of an appropriate b is guaranteed by the properties of relation $\leq_\delta$, but sometimes we need to look for it outside V.

Note that in this case element a has an R-path in $\mathfrak{A}'$ to every element from V'. Indeed, on a path from a to an element of V there must be an element, say b, which has an R-edge to a point from V. This element b will be made R^+-connected to all elements from V; in particular, if V is non-singular it will have R-edges to infinitely many elements of V.

Case 3: In $\mathfrak{A}$ there is an R-path from an element of V to a. Proceed analogously to Case 2.

Modifying the remaining part of the model. To complete the construction of $\mathfrak{A}'$ consider a pair of elements $a_1, a_2 \in A \setminus V$. If $\mathfrak{A} \models a_1R^+b \wedge b'R^+a_2$ (or symmetrically $\mathfrak{A} \models a_2R^+b \wedge b'R^+a_1$) for some elements $b, b' \in V$ then a_1 becomes R^+-connected to a_2 in $\mathfrak{A}'$, even if they are not connected in $\mathfrak{A}$. Note that in this case $a_1 \in \mathcal{A}$ and $a_2 \in \mathcal{B}$. This means that there is a pair of realizations a_1', a_2' of $\mathrm{tp}(a_1)$ and $\mathrm{tp}(a_2)$ in $\mathfrak{A}$ such that $\mathfrak{A} \models a_1'R^+a_2'$. We set in this case $\mathrm{tp}_{\mathfrak{A}'}(a_1, a_2) = \mathrm{tp}_{\mathfrak{A}}(a_1', a_2')$ (and proceed analogously in the symmetric case). In the opposite case there is no R^+-path in $\mathfrak{A}'$ between a_1 and a_2 and we can safely set $\mathrm{tp}_{\mathfrak{A}'}(a_1, a_2) = \mathrm{tp}_{\mathfrak{A}}(a_1, a_2)$.

▶ **Proposition 13.** Let $\mathfrak{A}$ be a model of an $\forall^2_{TC}[1]$ formula φ. Then there exists a model $\mathfrak{A}' \models \varphi$ in which all zones are either single cliques or infinite, regular chains of cliques, with regular connections among zones.

Proof. We simply repeat the described procedure successively to all zones, obtaining finally a model of a desired shape. ◀

6.4 Decidability procedure

A structure of a shape as in Proposition 13 can be described in a natural way. Such a description contains for every zone a sequence of clique types guaranteed by Proposition 12, patterns of connections among them, and for every pair of zones a pattern of connection between every clique type from the first zone and every clique type from the second zone.

To check if a given formula φ in $\forall^2_{TC}[1]$ has a model we guess such a description of a regular model. Verifying that a guessed description indeed produces a model of φ is easy and can be done in polynomial time with respect to its size. As the number of zones is bounded doubly exponentially in the size of the signature, and thus also in $|\varphi|$, the whole description of a regular structure is also bounded doubly exponentially. Thus we obtain:

▶ **Theorem 14.** *The satisfiability problem for* $\forall^2_{TC}[1]$ *is decidable in 2-*NExpTime.

7 NExpTime-upper bound for formulas with three constants

In this section we show that $\forall^2_{TC}[1,3]$, even though it lacks a finite model property, is still decidable in NExpTime. For a given structure $\mathfrak{A}$ we say that a sequence $V_1,\ldots,V_k$ of zones is a *path of zones* if for each i there exist $v_i \in V_i, v_{i+1} \in V_{i+1}$ such that $\mathfrak{A} \models v_i R v_{i+1}$. Note that in this case, in models guaranteed by Proposition 13 a path from each element of V_i to each element of V_j exists for $i < j$.

▶ **Definition 15.** Let $\mathfrak{A}$ be a model with regular zones as in Proposition 13. We say that $\mathfrak{A}$ is *downward fork-like* if it consists of four zones $V_0,\ldots,V_3$, containing all constants, and some number of zones forming a path from V_1 to V_0, a path from V_0 to V_2, and a path from V_0 to V_3. Similarly $\mathfrak{A}$ is *upward fork-like* if it consists of four zones $V_0,\ldots,V_3$, containing all constants, and some number of zones forming a path from V_0 to V_1, a path from V_2 to V_0, and a path from V_3 to V_0. A structure is *fork-like* if it is downward or upward fork-like. Zone V_0 is called a *splitting zone* of the structure. We start from the following observation.

▶ **Lemma 16.** *If an* $\forall^2_{TC}[1,3]$ *formula* φ *has a fork-like model* $\mathfrak{A}$ *then it has a fork-like model in which the number of zones is bounded exponentially in* $|\varphi|$.

Proof. We show a proof for the case in which $\mathfrak{A}$ is downward fork-like. The case of an upward fork-like structure is analogous. Let $\mathfrak{A}'$ be the R-saturation of $\mathfrak{A}$. Note that R-saturation does not change the division into cliques and zones. Let $V_0,\ldots,V_3$ be as in Definition 15. Let π_{10} be some shortest path of zones from V_1 to V_0, π_{02} a shortest path of zones from V_0 to V_2 and π_{03} a shortest paths of zones from V_0 to V_3. Note that $\mathfrak{A}'' = \mathfrak{A}' \restriction \pi_{10} \cup \pi_{02} \cup \pi_{03}$ is still a model of φ. By R-saturation of $\mathfrak{A}'$ and an argument similar to the one used in the proof of Lemma 5 the number of zones in $\mathfrak{A}''$ is bounded exponentially in the size of φ. ◀

Our plan is to show that every satisfiable formula φ in $\forall^2_{TC}[1,3]$ has either a finite, exponentially bounded model, or a fork-like model.

Let $\mathfrak{A} \models \varphi$ be a regular model guaranteed by Proposition 13. Let c_1, c_2, c_3 be the elements satisfying K_1, K_2, K_3, resp.

Simple cases.

- If two of c_1, c_2, c_3 belong to the same zone, then they belong to the same clique. In this case we may construct a finite model as in Section 4.
- If one of the constants, say c_3 is in free position with both the remaining constants, then we construct a model consisting of the cliques of c_1 and c_2, a path between them, if such a path exists, and the clique of c_3. The path between c_1 and c_2 can be then shortened to an exponential length as in Section 4.
- If there exists a path from one of the constants to another, containing the third one then again we may use the construction from Section 4 to obtain a path of exponential size.

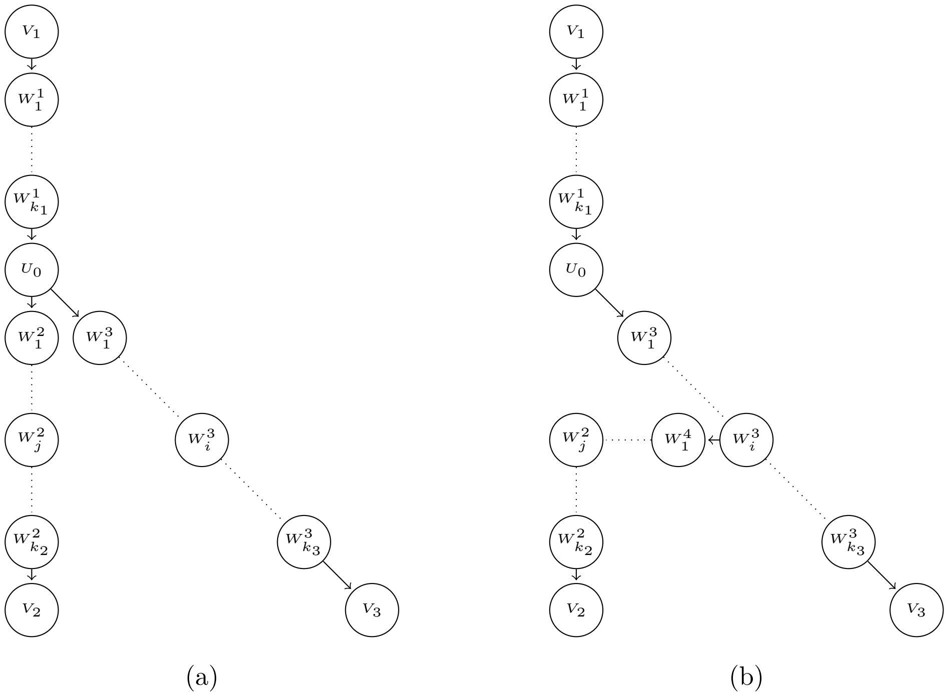


Figure 4 Fork-like structures $\mathfrak{A}_0$ and $\mathfrak{A}_1$ from the proof.

As demonstrated, in each of the above cases there exists a finite, exponentially bounded model of φ.

Fork-like case. A more interesting case is when the constants belong to three distinct zones, two of them, say c_1, c_2 are in free position, and the third one, c_3, can reach both c_1 and c_2 by R-paths, or, symmetrically, can be reached from both c_1 and c_2 by R-paths. Assume, e.g., that $\mathfrak{A} \models c_1R^+c_2 \wedge c_1R^+c_3 \wedge \neg c_2R^+c_3 \wedge \neg c_3R^+c_2$. Let V_1, V_2, V_3 be the zones of c_1, c_2, resp. c_3. Let π_{12} be a path of zones $V_1, W_1^1, \ldots, W_{k_1}^1, U_0, W_1^2, \ldots, W_{k_2}^2, V_2$ from V_1 to V_2, with U_0 being a zone from which a path to V_3 exists. Let π_{03} be a path of zones $U_0, W_1^3, W_2^3, \ldots W_{k_3}^3, V_3$, from U_0 to V_3. See Fig. 4(a).

Note that $\mathfrak{A}_0 \restriction \pi_{12} \cup \pi_{03}$ is a downward fork-like structure, splitting at zone U_0. Observe also that in $\mathfrak{A}_0$ the formula φ cannot be violated by a pair of elements, such that one of them belongs to the fragment $V_1, \ldots, U_0$ of π_{12}. However, it is not necessarily the case that $\mathfrak{A}_0 \models \varphi$, as some elements belonging to zones located below U_0 may be required to be connected by R-paths. Assume e.g. that an element from W_i^3 is connected in $\mathfrak{A}$ to an element in W_j^2. Let $W_i^3, W_1^4, \ldots, W_{k_4}^4, W_j^2$ be a path of zones. Observe now that the structure $\mathfrak{A}'$ consisting of the path of zones $V_1, W_1^1, \ldots, W_{k_1}^1, U_0, W_1^3, \ldots, W_{i-1}^3, W_i^3, W_1^4, \ldots, W_{k_4}^4, W_j^2, W_{j+1}^2, \ldots, W_{k_2}^2, V_2$, and the path $W_{i+1}^3, \ldots, W_{k_3}^3, V_3$ is a downward fork-like structure splitting at zone W_i^3. Denote $U_1 = W_i^3$, and observe that U_1 is located below U_0. See Fig. 4(b). If $\mathfrak{A}_1$ is still not a model of φ we repeat the above step obtaining a fork-like structure $\mathfrak{A}_2$, splitting at U_2, such that U_2 is located below U_1, and so on. Thus a descending sequence of zones $U_0, U_1, \ldots$ is formed, and as the number of zones is finite this process must eventually end in a structure which is a model of φ.

The described construction, together with Lemma 16 allows us to state:

▶ **Theorem 17.** *The satisfiability problem for* $\forall^2_{TC}[1,3]$ *is in* NExpTime.

8 Related undecidability results

To complete the picture we observe that the decidable fragment we have identified is very close to the border between decidability and undecidability. Namely, we show that adding a third variable or the transitive closure of a second binary symbol lead to undecidability.

▶ **Theorem 18.** *The satisfiability and the finite satisfiability problems for* $\forall^3_{TC}[1]$ *and* $\forall^2_{TC}[2]$ *are undecidable.*

The proof for $\forall^3_{TC}[1]$ can be obtained by a slight refinement of the proof of Corollary 9 from [7], which states that $\forall^4_{TC}[1]$ is undecidable. In that proof a snake-like path from the upper-left corner to the lower-right corner of the grid is enforced. Additional R-edges, necessary to define vertical adjacency relation, are enforced by a *completing squares* formula with four variables. If we allow for additional diagonal R-edges then this formula can be replaced by a *completing triangles* formula with three variables. We also remark that this proof requires only a single constant: to mark the upper-left corner of the grid. We require this constant to lie on a cycle and accept an incoming edge only from the opposite corner of the grid.

The proof for $\forall^2_{TC}[2]$ can be obtained by an adaptation of the proof of the undecidability of FO^2 with two transitive relations from [10]. This adaptation uses similar ideas to the proof of the 2-ExpTime-lower bound for $\forall^2_{TC}[1,4]$ from Section 5: appropriate neighbors of elements of the grid, which in the proof from [10] are enforced explicitly by formulas with existential quantifiers in our case can be enforced by requiring that some elements have paths to or from some constants, and by appropriate restriction of of 1-types which may be related by R-edges.

9 Conclusions

We have identified an interesting decidable fragment of two-variable logic with transitive closure operator, $\forall^2_{TC}[1]$. This fragment, even though does not allow explicitly for existential quantifiers, is sufficiently strong to admit infinity axioms and encodings of alternating Turing machines with exponentially bounded space.

Regarding the influence of the number of constants k on the finite model property and the complexity of $\forall^2_{TC}[1,k]$ we have drawn the following picture.

complexity:	NExpTime-complete				between 2-ExpTime and 2-NExpTime				
finite model property:	yes			no (infinity axioms)					
# of constants:	0	1	2	3	4	$\cdots$	l	$\cdots$	unbounded

In fact, our construction of a regular model $\mathfrak{A}'$ of φ from its arbitrary model $\mathfrak{A}$ retains more properties than those expressible in $\forall^2_{TC}[1]$. In particular $\mathfrak{A}'$ realizes only clique types realized in $\mathfrak{A}$. Thus we may add for free to our language existential statements of the form $\forall x(\chi_1(x) \rightarrow \exists y(xR^+y \wedge \chi_2(y)))$ or $\forall x(\chi_1(x) \rightarrow \exists y(yR^+x \wedge \chi_2(y)))$, with χ_1, χ_2 quantifier-free.

Without major difficulties it is possible to extend our construction even to a more expressive logic, namely to the fragment of FO^2 with the transitive closure of relation R, with the only restriction that existential subformulas are of the form $\exists y(xR^+y \wedge \psi(x,y))$, $\exists y(yR^+x \wedge \psi(x,y))$ (or formulas obtained by switching the role of x and y). In other words, existential quantifiers are *guarded* by atomic predicates built from R^+.

An important open question arises:

▶ **Open Question 1.** Is the whole two-variable fragment of first-order logic, FO^2, decidable when extended by transitive closure of a fixed binary relation?

In a recent paper [17] it is shown that FO^2 is decidable with one transitive relation. We believe that combining the techniques from that paper with some ideas from our paper may lead to a positive answer to the given open question.

We also leave two open question regarding $\forall^2_{TC}[1]$:

▶ **Open Question 2.** What is the exact complexity of the satisfiability problem for $\forall^2_{TC}[1]$?

▶ **Open Question 3.** Is the finite satisfiability problem for $\forall^2_{TC}[1]$ decidable?

References

1 A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

2 W. Charatonik and P. Witkowski. On the complexity of the Bernays-Schönfinkel class with datalog. In *LPAR*, volume 6397 of *LNCS*, pages 187–201, 2010.

3 M. J. Fisher and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.

4 Martin Fürer. The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In *Logic and Machines*, volume 171 of *LNCS*, pages 312–319, 1983.

5 E. Grädel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic*, 3(1):53–69, 1997.

6 E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logic. In *STACS*, volume 1200 of *LNCS*, pages 249–260, 1997.

7 N. Immerman, A. Moshe Rabinovich, T. W. Reps, S. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, volume 3210 of *LNCS*, pages 160–174, 2004.

8 A.S. Kahr, E.F. Moore, and H. Wang. Entscheidungsproblem reduced to the ∀∃∀ case. 1962.

9 E. Kieroński. The two-variable guarded fragment with transitive guards is 2ExpTime-Hard. In *Proc. Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS*, volume 2620 of *LNCS*, pages 299–312, 2003.

10 E. Kieroński. Results on the guarded fragment with equivalence or transitive relations. In *CSL*, volume 3634 of *LNCS*, pages 309–324, 2005.

11 E. Kieroński, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. *Accepted for LICS*, 2012.

12 E. Kieroński and L. Tendera. On finite satisfiability of the guarded fragment with equivalence or transitive guards. In *LPAR*, volume 4790 of *LNAI*, pages 318–332, 2007.

13 H. R. Lewis. Complexity results for classes of quantificational formulas. *Journal of Computer and System Sciences*, 21:317–353, 1980.

14 J. Michaliszyn. Decidability of the guarded fragment with the transitive closure. In *ICALP (2)*, volume 5556 of *LNCS*, pages 261–272, 2009.

15 M. Mortimer. On languages with two variables. *Zeitschr. f. Logik und Grundlagen d. Math.*, 21:135–140, 1975.

16 W. Szwast and L. Tendera. On the decision problem for the guarded fragment with transitivity. In *LICS*, pages 147–156, 2001.

17 W. Szwast and L. Tendera. FO^2 with one transitive relation is decidable. *Unpublished*, 2012.

Connection Matrices and the Definability of Graph Parameters*

Tomer Kotek and Johann A. Makowsky

Faculty of Computer Science
Technion–Israel Institute of Technology
Haifa Israel
{tkotek,janos}@cs.technion.ac.il

Abstract

In this paper we extend the Finite Rank Theorem for connection matrices of graph parameters definable in Monadic Second Order Logic with modular counting CMSOL of B. Godlin, T. Kotek and J.A. Makowsky, [16, 30], and demonstrate its vast applicability in simplifying known and new non-definability results of graph properties and finding new non-definability results for graph parameters. We also prove a Feferman-Vaught Theorem for the logic CFOL, First Order Logic with the modular counting quantifiers.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Model theory, finite model theory, graph invariants

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.411

1 Introduction

Difficulties in proving non-definability

Proving that a graph property P is not definable in first order logic FOL can be a challenging task, especially on graphs with an additional linear order on the vertices. Proving that a graph property such as 3-colorability, which is definable in monadic second order logic MSOL, is not definable in fixed point logic on ordered graphs amounts to solving the famous $\mathbf{P} \neq \mathbf{NP}$ problem.

In the case of FOL and MSOL properties the basic tools for proving non-definability are the various *Ehrenfeucht-Fraïssé games* also called *pebble games*. However, proving the existence of winning strategies for these games can be exasperating. Two additional tools can be used to make the construction of such winning strategies easier and more transparent: the *composition of winning strategies* and the use of *locality properties* such as Hanf locality and Gaifman locality. These techniques are by now well understood, even if not always simple to apply, and are described in monographs such as [11, 24]. However these techniques are not easily applicable for stronger logics, such as CMSOL, monadic second order logic with modular counting. Furthermore, the pebble game method or the locality method may be difficult to use when dealing with ordered structures or when proving non-definability for the case where the definition may use an order relation on the universe in an *order-invariant* way.

* The first author was partially supported by the Fein Foundation and the Graduate School of the Technion–Israel Institute of Technology. The second author was partially supported by the Israel Science Foundation for the project “Model Theoretic Interpretations of Counting Functions” (2007-2011) and the Grant for Promotion of Research by the Technion–Israel Institute of Technology.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 411–425
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The notion of definability was extended in [2, 3] to integer valued graph parameters, and in [9, 27, 29, 22, 23] to real or complex valued graph parameters and graph polynomials. In [27] and [22] graph polynomials definable in MSOL respectively SOL were introduced. The techniques of pebble games and locality do not lend themselves easily, or are not useful at all, for proving non-definability in these cases.

Connection matrices

Connection matrices were introduced in [15, 26] by M. Freedman, L. Lovász and A. Schrijver where they were used to characterize graph homomorphism functions. Let f be a real or complex valued graph parameter. A k-connection matrix $M(\sqcup_k, f)$ is an infinite matrix, where the rows and columns are indexed by finite k-labeled graphs G_i and the entry $M(\sqcup_k, f)_{i,j}$ is given by the value of $f(G_i \sqcup_k G_j)$. Here $\sqcup_k$ denotes the *k-sum* operation on G_i and G_j, i.e. the operation of taking the disjoint union of G_i and G_j and identifying vertices with the corresponding k-many labels.

In [16] connection matrices were used to show that certain graph parameters and polynomials are not MSOL-definable. The main result of [16] is the *Finite Rank Theorem*, which states that the connection matrices of CMSOL-definable graph polynomials have finite rank. Connection matrices and the Finite Rank Theorem were generalized in [30] to matrices $M(\Box, f)$ where $\Box$ is a binary operation on labeled graphs subject to a smoothness condition depending on the logic one wants to deal with. However, very few applications of the Finite Rank Theorem were given.

Properties not definable in CFOL and CMSOL

The purpose of this paper lies in the demonstration that the Finite Rank Theorem is a *truly manageable tool* for proving non-definability which leaves no room for hand-waving arguments. To make our point we discuss graph properties (not)-definable in CFOL and CMSOL, i.e., First Order Logic, respectively Monadic Second Order Logic with modular counting quantifiers $\mathrm{D}_{m,i}x\phi(x)$ which say that the number of elements satisfying ϕ equals i modulo m. We also discuss the corresponding (non)-definability questions in CMSOL for graph parameters and graph polynomials. Although one can derive pebble games for these two logics, see e.g. [21, 32], using them to prove non-definability may be very awkward.

Instead we use a Feferman-Vaught-type Theorem for CFOL for disjoint unions and Cartesian products, Theorem 3.3, which seems to be new for the case of products. The corresponding theorem for disjoint unions, Theorem 3.2(i), for CMSOL was proven by B. Courcelle [7, 8, 27].

The proof of the Finite Rank Theorem for these logics follows from the Feferman-Vaught-type theorems. The details will be spelled out in Section 3.

With the help of the Finite Rank Theorem we give new and uniform proofs for the following:

(i) Using connection matrices for various generalizations of the Cartesian product $\times_\Phi$ we prove non-definability of the following properties in CFOL with the vocabulary of graphs $\langle V, E, < \rangle$ with linear order:

- Forests, bipartite graphs, chordal graphs, perfect graphs, interval graphs, block graphs (every biconnected component, i.e., every block, is a clique), parity graphs (any two induced paths joining the same pair of vertices have the same parity);
- Trees, connected graphs;

- Planar graphs, cactus graphs (graphs in which any two cycles have at most one vertex in common) and pseudo-forests (graphs in which every connected component has at most one cycle).

The case of connected graphs was also shown undefinable in CFOL by. J. Nurmonen in [32] using his version of the pebble games for CFOL.

(ii) Using connection matrices for various generalizations of the disjoint union $\sqcup_\Phi$ we prove non-definability of the following properties in CMSOL with the vocabulary of graphs $\langle V, E, < \rangle$ with linear order:

- Hamiltonicity (via cycles or paths), graphs having a perfect matching, cage graphs (regular graphs with as few vertices as possible for their girth), well-covered graphs (where every minimal vertex cover has the same size as any other minimal vertex cover). Here $\sqcup_\Phi$ is the join operation $\bowtie$.
- The class of graphs which have a spanning tree of degree at most 3. Here $\sqcup_\Phi$ is a modified join operation.

(iii) Using connection matrices for various generalizations of the disjoint union $\sqcup_\Phi$ we prove non-definability of the following properties in CMSOL with the vocabulary of hypergraphs $\langle V, E; R, < \rangle$ with linear order:

- Regular graphs and bi-degree graphs.
- Graphs with average degree at most $\frac{|V|}{2}$.
- Aperiodic digraphs (where the greatest common divisor of the lengths of all cycles in the graph is 1).
- Asymmetric (also called rigid) graphs (i.e. graphs which have no non-trivial automorphisms).

Graph parameters and graph polynomials not definable in CMSOL

A graph parameter is CMSOL-definable if it is the evaluation of a CMSOL-definable graph polynomial. The precise definition of definability of graph polynomials is given in Section 6. Most prominent graph polynomials turned out to be definable in CMSOL, sometimes using a linear order on the vertices in an order-invariant way, among them the various Tutte polynomials, interlace polynomials, matching polynomials, and many more, cf. [28]. This led the second author to express his belief in [28] that all "naturally occurring graph polynomials" are CMSOL-definable. However, in [16] it was shown, using connection matrices, that the graph polynomial counting harmonious colorings is not CMSOL-definable. A vertex coloring is harmonious if each pair of colors appears at most once at the two end points of an edge, cf. [12, 20]. That this is indeed a graph polynomial was shown in [23]. However, the main thrust of [16] consists in showing that certain graph parameters are not evaluations of the standard prominent graph polynomials.

In Section 7, we use connection matrices to show that many "naturally occurring graph polynomials" are not CMSOL-definable. All these examples count various colorings and are graph polynomials by [23]. The corresponding notion of coloring is studied extensively in the literature.

To illustrate this we show that the following graph polynomials are not CMSOL-definable:

- $\chi_{rainbow}(G, k)$ is the number of *path-rainbow connected k-colorings*, which are functions $c : E(G) \to [k]$ such that between any two vertices $u, v \in V(G)$ there exists a path where all the edges have different colors.
- For every fixed $t \in \mathbb{N}$, $\chi_{mcc(t)}(G, k)$ is the number of vertex k-colorings $f : V(G) \to [k]$ for which no color induces a subgraph with a connected component of size larger than t.

- $\chi_{convex}(G, k)$ is the number of *convex colorings*, which are vertex k-colorings $f : V(G) \to [k]$ such that every color induces a connected subgraph of G.

Path-rainbow connected colorings were introduced in [6] and their complexity was studied in [5]. $mcc(t)$-colorings were studied in [1], [25] and [14]. Note $\chi_{mcc(1)}(G, k)$ is the chromatic polynomial. Convex colorings were studied for their complexity e.g. in [31] and [17]. From [23] we get that $\chi_{rainbow}(G, k)$, $\chi_{mcc(t)}(G, k)$, and $\chi_{convex}(G, k)$ are graph polynomials with k as the variable.

In Section 7 more examples of graph polynomials and graph parameters not definable in CMSOL are given.

Outline of the paper

We assume the reader is familiar with the basics of finite model theory [11, 24] and graph theory [4, 10].

In Section 2 we illustrate the use of connection matrices in the case of regular languages. This serves as a "warm-up" exercise. In Section 3 we introduce the general framework for connection matrices of graph properties, i.e., boolean graph parameters, and of properties of general τ-structures. In Section 4 we spell out the advantages and limitations of the method of connection matrices in proving non-definability. In Section 5 we illustrate the use of connection matrices and the Finite Rank Theorem for proving non-definability of properties. In Section 6 we recall the framework of definable graph polynomials and τ-polynomials and the corresponding definable numeric parameters, and in Section 7 we show how to prove non-definability of these.

2 Connection Matrices for Regular Languages

Our first motivating examples deal with regular languages and the operation of concatenation $\circ$. By the well-known Büchi-Elgot-Trakhtenbrot Theorem, see [11, 24], a language $L \subseteq \Sigma^*$ is regular if and only if the class S_L of ordered structures representing the words of L is definable in MSOL (or equivalently in CMSOL or $\exists$MSOL, the existential fragment of MSOL). The connection matrix $M(\circ, L)$ with columns and rows indexed by all words of Σ^* is defined by $M(\circ, L)_{u,v} = 1$ iff $u \circ v \in L$.

The Myhill-Nerode Theorem and the Pumping Lemma for regular languages, see [19, 18], can be used to derive the following properties of $M(\circ, L)$:

▶ Proposition 2.1. Let $L \subseteq \Sigma^*$ be a regular language.

(i) There is a finite partition $\{U_1, \ldots, U_k\}$ of Σ^* such that the sub-matrices obtained from restricting $M(\circ, L)$ to $M(\circ, L)^{[U_i, U_j]}$ have constant entries.

(ii) In particular, the infinite matrix $M(\circ, L)$ has finite rank over any field $\mathcal{F}$.

(iii) $M(\circ, L)$ has an infinite sub-matrix of rank at most 1.

Now we can also look at counting functions and numeric parameters of words, such as the length $\ell(w)$ of a word w or the number of words $s_L(w)$ in a language L which are (connected) sub-words of a given word w. The corresponding connection matrices $M(\circ, \ell)$ and $M(\circ, s_L)$ defined by $M(\circ, \ell)_{u,v} = \ell(u \circ v)$ and $M(\circ, s_L)_{u,v} = s_L(u \circ v)$ respectively do not satisfy (i) and (iii) above, but still have finite rank. On the other hand the function $m_L(w)$ which gives the maximal size of a word in L which occurs as a connected sub-word in w gives rise to connection matrix $M(\bar{\circ}, s_L)$ of infinite rank. Here $u \bar{\circ} v = u \circ a \circ v$ where $a \notin \Sigma$ and therefore $m_L(u \bar{\circ} v) = \max\{m_L(u), m_L(v)\}$.

We can use these connection matrices to show that $L_1 = \{0^n \circ 1^n : n \in \mathbb{N}\}$ is not regular, by noting that the sub-matrix $M(\circ, L_1)$ with columns indexed by 0^n and rows indexed by 1^n has 0 everywhere but in the diagonal, hence has infinite rank, contradicting (ii) of Proposition 2.1.

The numeric parameters on words ℓ, s_L are MSOL-definable as follows: $\ell(w) = \sum_{u <_{in} w} 1$, where $u <_{in} w$ means that u is a proper possibly empty initial segment of w. Similarly, $s_L(w) = \sum_{u <_{sw} w} 1$, where $u <_{sw} w$ denotes the relation u is a connected sub-word of w. We shall give a general definition of MSOL-definable numeric parameter in Section 6. But we state here already

▶ Proposition 2.2. The connection matrices $M(\circ, f)$ and $M(\bar{\circ}, f)$ have finite rank, provided f is MSOL-definable.

▶ Corollary 2.3. The function $m_L(w)$ is not MSOL-definable.

3 Connection Matrices for Properties: The Framework

Let τ be a purely relational finite vocabulary which may include constant symbols and may include a distinguished binary relation symbol for a linear order. A *τ-property* is a class of finite τ-structures closed under τ-isomorphisms. If the context is clear we just speak of properties and isomorphisms. We denote by SOL(τ) the set of SOL formulas over τ. A sentence is a formula without free variables.

Let $\mathcal{L}$ be a subset of SOL. $\mathcal{L}$ is a *fragment of* SOL if the following conditions hold:

(i) For every finite relational vocabulary τ the set of $\mathcal{L}(\tau)$ formulas contains all the atomic τ-formulas and is closed under boolean operations and renaming of relation and constant symbols.

(ii) $\mathcal{L}$ is equipped with a notion of *quantifier rank* and we denote by $\mathcal{L}_q(\tau)$ the set of formulas of quantifier rank at most q. The quantifier rank is sub-additive under substitution of sub-formulas,

(iii) The set of formulas of $\mathcal{L}_q(\tau)$ with a fixed set of free variables is, up to logical equivalence, finite.

(iv) Furthermore, if $\phi(x)$ is a formula of $\mathcal{L}_q(\tau)$ with x a free variable of $\mathcal{L}$, then there is a formula ψ logically equivalent to $\exists x \phi(x)$ in $\mathcal{L}_{q'}(\tau)$ with $q' \geq q + 1$.

Typical fragments are FOL and MSOL. CMSOL and the fixed point logics IFPL and FPL and their corresponding finite variable subsets correspond to fragments of SOL if we replace the counting or fixed-point operators by their SOL-definitions.

For two τ-structures $\mathfrak{A}$ and $\mathfrak{B}$ we define the *equivalence relation of $\mathcal{L}_q(\tau)$- non-distinguishability*, and we write $\mathfrak{A} \equiv_q^{\mathcal{L}} \mathfrak{B}$, if they satisfy the same sentences from $\mathcal{L}_q(\tau)$.

Let $s : \mathbb{N} \to \mathbb{N}$ be a function. A binary operation $\Box$ between τ-structures is called *(s,$\mathcal{L}$)-smooth*, if for all $q \in \mathbb{N}$ whenever $\mathfrak{A}_1 \equiv_{q+s(q)}^{\mathcal{L}} \mathfrak{B}_1$ and $\mathfrak{A}_2 \equiv_{q+s(q)}^{\mathcal{L}} \mathfrak{B}_2$ then

$$\mathfrak{A}_1 \Box \mathfrak{A}_2 \equiv_q^{\mathcal{L}} \mathfrak{B}_1 \Box \mathfrak{B}_2.$$

If $s(q)$ is identically 0 we omit it.

For two τ-structures $\mathfrak{A}$ and $\mathfrak{B}$, we denote by $\mathfrak{A} \sqcup \mathfrak{B}$ the *disjoint union*, which is a τ-structure; $\mathfrak{A} \sqcup_{rich} \mathfrak{B}$ the *rich disjoint union* which is the disjoint union augmented with two unary predicates for the universes A and B respectively; $\mathfrak{A} \times \mathfrak{B}$ the *Cartesian product*, which is a τ-structure; and for graphs G, H by $G \bowtie H$ the *join* of two graphs, see [10].

A *$\mathcal{L}$-transduction* of τ-structures into σ-structures is given by defining a σ-structure inside a given τ-structure. The universe of the new structure may be a definable subset of an

m-fold Cartesian product of the old structure. If $m = 1$ we speak of *scalar* and otherwise of *vectorized* transductions. For every k-ary relation symbol $R \in \sigma$ we need a τ-formula in $k \cdot m$ free individual variables to define it. We denote by Φ a sequence of τ-formulas which defines a transduction. We denote by $\Phi^\star$ the map sending τ-structures into σ-structures induced by Φ. We denote by $\Phi^\sharp$ the map sending σ-formulas into τ-formulas induced by Φ. For a σ-formula $\Phi^\sharp(\theta)$ is the *backward translation* of θ into a τ-formula. Φ is *quantifier-free* if all its formulas are from $\mathrm{FOL}_0(\tau)$. We skip the details, and refer the reader to [24, 27].

A fragment $\mathcal{L}$ is *closed under scalar transductions*, if for Φ such that all the formulas of Φ are in $\mathcal{L}(\tau)$, Φ scalar, and $\theta \in \mathcal{L}(\sigma)$, the backward substitution $\Phi^\sharp(\theta)$ is also in $\mathcal{L}(\tau)$. A fragment of SOL is called *tame* if it is closed under scalar transductions. FOL, MSOL and CMSOL are all tame fragments. So are their finite variable versions.

FOL and SOL are also closed under vectorized transductions, but the monadic fragments MSOL and CMSOL are not.

We shall frequently use the following:

▶ Proposition 3.1. Let Φ define a $\mathcal{L}$-transduction from τ-structures to σ-structures where each formula is of quantifier rank at most q. Let θ be a $\mathcal{L}(\sigma)_r$-formula. Then

$$\Phi^\star(\mathfrak{A}) \models \theta \text{ iff } \mathfrak{A} \models \Phi^\sharp(\theta)$$

and $\Phi^\sharp(\theta)$ is in $\mathcal{L}(\tau)_{q+r}$.

▶ Proposition 3.2 (Smooth operations).

(i) The rich disjoint union $\sqcup_{rich}$ of τ-structures and therefore also the disjoint union are FOL-smooth, MSOL-smooth and CMSOL-smooth. They are not SOL-smooth.

(ii) The Cartesian product $\times$ of τ-structures is FOL-smooth, but not MSOL-smooth

(iii) Let Φ be a quantifier-free scalar transduction of τ-structures into τ-structures and let $\Box$ be an $\mathcal{L}$-smooth operation. Then the operation $\Box_\Phi(\mathfrak{A}, \mathfrak{B}) = \Phi^\star(\mathfrak{A} \Box \mathfrak{B})$ $\mathcal{L}$-smooth. If Φ has quantifier rank at most k, it is $(k, \mathcal{L})$-smooth.

Sketch of proof. (i) is shown for FOL and MSOL using the usual pebble games. For CMSOL one can use Courcelle's version of the Feferman-Vaught Theorem for CMSOL, cf. [7, 8, 27]. (ii) is again shown using the pebble game for FOL. (iii) follows from Proposition 3.1. The negative statements are well-known, but also follow from the developments in the sequel. ◀

▶ Theorem 3.3 (Feferman-Vaught Theorem for CFOL).

(i) The rich disjoint union $\sqcup_{rich}$ of τ-structures, and therefore the disjoint union, too, is CFOL-smooth.

(ii) The Cartesian product $\times$ of τ-structures is CFOL-smooth.

Sketch of proof. The proof does not use pebble games, but Feferman-Vaught-type reduction sequences. (i) can be proven using the same reduction sequences which are used in [7, 8]. (ii) is proven using modifications of the reduction sequences as given in detail in [27, Theorem 1.6]. ◀

To the best of our knowledge, (ii) of Theorem 3.3 has not been stated in the literature before.

▶ Remark. We call this a Feferman-Vaught Theorem, because our proof actually computes the reduction sequences explicitly. However, this is not needed here, so we refer the reader to [27] for the definition of reduction sequences. One might also try to prove the theorem using the pebble games defined in [32], but at least for the case of the Cartesian product, the proof would be rather complicated and less transparent.

▶ Theorem 3.4 (Finite Rank Theorem for tame $\mathcal{L}$, [16, 30]).
Let $\mathcal{L}$ be a tame fragment of SOL. Let $\Box$ be a binary operation between τ-structures which is $\mathcal{L}$-smooth. Let $\mathcal{P}$ be a τ-property which is definable by a $\mathcal{L}$-formula ψ and $M(\Box, \psi)$ be the connection matrix defined by

$$M(\Box, \psi)_{\mathfrak{A},\mathfrak{B}} = 1 \text{ iff } \mathfrak{A}\Box\mathfrak{B} \models \psi \text{ and } 0 \text{ otherwise} .$$

Then

(i) There is a finite partition $\{U_1, \ldots, U_k\}$ of the (finite) τ-structures such that the sub-matrices obtained from restricting $M(\Box, \psi)$ to $M(\Box, \psi)^{[U_i, U_j]}$ have constant entries.

(ii) In particular, the infinite matrix $M(\Box, \psi)$ has finite rank over any field $\mathcal{F}$.

(iii) $M(\Box, \psi)$ has an infinite sub-matrix of rank at most 1.

Sketch of proof. (i) follows from the definition of a tame fragment and of smoothness and the fact that there are only finitely many formulas (up to logical equivalence) in $\mathcal{L}(\tau)_q$. (ii) and (iii) follow from (i). ◀

4 Merits and Limitations of Connection Matrices

Merits

The advantages of the Finite Rank Theorem for tame $\mathcal{L}$ in proving that a property is not definable in $\mathcal{L}$ are the following:

(i) It suffices to prove that certain binary operations on graphs (τ-structures) are $\mathcal{L}$-smooth operations.

(ii) Once the $\mathcal{L}$-smoothness of a binary operation has been established, proofs of non-definability become surprisingly simple and transparent. One of the most striking examples is the fact that asymmetric (rigid) graphs are not definable in CMSOL, cf. Corollary 5.7.

(iii) Many properties can be proven to be non-definable using the same or similar sub-matrices, i.e., matrices with the same row and column indices. This is well illustrated in the examples of Section 5.

Limitations

The classical method of proving non-definability in FOL using pebble games is complete in the sense that a property is FOL$(\tau)_q$-definable iff the class of its models is closed under game equivalence of length q. Using pebble games one proves easily that the class of structures without any relations of even cardinality, EVEN, is not FOL-definable. This cannot be proven using connection matrices in the following sense:

▶ Proposition 4.1. Let Φ be a quantifier-free transduction between τ-structures and let $\Box_\Phi$ be the binary operation on τ-structures:

$$\Box_\Phi(\mathfrak{A}, \mathfrak{B}) = \Phi^\star(\mathfrak{A} \sqcup_{rich} \mathfrak{B})$$

Then the connection matrix $M(\Box_\Phi, \text{EVEN})$ satisfies the properties (i)-(iii) of Theorem 3.4.

5 Proving Non-definability of Properties

Non-definability on CFOL

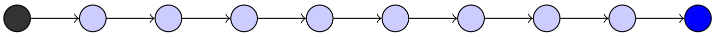
We will prove non-definability in CFOL using Theorem 3.3 for Cartesian products combined with FOL transductions. It is useful to consider a slight generalization of the Cartesian product as follows. We add two constant symbols *start* and *end* to our graphs. In $G^1 \times G^2$ the symbol *start* is interpreted as the pair of vertices $(v^1_{start}, v^2_{start})$ from G^1 and G^2 respectively such that v^i_{start} is the interpretation of $start_i$ (i.e. *start* in G^i) for $i = 1, 2$.

The transduction $\Phi_{sym}(x, y) = E_D(x, y) \vee E_D(y, x)$ transforms a digraph $D = (V_D, E_D)$ into an undirected graph whose edge relation is the symmetric closure of the edge relation of the digraph.

The following transduction Φ_F transforms the Cartesian product of two directed graphs $G^i = (V_1, E_1, v^i_{start}, v^i_{end})$ with the two constants $start_i$ and end_i, $i = 1, 2$ into a certain digraph. It is convenient to describe Φ_F as a tranduction of the two input graphs G^1 and G^2:

$$\begin{aligned}\Phi_F\left((v_1, v_2), (u_1, u_2)\right) \quad &= \quad (E_1(v_1, u_1) \wedge E_2(v_2, u_2)) \vee \\ &\quad\quad ((v_1, v_2), (u_1, u_2)) = ((start_1, start_2), (end_1, end_2))\end{aligned}$$

Consider the transduction obtained from Φ_F by applying Φ_{sym} when the input graphs are directed paths $P^i_{n_i}$ of length n_i. The input graphs look like this:



The result of the application of the transduction is given in Figure 1. The result of the transduction has a cycle iff $n_1 = n_2$. The length of this cycle is n_1. Hence, the connection sub-matrix with rows and columns labeled by directed paths of odd (even) length has ones on the main diagonal and zeros everywhere else, so it has infinite rank. Thus we have shown:

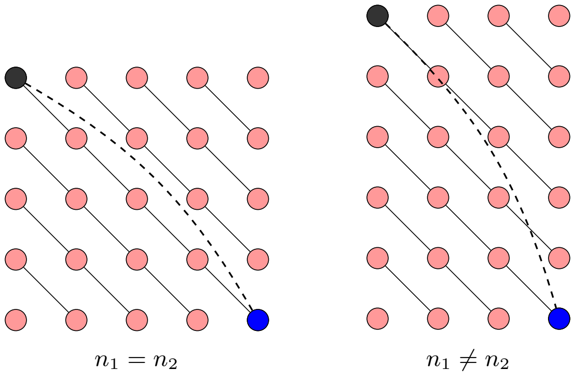


Figure 1 The result of applying Φ_F and then Φ_{sym} on the two directed paths. There is a cycle iff the two directed paths are of the same length.

▶ Theorem 5.1. The graphs without cycles of odd (even) length are not CFOL-definable even in the presence of a linear order.

▶ Corollary 5.2. Not definable in CFOL with order are:

(i) Forests, bipartite graphs, chordal graphs, perfect graphs
(ii) interval graphs (cycles are not interval graphs)
(iii) Block graphs (every biconnected component is a clique)

(iv) Parity graphs (any two induced paths joining the same pair of vertices have the same parity)

The transduction

$$\begin{aligned}\Phi_T\left((v_1,v_2),(u_1,u_2)\right) &= (E_1(v_1,u_1)\wedge E_2(v_2,u_2))\vee\\ &\quad (v_1=u_1=start_1\wedge E(v_2,u_2))\vee\\ &\quad (v_1=u_1=end_1\wedge E(v_2,u_2)),\end{aligned}$$

combined with Φ_{sym} transforms the two directed paths into the structures in Figure 2.

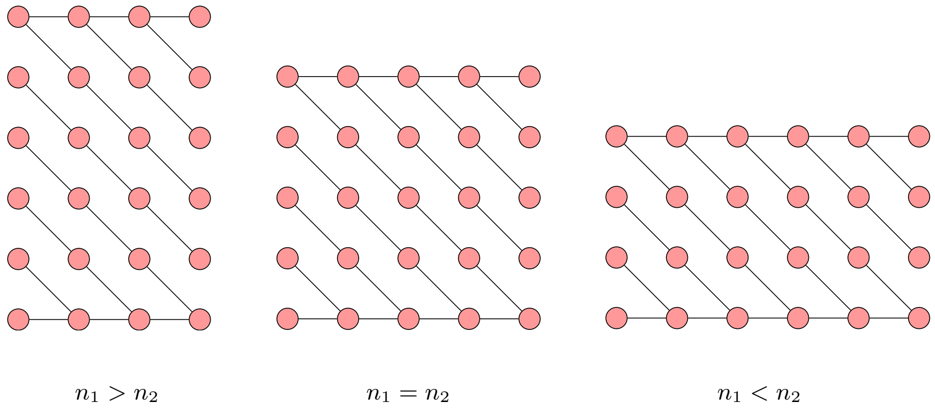


Figure 2 The result of applying Φ_T and then Φ_{sym} on two directed paths. We get a tree iff the two directed paths are of equal length.

So, the result of the transduction is a tree iff $n_1 = n_2$. It is connected iff $n_1 \leq n_2$. Hence, both the connection matrices with directed paths as row and column labels of the property of being a tree and of connectivity have infinite rank.

▶ Theorem 5.3. The properties of being a tree or a connected graph are not CFOL-definable even in the presence of linear order.

For our next connection matrix we use the 2-sum of the following two 2-graphs:

(i) the 2-graph (G, a, b) obtained from K_5 by choosing two vertices a and b and removing the edge between them

(ii) the symmetric closure of the Cartesian product of the two digraphs $P^1_{n_1}$ and $P^2_{n_2}$:

We denote this transduction by Φ_P, see Figure 3.

So, the result of this construction has a clique of size 5 as a minor iff $n_1 = n_2$. It can never have a $K_{3,3}$ as a minor.

▶ Theorem 5.4. The class of planar graphs is not CFOL-definable on ordered graphs.

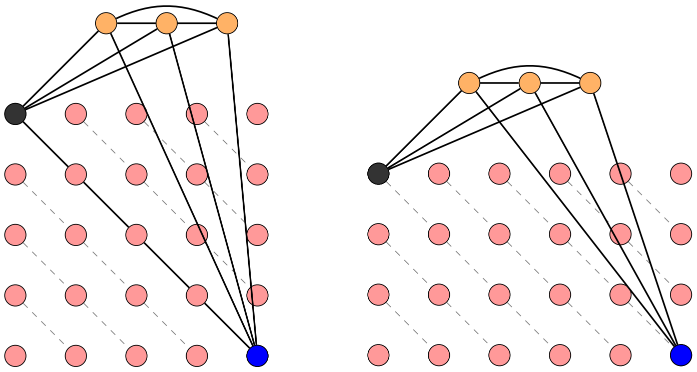
If we modify the above construction by taking K_3 instead of K_5 and making $(start_1, start_2)$ and (end_1, end_2) adjacent, we get

▶ Corollary 5.5. The following classes of graphs are not CFOL-definable on ordered graphs.

(i) Cactus graphs, i.e. graphs in which any two cycles have at most one vertex in common.

(ii) Pseudo-forests, i.e. graphs in which every connected component has at most one cycle.

The case of connected graphs was also shown non-definable in CFOL by. J. Nurmonen in [32] using his version of the pebble games for CFOL.



$n_1 = n_2$ $n_1 \neq n_2$

Figure 3 The result of Φ_P on two directed paths. The graph obtained here is planar iff the two directed paths are of equal length.

Non-definability in CMSOL

Considering the connection matrix where the rows and columns are labeled by the graphs on n vertices but without edges E_n, the graph $E_i \bowtie E_j = K_{i,j}$ is

(i) Hamiltonian iff $i = j$;
(ii) has a perfect matching iff $i = j$;
(iii) is a cage graph (a regular graph with as few vertices as possible for its girth) iff $i = j$;
(iv) is a well-covered graph (every minimal vertex cover has the same size as any other minimal vertex cover) iff $i = j$.

All of these connection matrices have infinite rank, so we get

▶ Corollary 5.6. None of the properties above are CMSOL-definable as graphs even in the presence of an order.

Using a modification $\overline{\bowtie}$ of the join operation used in [8, Remark 5.21] one can show the same for the class of graphs which have a spanning tree of degree at most 3. For any fixed natural number $d > 3$, by performing a transduction on $G \overline{\bowtie} H$ which attaches $d - 3$ new vertices as pendants to each vertex of $G \overline{\bowtie} H$, one can extend the non-definability result to the class of graphs which have a spanning tree of degree at most d.

For the language of hypergraphs we cannot use the join operation, since it is not smooth. Note also that Hamiltonian and having a perfect matching are both definable in CMSOL in the language of hypergraphs. But using the connection sub-matrices of the disjoint union we still get:

(i) Regular: $K_i \sqcup K_j$ is regular iff $i = j$;
(ii) A generalization of regular graphs are *bi-degree* graphs, i.e., graphs where every vertex has one of two possible degrees. $K_i \sqcup (K_j \sqcup K_1)$ is a bi-degree graph iff $i = j$.
(iii) The average degree of $K_i \sqcup K_j$ is at most $\frac{|V|}{2}$ iff $i = j$;
(iv) A digraph is *aperiodic* if the common denominator of the lengths of all cycles in the graph is 1. We denote by C_i^d the directed cycle with i vertices. For prime numbers p, q the digraphs $C_p \sqcup C_q$ is aperiodic iff $p \neq q$.
(v) A graph is *asymmetric (or rigid)* if it has no non-trivial automorphisms. It was shown by P. Erdös and A. Rényi [13] that almost all finite graphs are asymmetric. So there is

an infinite set $I \subseteq \mathbb{N}$ such that for $i \in I$ there is an asymmetric graph R_i of cardinality i. $R_i \sqcup R_j$ is asymmetric iff $i \neq j$.

▶ Corollary 5.7. None of the properties above are CMSOL-definable as hypergraphs even in the presence of an order.

▶ Remark. The case of asymmetric graphs illustrates that it is not always necessary to find explicit infinite families of graphs whose connection matrices are of infinite rank in order to show that such a family exists.

6 $\mathcal{L}$-Definable Graph Polynomials and Graph Parameters

$\mathcal{L}(\tau)$-polynomials

Here we follow closely the exposition from [23]. Let $\mathcal{L}$ be a tame fragment of SOL. We are now ready to introduce the $\mathcal{L}$-definable polynomials. They are defined for τ-structures and generally are called $\mathcal{L}(\tau)$ *invariants* as they map τ-structures into some commutative semi-ring $\mathcal{R}$, which contains the semi-ring of the integers $\mathbb{N}$, and are invariant under τ-isomorphisms. If τ is the vocabulary of graphs or hypergraphs, we speak of graph invariants and graph polynomials.

For our discussion $\mathcal{R} = \mathbb{N}$ or $\mathcal{R} = \mathbb{Z}$ suffices, but the definitions generalize. Our polynomials have a fixed finite set of variables (indeterminates, if we distinguish them from the variables of $\mathcal{L}$), $\mathbf{X}$.

▶ **Definition 1** ($\mathcal{L}$-monomials). Let $\mathcal{M}$ be a τ-structure. We first define the $\mathcal{L}$-definable *$\mathcal{M}$-monomials.* inductively.

(i) Elements of $\mathbb{N}$ are $\mathcal{L}$-definable $\mathcal{M}$-monomials.

(ii) Elements of $\mathbf{X}$ are $\mathcal{L}$-definable $\mathcal{M}$-monomials.

(iii) Finite products of monomials are $\mathcal{L}$-definable $\mathcal{M}$-monomials.

(iv) Let $\phi(a)$ be a $\tau \cup \{a\}$-formula in $\mathcal{L}$, where a is a constant symbol not in τ. Let t be a $\mathcal{M}$-monomial. Then $\prod_{a:\langle \mathcal{M},a\rangle \models \phi(a)} t$ is a $\mathcal{L}$-definable $\mathcal{M}$-monomial.

The monomial t may depend on relation or function symbols occurring in ϕ.

Note the degree of a monomial is polynomially bounded by the cardinality of $\mathcal{M}$.

▶ **Definition 2** ($\mathcal{L}$-polynomials). The $\mathcal{M}$-polynomials definable in $\mathcal{L}$ are defined inductively:

(i) $\mathcal{M}$-monomials are $\mathcal{L}$-definable $\mathcal{M}$-polynomials.

(ii) Let $\phi(\overline{a})$ be a $\tau \cup \{\overline{a}\}$-formula in $\mathcal{L}$ where $\overline{a} = (a_1, \ldots, a_m)$ is a finite sequence of constant symbols not in τ. Let t be a $\mathcal{M}$-polynomial. Then $\sum_{\overline{a}:\langle \mathcal{M},\overline{a}\rangle \models \phi(\overline{a})} t$ is a $\mathcal{L}$-definable $\mathcal{M}$-polynomial.

(iii) Let $\phi(\overline{R})$ be a $\tau \cup \{\overline{R}\}$-formula in $\mathcal{L}$ where $\overline{R} = (R_1, \ldots, R_m)$ is a finite sequence of relation symbols not in τ. Let t be a $\mathcal{M}$-polynomial definable in $\mathcal{L}$. Then $\sum_{\overline{R}:\langle \mathcal{M},\overline{R}\rangle \models \phi(\overline{R})} t$ is a $\mathcal{L}$-definable $\mathcal{M}$-polynomial.

The polynomial t may depend on relation or function symbols occurring in ϕ.

An $\mathcal{M}$-polynomial $p_{\mathcal{M}}(\mathbf{X})$ is an expression with parameter $\mathcal{M}$. The family of polynomials, which we obtain from this expression by letting $\mathcal{M}$ vary over all τ-structures, is called, by abuse of terminology, a $\mathcal{L}(\tau)$-polynomial.

Among the $\mathcal{L}$-definable polynomials we find most of the known graph polynomials from the literature, cf. [28, 23]. *$\mathcal{L}$-definable numeric graph parameters* are evaluations of $\mathcal{L}$-definable polynomials and take values in $\mathcal{R}$. *$\mathcal{L}$-definable properties* are special cases of numeric parameters which have boolean values.

Some simple graph parameters are even FOL-definable, e.g. $|V|$, the number of vertices and $|E|$, the number of edges. However, we leave the discussion of FOL-definable parameters for the journal version of this paper, and concentrate on tame fragments of SOL which do have second order variables.

Sum-like operations

For the proof of the The Finite Rank Theorem for $\mathcal{L}$-polynomials which involve second order variables it is not enough that the binary operation $\Box$ on τ-structures be $\mathcal{L}$-smooth. We need a way to uniquely decompose the relation over which we perform summation in $\mathfrak{A}\Box\mathfrak{B}$ into relations in $\mathfrak{A}$ and $\mathfrak{B}$ respectively, from which we can reconstruct the relation in $\mathfrak{A}\Box\mathfrak{B}$. For our discussion here it suffices to restrict $\Box$ to *$\mathcal{L}$-sum-like operations.* $\mathfrak{A}\Box\mathfrak{B}$ is *$\mathcal{L}$-sum-like* if there is a *scalar* $\mathcal{L}$-transduction Φ such that

$$\mathfrak{A}\Box\mathfrak{B} = \Phi^{\star}(\mathfrak{A} \sqcup_{rich} \mathfrak{B}).$$

An operation is *$\mathcal{L}$-product-like* if instead of scalar transductions we also allow *vectorized* transductions. Typically, the Cartesian product is FOL-product-like, but not sum-like. The k-sum and the join operation on graphs are FOL-sum-like (but, in the case of join, not on hypergraphs).

The Finite Rank Theorem for $\mathcal{L}$-polynomials

Now we can state the Finite Rank Theorem for $\mathcal{L}$-polynomials. The proof uses the same techniques as in [9, 27].

▶ Theorem 6.1 (The Finite Rank Theorem for $\mathcal{L}$-polynomials).
Let $\mathcal{L}$ be a tame fragment of SOL and $\Box$ be an $\mathcal{L}$-sum-like operation between τ-structures which is $(s, \mathcal{L})$-smooth. Let P be an $\mathcal{L}(\tau)$-polynomial. Then the connection matrix $M(\Box, P)$ has finite rank.

▶ Remark. In [16] the theorem was only formulated for k-sums, and the join operation and for the logic CMSOL.

7 Non-definability of $\mathcal{L}(\tau)$-invariants

7.1 Numeric $\mathcal{L}(\tau)$-parameters

Theorem 6.1 can be used to show that many τ-parameters are not $\mathcal{L}$-definable.

$\Box$-maximizing and $\Box$-minimizing parameters

We say a τ-parameter f is *$\Box$-maximizing* (*$\Box$-minimizing*) if there exist an infinite sequence of non-isomorphic τ-structures $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots, \mathcal{A}_i, \ldots$ such that for any $i \neq j$,

$$f(\mathcal{A}_i \Box \mathcal{A}_j) = \max\{f(\mathcal{A}_i), f(\mathcal{A}_j)\}\,.$$

Furthermore, if f is unbounded on $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \ldots$ then f is *unboundedly $\Box$-maximizing.* Analogously we define *(unboundedly) $\Box$-minimizing.*

▶ Proposition 7.1. If f is a unboundedly $\Box$-maximizing ($\Box$-minimizing) τ-parameter, then $M(f, \Box)$ has infinite rank.

Using Proposition 7.1 we show that many τ-parameters are not CMSOL-definable:

▶ Proposition 7.2. Let $\mathcal{L}$ be tame and $\sqcup$-smooth. The following graph parameters are not $\mathcal{L}$-definable in the language of hypergraphs. In particular they are not CMSOL-definable. *Spectral radius, chromatic number, acyclic chromatic number, arboricity, star chromatic number, clique number, Hadwiger number, Hajós number, tree-width, path-width, clique-width, edge chromatic number, Thue number, maximum valency, circumference, longest path, maximal connected planar (bipartite) induced subgraph, boxicity, minimal eigenvalue, spectral gap, girth, degeneracy,* and *minimum valency.*

Proof. All these graph parameters g are unboundedly $\sqcup$-maximizing or $\sqcup$-minimizing. ◀

Variations of the notions of $\Box$-maximizing or $\Box$-minimizing τ-parameters can also lead to non-definability results, e.g.:

▶ Proposition 7.3. Under the same assumption on $\mathcal{L}$ as before, the number of connected components (blocks, simple cycles, induced paths) of maximum (minimum) size is not $\mathcal{L}$-definable in the language of hypergraphs.

Proof. Consider the connection matrix of graphs $i \cdot K_i$ which consist of the disjoint union of i cliques of size i with the operation of disjoint union. We denote the number of connected components of maximum size in a graph G by $\#_{\mathrm{max-cc}}(G)$. Then

$$\#_{\mathrm{max-cc}}(nK_n \sqcup mK_m) = \begin{cases} \max\{n, m\} & n \neq m \\ n + m & n = m \end{cases}$$

So $M(\#_{\mathrm{max-cc}}, \sqcup)$ is of infinite rank. The other cases are proved similarly. ◀

7.2 τ-polynomials

Here we use the method of connection matrices for showing that (hyper)graph polynomials are not MSOL-definable. Some of the material here is taken from the first author's thesis [33]. As examples we consider the polynomials $\chi_{rainbow}(G,k)$, $\chi_{mcc(t)}(G,k)$, and $\chi_{convex}(G,k)$, which were defined in the introduction.

To show that none of $\chi_{rainbow}(G,k)$, $\chi_{mcc(t)}(G,k)$, or $\chi_{convex}(G,k)$ are CMSOL-polynomials in the language of graphs, and that neither $\chi_{rainbow}(G,k)$ nor $\chi_{convex}(G,k)$ are CMSOL-polynomials in the language of hypergraphs, we prove the following general proposition:

▶ Lemma 7.4. Given a τ-parameter p, a binary operation $\Box$ on τ-structures and an infinite sequence of non-isomorphic τ-structures $\mathcal{A}_i, i \in \mathbb{N}$, let $f : \mathbb{N} \to \mathbb{N}$ be an unbounded function such that for every $\lambda \in \mathbb{N}$, $p(\mathcal{A}_i \Box \mathcal{A}_j, \lambda) = 0$ iff $i + j > f(\lambda)$. Then the connection matrix $M(\Box, p)$ has infinite rank.

Proof. Let $\lambda \in \mathbb{N}$ and let p_λ be the graph parameter given by $p_\lambda(G) = p(G, \lambda)$. The restriction of the connection matrix $M(p_\lambda, \Box)$ to the rows and columns corresponding to $\mathcal{A}_i$, $0 \leq i \leq f(\lambda) - 1$, yields a finite triangular matrix with non-zero diagonal. Hence the rank of $M(p_\lambda, \Box)$ is at least $f(\lambda) - 1$.

Using that f is unbounded, we get that $M(p, \Box)$ contains infinitely many finite submatrices with ranks which tend to infinity. Hence, the rank of $M(p, \Box)$ is infinite, ◀

We now use Lemma 7.4 to compute connection matrices where $\Box$ is the disjoint union $\sqcup$, the 1-sum $\sqcup_1$ or the join $\bowtie$.

▶ Proposition 7.5. The following connection matrices have infinite rank:

(i) $M(\sqcup_1, \chi_{rainbow}(G,k))$;
(ii) $M(\sqcup_1, \chi_{convex}(G,k))$;
(iii) For every $t > 0$ the matrix $M(\bowtie, \chi_{mcc(t)}(G,k))$;

Proof.
(i) For $\chi_{rainbow}(G,k)$, we use that the 1-sum of paths with one end labeled is again a path with $P_i \sqcup_1 P_j = P_{i+j-1}$ and that $\chi_{rainbow}(P_r,k) = 0$ iff $r > k+3$.
(ii) For $\chi_{convex}(G,k)$, we use edgeless graphs and disjoint union $E_i \sqcup E_j = E_{i+j}$ and that $\chi_{convex}(E_r,k) = 0$ iff $r > k$.
(iii) For $\chi_{mcc(t)}(G,k)$ we use the join and cliques, $K_i \bowtie K_j = K_{i+j}$ and that $\chi_{mcc(t)}(K_r,k) = 0$ iff $r > kt$.

◀

▶ **Corollary 3.**
(i) $\chi_{rainbow}(G,k)$ *and* $\chi_{convex}(G,k)$ *are not* CMSOL-*definable in the language of graphs and hypergraphs.*
(ii) $\chi_{mcc(t)}(G,k)$ *(for any fixed* $t > 0$*) is not* CMSOL-*definable in the language of graphs.*

Proof. (i) The 1-sum and the disjoint union are CMSOL-sum-like and CMSOL-smooth for hypergraphs. (ii) The join is only CMSOL-sum-like and CMSOL-smooth for graphs. ◀

The same method yields non-definability results for other graph polynomials which arise by counting other graph colorings from the literature, such as acyclic colorings, non-repetitive colorings, t-improper colorings, co-colorings, sub-colorings and G-free colorings.

References

1 N. Alon, G. Ding, B. Oporowski, and D. Vertigan. Partitioning into graphs with only small components. *Journal of Combinatorial Theory, Series B*, 87(2):231–243, 2003.

2 S. Arnborg, J. Lagergren, and D. Seese. Problems easy for tree-decomposable graphs (extended abstract). In *ICALP*, pages 38–51, 1988.

3 S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree decomposable graphs. *Journal of Algorithms*, 12:308–340, 1991.

4 B. Bollobás. *Modern Graph Theory.* Springer, 1999.

5 S. Chakraborty, E. Fischer, A. Matsliah, and R. Yuster. Hardness and algorithms for rainbow connection. *Journal of Combinatorial Optimization*, 21(3):330–347, 2011.

6 G. Chartrand, G.L. Johns, K.A. McKeon, and P. Zhang. Rainbow connection in graphs. *Mathematica Bohemica*, 133(1):85–98, 2008.

7 B. Courcelle. Graph rewriting: An algebraic and logic approach. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 2, chapter 5, pages 193–242. Elsevier Science Publishers, 1990.

8 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-order Logic, a Language Theoretic Approach.* Cambridge University Press, 2012.

9 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.

10 R. Diestel. *Graph Theory.* Graduate Texts in Mathematics. Springer, 3 edition, 2005.

11 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer Verlag, 1995.

12 K. Edwards. The harmonious chromatic number and the achromatic number. In R. A. Bailey, editor, *Survey in Combinatorics*, volume 241 of *London Math. Soc. Lecture Note Ser.*, pages 13–47. Cambridge Univ. Press, 1997.

13 P. Erdös and A. Rényi. Asymmetric graphs. *Acta Mathematica Hungaric*, 14(3):295–315, 1963.

14 A. Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *Electric Journal of Combinatorics*, 11(1), 2004.

15 M. Freedman, László Lovász, and A. Schrijver. Reflection positivity, rank connectivity, and homomorphisms of graphs. *Journal of AMS*, 20:37–51, 2007.

16 B. Godlin, T. Kotek, and J.A. Makowsky. Evaluation of graph polynomials. In *34th International Workshop on Graph-Theoretic Concepts in Computer Science, WG08*, volume 5344 of *Lecture Notes in Computer Science*, pages 183–194, 2008.

17 A.J. Goodall and S.D. Noble. Counting cocircuits and convex two-colourings is #P-complete, 2008. http://arxiv.org/abs/0810.2042.

18 M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

19 J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, 1980.

20 J.E. Hopcroft and M.S. Krishnamoorthy. On the harmonious coloring of graphs. *SIAM J. Algebraic Discrete Methods*, 4:306–311, 1983.

21 Phokion G. Kolaitis and Jouko A. Väänänen. Generalized quantifiers and pebble games on finite structures. *Ann. Pure Appl. Logic*, 74(1):23–75, 1995.

22 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.

23 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 207–242. American Mathematical Society, 2011.

24 L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

25 N. Linial, J. Matousek, O. Sheffet, and G. Tardos. Graph coloring with no large monochromatic components. *Combinatorics, Probability, and Computing*, 17.4:577–589, 2008.

26 L. Lovász. Connection matrics. In G. Grimmet and C. McDiarmid, editors, *Combinatorics, Complexity and Chance, A Tribute to Dominic Welsh*, pages 179–190. Oxford University Press, 2007.

27 J.A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126.1-3:159–213, 2004.

28 J.A. Makowsky. From a zoo to a zoology: Towards a general theory of graph polynomials. *Theory of Computing Systems*, 43:542–562, 2008.

29 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.

30 Johann A. Makowsky. Connection matrices for msol-definable structural invariants. In *ICLA*, pages 51–64, 2009.

31 S. Moran and S. Snir. Efficient approximation of convex recolorings. *Journal of Computer and System Sciences*, 73(7):1078–1089, 2007.

32 Juha Nurmonen. Counting modulo quantifiers on finite structures. *Inf. Comput.*, 160(1-2):62–87, 2000.

33 T. Kotek. *Definability of Combinatorial Functions.* PhD thesis, Technion—Israel Institute of Technology, 2012.

The FO^2 alternation hierarchy is decidable

Manfred Kufleitner[1] **and Pascal Weil**[2]

1 **University of Stuttgart, Germany** *
kufleitner@fmi.uni-stuttgart.de

2 **CNRS, LaBRI, UMR5800, F-33400 Talence, France** †
Univ. Bordeaux, LaBRI, UMR5800, F-33400 Talence, France
pascal.weil@labri.fr

Abstract

We consider the two-variable fragment $\mathrm{FO}^2[<]$ of first-order logic over finite words. Numerous characterizations of this class are known. Thérien and Wilke have shown that it is decidable whether a given regular language is definable in $\mathrm{FO}^2[<]$. From a practical point of view, as shown by Weis, $\mathrm{FO}^2[<]$ is interesting since its satisfiability problem is in NP. Restricting the number of quantifier alternations yields an infinite hierarchy inside the class of $\mathrm{FO}^2[<]$-definable languages. We show that each level of this hierarchy is decidable. For this purpose, we relate each level of the hierarchy with a decidable variety of finite monoids.

Our result implies that there are many different ways of climbing up the $\mathrm{FO}^2[<]$-quantifier alternation hierarchy: deterministic and co-deterministic products, Mal'cev products with definite and reverse definite semigroups, iterated block products with $\mathcal{J}$-trivial monoids, and some inductively defined omega-term identities. A combinatorial tool in the process of ascension is that of condensed rankers, a refinement of the rankers of Weis and Immerman and the turtle programs of Schwentick, Thérien, and Vollmer.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages.

Keywords and phrases first-order logic, regular language, automata theory, semigroup, ranker

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.426

1 Introduction

The investigation of logical fragments has a long history. McNaughton and Papert [17] showed that a language over finite words is definable in first-order logic $\mathrm{FO}[<]$ if and only if it is star-free. Combined with Schützenberger's characterization of star-free languages in terms of finite aperiodic monoids [23], this leads to an algorithm to decide whether a given regular language is first-order definable. Many other characterizations of this class have been given over the past 50 years, see [3] for an overview. Moreover, mainly due to its relation to linear temporal logic [7], it became relevant to a large number of application fields, such as verification.

Very often one is interested in fragments of first-order logic. From a practical point of view, the reason is that smaller fragments often yield more efficient algorithms for computational problems such as satisfiability. For example, satisfiability for $\mathrm{FO}[<]$ is non-elementary [26], whereas the satisfiability problem for first-order logic with only two variables is in NP, *cf.* [39]. And on the theoretical side, fragments form the basis of a descriptive complexity theory

* The first author was supported by the German Research Foundation (DFG) under grant DI 435/5-1.
† The second author was supported by the grant ANR 2010 BLAN 0202 01 FREC.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 426–439
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

inside the regular languages: the simpler a logical formula defining a language, the easier the language. Moreover, in contrast to classical complexity theory, in some cases one can actually decide whether a given language has a particular property. From both the practical and the theoretical point of view, several natural hierarchies have been considered in the literature: the quantifier alternation hierarchy inside FO[$<$] which coincides with the Straubing-Thérien hierarchy [28, 32], the quantifier alternation hierarchy inside FO[$<, +1$] with a successor predicate $+1$ which coincides with the dot-depth hierarchy [2, 36], the until hierarchy of temporal logic [34], and the until-since hierarchy [35]. Decidability is known for the levels of the until and the the until-since hierarchies, and only for the very first levels of the alternation hierarchies, see *e.g.* [4, 21].

Fragments are usually defined by restricting resources in a formula. Such resources can be the predicates which are allowed, the quantifier depth, the number of quantifier alternations, or the number of variables. When the quantifier depth is restricted, only finitely many languages are definable over a fixed alphabet: decidability of the membership problem is not an issue in this case. When restricting the number of variables which can be used (and reused), then first-order logic $\mathrm{FO}^3[<]$ with three variables already has the full expressive power of FO[$<$], see [6, 7]. On the other hand, first-order logic $\mathrm{FO}^2[<]$ with only two variables defines a proper subclass. The languages definable in $\mathrm{FO}^2[<]$ have a huge number of different characterizations, see *e.g.* [4, 30, 31]. For example, $\mathrm{FO}^2[<]$ has the same expressive power as $\Delta_2[<]$; the latter is a fragment of FO[$<$] with two blocks of quantifiers [33].

Turtle programs are one of these numerous descriptions of $\mathrm{FO}^2[<]$-definable languages [24]. They are sequences of instructions of the form "go to the next a-position" and "go to the previous a-position". Using the term *ranker* for this concept and having a stronger focus on the order of positions defined by such sequences, Weis and Immerman [40] were able to give a combinatorial characterization of the alternation hierarchy $\mathrm{FO}^2_m[<]$ inside $\mathrm{FO}^2[<]$. Straubing [29] gave an algebraic characterization of $\mathrm{FO}^2_m[<]$. But neither result yields the decidability of $\mathrm{FO}^2_m[<]$-definability for $m > 2$. In some sense, this is the opposite of a previous result of the authors [15, Thm. 6.1], who give necessary and sufficient conditions which helped to decide the $\mathrm{FO}^2_m[<]$-hierarchy with an error of at most one. In this paper we give a new algebraic characterization of $\mathrm{FO}^2_m[<]$, and this characterization immediately yields decidability.

The algebraic approach to the membership problem of logical fragments has several advantages. In favorable cases, it opens the road to decidability procedures. Moreover, it allows a more *semantic* comparison of fragments; for example, the equality $\mathrm{FO}^2[<] = \Delta_2[<]$ was obtained by showing that both $\mathrm{FO}^2[<]$ and $\Delta_2[<]$ correspond to the same variety of finite monoids, namely **DA** [22, 33].

Building on previous detailed knowledge of the lattice of *band* varieties (varieties of idempotent monoids), Trotter and Weil defined a sub-lattice of the lattice of subvarieties of **DA** [37], which we call the $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy. These varieties have many interesting properties and in particular, each $\mathbf{R}_m$ (resp. $\mathbf{L}_m$) is efficiently decidable (by a combination of results of Trotter and Weil [37], Kufleitner and Weil [14], and Straubing and Weil [27], see Section 3 for more details). Moreover, one can climb up the $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy algebraically, using Mal'cev products, see [14] and Section 2 below; language-theoretically, in terms of alternated closures under deterministic and co-deterministic products [19, 15]; and combinatorially using *condensed* rankers, see [13, 16] and Section 2.

We relate the $\mathrm{FO}^2[<]$ quantifier alternation hierarchy with the $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy. More precisely, the main result of this paper is that a language is definable in $\mathrm{FO}^2_m[<]$ if and only if it is recognized by a monoid in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, thus establishing the decidability of

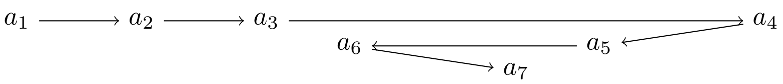


Figure 1 The positions defined by r in u, when $r = \mathsf{X}_{a_1}\mathsf{X}_{a_2}\mathsf{X}_{a_3}\mathsf{X}_{a_4}\mathsf{Y}_{a_5}\mathsf{Y}_{a_6}\mathsf{X}_{a_7}$ is condensed on u.

each $\mathrm{FO}^2_m[<]$. This result was first conjectured in [13], where one inclusion was established. Our proof combines a technique introduced by Klíma [8] and a substitution idea [10] with algebraic and combinatorial tools inspired by [15]. The proof is by induction and the base case is Simon's Theorem on piecewise testable languages [25].

2 Preliminaries

Let A be a finite alphabet and let A^* be the set of all finite words over A. The *length* $|u|$ of a word $u = a_1 \cdots a_n$, $a_i \in A$, is n and its *alphabet* is $\mathsf{alph}(u) = \{a_1, \ldots, a_n\} \subseteq A$. A position i of $u = a_1 \cdots a_n$ is an *a-position* if $a_i = a$. A factorization $u = u_- a u_+$ is the *a-left factorization* of u if $a \notin \mathsf{alph}(u_-)$, and it is the *$a$-right factorization* if $a \notin \mathsf{alph}(u_+)$, *i.e.*, we factor at the first or at the last a-position.

2.1 Rankers

A *ranker* is a nonempty word over the alphabet $\{\mathsf{X}_a, \mathsf{Y}_a \mid a \in A\}$. It is interpreted as a sequence of instructions of the form “go to the next a-position” and “go to the previous a-position”. More formally, for $u = a_1 \cdots a_n \in A^*$ and $x \in \{0, \ldots, n+1\}$ we let

$$\mathsf{X}_a(u,x) = \min\{y \mid y > x \text{ and } a_y = a\}, \qquad \mathsf{X}_a(u) = \mathsf{X}_a(u,0),$$
$$\mathsf{Y}_a(u,x) = \max\{y \mid y < x \text{ and } a_y = a\}, \qquad \mathsf{Y}_a(u) = \mathsf{Y}_a(u,n+1).$$

Here, both the minimum and the maximum of the empty set are undefined. The modality X_a is for “neXt-a” and Y_a is for “Yesterday-a”. For $r = \mathsf{Z}\,s$, $\mathsf{Z} \in \{\mathsf{X}_a, \mathsf{Y}_a \mid a \in A\}$, we set

$$r(u,x) = s(u, \mathsf{Z}(u,x)), \qquad r(u) = s(u, \mathsf{Z}(u)).$$

In particular, rankers are executed (as a set of instructions) from left to right. Every ranker r either defines a unique position in a word u, or it is undefined on u. For example, $\mathsf{X}_a\,\mathsf{Y}_b\,\mathsf{X}_c(bca) = 2$ and $\mathsf{X}_a\,\mathsf{Y}_b\,\mathsf{X}_c(bac) = 3$ whereas $\mathsf{X}_a\,\mathsf{Y}_b\,\mathsf{X}_c(cabc)$ and $\mathsf{X}_a\,\mathsf{Y}_b\,\mathsf{X}_c(bcba)$ are undefined. A ranker r is *condensed* on u if it is defined and, during the execution of r, no previously visited position is overrun [15]. One can think of condensed rankers as *zooming in* on the position they define, see Figure 1. More formally $r = \mathsf{Z}_1 \cdots \mathsf{Z}_k$, $\mathsf{Z}_i \in \{\mathsf{X}_a, \mathsf{Y}_a \mid a \in A\}$, is *condensed* on u if there exists a chain of open intervals

$$(0; |u|+1) = (x_0; y_0) \supset (x_1; y_1) \supset \cdots \supset (x_{n-1}; y_{n-1}) \ni r(u)$$

such that for all $1 \leq \ell \leq n-1$ the following properties are satisfied:

- If $\mathsf{Z}_\ell\,\mathsf{Z}_{\ell+1} = \mathsf{X}_a\,\mathsf{X}_b$, then $(x_\ell; y_\ell) = (\mathsf{X}_a(u, x_{\ell-1}); y_{\ell-1})$.
- If $\mathsf{Z}_\ell\,\mathsf{Z}_{\ell+1} = \mathsf{Y}_a\,\mathsf{Y}_b$, then $(x_\ell; y_\ell) = (x_{\ell-1}; \mathsf{Y}_a(u, y_{\ell-1}))$.
- If $\mathsf{Z}_\ell\,\mathsf{Z}_{\ell+1} = \mathsf{X}_a\,\mathsf{Y}_b$, then $(x_\ell; y_\ell) = (x_{\ell-1}; \mathsf{X}_a(u, x_{\ell-1}))$.
- If $\mathsf{Z}_\ell\,\mathsf{Z}_{\ell+1} = \mathsf{Y}_a\,\mathsf{X}_b$, then $(x_\ell; y_\ell) = (\mathsf{Y}_a(u, y_{\ell-1}); y_{\ell-1})$.

For example, $\mathsf{X}_a\,\mathsf{Y}_b\,\mathsf{X}_c$ is condensed on bca but not on bac.

The *depth* of a ranker is its length as a word. A *block* of a ranker is a maximal factor of the form $\mathsf{X}_{a_1} \cdots \mathsf{X}_{a_k}$ or of the form $\mathsf{Y}_{b_1} \cdots \mathsf{Y}_{b_\ell}$. A ranker with m blocks changes direction $m-1$

times. By $R_{m,n}$ we denote the class of all rankers with depth at most n and with up to m blocks. We write $R^{\mathsf{X}}_{m,n}$ for the set of all rankers in $R_{m,n}$ which start with an X_a-modality and we write $R^{\mathsf{Y}}_{m,n}$ for all rankers in $R_{m,n}$ which start with a Y_a-modality.

We define $u \rhd_{m,n} v$ if the same rankers in $R^{\mathsf{X}}_{m,n} \cup R^{\mathsf{Y}}_{m-1,n-1}$ are condensed on u and v. Similarly, $u \lhd_{m,n} v$ if the same rankers in $R^{\mathsf{Y}}_{m,n} \cup R^{\mathsf{X}}_{m-1,n-1}$ are condensed on u and v. The relations $\rhd_{m,n}$ and $\lhd_{m,n}$ are finite index congruences [15, Lem. 3.13].

The *order type* $\mathrm{ord}(i,j)$ is one of $\{<,=,>\}$, depending on whether $i<j$, $i=j$, or $i>j$, respectively. We define $u \equiv_{m,n} v$ if

- the same rankers in $R_{m,n}$ are defined on u and v,
- for all $r \in R^{\mathsf{X}}_{m,n}$ and $s \in R^{\mathsf{Y}}_{m,n-1}$: $\mathrm{ord}(r(u),s(u)) = \mathrm{ord}(r(v),s(v))$,
- for all $r \in R^{\mathsf{Y}}_{m,n}$ and $s \in R^{\mathsf{X}}_{m,n-1}$: $\mathrm{ord}(r(u),s(u)) = \mathrm{ord}(r(v),s(v))$,
- for all $r \in R^{\mathsf{X}}_{m,n}$ and $s \in R^{\mathsf{X}}_{m-1,n-1}$: $\mathrm{ord}(r(u),s(u)) = \mathrm{ord}(r(v),s(v))$,
- for all $r \in R^{\mathsf{Y}}_{m,n}$ and $s \in R^{\mathsf{Y}}_{m-1,n-1}$: $\mathrm{ord}(r(u),s(u)) = \mathrm{ord}(r(v),s(v))$.

▶ Remark. For $m=1$, each of the families $(\equiv_{1,n})_n$, $(\rhd_{1,n})_n$, and $(\lhd_{1,n})_n$ defines the class of piecewise testable languages, see *e.g.* [8, 25]. Recall that a language $L \subseteq A^*$ is *piecewise testable* if it is a Boolean combination of languages of the form $A^*a_1A^* \cdots a_kA^*$ ($k \geq 0$, $a_1,\dots,a_k \in A$).

2.2 First-order Logic

We denote by FO[<] the first-order logic over words interpreted as labeled linear orders. The atomic formulas are $\top$ (for *true*), $\bot$ (for *false*), the unary predicates $\mathbf{a}(x)$ (one for each $a \in A$), and the binary predicate $x<y$ for variables x and y. Variables range over the linearly ordered positions of a word and $\mathbf{a}(x)$ means that x is an a-position. Apart from the Boolean connectives, we allow composition of formulas using existential quantification $\exists x\colon \varphi$ and universal quantification $\forall x\colon \varphi$ for $\varphi \in \mathrm{FO}[<]$. The semantics is as usual. A *sentence* in FO[<] is a formula without free variables. For a sentence φ the *language defined by* φ, denoted by $L(\varphi)$, is the set of all words $u \in A^*$ which model φ.

The fragment $\mathrm{FO}^2[<]$ of first-order logic consists of all formulas which use at most two different names for the variables. This is a natural restriction, since FO with three variables already has the full expressive power of FO. A formula $\varphi \in \mathrm{FO}^2[<]$ is in $\mathrm{FO}^2_m[<]$ if, on every path of its parse tree, φ has at most m blocks of alternating quantifiers.

Note that $\mathrm{FO}^2_1[<]$-definable languages are exactly the piecewise testable languages, *cf.* [29]. For $m \geq 2$, we rely on the following important result, due to Weis and Immerman [40, Thm. 4.5].

▶ **Theorem 1.** *A language L is definable in $\mathrm{FO}^2_m[<]$ if and only if there exists an integer $n \geq 1$ such that L is a union of $\equiv_{m,n}$-classes.*

▶ Remark. The definition of $\equiv_{m,n}$ above is formally different from the conditions in Weis and Immerman's [40, Thm. 4.5]. A careful but elementary examination reveals that they are actually equivalent.

2.3 Algebra

A monoid M *recognizes* a language $L \subseteq A^*$ if there exists a morphism $\varphi : A^* \to M$ such that $L = \varphi^{-1}\varphi(L)$. If $\varphi : A^* \to M$ is a morphism, then we set $u \equiv_\varphi v$ if $\varphi(u) = \varphi(v)$. The join $\equiv_1 \vee \equiv_2$ of two congruences $\equiv_1$ and $\equiv_2$ is the least congruence containing $\equiv_1$ and $\equiv_2$. An element u is *idempotent* if $u^2 = u$. The set of all idempotents of a monoid M is denoted by

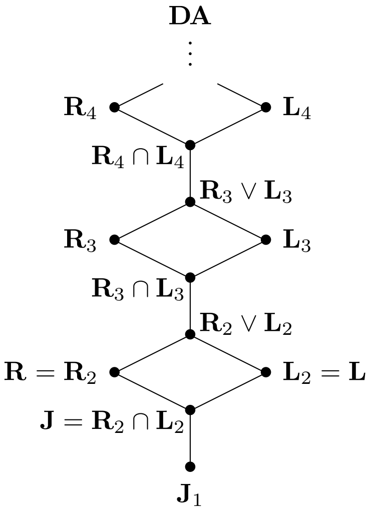

$\mathbf{DA}$
$\mathbf{R}_4$
$\mathbf{L}_4$
$\mathbf{R}_4 \cap \mathbf{L}_4$
$\mathbf{R}_3 \vee \mathbf{L}_3$
$\mathbf{R}_3$
$\mathbf{L}_3$
$\mathbf{R}_3 \cap \mathbf{L}_3$
$\mathbf{R}_2 \vee \mathbf{L}_2$
$\mathbf{R} = \mathbf{R}_2$
$\mathbf{L}_2 = \mathbf{L}$
$\mathbf{J} = \mathbf{R}_2 \cap \mathbf{L}_2$
$\mathbf{J}_1$

Figure 2 The $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy.

$E(M)$. For every finite monoid M there exists an integer $\omega \geq 1$ such that u^ω is idempotent for all $u \in M$. *Green's relations* $\mathcal{J}$, $\mathcal{R}$, and $\mathcal{L}$ are an important concept to describe the structural properties of a monoid M: we set $u \leq_\mathcal{J} v$ (resp. $u \leq_\mathcal{R} v$, $u \leq_\mathcal{L} v$) if $u = pvq$ (resp. $u = vq$, $u = pv$) for some $p, q \in M$. We also define $u \mathrel{\mathcal{J}} v$ (resp. $u \mathrel{\mathcal{R}} v$, $u \mathrel{\mathcal{L}} v$) if $u \leq_\mathcal{J} v$ and $v \leq_\mathcal{J} u$ (resp. $u \leq_\mathcal{R} v$ and $v \leq_\mathcal{R} u$, $u \leq_\mathcal{L} v$ and $v \leq_\mathcal{L} u$). The strict version $<_\mathcal{J}$ of $\leq_\mathcal{J}$ is defined by $u <_\mathcal{J} v$ if $u \leq_\mathcal{J} v$ but not $u \mathrel{\mathcal{J}} v$. The relations $<_\mathcal{R}$ and $<_\mathcal{L}$ are defined similarly. A monoid M is *$\mathcal{J}$-trivial* (resp. *$\mathcal{R}$-trivial*, *$\mathcal{L}$-trivial*) if $\mathcal{J}$ (resp. $\mathcal{R}$, $\mathcal{L}$) is the identity relation on M. We define the relations $\sim_\mathbf{K}$, $\sim_\mathbf{D}$, and $\sim_\mathbf{LI}$ on M as follows:

- $u \sim_\mathbf{K} v$ if and only if, for all $e \in E(M)$, we have either $eu, ev <_\mathcal{J} e$, or $eu = ev$.
- $u \sim_\mathbf{D} v$ if and only if, for all $f \in E(M)$, we have either $uf, vf <_\mathcal{J} f$, or $uf = vf$.
- $u \sim_\mathbf{LI} v$ if and only if, for all $e, f \in E(M)$ such that $e \mathrel{\mathcal{J}} f$, we have either $euf, evf <_\mathcal{J} e$, or $euf = evf$.

The relations $\sim_\mathbf{K}$, $\sim_\mathbf{D}$ and $\sim_\mathbf{LI}$ are congruences [9]. If $\mathbf{V}$ is a class of finite monoids, we say that a monoid M is in $\mathbf{K} \mathbin{\textcircled{m}} \mathbf{V}$ (resp. $\mathbf{D} \mathbin{\textcircled{m}} \mathbf{V}$, $\mathbf{LI} \mathbin{\textcircled{m}} \mathbf{V}$) if $M/{\sim_\mathbf{K}} \in \mathbf{V}$ (resp. $M/{\sim_\mathbf{D}} \in \mathbf{V}$, $M/{\sim_\mathbf{LI}} \in \mathbf{V}$). The classes $\mathbf{K} \mathbin{\textcircled{m}} \mathbf{V}$, $\mathbf{D} \mathbin{\textcircled{m}} \mathbf{V}$ and $\mathbf{LI} \mathbin{\textcircled{m}} \mathbf{V}$ are called *Mal'cev products* and they are usually defined in terms of relational morphisms. In the present context however, the definition above will be sufficient [9], see [5]. We will need the following classes of finite monoids:

- $\mathbf{J}_1$ consists of all finite commutative monoids satisfying $x^2 = x$.
- $\mathbf{J}$ (resp. $\mathbf{R}$, $\mathbf{L}$) consists of all finite $\mathcal{J}$-trivial (resp. $\mathcal{R}$-trivial, $\mathcal{L}$-trivial) monoids.
- $\mathbf{A}$ consists of all finite monoids satisfying $x^{\omega+1} = x^\omega$. Monoids in $\mathbf{A}$ are called *aperiodic.*
- $\mathbf{DA}$ consists of all finite monoids satisfying $(xy)^\omega x (xy)^\omega = (xy)^\omega$.
- $\mathbf{R}_1 = \mathbf{L}_1 = \mathbf{J}$, $\mathbf{R}_{m+1} = \mathbf{K} \mathbin{\textcircled{m}} \mathbf{L}_m$, $\mathbf{L}_{m+1} = \mathbf{D} \mathbin{\textcircled{m}} \mathbf{R}_m$.

It is well known that

$$\begin{aligned} &\mathbf{DA} = \mathbf{LI} \mathbin{\textcircled{m}} \mathbf{J}_1, \ \mathbf{R}_2 = \mathbf{R}, \ \mathbf{L}_2 = \mathbf{L}, \ \mathbf{R} \cap \mathbf{L} = \mathbf{J}, \text{and} \\ &\mathbf{R}_m \cup \mathbf{L}_m \ \subseteq \ \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1} \ \subsetneq \ \mathbf{DA} \ \subsetneq \ \mathbf{A} \end{aligned}$$

see *e.g.* [20]. The $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy is depicted in Figure 2.

2.4 The variety approach to the decidability of $\mathrm{FO}^2_m[<]$

Classes of finite monoids that are closed under taking submonoids, homomorphic images and finite direct products are called *pseudovarieties*. The classes of finite monoids $\mathbf{J}_1$, $\mathbf{J}$, $\mathbf{A}$, $\mathbf{DA}$, $\mathbf{R}_m$ and $\mathbf{L}_m$ introduced above are all pseudovarieties.

If $\mathbf{V}$ is a pseudovariety of monoids, then the class $\mathcal{V}$ of languages recognized by a monoid in $\mathbf{V}$ is called a *variety of languages*. Eilenberg's variety theorem (see *e.g.* [18, Annex B]) shows that varieties of languages are characterized by natural closure properties, and that the correspondence $\mathbf{V} \mapsto \mathcal{V}$ is onto. Elementary automata theory shows in addition that a language L is recognized by a monoid in a pseudovariety $\mathbf{V}$ if and only the syntactic monoid of L is in $\mathbf{V}$. It follows that if $\mathbf{V}$ has a decidable membership problem, then so does the corresponding variety of languages $\mathcal{V}$.

Simon's Theorem on piecewise testable languages [8, 25] is an important instance of this Eilenberg correspondence: a language L is recognizable by a monoid in $\mathbf{J}$ if and only if L is piecewise testable (and hence, as we already observed, if and only if L is definable in $\mathrm{FO}^2_1[<]$). Simon's result implies the decidability of piecewise testability.

It immediately follows from the definition that membership in $\mathbf{R}_m$ and $\mathbf{L}_m$ is decidable for all m since membership in $\mathbf{J}$ is decidable (see Corollary 8 for a more precise statement). Many additional properties of the pseudovarieties $\mathbf{R}_m$ and $\mathbf{L}_m$, and of the corresponding varieties of languages were established by the authors [14, 15, 37]. We will use in particular the following results, respectively [15, Cor. 3.15] and [14, Thms. 2.5 and 3.1].

▶ **Proposition 2.** *An A-generated monoid M is in $\mathbf{R}_m$ (resp. $\mathbf{L}_m$) if and only if there exists an integer n such that M is a quotient of $A^*/\triangleright_{m,n}$ (resp. $A^*/\triangleleft_{m,n}$).*

Let $x_1, x_2, \ldots$ be a sequence of variables. For each word u, we denote by $\overline{u}$ the mirror image of u, that is, the word obtained by reading u from right to left. Let $G_2 = x_2x_1$, $I_2 = x_2x_1x_2$ and, for $m \geq 2$, $G_{m+1} = x_{m+1}\overline{G_m}$ and $I_{m+1} = G_{m+1}x_{m+1}\overline{I_m}$. Finally, let φ be the substitution given by

$$\varphi(x_1) = (x_1^\omega x_2^\omega x_1^\omega)^\omega, \quad \varphi(x_2) = x_2^\omega,$$
$$\text{and, for } m \geq 2, \quad \varphi(x_{m+1}) = (x_{m+1}^\omega \varphi(G_m\overline{G_m})^\omega x_{m+1}^\omega)^\omega.$$

▶ **Proposition 3.** *$\mathbf{R}_m$ (resp. $\mathbf{L}_m$) is the class of finite monoids satisfying the identities $(xy)^\omega x(xy)^\omega = (xy)^\omega$ and $\varphi(G_m) = \varphi(I_m)$ (resp. $\varphi(\overline{G_m}) = \varphi(\overline{I_m})$).*

Straubing [29] and Kufleitner and Lauser [11, Cor. 1] established, by different means, that for each $m \geq 1$, the class of $\mathrm{FO}^2_m[<]$-definable languages forms a variety of languages, and we denote by $\mathbf{FO}^2_m$ the corresponding pseudovariety. In particular, $\mathbf{FO}^2_1 = \mathbf{J}$. Our strategy for establishing the decidability of $\mathrm{FO}^2_m[<]$-definability, is to establish the decidability of membership in $\mathbf{FO}^2_m$.

It is to be noted that neither Straubing's result, nor Kufleitner and Lauser's result implies the decidability of $\mathbf{FO}^2_m$. Straubing's result is the following [29, Thm. 4].

▶ **Theorem 4.** *For $m \geq 1$, $\mathbf{FO}^2_{m+1} = \mathbf{FO}^2_m ** \mathbf{J}$, where $**$ denotes the two-sided semidirect product.*

We refer the reader to [29] for the definition of the two-sided wreath product, which is also called the block product in the literature. As discussed by Straubing, this exact algebraic characterization of $\mathbf{FO}^2_m$ implies the decidability of $\mathbf{FO}^2_2$ but not of the other levels of the hierarchy. Straubing however conjectured that the following holds [29, Conj. 10].

▶ **Conjecture 5** (Straubing). *Let $u_1 = (x_1x_2)^\omega$, $v_1 = (x_2x_1)^\omega$ and, for $m \geq 1$,*

$$u_{m+1} = (x_1 \cdots x_{2n}x_{2n+1})^\omega u_n(x_{2n+2}x_1 \cdots x_{2n})^\omega$$
$$v_{m+1} = (x_1 \cdots x_{2n}x_{2n+1})^\omega v_n(x_{2n+2}x_1 \cdots x_{2n})^\omega.$$

Then a monoid is in $\mathbf{FO}^2_m$ if and only if it satisfies $x^{\omega+1} = x^\omega$ and $u_m = v_m$.[1]

If established, this conjecture would prove the decidability of each $\mathbf{FO}^2_m$. The authors on the other hand proved the following [15, Thm. 5.1].

▶ **Theorem 6.** *If a language L is recognized by a monoid in the join $\mathbf{R}_m \vee \mathbf{L}_m$, then L is definable in $\mathrm{FO}^2_m[<]$; and if L is definable in $\mathrm{FO}^2_m[<]$, then L is recognized by a monoid in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$.*

3 The FO^2 alternation hierarchy is decidable

We tighten the connection between the alternation hierarchy within $\mathrm{FO}^2[<]$ and the $\mathbf{R}_m$-$\mathbf{L}_m$-hierarchy and we prove the following result.

▶ **Theorem 7.** *A language $L \subseteq A^*$ is definable in $\mathrm{FO}^2_m[<]$ if and only if it is recognizable by a monoid in $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$.*

Theorem 7 immediately yields a decidability result.

▶ **Corollary 8.** *For each $m \geq 1$, it is decidable whether a given regular language L is $\mathrm{FO}^2_m[<]$-definable. This decision can be achieved in* LOGSPACE *on input the multiplication table of the syntactic monoid of L, and in* PSPACE *on input its minimal automaton.*

Moreover, given an $\mathrm{FO}^2[<]$-definable language L, one can compute the least integer m such that L is $\mathrm{FO}^2_m[<]$-definable.

Proof. We already observed that the $\mathbf{R}_m$ and $\mathbf{L}_m$ are decidable, and that each is described by two omega-term identities (Proposition 3). The decidability statement follows immediately. The complexity statement is a consequence of Straubing and Weil's [27, Thm. 2.19]. The computability statement follows immediately. ◀

We now turn to the proof of Theorem 7. One implication was established in Theorem 6. To prove the reverse implication, we prove Proposition 9 below, which establishes that every language recognized by a monoid $M \in \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$ is a union of $\equiv_{m,n}$-classes for some integer n depending on M. Theorem 7 follows, in view of Theorem 1.

▶ **Proposition 9.** *For every $m \geq 1$ and every morphism $\varphi\colon A^* \to M$ with $M \in \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$ there exists an integer n such that $\equiv_{m,n}$ is contained in $\equiv_\varphi$.*

Before we embark in the proof of Proposition 9, we record several algebraic and combinatorial lemmas.

[1] While this paper was under revision, Krebs and Straubing announced a proof of Conjecture 5, see `arXiv:1205.4802`.

$$\begin{tikzcd}
A^* \arrow[r, "\varphi"] \arrow[d, "\mathsf{alph}"'] & M \arrow[d, "\pi"] \\
\mathcal{P}(A) \arrow[r, "\psi"'] & M/{\sim_{\mathbf{LI}}}
\end{tikzcd}$$

Figure 3 $M \in \mathbf{DA} = \mathbf{LI} \mathbin{\textcircled{m}} \mathbf{J}_1$.

3.1 A collection of technical lemmas

▶ **Lemma 10.** *Let M be a finite monoid. If $s \mathrel{\mathcal{R}} sx$ and $x \sim_{\mathbf{K}} y$, then $sx = sy$. If $s \mathrel{\mathcal{L}} xs$ and $x \sim_{\mathbf{D}} y$, then $xs = ys$.*

Proof. Let $z \in M$ such that $sxz = s$. We have $(xz)^\omega x \mathrel{\mathcal{J}} (xz)^\omega$. Now, $x \sim_{\mathbf{K}} y$ implies $(xz)^\omega x = (xz)^\omega y$. Thus $sx = s(xz)^\omega x = s(xz)^\omega y = sy$. The second statement is left-right symmetric. ◀

The following lemma illustrates an important structural property of monoids in $\mathbf{DA}$.

▶ **Lemma 11.** *Let $\varphi\colon A^* \to M$ be a surjective morphism onto $M \in \mathbf{DA}$ and let $x, y, z \in A^*$ such that $\varphi(x) \mathrel{\mathcal{R}} \varphi(xy)$ and $\mathsf{alph}(z) \subseteq \mathsf{alph}(y)$. Then $\varphi(x) \mathrel{\mathcal{R}} \varphi(xz)$.*

Proof. The map $\mathsf{alph}\colon A^* \to \mathcal{P}(A)$ can be seen as a morphism, where the product on the power set $\mathcal{P}(A)$ of A is the union operation. Since $M \in \mathbf{DA}$, we have $M/{\sim_{\mathbf{LI}}} \in \mathbf{J}_1$; let $\pi\colon M \to M/{\sim_{\mathbf{LI}}}$ be the projection morphism. It is easily verified that there exists a morphism $\psi\colon \mathcal{P}(A) \to M/{\sim_{\mathbf{LI}}}$ such that $\psi \circ \mathsf{alph} = \pi \circ \varphi$, see Figure 3.

By assumption, $\varphi(x) = \varphi(xyt)$ for some $t \in A^*$, and hence $\varphi(x) = \varphi(x)\varphi(yt)^\omega$. Since $\mathsf{alph}((yt)^\omega) = \mathsf{alph}((yt)^\omega z (yt)^\omega)$, we have $\varphi(yt)^\omega \sim_{\mathbf{LI}} \varphi(yt)^\omega \varphi(z) \varphi(yt)^\omega$. Applying the definition of $\sim_{\mathbf{LI}}$ with $e = f = \varphi(yt)^\omega$, it follows that $\varphi(yt)^\omega = \varphi(yt)^\omega \varphi(z) \varphi(yt)^\omega$ and we now have

$$\varphi(x) = \varphi(x)\varphi(yt)^\omega = \varphi(x)\varphi(yt)^\omega\varphi(z)\varphi(yt)^\omega = \varphi(x)\varphi(z)\varphi(yt)^\omega.$$

Therefore $\varphi(x) \mathrel{\mathcal{R}} \varphi(xz)$, which concludes the proof. ◀

A proof of the following lemma can be found in [15, Prop. 3.6 and Lem. 3.7].

▶ **Lemma 12.** *Let $m \geq 2$, $u, v \in A^*$, $a \in A$.*

1. *If $u \triangleright_{m,n} v$ and $u = u_- a u_+$ and $v = v_- a v_+$ are a-left factorizations, then $u_- \triangleright_{m,n-1} v_-$ and $u_+ \triangleright_{m,n-1} v_+$.*
2. *If $u \triangleright_{m,n} v$ and $u = u_- a u_+$ and $v = v_- a v_+$ are a-right factorizations, then $u_- \triangleright_{m,n-1} v_-$ and $u_+ \triangleleft_{m-1,n-1} v_+$.*

Dual statements hold for $u \triangleleft_{m,n} v$.

▶ **Lemma 13.** *Let $m, n \geq 2$ and let $u = u_- a u_+$ and $v = v_- a v_+$ be a-left factorizations. If $u \equiv_{m,n} v$, then $u_- \equiv_{m-1,n-1} v_-$ and $u_+ \equiv_{m,n-1} v_+$. A dual statement holds for the factors of the a-right factorizations of u and v.*

Proof. We first show $u_- \equiv_{m-1,n-1} v_-$. Consider a ranker $r \in R_{m-1,n-1}$, supposing first that $r \in R^{\mathsf{X}}_{m-1,n-1}$. Then r is defined on u_- if and only if r is defined on u and $\mathrm{ord}(r'(u), \mathsf{X}_a(u))$ is $<$ for every nonempty prefix r' of r. By definition of $\equiv_{m,n}$, this is equivalent to r being defined on v_-. If instead $r \in R^{\mathsf{Y}}_{m-1,n-1}$, then r is defined on u_- if and only if $\mathsf{X}_a\, r \in R_{m,n}$ is defined on u and $\mathrm{ord}(\mathsf{X}_a\, r'(u), \mathsf{X}_a(u))$ is $<$ for every nonempty prefix r' of r. Again, this is equivalent to r being defined on v_- since $u \equiv_{m,n} v$. Thus, the same rankers in $R_{m-1,n-1}$ are defined on u_- and v_-.

Now consider rankers $r \in R^{\mathsf{X}}_{m-1,n-1}$ and $s \in R^{\mathsf{Y}}_{m-1,n-2}$, which we can assume to be defined on both u_- and v_-. Then the order types induced by r and s on u_- and v_- are equal, since $\mathrm{ord}(r(u_-), s(u_-)) = \mathrm{ord}(r(u), \mathsf{X}_a\, s(u)) = \mathrm{ord}(r(v), \mathsf{X}_a\, s(v)) = \mathrm{ord}(r(v_-), s(v_-))$ and $\mathsf{X}_a\, s \in R^{\mathsf{X}}_{m,n-1}$.

The same reasoning applies if $r \in R^{\mathsf{Y}}_{m-1,n-1}$ and $s \in R^{\mathsf{X}}_{m-1,n-2}$ (resp. if $r \in R^{\mathsf{X}}_{m-1,n-1}$ and $s \in R^{\mathsf{X}}_{m-1,n-2}$, if $r \in R^{\mathsf{Y}}_{m-1,n-1}$ and $s \in R^{\mathsf{Y}}_{m-2,n-2}$) since in that case, $\mathrm{ord}(r(u_-), s(u_-)) = \mathrm{ord}(\mathsf{X}_a\, r(u), s(u))$ (resp. $\mathrm{ord}(r(u), s(u))$, $\mathrm{ord}(\mathsf{X}_a\, r(u), \mathsf{X}_a\, s(u))$). Therefore, $u_- \equiv_{m-1,n-1} v_-$.

We now verify that $u_+ \equiv_{m,n-1} v_+$. The proof is very similar to the first part and deviates only in technical details. Consider a ranker $r \in R_{m,n-1}$, say, in $R^{\mathsf{X}}_{m,n-1}$. Then r is defined on u_+ if and only if $\mathsf{X}_a\, r \in R_{m,n}$ is defined on u and $\mathrm{ord}(\mathsf{X}_a\, r'(u), \mathsf{X}_a(u))$ is $>$ for every nonempty prefix r' of r. Again, this is equivalent to r being defined on v_+ since $u \equiv_{m,n} v$. If instead $r \in R^{\mathsf{Y}}_{m,n-1}$, then r is defined on u_+ if and only if r is defined on u and $\mathrm{ord}(r'(u), \mathsf{X}_a(u))$ is $>$ for every nonempty prefix r' of r, which is equivalent to r being defined on v_+. Thus, the same rankers in $R_{m,n-1}$ are defined on u_+ and v_+.

Now consider rankers $r \in R^{\mathsf{X}}_{m,n-1}$ and $s \in R^{\mathsf{Y}}_{m,n-2}$, both defined on u_+ and v_+. Then the order types induced by r and s on u_+ and v_+ are equal, since $\mathrm{ord}(r(u_+), s(u_+)) = \mathrm{ord}(\mathsf{X}_a\, r(u), s(u))$ and $\mathsf{X}_a\, r \in R^{\mathsf{X}}_{m,n}$.

Again, a similar verification guarantees that the order types induced by r and s on u_+ and v_+ are equal also if $r \in R^{\mathsf{Y}}_{m,n-1}$ and $s \in R^{\mathsf{X}}_{m,n-2}$, or if $r \in R^{\mathsf{X}}_{m,n-1}$ and $s \in R^{\mathsf{X}}_{m-1,n-2}$, or if $r \in R^{\mathsf{Y}}_{m,n-1}$ and $s \in R^{\mathsf{Y}}_{m-1,n-2}$. This shows $u_+ \equiv_{m,n-1} v_+$ which completes the proof. ◀

▶ **Lemma 14.** *Let $m, n \geq 2$ and let $u = u_- a u_0 b u_+$ and $v = v_- a v_0 b v_+$ describe b-left and a-right factorizations (that is, $a \notin \mathsf{alph}(u_0 b u_+) \cup \mathsf{alph}(v_0 b v_+)$ and $b \notin \mathsf{alph}(u_- a u_0) \cup \mathsf{alph}(v_- a v_0)$). If $u \equiv_{m,n} v$, then $u_0 \equiv_{m-1,n-1} v_0$.*

Proof. A ranker $r \in R^{\mathsf{X}}_{m-1,n-1}$ is defined on u_0 if and only if $\mathsf{Y}_a\, r \in R_{m,n}$ is defined on u and $\mathrm{ord}(\mathsf{Y}_a\, r'(u), \mathsf{Y}_a(u))$ is $>$ and $\mathrm{ord}(\mathsf{Y}_a\, r'(u), \mathsf{X}_b(u))$ is $<$ for every nonempty prefix r' of r. Similarly, a ranker $r \in R^{\mathsf{Y}}_{m-1,n-1}$ is defined on u_0 if and only if $\mathsf{X}_b\, r \in R_{m,n}$ is defined on u and $\mathrm{ord}(\mathsf{X}_b\, r'(u), \mathsf{Y}_a(u))$ is $>$ and $\mathrm{ord}(\mathsf{X}_b\, r'(u), \mathsf{X}_b(u))$ is $<$ for every nonempty prefix r' of r. Thus, if $u \equiv_{m,n} v$, then the same rankers in $R_{m-1,n-1}$ are defined on u_0 and v_0.

Now consider rankers $r \in R^{\mathsf{X}}_{m-1,n-1}$ and $s \in R^{\mathsf{Y}}_{m-1,n-2}$ (resp. $r \in R^{\mathsf{Y}}_{m-1,n-1}$ and $s \in R^{\mathsf{X}}_{m-1,n-2}$), defined on both u_0 and v_0. Then $\mathrm{ord}(r(u_0), s(u_0)) = \mathrm{ord}(\mathsf{Y}_a\, r(u), \mathsf{X}_b\, s(u))$ (resp. $\mathrm{ord}(\mathsf{X}_b\, r(u), \mathsf{Y}_a\, s(u))$). Since $u \equiv_{m,n} v$, $\mathsf{Y}_a\, r \in R^{\mathsf{Y}}_{m,n}$ and $\mathsf{X}_b\, s \in R^{\mathsf{X}}_{m,n-1}$ (resp. $\mathsf{X}_b\, r \in R^{\mathsf{X}}_{m,n}$ and $\mathsf{Y}_a\, s \in R^{\mathsf{Y}}_{m,n-1}$), the order types defined by r and s on u_0 and v_0 are equal.

If $m = 2$, we are done proving that $u_0 \equiv_{m-1,n-1} v_0$. We now assume that $m \geq 3$. Let $r \in R^{\mathsf{X}}_{m-1,n-1}$ and $s \in R^{\mathsf{X}}_{m-2,n-2}$ (resp. $r \in R^{\mathsf{Y}}_{m-1,n-1}$ and $s \in R^{\mathsf{Y}}_{m-2,n-2}$) be defined on both u_0 and v_0. Then $\mathrm{ord}(r(u_0), s(u_0)) = \mathrm{ord}(\mathsf{Y}_a\, r(u), \mathsf{Y}_a\, s(u))$ (resp. $\mathrm{ord}(\mathsf{X}_b\, r(u), \mathsf{X}_b\, s(u))$). By the same reasoning as above, the order type defined by v on u_0 and v_0 is the same since $\mathsf{Y}_a\, r \in R^{\mathsf{Y}}_{m,n}$ and $\mathsf{Y}_a\, s \in R^{\mathsf{Y}}_{m-1,n-1}$ (resp. $\mathsf{X}_b\, r \in R^{\mathsf{X}}_{m,n}$ and $\mathsf{X}_b\, s \in R^{\mathsf{X}}_{m-1,n-1}$). This concludes the proof of the lemma. ◀

3.2 Proof of Proposition 9

The proof is by induction on m. We already observed that L is $\mathrm{FO}^2_1[<]$-definable if and only if it is piecewise testable, if and only if it is accepted by a monoid in $\mathbf{J}$. Since $\mathbf{J} = \mathbf{R}_2 \cap \mathbf{L}_2$, Proposition 9 holds for $m = 1$. We now assume that $m \geq 2$.

$$\begin{tikzcd}
 & & A^* \arrow[d, "\varphi"] \arrow[ddll] \arrow[ddl, bend right, "\rho"'] \arrow[ddr, bend left, "\lambda"] \arrow[ddrr] \arrow[dddrrr, bend left] & & & \\
 & & M \arrow[dl, "\pi_{\mathbf{D}}"'] \arrow[dr, "\pi_{\mathbf{K}}"] & & & \\
A^*/\rhd_{m,n} \arrow[r] & M/{\sim_{\mathbf{D}}} \arrow[dr] & & M/{\sim_{\mathbf{K}}} \arrow[dl] & A^*/\lhd_{m,n} \arrow[l] & \\
 & & A^*/(\equiv_\rho \vee \equiv_\lambda) & & & A^*/{\equiv_{m-1,n}} \arrow[lll]
\end{tikzcd}$$

Figure 4 A commutative diagram.

Let $\varphi\colon A^* \to M$ be a morphism with $M \in \mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$. We note that it suffices to prove Proposition 9 for the morphism $\varphi'\colon A^* \to M \times 2^A$ given by $\varphi'(u) = (\varphi(u), \mathsf{alph}(u))$. Observe that, for $u, v \in A^*$,

$$\varphi'(u) \sim_{\mathbf{D}} \varphi'(v) \text{ (resp. } \varphi'(u) \sim_{\mathbf{K}} \varphi'(v)) \quad \text{implies} \quad \mathsf{alph}(u) = \mathsf{alph}(v). \tag{1}$$

Indeed we have $\varphi'(u)\varphi'(u)^\omega = \varphi'(u)^\omega$ (since M is aperiodic): then $\varphi'(u) \sim_{\mathbf{D}} \varphi'(v)$ implies that $\varphi'(v)\varphi'(u)^\omega = \varphi'(u)\varphi'(u)^\omega$ and by definition of φ', $\mathsf{alph}(v)$ is contained in $\mathsf{alph}(u)$. By symmetry, u and v have the same alphabetical content and the same holds for $\sim_{\mathbf{K}}$.

To lighten up the notation, we dispense with the consideration of φ' and we assume that φ satisfies Property (1). Moreover, we assume that φ is surjective.

Let $\pi_{\mathbf{D}} : M \to M/{\sim_{\mathbf{D}}}$ and $\pi_{\mathbf{K}} : M \to M/{\sim_{\mathbf{K}}}$ be the natural morphisms. By definition of $\mathbf{R}_{m+1}$ and $\mathbf{L}_{m+1}$, we have $M/{\sim_{\mathbf{D}}} \in \mathbf{R}_m$ and $M/{\sim_{\mathbf{K}}} \in \mathbf{L}_m$. Let $\rho = \pi_{\mathbf{D}} \circ \varphi$ and $\lambda = \pi_{\mathbf{K}} \circ \varphi$, see Figure 4. The monoid $A^*/(\equiv_\rho \vee \equiv_\lambda)$ is a quotient of both $M/{\sim_{\mathbf{D}}}$ and $M/{\sim_{\mathbf{K}}}$, so $A^*/(\equiv_\rho \vee \equiv_\lambda) \in \mathbf{R}_m \cap \mathbf{L}_m$ and there exists $n \geq 1$ such that

- $\rhd_{m,n}$ is contained in $\equiv_\rho$ and $\lhd_{m,n}$ is contained in $\equiv_\lambda$ (by Proposition 2),
- $\equiv_{m-1,n}$ is contained in $\equiv_\rho \vee \equiv_\lambda$ (by induction).

We show that $\equiv_{m,n+2|M|}$ is contained in $\equiv_\varphi$. Let $u \equiv_{m,n+2|M|} v$. Consider the $\mathcal{R}$-factorization of u, *i.e.*, $u = s_1 a_1 \cdots s_k a_k s_{k+1}$ with $a_i \in A$ and $s_i \in A^*$ such that $1 = \varphi(s_1)$ and for all $1 \leq i \leq k$:

$$\varphi(s_1 a_1 \cdots s_i) >_{\mathcal{R}} \varphi(s_1 a_1 \cdots s_i a_i) \mathrel{\mathcal{R}} \varphi(s_1 a_1 \cdots s_i a_i s_{i+1}).$$

Since the number of $\mathcal{R}$-classes is at most $|M|$, we have $k < |M|$. Similarly, let $v = t_1 b_1 \cdots t_{k'} b_{k'} t_{k'+1}$ with $b_i \in A$ and $t_i \in A^*$ be the $\mathcal{L}$-factorization of v such that $\varphi(t_{k'+1}) = 1$ and for all $1 \leq i \leq k'$:

$$\varphi(t_i b_i t_{i+1} \cdots b_{k'} t_{k'+1}) \mathrel{\mathcal{L}} \varphi(b_i t_{i+1} \cdots b_{k'} t_{k'+1}) <_{\mathcal{L}} \varphi(t_{i+1} \cdots b_{k'} t_{k'+1}).$$

As before, we have $k' < |M|$. By Lemma 11 (applied with $x = s_1 \cdots s_{i-1} a_{i-1}$, $y = s_i$ and $z = a_i$), we have $a_i \notin \mathsf{alph}(s_i)$; and similarly, $b_i \notin \mathsf{alph}(t_{i+1})$. Therefore, the positions of the a_i's in u are exactly the positions visited by the ranker $r = \mathsf{X}_{a_1} \cdots \mathsf{X}_{a_k}$, and the positions of the b_i's in v are exactly the positions visited by the ranker $s = \mathsf{Y}_{b_{k'}} \cdots \mathsf{Y}_{b_1}$. Since $u \equiv_{m,n+2|M|} v$, each of the rankers r and s is defined on both u and v, and all the positions visited by the rankers r and s occur in the same order in u as in v. We call these positions *special.* Let

$$u = u_1c_1 \cdots u_\ell c_\ell u_{\ell+1}$$
$$v = v_1c_1 \cdots v_\ell c_\ell v_{\ell+1}$$

be obtained by factoring u and v at all the special positions. We have $\ell \leq k + k' < 2\,|M|$. We say that a special position is *red* if it is visited by r, and that it is *green* if it is visited by s. Some special positions may be both red and green, which means that more than one of the cases below may apply.

For u the above factorization is a refinement of the $\mathcal{R}$-factorization; and for v it is a refinement of the $\mathcal{L}$-factorization. In particular, $\varphi(u_1) = 1$, $\varphi(v_{\ell+1}) = 1$ and

$$\varphi(u_1 \cdots u_{i-1}c_{i-1})\ \mathcal{R}\ \varphi(u_1 \cdots u_{i-1}c_{i-1}u_i) \qquad \text{for } 1 < i \leq \ell + 1, \qquad (\mathrm{Eq}(\mathcal{R}))$$
$$\varphi(v_ic_iv_{i+1} \cdots c_\ell)\ \mathcal{L}\ \varphi(c_iv_{i+1} \cdots c_\ell) \qquad \text{for } 1 \leq i \leq \ell. \qquad (\mathrm{Eq}(\mathcal{L}))$$

In order to prove $u \equiv_\varphi v$, we show that we can gradually substitute u_i for v_i in the product $v_1c_1 \cdots v_\ell c_\ell v_{\ell+1} = v$, starting from $i = 1$, while maintaining $\equiv_\varphi$-equivalence. Namely we show that, for each i, it holds

$$u_1 \cdots u_{i-1}c_{i-1}\, u_i\, c_iv_{i+1} \cdots v_{\ell+1} \equiv_\varphi u_1 \cdots u_{i-1}c_{i-1}\, v_i\, c_iv_{i+1} \cdots v_{\ell+1}. \qquad (\mathrm{Eq(i)})$$

Let h_0 be the index of the leftmost red position: then $c_{h_0} = a_1$ and $s_1 = u_1c_1 \cdots u_{h_0}$. Since $\varphi(s_1) = 1$ and M is aperiodic, the φ-image of every letter in s_1 is 1. Applying Lemma 13 to the a_1-left factorizations of u and v, we find that $u_1c_1 \cdots u_{h_0-1} \equiv_{m-1,n-1} v_1c_1 \cdots v_{h_0-1}$ and in particular, these words have the same alphabet. It follows that $\varphi(u_i) = \varphi(v_i) = 1$ for all $i \leq h_0$, and hence $(\mathrm{Eq}(i))$ holds for all $i \leq h_0$.

The right-left dual of this reasoning establishes that $\varphi(u_i) = \varphi(v_i) = 1$ for all the u_i, v_i to the right of the last (rightmost) green position; let j_0 be its index. In particular, $(\mathrm{Eq}(i))$ also holds for all $i > j_0$.

We now assume that $h_0 < i \leq j_0$ and we let $h - 1$ be the index of the first red position to the left of i and j be the index of the first green position to the right of i: we have $h_0 < h \leq i \leq j \leq j_0$.

Case 1: $h = i$ ($i - 1$ is red)

We have $u \vartriangleright_{m,n+2|M|} v$. By Lemma 12 (1), a sequence of at most $i - 1$ left-factorizations yields $u_ic_i \cdots u_{\ell+1} \vartriangleright_{m,n+2|M|-i+1} v_ic_i \cdots v_{\ell+1}$. If i is red, then by Lemma 12 (1), after one c_i-left-factorization, we see that $u_i \vartriangleright_{m,n+2|M|-i} v_i$. If i is not red, then i is green and by Lemma 12 (2), after at most $\ell - i$ right-factorizations, we find that u_i and v_i are $\vartriangleright_{m,n+2|M|-i-(\ell-i)}$-equivalent. In any case, we have $u_i \vartriangleright_{m,n} v_i$ and thus $u_i \equiv_\rho v_i$ (*i.e.*, $\varphi(u_i) \sim_{\mathbf{D}} \varphi(v_i)$) by the choice of n. In view of $(\mathrm{Eq}(\mathcal{L}))$, Lemma 10 now implies

$$u_ic_iv_{i+1} \cdots c_\ell v_{\ell+1} \equiv_\varphi v_ic_iv_{i+1} \cdots c_\ell v_{\ell+1}$$

and left multiplication by $u_1c_1 \cdots c_{i-1}$ yields $(\mathrm{Eq}(i))$.

Case 2: $j = i$ (i is green)

As in Case 1, we see that $u_i \equiv_\lambda v_i$. $(\mathrm{Eq}(\mathcal{R}))$ and Lemma 10 then implies

$$u_1c_1 \cdots u_{i-1}c_{i-1}u_i \equiv_\varphi u_1c_1 \cdots u_{i-1}c_{i-1}v_i,$$

and right multiplication by $c_iv_{i+1} \cdots v_{\ell+1}$ yields $(\mathrm{Eq}(i))$.

Case 3: $h < i < j$ ($i - 1$ is not red and i is not green)

By Lemma 13, after at most $h - 1$ left factorizations and $\ell - j + 1$ right factorizations, we obtain $u_h c_h \cdots u_j \equiv_{m,n+j-h} v_h c_h \cdots v_j$ (since $n + j - h \leq n + 2\,|M| - (h-1) - (\ell - j + 1)$). Lemma 14, applied with $a = c_{i-1}$ and $b = c_i$, then yields $u_i \equiv_{m-1,n} v_i$. Since $\equiv_{m-1,n}$ is contained in $\equiv_\lambda \vee \equiv_\rho$, there exist words $w_1, \ldots, w_d$ such that

$$v_i = w_1 \equiv_\rho w_2 \equiv_\lambda w_3 \equiv_\rho \cdots \equiv_\lambda w_{d-2} \equiv_\rho w_{d-1} \equiv_\lambda w_d = u_i.$$

After the discussion at the beginning of this section, we have $\mathsf{alph}(v_i) = \mathsf{alph}(w_2) = \cdots = \mathsf{alph}(w_{d-1}) = \mathsf{alph}(u_i)$. Thus, by Lemma 11, we have $\varphi(pu_i) \mathrel{\mathcal{R}} \varphi(p)$ if and only if $\varphi(pw_g) \mathrel{\mathcal{R}} \varphi(p)$, and $\varphi(v_i q) \mathrel{\mathcal{L}} \varphi(q)$ if and only if $\varphi(w_g q) \mathrel{\mathcal{L}} \varphi(q)$ for all $p, q \in A^*$. As in Cases 1 and 2, we conclude that for each $1 \leq e < d$,

- if $w_e \equiv_\rho w_{e+1}$, then

$$w_e c_i \cdots c_\ell v_{\ell+1} \equiv_\varphi w_{e+1} c_i \cdots c_\ell v_{\ell+1}, \text{ and thus}$$
$$u_1 c_1 \cdots u_i c_{i-1} w_e c_i \cdots c_\ell v_{\ell+1} \equiv_\varphi u_1 c_1 \cdots u_i c_{i-1} w_{e+1} c_i \cdots c_\ell v_{\ell+1};$$

- and if $w_e \equiv_\lambda w_{e+1}$, then

$$u_1 c_1 \cdots c_{i-1} w_e \equiv_\varphi u_1 c_1 \cdots c_{i-1} w_{e+1}, \text{ and thus}$$
$$u_1 c_1 \cdots c_{i-1} w_e c_i v_{i+1} \cdots c_\ell v_{\ell+1} \equiv_\varphi u_1 c_1 \cdots c_{i-1} w_{e+1} c_i v_{i+1} \cdots c_\ell v_{\ell+1}.$$

It follows by transitivity of $\equiv_\varphi$ that (Eq(i)) holds.

Concluding the proof

We have now established (Eq(i)) for every $1 \leq i \leq \ell+1$. It follows immediately, by transitivity, that $u \equiv_\varphi v$. ◀

4 Conclusion

We have shown that for each $m \geq 1$, it is decidable whether a given regular language is $\mathrm{FO}^2_m[<]$-definable. Previous results in the literature only showed decidability for levels 1 and 2 of this quantifier alternation hierarchy. Our decidability result follows from the proof that $\mathbf{FO}^2_m$ (the pseudovariety of finite monoids corresponding to the $\mathrm{FO}^2_m[<]$-definable languages) is equal to the intersection $\mathbf{R}_{m+1} \cap \mathbf{L}_{m+1}$, which was known to be decidable.

This result implies the decidability of the levels of the hierarchy given by $\mathbf{V}_1 = \mathbf{J}$ and $\mathbf{V}_{m+1} = \mathbf{V}_m ** \mathbf{J}$, since Straubing showed that $\mathbf{V}_m = \mathbf{FO}^2_m$ [29]. Straubing used general results of Almeida and Weil on two-sided semidirect products to deduce from this that $\mathbf{FO}^2_2$ is decidable, but these results do not extend to $\mathbf{FO}^2_m$ when $m > 2$ ([1, 38], see [29, Sec. 5] for a discussion). In particular, this shows that Conjecture 5 holds for $m = 2$.

We also showed that the decision procedure whether a regular language L is FO^2_m-definable, is in LOGSPACE on input the multiplication table of the syntactic monoid of L, and in PSPACE on input the minimal automaton of L. The result behind this statement is the fact that membership in $\mathbf{R}_m$ and in $\mathbf{L}_m$ is characterized by a small set of (rather complicated) identities. Another equational description of $\mathbf{R}_m$ and $\mathbf{L}_m$ was recently given by Kufleitner and Lauser [12]. Straubing conjectured a different and simpler set of identities for $\mathbf{FO}^2_m$, see Conjecture 5 above. While this paper was under revision, Krebs and Straubing announced that they succeeded in extending the latter set of identities to all the levels of the hierarchy, see `arXiv:1205.4802`.

References

1 Jorge Almeida and Pascal Weil. Profinite categories and semidirect products. *Journal of Pure and Applied Algebra*, 123:1–50, 1998.

2 Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.

3 Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 261–306. Amsterdam University Press, 2008.

4 Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008. Special issue DLT'07.

5 Tom E. Hall and Pascal Weil. On radical congruence systems. *Semigroup Forum*, 59:56–73, 1999.

6 Neil Immerman and Dexter Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, November 1989.

7 Johan Anthony Willem Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, 1968.

8 Ondřej Klíma. Piecewise testable languages via combinatorics on words. *Discrete Mathematics*, 311(20):2124–2127, 2011.

9 Kenneth Krohn, John L. Rhodes, and Bret Tilson. Homomorphisms and semilocal theory. In Michael A. Arbib, editor, *Algebraic Theory of Machines, Languages, and Semigroups*, chapter 8, pages 191–231. Academic Press, New York and London, 1968.

10 Manfred Kufleitner and Alexander Lauser. Languages of dot-depth one over infinite words. In *LICS'11, Proceedings*, pages 23–32. IEEE Computer Society, 2011.

11 Manfred Kufleitner and Alexander Lauser. Lattices of logical fragments over words. In *ICALP'12, Proceedings*, volume 7392 of *LNCS*, pages 275–286. Springer, 2012.

12 Manfred Kufleitner and Alexander Lauser. The join levels of the Trotter-Weil hierarchy are decidable. In *MFCS'12, Proceedings*, *LNCS*. Springer, 2012. To appear.

13 Manfred Kufleitner and Pascal Weil. On FO^2 quantifier alternation over words. In *MFCS'09, Proceedings*, volume 5734 of *LNCS*, pages 513–524. Springer, 2009.

14 Manfred Kufleitner and Pascal Weil. On the lattice of sub-pseudovarieties of **DA**. *Semigroup Forum*, 81(2):243–254, 2010.

15 Manfred Kufleitner and Pascal Weil. On logical hierarchies within FO^2-definable languages. *Log. Methods Comput. Sci.*, 2012. To appear.

16 Kamal Lodaya, Paritosh K. Pandya, and Simoni S. Shah. Marking the chops: an unambiguous temporal logic. In *IFIP TCS'08, Proceedings*, volume 273 of *IFIP*, pages 461–476. Springer, 2008.

17 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. The MIT Press, 1971.

18 Dominique Perrin and Jean-Éric Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, Amsterdam, 2004.

19 Jean-Éric Pin. Propriétés syntactiques du produit non ambigu. In W. Kuich, editor, *Proc.7th International Colloquium Automata, Languages and Programming (ICALP'80)*, volume 85 of *Lecture Notes in Computer Science*, pages 483–499, Heidelberg, 1980. Springer-Verlag.

20 Jean-Éric Pin. *Varieties of Formal Languages*. North Oxford Academic, London, 1986.

21 Jean-Éric Pin. Syntactic semigroups. In *Handbook of Formal Languages*, volume 1, pages 679–746. Springer-Verlag, Berlin, 1997.

22 Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory Comput. Syst.*, 30(4):383–422, 1997.

23 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control*, 8:190–194, 1965.

24 Thomas Schwentick, Denis Thérien, and Heribert Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *DLT'01, Proceedings*, volume 2295 of *LNCS*, pages 239–250. Springer, 2002.

25 Imre Simon. Piecewise testable events. In H. Barkhage, editor, *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 22–23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer-Verlag, 1975.

26 Larry Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, TR 133, M.I.T., Cambridge, 1974.

27 Howard Straubing and Pascal Weil. An introduction to finite automata and their connection to logic. In D. D'Souza and Priti Shankar, editors, *Modern Applications of Automata Theory*, volume 2 of *IISc Research Monographs*, pages 3–43. World Scientific, 2012.

28 Howard Straubing. A generalization of the Schützenberger product of finite monoids. *Theor. Comput. Sci.*, 13:137–150, 1981.

29 Howard Straubing. Algebraic characterization of the alternation hierarchy in $\mathrm{FO}^2[<]$ on finite words. In Marc Bezem, editor, *CSL'11*, volume 12 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 525–537, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

30 Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages 2001, Proceedings*, pages 475–500. World Scientific, 2002.

31 Pascal Tesson and Denis Thérien. Logic meets algebra: The case of regular languages. *Log. Methods Comput. Sci.*, 3(1):1–37, 2007.

32 Denis Thérien. Classification of finite monoids: The language approach. *Theor. Comput. Sci.*, 14(2):195–208, 1981.

33 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC'98, Proceedings*, pages 234–240. ACM Press, 1998.

34 Denis Thérien and Thomas Wilke. Temporal logic and semidirect products: An effective characterization of the until hierarchy. *SIAM J. Comput.*, 31(3):777–798, 2001.

35 Denis Thérien and Thomas Wilke. Nesting until and since in linear temporal logic. *Theory of Computing Systems*, 37(1):111–131, 2004.

36 Wolfgang Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

37 Peter Trotter and Pascal Weil. The lattice of pseudovarieties of idempotent semigroups and a non-regular analogue. *Algebra Universalis*, 37(4):491–526, 1997.

38 Pascal Weil. Profinite methods in semigroup theory. *International Journal of Algebra and Computation*, 12:137–178, 2002.

39 Philipp Weis. *Expressiveness and succinctness of first-order logic on finite words.* PhD thesis, University of Massachusetts Amherst, 2011.

40 Philipp Weis and Neil Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Log. Methods Comput. Sci.*, 5(3), 2009.

Axiomatizing proof tree concepts in Bounded Arithmetic

Satoru Kuroda

Gunma Prefectural Women's University
1395-1, Kaminote, Tamamura, Gunma, 370-1193, Japan
satoru@gpwu.ac.jp

Abstract

We construct theories of Cook-Nguyen style two-sort bounded arithmetic whose provably total functions are exactly those in LOGCFL and LOGDCFL. Axiomatizations of both theories are based on the proof tree size characterizations of these classes. We also show that our theory for LOGCFL proves a certain formulation of the pumping lemma for context-free languages.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Bounded Arithmetic, LOGCFL, LOGDCFL, Proof tree

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.440

1 Introduction

The complexity classes LOGCFL and LOGDCFL are subclasses of P which gain a large popularity in the complexity theory community during the last two decades. The class LOGCFL consists of predicates logspace reducible to context free languages and LOGDCFL is defined in the same manner as LOGCFL but with deterministic context free languages.

LOGCFL lies between NL and AC^1 and LOGDCFL contains L but we do not know whether any of these inclusions is proper or not. Furthermore, we do not know the relationship between NL and LOGDCFL.

One of the main feature of LOGCFL is that it has several natural alternative characterizations such as NAuxPDA [9], alternating Turing machines [8] and semi-unbounded fan-in circuits [10]. Also LOGCFL has natural complete problems such as word problems for finite groupoids and acyclic conjunctive queries [4]. Moreover, it is worth noting that LOGCFL is closed under many operations including the closure under complementation [2].

On the contrary, much less is known about the class LOGDCFL. In particular, no natural complete problem is found so far.

In this paper we construct theories of bounded arithmetic for LOGCFL and LOGDCFL using their proof tree size characterizations. To author's knowledge, no such theory for LOGDCFL has been proposed so far. For LOGCFL, the author [6] defined a theory $V\text{-}Q^{SAC}$ which axiomatize the concept of SAC^1 circuits. However, the theory has an augmented language which represents generalized quantifier expressing that a SAC^1 circuit has an accepting tree on an input.

Our approach is very similar to the one developed by Cook and Nguyen [3], namely augmenting the theory V^0 by axioms expressing the concept of a given complexity class.

The concepts we use for constructing theories for LOGCFL and LOGDCFL are somewhat similar. That is, both concepts are based on circuits having polynomial proof tree size.

For LOGDCFL, McKenzie et.al. [7] proved that the class is identical to the class of predicates decidable by polynomial size Multiplex-Select circuits with polynomial proof tree

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 440–454
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

size. For LOGCFL, a similar and simpler characterization is known by Venkateswaran [10] as the class of polynomial size Boolean circuits having polynomial proof tree size.

Although these two characterizations have a similar nature, we encounter with a slightly different circumstance when axiomatizing them. Since LOGDCFL is a deterministic class, there exists a unique witness, namely a proof tree for an accepting input to a circuit, whereas, there may exist more than one polynomial size proof trees for a general circuit which witnesses LOGCFL predicates. Such a difference causes a difficulty in the witnessing argument of the corresponding theories. However, we show that a slight modification of the argument of Cook and Nguyen implies witnessing theorems for both theories.

We also show that our theory **V**LCFL for LOGCFL is finitely axiomatizable by formalizing LOGCFL complete problems.

Cook proposed a research program called Bounded Reverse Mathematics whose aim is to decide how much computational concepts are needed to prove mathematical theorems. This is achieved by deciding which theorems of mathematics are provable in a given bounded arithmetic theory (See [3] for an exposition). For this direction, we show that **V**LCFL proves a form of the pumping lemma for context-free languages.

The essential part of formalizing the standard proof of the pumping lemma in **V**LCFL is to translate parse trees into proof trees of Boolean circuits so that we can argue about context-free languages in terms of circuits.

By carefully examining the textbook style proof of the pumping lemma, it turns out that proof also uses combinatorial arguments including the pigeonhole principle and PATH problem which is NL complete. Moreover, the inductive argument in the proof can be expressed as a number induction for LOGCFL decidable predicate.

2 Preliminaries

Throughout the paper we deal with two-sort theories and complexity classes. So first we briefly review their basic notions.

Two-sort logic uses two types of variables; number variables denoted by lower case letters $x, y, z \cdots$ represent natural numbers while string variables denoted by upper case letters $X, Y, Z \cdots$ represent binary strings.

2.1 Two-sort complexity classes

A two-sort function is a function with two sort of arguments $\bar{x}$ and $\bar{X}$. A function $F(\bar{x}, \bar{X})$ is a string function if its range is in strings. A function $f(\bar{x}, \bar{X})$ is a number function if its range is in numbers. As usual, we denote strings functions by uppercase letters and number functions by lower case letter. Predicates are identified with 0-1 valued number functions.

In the two-sort formalization, number variables play subsidiary roles, so in defining complexity classes, their measures are with respect to the length of string parameters.

Let $R(x_1, \ldots, x_k, X_1, \ldots, X_n)$ be a two-sort relation. As in [3], number variables $x_1, \ldots, x_k$ are presented in unary and string variables $X_1, \ldots, X_n$ are in binary form.

We define two sort complexity classes LOGCFL and LOGDCFL as follows:

▶ **Definition 1.** LOGCFL is the class of predicates which are logspace reducible to some context-free languages. LOGDCFL is the class of predicates which are logspace reducible to some deterministic context-free languages.

2.2 Two-sort theories

The language L_2 of two sort theories contains the symbols $Z(x)$, $x+y$, $x \cdot y$, $|X|$, $x=y$ and $x \in Y$. We also write $x \in Y$ as $Y(x)$. We use two sort of quantifiers; number quantifiers $(\forall x)$ and $(\exists x)$ and string quantifiers $(\forall X)$ and $(\exists X)$. Bounded number quantifiers are of the form $(\forall x < t)$ and $(\exists x < t)$. String bounded quantifiers are of the form

$$(\forall X < t)\varphi(X) \equiv (\forall X)(|X| < t \to \varphi(X)), \text{ and } (\exists X < t)\varphi(X) \equiv (\exists X)(|X| < t \wedge \varphi(X)).$$

An L_2 formula is called bounded if all its quantifiers are either bounded number quantifiers or bounded string quantifiers. The class Σ_0^B consists of L_2 formulae whose quantifiers are bounded quantifiers only. The class Σ_1^B consists of L_2 formulae whose quantifiers are bounded quantifiers, positive occurrences of existential bounded string quantifiers, and negative occurrences of universal bounded string quantifiers.

Our base theory is the following theory:

▶ **Definition 2.** The L_2 theory V^0 consists of the following axioms:

- $BASIC_2$: finite number of defining axioms for symbols in L_2,
- extensionality axiom : $X = Y \leftrightarrow (|X| = |Y| \wedge (\forall i < |X|)(X(i) \leftrightarrow Y(i))$,
- Σ_0^B-COMP : $(\exists X)(\forall y < t)(X(y) \leftrightarrow \varphi(y))$, where $\varphi \in \Sigma_0^B$ does not contain X as a free variable.

▶ **Definition 3.** A function $F(\bar{x}, \bar{X})$ is Σ_1^B definable in a L_2-theory T if there exists a Σ_1^B formula $\varphi(\bar{x}, \bar{X}, Y)$ such that

- $T \vdash (\forall \bar{x})(\forall \bar{X})(\exists! Y)\varphi(\bar{x}, \bar{X}, Y)$, and
- $Y = F(\bar{x}, \bar{X}) \leftrightarrow \varphi(\bar{x}, \bar{X}, Y)$ holds in the standard model.

▶ **Theorem 4.** *A function is Σ_1^B definable in V^0 if and only if it is computable in AC^0.*

The following property is also well-known and useful:

▶ **Theorem 5.** *Let F be a Σ_1^B definable function of V^0 and $V^0(F)$ be the theory V^0 extended by the function symbol for F together with its defining axioms. Then $V^0(F)$ is an conservative extension of V^0.*

So we can use any Σ_1^B definable function in V^0 without increasing its strength. In particular, the following functions are Σ_1^B definable in V^0 which will be used in elsewhere in this paper:

- The number pairing function : $\langle x, y \rangle = \frac{(x+y+1)(x+y)}{2}$.
- The number μ-operator :

$$\mu_{z<y}\varphi(x) = \begin{cases} \text{the least } x < y \text{ with } \varphi(x) & \text{if exists,} \\ 0 & \text{otherwise.} \end{cases}$$

- The value of X at y : $(X)^y = \mu_{y<a} X(\langle y, z \rangle)$ for $|X| < \langle a, b \rangle$.
- The row function : $Z^{[x]}(i) \leftrightarrow (i < |Z| \wedge Z(x, i))$.

The main tool of this paper is the method for constructing theories for subclasses of P using complete problems which is developed by Cook and Nguyen. Details of this method can be found in [3], so we briefly overview the argument.

Suppose a function F has a Σ_0^B graph as

$$Y = F(X) \leftrightarrow (|Y| \leq t \wedge \delta_F(X < Y))$$

for some L_2 term t and Σ_0^B formula δ_F. Suppose further that V^0 proves the uniqueness of the value of F. Let C be the class of relations which are AC^0 reducible to F. In this case, we define a theory for C as follows:

▶ **Definition 6.** The L_2-theory **VC** is axiomatized by the axioms of V^0 and the following axiom:

$$(\exists Y \le b)(\forall i < b)\delta_F(X^{[i]}, Y^{[i]}).$$

Below we define theories with defining axiom for the function F as above and the equivalence to the theory **VC** is ensured by showing that the aggregate function is Σ_1^B definable within the theory.

▶ **Definition 7** (Aggregate Function)**.** Suppose that $F(\bar{x}, \bar{X})$ is such that for some L_2 term t

$$|F(\bar{x}, \bar{X})| \le t(\bar{x}, \bar{X}).$$

Then $F^*(b, \bar{Z}, \bar{X})$ is a function such that

$$|F^*(b, \bar{Z}, \bar{X})| \le \langle b, t(|\bar{Z}|, \bar{X})\rangle$$

and

$$F^*(b, \bar{Z}, \bar{X})(w) \leftrightarrow (\exists i < b)(\exists v < t)(w = \langle i, v\rangle \wedge F^*(b, (Z_1)^i, \dots, (Z_k)^i, X_1^{[i]}, \dots, X_n^{[i]})(v).$$

Finally, a direct connection between **VC** and FC is established:

▶ **Theorem 8** (Cook-Nguyen)**.** *A function is Σ_1^B definable in* **VC** *if and only if it is in* FC.

We also use universal conservative extension $\widehat{\mathbf{VC}}$ of **VC** which is defined as follows: let $\overline{V^0}$ be the theory V^0 extended by function symbols for all AC^0 functions together with their defining axioms. Let $\delta'_F(X, Y)$ be the quantifier free defining axiom for F which is equivalent to $\delta_F(X, Y)$ in $\overline{V^0}$ and define the function

$$Y = F'(X) \leftrightarrow (|Y| \le t \wedge \delta'_F(X, Y)). \quad (*)$$

▶ **Definition 9.** $\widehat{\mathbf{VC}}$ is the universal theory over language of $\overline{V^0}$ plus F' whose axioms are those for $\overline{V^0}$ and $(*)$.

Remark. Our theories defined below are slightly different from **VC** in that we add axiom scheme rather than a single axiom. Nevertheless, the witnessing and definability works by an almost similar argument.

3 A theory for LOGDCFL

We will define the theory **VLDCFL** by formalizing the concept of polynomial proof tree of polynomial size multiplex-select circuits.

A multiplex select circuit is a circuit whose only gates are multiplex select gates. Inputs to a multiplex select gate are grouped into a bundle of $k \in O(\log n)$ steering bits and 2^k equal size bundles of $l \in O(\log n)$ data bits. The gate outputs the value of the data input bundle d_i if the value of the steering input bundle is the binary expression of i. For detailed definition, see [7].

A proof tree of a circuit C on input X is a tree $PT(C, X)$ define inductively as follows:

- if C consists of a single gate then $PT(C, X)$ consists of its steering input bundle and the data bundle selected by the steering input on X.
- if C consists of more than two gates and g is the output gate of C then $PT(C, X)$ is defined as the tree rooted at g with subtrees $PT(C_s, X)$ and $PT(C_d, X)$ where C_s is the subcircuit with output at the steering input of g and C_d is the subcircuit with output at the data input selected by s on X.

Note that the computation of multiplex select circuits are deterministic in the sense that there exists an unique proof tree for each input.

McKenzie et.al. [7] showed that multiplex select circuits characterize the class LOGDCFL.

▶ **Theorem 10** (McKenzie et.al.). *A predicate is in* LOGDCFL *if and only if it is decidable by AC^0-uniform poly-size family of multiplex select circuits with polynomial proof tree size.*

We code a multiplex select gate by a triple $\langle g, c, k\rangle$ where g is the index of its steering input, k is the number of data inputs, and $c + j$ is the index of j-th data input for $j \leq k$.

As usual, a circuit is coded by a two-dimensional array but with semantics which differs from that of Boolean circuits. Recall that a non-input gate in a multiplex-select circuit has two types of inputs; one steering input and 2^l data inputs where l is the length of the steering input bundle. So we define AC^0 functions which specifies these inputs.

Let E be a two-dimensional array with $|E| = \langle n, n\rangle$. A gate $x < n$ with no input is regarded as an input data bundle with the data x. Otherwise, x is a non-input gate and we define its steering input $s(x, E)$ and the first data input $c(x, E)$ as follows:

$$s(x, E) = \mu y < x E(x, y),$$

$$c(x, E) = \begin{cases} \mu y < x(E(x, y) \wedge s(x, E)) < y & \text{if it exists,} \\ s(x, E) & \text{otherwise.} \end{cases}$$

We omit the parameter E if it is clear from the context.

The output of a multiplex select gate is given by the following rule:

> for a gate x, if it receives an input j from its steering input $s(x)$ then it outputs the input from the data input of index $\min(c(x) + j, x - 1)$.

We use the notion of degrees of gates in a multiplex-select circuit which is defined as:

▶ **Definition 11.** Let E be a circuit such that $|E| = \langle n, n\rangle$ and $x < n$. We define the degree $\deg(x, E)$ recursively as

$$\deg(x, E) = \begin{cases} 1 \quad \text{if } x \text{ is an input gate,} \\ \deg(s(x)) + \deg(\min(c(x) + j, x - 1)) \\ \quad\quad \text{if } x \text{ is an non-input and } s(x) \text{ outputs } j. \end{cases}$$

It is not difficult to see that a circuit family has polynomial size proof trees if and only if it has polynomial degrees.

We define an axiom expressing the concept of multiplex-select circuits with polynomial degrees. The idea is to code the computation of a given circuit by a string Z such that a

- if $degree(x)$ is bounded by a given polynomial $p(n)$ then $(Z)^x = \langle v_x, \deg(x)\rangle$ where v_x is the output of x, and
- otherwise $(Z)^x = \langle n, p(n)\rangle$.

Formally, define the L_2 formula $MSCDeg(n, x, E, Z)$ as the conjunction of the following formulae:

- x is an input gate: $(\forall y < x)\neg E(x, y) \to (Z)^x = \langle x, 1\rangle$,
- x is a non-input gate having a degree less than $p(n)$:

$$(\exists y < x)E(x, y) \to (Z)_1^{s(x)} + (Z)_1^{\min(c(x)+(Z)_0^{s(x)}, x-1)} < p(n) - 1 \wedge$$
$$(Z)^x = \langle (Z)_0^{\min(c(x)+(Z)_0^{s(x)}, x-1)}, (Z)_1^{s(x)} + (Z)_1^{\min(c(x)+(Z)_0^{s(x)}, x-1)}\rangle,$$

- x is an non-input gate having a degree greater than $p(n) - 1$:

$$(\exists y < x)E(x, y) \to (Z)_1^{s(x)} + (Z)_1^{\min(c(x)+(Z)_0^{s(x)}, x-1)} \geq p(n) \wedge (Z)^x = \langle n, p(n)\rangle.$$

For a L_2-term $p(n)$ we define the formula

$$MSCDeg\text{-}COMP_p : (\forall n)(\forall E < \langle n, n\rangle)(\exists Z\langle n, \langle n, p(n)\rangle\rangle)(\forall x < n)MSCDeg(n, x, E, Z).$$

▶ **Definition 12.** The L_2-theory **V**LDCFL is V^0 extended by $MSCDeg\text{-}COMP_p$ for each $p \in L_2$}.

Remark. We have made several simplifications in the formalization of multiplex-select circuits which is still general enough to capture the original ones in the sense that we can effectively compute a code $E(C, X)$ from the original circuit code C and an input X. Firstly, we can convert an $O(\log |X|)$ input bit bundle into a number representing a gate index using AC^0 function. The original definition allows each bundle to be extended by constant bits which also can be computed in AC^0.

Finally, the restriction of date inputs bit to consecutive ones as $c(x), \ldots c(x) + j$ can express the original circuit as follows. Let d be the i-th data input to x. We introduce a "copy" gate C_d of d having a single bit steering input and a single data input from d. Then we let the gate position of C_d to be $c(x) + i$.

▶ **Theorem 13.** *A function is Σ_1^B definable in* **V**LDCFL *if and only if it is* LOGDCFL-*computable.*

To prove Theorem 13, we use the formalization as in Cook-Nguyen's Book [3]. First we show the uniqueness of proof trees.

▶ **Lemma 14.** **V**LDCFL *proves the following:*

$$(\forall Z_0, Z_1 < n)[((\forall x < n)MSCDeg_p(n, x, E, Z_0) \wedge (\forall x < n)MSCDeg_p(n, x, E, Z_1)) \\ \to Z_0 = Z_1].$$

(Proof). Since $MSCDeg_p$ is a Σ_0^B formula, this follows from an Σ_0^B-IND instance which is available in V^0. The most essential part is to show that the aggregate function for $MSCDeg_p$ is Σ_1^B-definable in **V**LDCFL.

▶ **Lemma 15.** *The aggregate function for $MSCDeg\text{-}COMP_p$ defined by the formula*

$$(\forall n)(\forall E < \langle\langle n, n\rangle, b\rangle)(\exists Y)(\forall i < b)(\forall x < n)MSCDeg_p(n, x, E^{[i]}, Y^{[i]})$$

is Σ_1^B-definable in **V**LDCFL.

The proof is more or less identical to that for VP (Lemma VIII 1.10 in [3]).

Now, Theorem 13 follows from the following:

▶ **Lemma 16.** *The FAC^0 closure of functions F_p with defining axiom*

$$Y = F_p(n, E) \Leftrightarrow |Y| \leq t_p \wedge (\forall x < n)MSCDeg_p(n, x, E, Y)$$

for $p \in L_2$, denoted by $FAC^0(DegP)$, is identical to $\mathcal{F}$LOGDCFL.

(Proof). To show that $FAC^0(DegP) \subseteq \mathcal{F}\textsc{logdcfl}$ we remark that $F_p \in \mathcal{F}\textsc{logdcfl}$ for all $p \in L_2$.

For the converse inclusion, it suffices to show that any bit in $F \in \mathcal{F}\textsc{logdcfl}$ can be computed using some F_p together with AC^0 operations.

Using the technique developed by Cook and Nguyen [3], we can prove Theorem 13 from Lemmata 14, 15 and 16.

Since LOGDCFL contains L, we expect that the same inclusion holds for corresponding theories. The theory **VL** is axiomatized by V^0 together with the following axiom:

$$PATH \equiv Unique(a, E) \to (\exists P \leq \langle a, a \rangle)\delta_{PATH}(a, E, P),$$

where

$$Unique(a, E) \equiv (\forall x < a)(\exists! y < a)E(x, y),$$
$$\delta_{PATH}(a, E, P) \equiv (P)^0 = 0 \land (\forall v < a)(E((P)^v, (P)^{v+1}) \land (P)^{v+1} < a).$$

▶ **Theorem 17.** $\mathbf{V}\textsc{ldcfl}$ *contains* **VL**.

(Proof). It suffices to show that $\mathbf{V}\textsc{ldcfl}$ proves $PATH$. Let E satisfy $Unique(a, E)$. For each $x < a$, we denote the unique y such that $E(x, y)$ by $d(x)$. We first design a multiplex select gate such that if the steering input is x then it outputs $d(x)$. This gate is easily constructed by setting its data inputs as $d(0), \ldots, d(a-1)$. Then we connect copies of this gate in a sequential manner, that is the first gate receives its steering input from x, the second gate from the first gate, and so on. Furthermore, all gates receive data inputs from $d(0), \ldots, d(a-1)$.

Suppose a node $b < n$ is reachable from a by a path of length $k \leq n$. Then it is easy to see that there exists a gate in this circuit with label $\langle b, 2k+1 \rangle$.

The above argument can be formalized in $\mathbf{V}\textsc{ldcfl}$, that is, for $p(n) \leq 2n+1$, we have

$$(\exists Z)(\forall x < n)MSCDeg_p(n, x, E', Z)$$

where E' is the code for the above circuit. Furthermore, the witness P in $PATH$ can be constructed from Z using Σ_0^B-COMP.

4 A theory for LOGCFL

We now turn to show that the axiomatization of proof tree size concept also captures the class LOGCFL. First we recall the following circuit characterization of the class:

▶ **Theorem 18** (Venkateswaran [10]). LOGCFL *is the class of predicates which are computable by polynomial size circuits with polynomial proof trees. This equivalence is true even if we restrict circuits to semi-unbounded fan-in.*

(Proof). The former statement is due to [10]. So we prove the latter part.

It is readily seen that a circuit can be transformed to an equivalent semi-unbounded fan-in circuit with polynomial increase in size by the transformation of unbounded fan-in AND gates by a fan-in two subcircuits of $O(\log n)$ depth. It remains to show that the degree of the resulting circuit also has polynomial increase. For this, it suffices to show that the replacement of unbounded fan-in AND-gates by subcircuits yields polynomial increase in the degree,that is the number of nodes in a proof tree.

Let g be an AND-gate in the original circuit having inputs from $h_0, \ldots, h_{k-1}$ and g' be the gate on the top of the subcircuit which is a substitute for g. It is not difficult to see that $\deg(g') = \deg(g) + k$. More strictly, this is proved by induction on the depth of gates.

The advantage of the semi-unbounded fan-in restriction is that we do not need vector summation to compute degrees. Thus we can axiomatize our theory based on V^0 rather than $\mathbf{VTC^0}$.

Now we define an axiom expressing polynomial degrees in a similar manner as **VP** of Cook-Nguyen [3]. First we will give an intuitive idea: Let E be a two-dimensional array coding a circuit. The input-output relation of E differs from that for multiplex select circuits.

Let $G(x)$ be an unary predicate determining the type of a gate x so that x is an AND gate if $G(x)$ and an OR gate otherwise. If $G(x)$ holds then x receives inputs from two gates defined by

$$\begin{aligned} c_0(x) &= \mu y < xE(x,y), \\ c_1(x) &= \mu y < x(E(x,y) \wedge c_0(x) < y). \end{aligned}$$

If $\neg G(x)$ then x receives inputs from all $y < x$ such that $E(x,y)$.

The computation of a circuit given by E and G as above is coded by a string Z such that

$$\begin{aligned} &(Z)^0 = p(n),\ (Z)^1 = 1, \\ &(Z)^x = \begin{cases} \deg(x) & \text{if } x \text{ outputs 1 and } \deg(x) < p(n), \\ p(n) & \text{otherwise} \end{cases} \end{aligned}$$

for $x \geq 2$.

Putting these altogether, we formally define

$$\begin{aligned} &MCVDeg_p(n, E, G, Z) \Leftrightarrow \\ &(Z)^0 = p(n) \wedge (Z)^1 = 1 \wedge \\ &(\forall x < n)(x \geq 2 \rightarrow \\ &(G(x) \wedge (((Z)^{c_0(x)} + (Z)^{c_1(x)} < p(n) - 1 \wedge (Z)^x = (Z)^{c_0(x)} + (Z)^{c_1(x)} + 1) \vee \\ &((Z)^{c_0(x)} + (Z)^{c_1(x)} \geq p(n) - 1 \wedge (Z)^x = p(n))) \vee \\ &(\neg G(x) \wedge ((\exists y < x)((Z)^y < p(n) - 1 \wedge E(x,y) \wedge (Z)^x = (Z)^y + 1) \vee \\ &((\forall y < x)(E(x,y) \rightarrow (Z)^y \geq p(n) - 1 \wedge (Z)^x = p(n)))). \end{aligned}$$

where $p \in L_2$.

▶ **Definition 19.** We define the L_2-theory $\mathbf{V}$LCFL as

$$V^0 + \{(\forall n)(\forall E < \langle n, n\rangle)(\forall G < n)(\exists Z < \langle n, p(n)\rangle MCVDeg_p(n, E, G, Z) : p \in L_2\}.$$

It is possible to define our theory using SAC^1 circuits which can be formalized by a single axiom. Nevertheless, we choose the above axiomatization as it is easier to formalize other computational concepts within the theory. First we show that our theories preserve the inclusion relation of corresponding complexity classes:

▶ **Theorem 20.** $\mathbf{V}$LCFL *proves* $CONN$*, thus it contains* **VNL***.*

(Proof). We argue in $\mathbf{V}$LCFL. Let $E < \langle n, n\rangle$ be given. We need to show that

$$(\exists Y < \langle a, a\rangle)(\delta_{CONN}(a, E, Y) \wedge Y(a, 1)).$$

To this end, we construct a graph $C < \langle\langle a, a\rangle, \langle a, a\rangle\rangle$ such that

$$C(0, x, 1, y) \leftrightarrow x = 0 \wedge E(0, y)$$

and

$$C(n, x, m, y) \leftrightarrow m = n + 1 \wedge E(x, y)$$

hold. That is, C has layers such that each column contains copies of all nodes in E. The edge relation of C is given so that the node x in i-th layer has an edge to the node y in $(i+1)$-st layer if $E(x,y)$ holds for $i > 0$. Furthermore, on 0-th layer, only the node 0 has an edge to the node y in the first layer if $E(0,y)$.

Now we show that a small modification of C yields a circuit deciding the axiom CONN for **VNL**. In the new circuit C', we have the first column containing 0 and 1, say $\langle 0,0\rangle$ and $\langle 1,0\rangle$, and the number of columns of C are incremented by 1 in C'. Thus the gate $\langle i,j\rangle$ in C corresponds to $\langle i,j+1\rangle$ in C'. Thus we define

$$C'(n,x,m,y) \leftrightarrow (n=0 \wedge x=1 \wedge m=1 \wedge y=0) \vee (C(n-1,x,m-1,y) \wedge n>0 \wedge m>0).$$

Furthermore, we let G be such that $(\forall n)(\forall x)G(n,x)$, that is, all gates in C' are AND gates.

Now let $p(n) = n$. Then we have $(\exists Z)MCVDeg_p(n,C',G,Z)$ and we have an AC^0 function F such that

$$MCVDeg_p(n,C',G,Z) \rightarrow \delta_{CONN}(n,E,F(Z)).$$

Also note that a proof tree of a gate $\langle i,x\rangle$ is a path from $\langle 0,1\rangle$ to $\langle i,x\rangle$. Thus we are done.

▶ **Theorem 21.** **V**LCFL *contains* **V**LDCFL.

(Proof). It suffices to show that **V**LCFL proves $MSCDeg\text{-}COMP_p$ for any $p \in L_2$. This is achieved by giving circuits simulating multiplex-select circuits which have polynomial size and polynomial proof size. Let g be an MS-gate with the steering input s and data inputs $d_0,\dots,d_l$. The Boolean circuit C_g simulating g can be obtained by building the following subcircuits:

- C^i_{sel} outputs 1 if the steering input s selects the data input d_i, and
- C^j_{out} outputs the bits of the j-th output bit of g.

Then C_g is defined as

$$\bigvee_{1\le i\le l} (C^i_{sel} \wedge C^i_{out}).$$

The subcircuit C^i_{sel} is defined as $\bigwedge_{1\le j\le l} l_j$ where

$$l_i = \begin{cases} s_j & \text{if the } j\text{-th bit of } i \text{ is } 1\\ \neg s_j & \text{otherwise.}\end{cases}$$

Then C^j_{out} is computed by the circuit

$$\bigvee_{1\le i\le l} (C^i_{sel} \wedge d^j_i).$$

It is not difficult to see that the above construction can be described by Σ^B_0 relation. So we have the Σ^B_0 description of the translation of the whole MS-circuit C by an equivalent Boolean circuit.

Although the above construction does not yields semi-unbounded fan-in circuits, it can be converted into an equivalent SAC^1 circuit.

It is also true that **V**LCFL is contained in the theory for AC^1 which is axiomatized by V^0 and the axiom

$$(\exists Y \le \langle |n|+1,n\rangle \delta_{LMCV}(n,|n|,E,G,I,Y)$$

where

$$\begin{aligned} \delta_{LMCV}(n,d,E,G,I,Y) \equiv\ & (\forall x<n)(\forall z<d)((Y(0,x)\leftrightarrow I(x))\wedge\\ (Y(z+1,x)\leftrightarrow\ & (G(z+1,x)\wedge(\forall u<n)(E(z,u,x)\rightarrow Y(z,u)))\vee\\ & (\neg G(z+1,x)\wedge(\exists u<n)(E(z,u,x)\wedge Y(z,u)))).\end{aligned}$$

▶ **Theorem 22.** Vlcfl *is contained in* $\mathbf{VAC^1}$.

Actually, we prove a stronger theorem stating that any polynomial size circuit family with polynomial proof tree size has an equivalent SAC^1 circuits.

▶ **Theorem 23.** *Let* $p, q \in L_2$. $\mathbf{VTC^0}$ *proves that for any circuit* C *with size* p *and proof tree size* q *there exists a semi-unbounded circuit* C' *such that for any input* X,

$$C \textit{ accepts } X \textit{in proof tree size } q \Leftrightarrow C' \textit{ accepts } X.$$

(Proof). We use the idea similar to realizable pairs of Ruzzo [8]. Let g be a gate of C and Γ be a set of nodes in C. W define $C_{g,\Gamma}$ to be a directed acyclic subgraph of C whose root is g and sinks are Γ. We say that $C_{g,\Gamma}$ is realizable within size p and proof tree size q if $size(C_{g,\Gamma}) \leq p$ and g has proof tree size $\leq q$ provided that all nonleaves of Γ are assigned the value 1.

It suffices to construct a semi-unbounded fan-in circuit deciding whether $C_{g,\Gamma}$ is realizable with size p and proof tree size q since $C = C_{g_0,\emptyset}$ is realizable if and only if C accepts the input where g_0 is the output gate of C. So we construct a recursive procedure $realize(C, p, q)$ which works as follows:

In each recursive step, $realize(C, p, q)$ splits C into two parts C_0 and C_1 each having approximately half size of C and recursively check whether $realize(C_0, p/2, i)$ and $realize(C_1, p/2, j)$ where $i + j = q$. This procedure is given as follows:

```
procedure realize(C,p,q);
begin
if $C$ consists of a single gate $g$ then if $g$ has the value 1 and $p, q \geq 1$
then ACCEPT else REJECT
else
guess $i$ with $0 < i < q$ and a gate $s$ in $C$;
$C_0$ := subcircuit of $C$ rooted at $s$; $C_1 := C \setminus C_0$ with $s$ replaced by 1;
check in parallel
$realize(C_0, p/2, i)$ AND $realize(C_1, p/2, q - i)$
end
```

We will show that this algorithm can be converted into logarithmic depth semi-unbounded fan-in circuits.

First we remark that checking whether $size(C) < p$ can be done by an NC^1 circuit using threshold circuits.

Since the number of the nestings of recursive calls is logarithmic in p, the total number of subcircuits C_0s and C_1s created along the execution of the procedure is bounded by polynomial. Thus the number of gates used in checking circuit sizes is polynomially bounded.

Now each recursive steps can be expressed by an AND of $size(C) < p$ and the OR of the following subroutines:

- the base case with a single node g,
- $realize(C_0, q/2, i)$ AND $realize(C_1, p/2, q - i)$ for $0 < i < q$ and all nodes in C.

The fan-in of this OR gate is $O(p \cdot q)$ which is polynomial. Furthermore, all AND gates required is bounded fan-in.

By recalling that the number of the nestings of recursive calls is $\log p = O(\log n)$, we conclude that the procedure $realize$ can be transformed into an SAC^1 circuit family.

It is not hard, though tedious, to show that the above construction can be demonstrated by Σ_0^B relation. To check the correctness of the construction, we need counting argument for the evaluation of circuit sizes. So the whole argument can be formalized in $\mathbf{VTC^0}$.

▶ **Corollary 24.** $\mathbf{V}$LCFL *and* V-Q^{SAC} *proves the same* L_2 *sentences.*

Now we show that $\mathbf{V}$LCFL corresponds to LOGCFL.

▶ **Theorem 25.** *A function is* Σ_1^B *definable in* $\mathbf{V}$LCFL *if and only if it is in* $\mathcal{F}$LOGCFL*.*

Note that the witness Z for $MCVDeg_p$ axiom is not always optimal in a sense that the degree of the output of Z is larger than $p(n)$ while there is another witness Z' whose output degree is less than or equal to $p(n)$. Nevertheless, we can avoid such undesirable cases by considering SAC^1 circuits.

▶ **Theorem 26** (Definability for $\mathbf{V}$LCFL)**.** *If a function of polynomial growth is bitwise computable in* LOGCFL *then it is* Σ_1^B *definable in* $\mathbf{V}$LCFL*.*

(Proof Sketch). In [6], it is shown that V-Q^{SAC} proves that SAC^1 is closed under complementation. By Corollary 24, this is also provable in $\mathbf{V}$LCFL.

Moreover, any SAC^1 circuit C is a polynomial size circuit having polynomial proof tree size such that for any input X and any Z witnessing $MCVP$ axiom for C, C outputs true if and only if the degree of the output gate is less than $p(|X|)$ for a given proof tree bound p.

Let $F(X)$ be a LOGCFL function with polynomial growth. To show that $\mathbf{V}$LCFL Σ_1^B defines F, it suffices to show that circuits can be combined into a multi output circuit. This can be done using circuits and their complementary circuits which is also an SAC^1 circuit provably in $\mathbf{V}$LCFL.

For witnessing we use the argument using the conservative universal extension $\widehat{\mathbf{V}\text{LCFL}}$ of $\mathbf{V}$LCFL.

▶ **Theorem 27** (Witnessing for $\mathbf{V}$LCFL)**.** *If a function is* Σ_1^B *definable in* $\mathbf{V}$LCFL *then it is of polynomial growth and bitwise computable in* LOGCFL*.*

(Proof). We define the universal conservative extension $\widehat{\mathbf{V}\text{LCFL}}$ by introducing for each $p \in L_2$ a function symbol F_p such that

$$F_p(n, E, G) = Z \leftrightarrow MCVDeg_p(n, E, G, Z).$$

Using Herbrand-style argument, we can show that provably total functions of $\widehat{\mathbf{V}\text{LCFL}}$ are in the AC^0 closure of $\{F_p : p \in L_2\}$. As LOGCFL is closed under AC^0 operations, it suffices to show that F_p is witnessed by a LOGCFL algorithm. In order to compute F_p we modify NAuxPDA machine simulating polynomial size circuits with polynomial proof trees as in [7]. The machine makes an depth-first search through a circuit and at each step, it computes the degree of the currently visiting gate using the work tape. This can be done in logarithmic space by expressing degrees in binary. Therefore, this gives an NAuxPDA computing F_p.

5 Finite axiomatizability

In this section we show that the theory $\mathbf{V}$LCFL is finitely axiomatizable. The idea is to formalize LOGCFL complete problems such as acyclic conjunctive query problem (Gottlob et.al. [4]) or word problem for finite groupoids (Bedard et.al. [1]).

First we modify the theory **VLCFL** so that it can argue about circuit families which have Σ_0^B direct connection languages. For an L_2-formula $\varphi(x, y)$ we define the string

$$E_\varphi(n)(x, y) \Leftrightarrow x < n \wedge y < n \wedge \varphi(x, y).$$

Similarly for an L_2-formula $\psi(x)$ we define the string

$$G_\psi(n)(x) \Leftrightarrow x < n \wedge \psi(x).$$

Then we define the theory

$$\mathbf{VLCFL}' \equiv V^0 + \{(\forall n)(\exists Z < \langle n, p(n)\rangle MCVDeg_p(n, E_\varphi(n), G_\psi(n), Z) : \varphi, \psi \in \Sigma_0^B, p \in L_2\}.$$

It is not difficult to see that

▶ **Lemma 28.** $\mathbf{VLCFL}' = \mathbf{VLCFL}$.

Using $\mathbf{VLCFL}'$ we have

▶ **Theorem 29.** $\mathbf{VLCFL}'$ *and hence* **VLCFL** *is finitely axiomatizable.*

(Proof Sketch). We can formalize the acyclic conjunctive query problem in L_2 and define the direct connection language φ and ψ of the polynomial size circuit family deciding it in proof tree size $p(n)$. Then in $\mathbf{VLCFL}'$ we can show that any circuit family with polynomial proof tree size can be reduced to it. Thus $\mathbf{VLCFL}'$ is finitely axiomatizable.
Remark. Although it is likely that the same argument holds for **VLDCFL**, we do not know whether there exists a complete problem for LOGDCFL.

6 Provability of the pumping lemma for CFLs

We now turn to the problem of the provability of the pumping lemma. Intuitively, the pumping lemma states that a sufficiently long word in a context-free language can be iterately "pumped up" to another word in the language. More precisely,

▶ **Theorem 30** (The pumping lemma for CFLs). *Let $G = (N, T, P, S)$ be a context-free grammar in Chomsky normal form. Suppose $W \in L(G)$ is such that $|W| \geq 2^{m-1} + 1$ where $m = |N|$. Then there exist $W_1, \ldots, W_5$ such that $W = W_1 \cdots W_5$, $|W_2W_4| \geq 1$, $|W_2W_3W_4| \leq 2^m$ and $W_1W_2^iW_3W_4^iW_5 \in L(G)$ for all $i \in \omega$.*

In order to formalize Theorem 32, we need expressions which refer to the exponentiation like $y \geq 2^x$. Recall that the relation $y = 2^x$ is Δ_0 definable in $I\Delta_0$ although the function is not total in the theory. Likewise, in L_2 we can define the exponentiation relation using a Σ_0^B formula. In particular, the assumption $|W| \geq 2^{s-1} + 1$ in Definition 31 below is expressible in the language of **VLCFL**.

Next we show how to code context-free grammars. Let $G = (N, T, P, S)$ be a CFG in Chomsky normal form. Since N and T are finite sets, We code it as $T = \{0, \ldots n-1\}$, $N = \{n, \ldots, n+s\}$, $S = n$. Recall that rules of context-free grammars in Chomsky normal form are either of the form $a \to bc$ or $a \to l$ for $a, b, c \in N$ and $l \in T$. We let $P \subseteq N \times (T \cup N^2)$ so that

$$P(a, x) \Leftrightarrow a \to x \text{ is a rule in } P.$$

We also write $a \to_P x$ instead of $P(a, x)$.

A parse tree T of height l is a layered binary tree such that each row $T^{[i]}$ is obtained from $T^{[i-1]}$ by applying rules in P to nonterminal symbols simultaneously. Each symbol $T^{[i]}[k]$

is of the form $\langle n, k' \rangle$ denoting that $u \leq n+s$ is a symbol assigned to the slot and k' is the index of its parent in $T^{[i-1]}$, where $X[y]$ is the value of X at y (denoted by $(X)^y$ in [3]).

More precisely, T is a parse tree if the following conditions hold:

- $T^{[0]}$ consists of a single nonterminal symbol a with $n \leq a \leq n+s$ and for all $i > 0$ and k, $T^{[i]}[k] = \langle u, k' \rangle$ for some $u \leq n+s$ and k'.
- If $T^{[i]}[k] = \langle u, k' \rangle$ with $u \geq n$ then $T^{[i]}[k+j] = \langle v, k' \rangle$ for $v \geq n$ and exactly one $j \in \{-1, 1\}$ such that $w \to_P uv$ for $w = (T^{[i-1]}[k'])_0$. Moreover, either one of the following holds:
  - there exists a unique k'' such that $T^{[i+1]}[k''] = \langle v, k \rangle$, $v < n$ and $u \to_P v$,
  - there exists a unique k'' such that $T^{[i+1]}[k''] = \langle v, k \rangle$, $T^{[i+1]}[k''+1] = \langle v', k \rangle$, $n \geq v, v' \geq n+s$ and $u \to_P vv'$.
- If $T^{[i]}[k] = \langle u, k' \rangle$ with $u < n$ then there exists a unique k'' such that $T^{[i+1]}[k''] = \langle v, k \rangle$ whenever $i \leq l$ where l is the number of rows in T. In addition, one of $(T^{[i-1]}[k'])_0 = u$ or $(T^{[i-1]}[k'])_0 \to u$ holds.
- For any i and $k < k'$ if $T^{[i+1]}[c]$ and $T^{[i+1]}[c]$ are children of $T^{[i]}[k]$ and $T^{[i]}[k']$ then $c < c'$.
- $(T^{[l]}[k]))_0 < n$ for all k.

Based on the above codings, it is easy, though tedious, to see that there exists a Σ_0^B formula

$$Parse(a, l, n, s.P, T, X) \equiv$$
$$T \text{ is a parse tree of length } l \text{ starting from } a \text{ which generates } X.$$

▶ **Definition 31.** We define $PL(n, s, P)$ to be the following formula:

$$(\forall X)(\exists T)Parse(s, l, n, n, T, P, X) \wedge |X| \geq 2^{s-1} + 1$$
$$\to (\exists X_1, X_2, X_3, X_4, X_5)(X = X_1X_2X_3X_4X_5 \wedge |X_2X_4| \geq 1 \wedge |X_2X_3X_4| \leq 2^s \wedge$$
$$(\forall i)(\exists l')(\exists T')Parse(a, l', n, n, P, T', X_1X_2^iX_3X_4^iX_5)).$$

As previously stated, this is provable in **V**LCFL, namely,

▶ **Theorem 32.** **V**LCFL *proves* $(\forall n)(\forall s)(\forall P)PL(n, s, P)$.

The main idea of the proof is to simulate CFGs by circuits so that the transformation of parse trees as in the standard proof of the pumping lemma as in [5] can be interpreted in terms of proof tree. Let (E, G) be a code of a circuit as in the axiom $MSCDeg_p$ but with n input gates whose values are unspecified and X be a string with $|X| = n$. Then we define (E, G) with input X by (E_X, G_X). Our main tool is the following:

▶ **Lemma 33.** *There exists an* AC^0 *function* $F(m, n, s, P) = (a, E, G)$ *which is* Σ_1^B *definable in* V^0 *and an* L_2*-term* $r(n, s, P, X)$ *such that* (E, G) *is a code of a semi-unbounded fan-in circuit with* m *unspecified input gates and* **V**LCFL *proves*

$$(\forall X)[(\exists T)Parse(s, l, n, n, P, T, X)$$
$$\leftrightarrow (\exists Z)(MCVP_r(a, E_X, G_X, Z) \wedge Z[a] < r(|X|, n, s, P))].$$

Moreover, there exists an AC^0 *function* $T(Z)$ *such that* V^0 *proves the following:*

$$(\forall X)[(MCVP_r(a, E_X, G_X, Z) \wedge Z[a] < r(|X|, n, s, P)) \to Parse(s, l, n, n, T(Z), P, X)].$$

(Proof Sketch). As in Example 1 in page 220 of Ruzzo [8], we construct a circuit checking whether an input string belongs to a given context-free language.

We construct a circuit which simulates a given CFG $G = (n, s, P)$ on input X. The idea is to introduce a gate which checks whether $a \Rightarrow^* x_i \cdots x_j$ for each nonterminal symbol a and a substring $x_i \cdots x_j$ of X.

In particular, the output gate is an OR gate checking whether $n \Rightarrow^* x_1 \cdots x_n$ (Recall that the start symbol is coded by n). This is done by introducing an AND gate for each rule $n \to ab$ and a partition $x_1 \cdots x_i / x_{i+1} \cdots x_n$. This AND gate gives an output to the output OR gate and receives inputs from gates checking $a \Rightarrow^* x_1 \cdots x_i$ and $b \Rightarrow^* x_{i+1} \cdots x_n$.

Gates which check whether $a \Rightarrow^* x_i \cdots x_j$ are defined similarly. Furthermore, at the input level, we connect subcircuits deciding that a given binary string is a code of a terminal symbol.

It is easy to see that the edge relation of this circuit is AC^0 computable. Also it can be seen that it has polynomial size and polynomial proof size and semi-unbounded fan-in.

We also remark that the above construction of the circuit ensures that its proof tree corresponds to a parse tree of $n \Rightarrow^* X$, so it is easy to construct an AC^0 function transforming a proof tree into a parse tree.

(Proof of Theorem 32) It suffices to show that the transformation of parse trees in the standard proof of the pumping lemma can be paraphrased in terms of proof trees in the circuit $F(n, s, P, X)$ of Lemma 33.

First we note that by Lemma 33, it follows that the predicate $(\exists T)Parse(s, l, n, n, T, P, X)$ is equivalent to an open formula in the universal conservative extension $\widehat{\mathbf{V}\text{LCFL}}$. So in the following we argue in $\widehat{\mathbf{V}\text{LCFL}}$.

Let $X \in L(G)$ be such that $|X| \geq 2^{s-1} + 1$. First we claim that there exists a partition $X_1, \ldots, X_5$ of X as in the proof of the pumping lemma.

By assumption, $(\exists T)Parse(s, l, n, n, P, T, X)$ holds for some l. So by Lemma 33 we have

$$(\exists Z)(MCVP_r(a, E_x, G_x, Z) \wedge Z[a] < r(m, n, s, P).$$

Since the circuit (E_X, G_X) is semi-unbounded fan-in, we can extract a proof tree from X which is binary branching.

Claim. There exists a path R from the output to some input in R which is longer than s.

(Proof of Claim). The construction of the circuit (E_X, G_X) ensures that any proof tree contains all bits of X. So by a combinatorial argument formalized in $\mathbf{V}$LCFL we have the claim.

From the above claim and the pigeonhole principle for $l(R) \to s$ where $l(R)$ is the length of the path R, we can choose two occurrences of some nonterminal symbol c.

In order to choose the position of such occurrences, we slightly modify the construction of the circuit in Lemma 33 so that the OR gate checking $a \Rightarrow^* x_i \cdots x_j$ is assigned a label a. Note that this modification is also AC^0 computable.

Let X' and X'' be substrings of X generated by the first and the second occurrences of c respectively. We can execute these substrings using the PATH axiom. Moreover, the subtree starting from a given node is also computable by PATH.

Let X_1, X_2, X_3, X_4, X_5 be such that $X = X_1 X' X_5$, $X' = X_2 X'' X_4$ and $X_3 = X''$. Define

$$pump(X, i) = pump(X_1, X_2, X_3, X_4, X_5, i) = X_1 X_2^i X_3 X_4^i X_5.$$

It is easy to see that $pump$ is Σ_1^B definable in V^0.

Now we show that $\widehat{\mathbf{V}\text{LCFL}}$ proves the following which we denote by $(\forall i)PT(i, X)$:

$$\begin{aligned}&(\forall i < |X|_m)(\exists Z)(MCVP_p(a_{pump(X,i)}, E_{pump(X,i)}, G_{pump(X,i)}, Z)\\ &\wedge Z[r(n, |s|, P, pump(i, X))] < p(|pump(i, X)|)).\end{aligned}$$

Let T' and T'' be the parse tree of X' and X'' respectively. From a parse tree T_i of $pump(X,i)$ we can construct a parse tree T_{i+1} of $pump(X,i+1)$ by replacing the parse tree of T'' by T'. Again we note that this operation is AC^0 computable. So we have

$$(\forall i)(PT(i,X) \to PT(i+1,X)).$$

It is easy to see that $PT(0,X)$. Recall that $PT(i,X)$ is equivalent to a Σ_0^B formula $\varphi(i,X)$ in the extended language and $\widehat{\mathbf{V}\text{LCFL}}$ proves Σ_0^B induction. So we conclude that $(\forall i)PT(i,X)$. Thus again by Lemma 33 we have

$$(\forall i < |X|_m)(\exists T)Parse(|s|,l,n,|s|,P,T,X).$$

which proves the theorem.

7 Concluding Remarks

It is an interesting problem to determine the lower bound on the provability of the pumping lemma for context-free languages. For our theories one might conjecture that **V**LDCFL does not prove the theorem $(\forall n)(\forall s)(\forall P)PL(n,s,P)$.

As we have pointed out in the proof of Theorem 32, we need several combinatorial tools which lie within the theory for NL in order to execute the proof of the pumping lemma. Also it is essential in the proof that context free languages are definable in the theory.

Based on these observations, we come to the problem of whether **V**LDCFL + $PATH$ proves $(\forall n)(\forall s)(\forall P)PL(n,s,P)$. Note that if it is not the case then **V**LDCFL + $PATH$ cannot define (nondeterministic) context-free languages which gives a strong evidence implying that both NL and LOGDCFL is strictly contained in LOGCFL.

References

1 F. Bédard, F. Lemieux and P. McKenzie, Extensions to Barrington's M-program model. Theoretical Computer Science, 107(1), 1993, pp.31–61.

2 A.Borodin, S.A.Cook, P.W.Dymond, W.L. Ruzzo, and M.Tompa, Two Applications of Inductive Counting for Complementation Problems. SIAM J. Comput. 18, 1989, pp. 559-578

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

4 G. Gottlob, N. Leone and F. Scarcello, The complexity of acyclic conjunctive queries. Journal of the ACM 48(3) 2001, pp. 431–498.

5 J. E. Hopcroft and J. D. Ullman, Introduction to automata theory, languages and computation. Addison-Wesley, 1979.

6 S. Kuroda, Generalized Quantifier and a Bounded Arithmetic theory for LOGCFL. Archive for Mathematical Logic, 46(5-6), 2007, pp489-516.

7 P. McKenzie, K. Reinhardt and V. Vinay, Circuits and Context-Free languages. Computing and Combinatorics. Lecture Notes in Computer Science, 1627, 1999, pp.194-203

8 W.L. Ruzzo, Tree-size bounded alternation. Journal of Computer and System Sciences, 21(2), 1980 pp.139–154.

9 I. Sudborough, On the tape complexity of deterministic context-free language, Journal of ACM, 25(3), 1978, pp.405–414.

10 H. Venkateswaran, Properties that characterize LOGCFL. Journal of Computer and System Sciences, 43(2), 1991, pp.380–404.

Isomorphisms of scattered automatic linear orders

Dietrich Kuske

Institut für Theoretische Informatik, Technische Universität Ilmenau, Germany

Abstract

We prove the undecidability of the existence of an isomorphism between scattered tree-automatic linear orders as well as the existence of automorphisms of scattered word automatic linear orders. For the existence of automatic automorphisms of word automatic linear orders, we determine the exact level of undecidability in the arithmetical hierarchy.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Automatic structures, isomorphism, automorphism

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.455

1 Introduction

Automatic structures form a class of computable structures for which a number of interesting problems is decidable: while, due to Rice's theorem, nothing is decidable about a computable structure (given as a tuple of Turing machines), validity of first-order sentences is decidable in automatic structures (given as a tuple of finite automata). This property of automatic structures was first observed and exploited in concrete settings by Büchi, by Elgot [12], and by Epstein et al. [13]. Hodgson [16] attempted a uniform treatment, but the systematic study really started with the work by Khoussainov and Nerode [19] and by Blumensath and Grädel [3, 4]. Over the last decade, a fair amount of results have been obtained, see e.g. the surveys [29, 1] as well as the list of open questions [20], for very recent results not covered by the mentioned articles, see e.g. [5, 11, 18, 17].

A rather basic question about two automatic structures is whether they are isomorphic. For ordinals and Boolean algebras, this problem was shown to be decidable together with a characterisation of the word-automatic members of these classes of structures. On the other hand, already Blumensath and Grädel [4] observed that this problem is undecidable in general. In [21], it is shown that the isomorphism problem is Σ^1_1-complete; a direct interpretation yields the same result for successor trees, for undirected graphs, for commutative monoids, for partial orders (of height 2), and for lattices (of height 4) [27]. Rubin [28] shows that the isomorphism problem for locally finite graphs is complete for Π^0_3. In [24], we show in particular that also the isomorphism problems of order trees and of linear orders are Σ^1_1-complete. For the handling of linear orders, our arguments rely heavily on "shuffle sums". Consequently, we construct linear orders that contain a copy of the rational line (a linear order not containing the rational line is called scattered, i.e., our result is shown for non-scattered linear orders). This is unavoidable since we also show that the isomorphism problem for scattered and word-automatic linear orders is reducible to true arithmetic (i.e., the first-order theory of $(\mathbb{N}, +, \cdot)$) and therefore much "simpler" than the isomorphism problem for arbitrary linear orders. But it is still conceivable that the isomorphism problem for scattered linear orders is decidable.

In this paper, we deal with automatic scattered linear orders. In particular, we prove the following three results:

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 455–469
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(1) There is a scattered linear order whose set of tree-automatic presentations is Π^0_1-hard (i.e. one can reduce the complement of the halting problem to this problem). This holds even if we fix the order relation on the set of all trees (Theorem 13). Hence also the isomorphism problem for tree-automatic scattered linear orders is Π^0_1-hard (Corollary 14).

(2) The existence of a non-trivial automorphism of a word-automatic scattered linear order is Σ^0_1-hard (i.e. the halting problem reduces to this problem, Corollary 6). Again, this holds even if we fix the linear order on the set of all words (Theorem 5). The existence of an automatic non-trivial automorphism is Σ^0_1-complete.

For regular languages ordered lexicographically, the existence of a non-trivial automorphism is decidable (Corollary 2), but it becomes undecidable for deterministic context-free languages (Theorem 8).

(3) The existence of a non-trivial automorphism of a tree-automatic scattered linear order is Σ^0_2-hard (i.e., one can reduce the set of Turing machines that accept a finite language to this problem, Theorem 17).

The proof of (2) uses an encoding of polynomials similarly to [24] but avoids the use of shuffle sums. The technique for proving (1) and (3) is genuinely new: One can understand a weighted automaton over the semiring $(\mathbb{N} \cup \{-\infty\}; \max, +)$ as a classical automaton with a partition of the set of transitions into two sets T_0 and T_1. The behavior of such a weighted automaton assigns numbers to words w, namely the maximal number of transitions from T_1 in an accepting run on the word w. Krob [23] showed that the equivalence problem for such weighted automata is Π^0_1-complete. The hardness results from (1) are based on a sharpening of Krob's result that can be found at [10]: there is a fixed weighted automaton such that the set of equivalent weighted automata is Π^0_1-hard (and therefore undecidable). A closer analysis of this proof, together with the techniques for proving (1) and (2), finally yields (3).

These results show that the existence of isomorphisms and of automorphisms is nontrivial for scattered linear orders that are described by word and tree automata, resp.

A complete version of this extended abstract can be found as arXiv:1204.5653.

2 Preliminaries

2.1 Tree and word automatic structures

Let Σ be some alphabet. A *Σ-tree* or just a *tree* is a finite partial mapping $t\colon \{0,1\}^* \to \Sigma$ such that $u0 \in \mathrm{dom}(t)$ implies $u \in \mathrm{dom}(t)$, and $u1 \in \mathrm{dom}(t)$ implies $u0 \in \mathrm{dom}(t)$ (note that we allow the empty tree $\emptyset$ with $\mathrm{dom}(\emptyset) = \emptyset$). A *(bottom up) tree automaton* is a tuple $\mathcal{A} = (Q, \iota, \Delta, F)$ where Q is a finite set of *states*, ι is the initial state, $\Delta \subseteq Q \times \Sigma \times Q^2$ is the *transition relation*, and $F \subseteq Q$ is the set of *final states*. A *run* of the tree automaton $\mathcal{A}$ on the tree t is a mapping $\rho\colon \mathrm{dom}(t) \to Q$ such that

$$(\rho(u), t(u), \rho'(u0), \rho'(u1)) \in \Delta \text{ with } \rho'(v) = \begin{cases} \rho(v) & \text{for } v \in \mathrm{dom}(t) \\ \iota & \text{otherwise} \end{cases}$$

holds for all $u \in \mathrm{dom}(t)$. The run ρ is *accepting* if $\rho'(\varepsilon) \in F$. The *language of the tree automaton* $\mathcal{A}$ is the set $L(\mathcal{A})$ of all trees t that admit an accepting run of $\mathcal{A}$ on t. A set L of trees is *regular* if there exists a tree automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$.

It is convenient to understand a *word* as a tree t with $\mathrm{dom}(t) \subseteq 0^*$ (then $t(\varepsilon)$ is the first letter of the word). Nevertheless, we will use standard notation for words like uv for the concatenation or ε for the empty word. A *word automaton* is a tree automaton

$\mathcal{A} = (Q, \iota, \Delta, F)$ with

$$(q, a, p_0, p_1) \in \Delta \implies p_1 = \iota \text{ and } q \neq \iota\,.$$

This condition ensures that word automata accept words, only.

Let $t_1, \ldots, t_n$ be trees and let $\# \notin \Sigma$. Then $\Sigma_\# = \Sigma \cup \{\#\}$ and the *convolution* $\otimes(t_1, t_2, \ldots, t_n)$ or $t_1 \otimes t_2 \otimes \cdots \otimes t_n$ is the $\Sigma_\#^n$-tree t with $\mathrm{dom}(t) = \bigcup_{1\le i\le n} \mathrm{dom}(t)$ and

$$t(u) = (t_1'(u), t_2'(u), \ldots, t_n'(u)) \text{ with } t_i'(u) = \begin{cases} t_i(u) & \text{if } u \in \mathrm{dom}(t_i) \\ \# & \text{otherwise.} \end{cases}$$

Note that the convolution of a tuple of words is a word, again. For an n-ary relation R on the set of all trees, we write $R^\otimes$ for the set of convolutions $\otimes(t_1, \ldots, t_n)$ with $(t_1, \ldots, t_n) \in R$. A relation R on the set of all trees is *automatic* if $R^\otimes$ is a regular tree language.

Let $\mathcal{S} = (L; R_1, \ldots, R_k)$ be a relational structure such that L is a set of trees. Then $\mathcal{S}$ is *tree-automatic* if the tree languages L and $R_i^\otimes$ for $1 \le i \le k$ are regular. The structure $\mathcal{S}$ is *word-automatic* if, in addition, L is a word language. A tuple of tree automata accepting L and $R_i^\otimes$ for $1 \le i \le k$ is called a *tree- or word-automatic presentation* of the structure $\mathcal{S}$.

2.2 Linear orders

For words u and v, we write $u \le_{\mathrm{pref}} v$ if u is a prefix of v. Let Σ be some set linearly ordered by $\le$. Then $\le_{\mathrm{lex}}$ denotes the lexicographic order on the set of words Σ^*: $u \le_{\mathrm{lex}} v$ if $u \le_{\mathrm{pref}} v$ or there are $x \in \Sigma^*$, $a, b \in \Sigma$ with $xa \le_{\mathrm{pref}} u$, $xb \le_{\mathrm{pref}} v$, and $a < b$. From the lexicographic order on Σ^*, we derive a linear order (denoted $\le_{\mathrm{lex}}^2$) on the set $\Sigma^* \otimes \Sigma^*$ of convolutions of words by

$$u \otimes v \le_{\mathrm{lex}}^2 u' \otimes v' :\iff u <_{\mathrm{lex}} u' \text{ or } u = u', v \le_{\mathrm{lex}} v'\,.$$

By $\le_{\mathrm{llex}}$, we denote the *length-lexicographic order* defined by $u \le_{\mathrm{llex}} v$ if $|u| < |v|$ or $|u| = |v|$ and $u \le_{\mathrm{lex}} v$. We extend this linear order $\le_{\mathrm{llex}}$ to trees. Let t be a tree. Then $t{\restriction}_{0^*}$ (more precisely, $t{\restriction}_{(0^*\cap\mathrm{dom}(t))}$) is a word that can be understood as the "main branch" of the tree t. For $u \in \{0,1\}^*$, let $t{\restriction}_u$ denote the subtree of t rooted at u (i.e., $\mathrm{dom}(t{\restriction}_u) = \{v \mid uv \in \mathrm{dom}(t)\}$ and $t{\restriction}_u(v) = t(uv)$ for $u \in \{0,1\}^*$ as well as $t{\restriction}_u = \emptyset$ for $u \notin \mathrm{dom}(t)$). Furthermore, $\tau(t)$ is the tuple of "side trees" of t, namely

$$\tau(t) = (t{\restriction}_{0^i1})_{0^i\in\mathrm{dom}(t)}\,.$$

We now define the extension $\le_{\mathrm{trees}}$ of $\le_{\mathrm{llex}}$ to trees setting $s <_{\mathrm{trees}} t$ if and only if

- $s{\restriction}_{0^*} <_{\mathrm{llex}} t{\restriction}_{0^*}$ or
- $s{\restriction}_{0^*} = t{\restriction}_{0^*}$ and there exists i (with $0^i \in \mathrm{dom}(s)$) such that $s{\restriction}_{0^j1} = t{\restriction}_{0^j1}$ for all $0 \le j < i$ and $s{\restriction}_{0^i1} <_{\mathrm{trees}} t{\restriction}_{0^i1}$.

In other words, we first compare the main branches of the trees s and t length-lexicographically and, if they are equal, compare the tuples $\tau(s)$ and $\tau(t)$ (length-)lexicographically (based on the extension $\le_{\mathrm{trees}}$ of the length-lexicographic order to trees). Since the "side trees" $t{\restriction}_{0^j1}$ of any tree t are properly smaller than the tree itself, the relation $\le_{\mathrm{trees}}$ is well-defined. Note that all the order relations $\le_{\mathrm{pref}}$, $\le_{\mathrm{lex}}$, $\le_{\mathrm{lex}}^2$, $\le_{\mathrm{llex}}$, and $\le_{\mathrm{trees}}$ are automatic.

A linear order $\mathcal{L}$ is *scattered* if there is no embedding of the rational line $(\mathbb{Q}; \le)$ into $\mathcal{L}$. Examples of scattered linear orders are the linear order of the non-negative integers ω, of the non-positive integers ω^*, or the linear order of size $n \in \mathbb{N}$ that we denote $\underline{n}$. If Σ is an alphabet with at least 2 letters, then $(\Sigma^*; \leq_{\text{llex}}) \cong \omega$ is scattered, too. On the other hand, if $a, b \in \Sigma$ are distinct letters, then $(\{aa, bb\}^* ab; \leq_{\text{lex}})$ is countably infinite, dense, and without endpoints, i.e., it is isomorphic to $(\mathbb{Q}; \leq)$ [6]. Hence $(\Sigma^*; \leq_{\text{lex}})$ is not scattered. From [22, Prop. 4.10], we know that the set of word-automatic presentations of scattered linear orders is decidable.

A linear order $\mathcal{L} = (L; \leq)$ is *rigid* if it does not admit any non-trivial automorphism, i.e., if the identity mapping $f\colon L \to L : x \mapsto x$ is the only automorphism of $\mathcal{L}$. The linear orders ω, ω^*, and $\underline{n}$ for $n \in \mathbb{N}$ are all rigid. On the other hand, $(\mathbb{Q}; \leq)$ as well as $(\mathbb{Z}; \leq)$ are not rigid.

Note that automorphisms of tree-automatic linear orders are binary relations on the set of all trees. Hence it makes sense to speak of an *automatic automorphism*. A tree-automatic structure is *automatically rigid* if it does not have any non-trivial automatic automorphisms.

Let $\mathcal{I} = (I; \leq)$ be a linear order and, for $i \in I$, let $\mathcal{L}_i = (L_i; \leq_i)$ be a linear order. Then the $\mathcal{I}$-*sum*[1] of these linear orders is defined by

$$\sum_{i \in (I;\leq)} \mathcal{L}_i = \left(\biguplus_{i \in I} L_i ; \bigcup_{i \in I} \leq_i \cup \bigcup_{\substack{i,j \in I \\ i<j}} (L_i \times L_j) \right) .$$

Intuitively, this $\mathcal{I}$-sum is obtained from the linear order $\mathcal{I}$ by replacing every element $i \in I$ by the linear order $(L_i; \leq_i)$.

For $\sum_{i \in \underline{2}} \mathcal{L}_i$, we simply write $\mathcal{L}_1 + \mathcal{L}_2$. If, for all $i \in I$, $\mathcal{L}_i = \mathcal{L}$, then we write $\mathcal{L} \cdot \mathcal{I}$ for $\sum_{i \in (I;\leq)} \mathcal{L}_i$. As an example, consider the linear order $\delta = \omega \cdot \omega^*$. This linear order is a descending chain of ascending chains. It will be used as "delimiter" in our constructions. Note that

$$\delta \cong (10^+1^+0; \leq_{\text{lex}})$$

where we assume $0 < 1$. Also for later use, we next define a regular set $D = \{t_{i,j} \mid i, j \geq 0\}$ of trees such that $\delta \cong (D; \leq_{\text{trees}})$. The alphabet of these trees will be the singleton $\{\$\}$ so that a tree is completely given by its domain. Then set inductively

$$\text{dom}(t_{0,j}) = \{\varepsilon, 0, 00\} \cup 1\{0^k \mid 0 \leq k \leq j\} \text{ and } \text{dom}(t_{i+1,j}) = \{\varepsilon, 0, 00\} \cup 01\, \text{dom}(t_{i,j})$$

The trees $t_{0,4}$ and $t_{2,2}$ are depicted in Figure 1 (left-arrows denote 0-sons, right-arrows denote 1-sons).

3 Automorphisms of linear orders on word languages

In this section, we consider linear orders on sets of words. The universe will be regular or context-free and the order will mainly be the lexicographic order $\leq_{\text{lex}}$ and its relative $\leq^2_{\text{lex}}$.

3.1 Regular universe and $\leq_{\text{lex}}$

In this section, we will show that the rigidity of a linear order $(L; \leq_{\text{lex}})$ with L regular is decidable. Even more, we show this decidability for regular words, i.e., word-automatic

[1] Shuffle sums mentioned in the introduction are special cases of this construction where $\mathcal{I} = (\mathbb{Q}; \leq)$ is the rational line and, for every $q \in \mathbb{Q}$, the set $\{r \in \mathbb{Q} \mid \mathcal{L}_q \cong \mathcal{L}_r\}$ is dense.

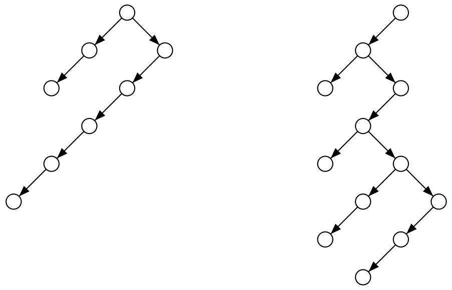


Figure 1 Two trees from D.

structures of the form $(L; \leq_{\text{lex}}, (P_a)_{a\in A})$ with L and $P_a \subseteq L$ regular languages of words for all $a \in A$. The study of these regular words was initiated by Courcelle [8] who was interested in the frontier of regular trees. Thomas proved that their isomorphism problem is decidable [30] (an alternative proof was given by Bloom and Ésik [2]) and the complexity of this problem was determined by Lohrey and Mathissen [25].

The outline of our proof is as follows (missing definitions are given below): Let ν be some regular word given by a tuple of word automata. The basic observation is that ν is rigid if and only if all its $\sim$-equivalence classes as well as the quotient $\nu/{\sim}$ are rigid. This allows to do induction since, after finitely many divisions of ν by $\sim$, we end up with a single $\sim$-equivalence class. The central problem therefore is to determine the $\sim$-equivalence classes (up to isomorphism, there are only finitely many) as well as the quotient $\nu/{\sim}$ and to decide whether a single $\sim$-equivalence class is rigid. For these calculations, we first represent the regular word ν by a "term" (Heilbrunner [15]) and then transform such terms using a technique by Bloom and Ésik [2].

An *extended word* is a labeled linear order with a finite set of labels. A *regular word* over the alphabet A is an extended word $(L; \leq, \lambda)$ with $\lambda \colon L \to A$ such that

- L and $\lambda^{-1}(a)$ for $a \in A$ are regular word languages over some alphabet Σ and
- $\leq$ is the lexicographic linear order $\leq_{\text{lex}}$.

A *term over A* uses constants $a \in A$ (standing for the extended word on $\underline{1}$ whose only element is labeled a) and the following operations:

- concatenation of words (denoted $\mu + \nu$)
- ω-power (denoted $\mu \cdot \omega$)
- ω^*-power (denoted $\mu \cdot \omega^*$)
- shuffle (denoted $[\nu_1, \nu_2, \ldots, \nu_k]^\eta$) for arbitrary $k \geq 1$.

The semantics of the concatenation, ω-power and ω^*-power generalize the corresponding operations for linear orders in the obvious way. To define the extended word $[\nu_1, \ldots, \nu_k]^\eta$, let $\lambda \colon \mathbb{Q} \to \{1, 2, \ldots, k\}$ be a mapping such that $\lambda^{-1}(i)$ is dense for all $1 \leq i \leq k$. Then set $\nu(q) = \nu_{\lambda(q)}$ for $q \in \mathbb{Q}$ and define

$$[\nu_1, \ldots, \nu_k]^\eta = \sum_{q\in(\mathbb{Q};\leq)} \nu(q)$$

as we did for linear orders. This operation is the extension of the shuffle sum from linear orders to extended words. It is well-defined in as far as the choice of the function λ does not influence the isomorphism type of the result. For a term t, let $|t|$ denote the extended word it describes.

Let $\nu = (L; \leq, \lambda)$ be an extended word. On the set L, we define an equivalence relation $\sim$ by $x \sim y$ if (where we assume $x \leq y$)

- the interval $[x, y]$ is finite or
- for any $x', y', z \in [x, y]$ with $x' < y'$, there exists $z' \in (x', y')$ with $\lambda(z) = \lambda(z')$.

As explained above, regular words with a single $\sim$-equivalence class are of particular importance in our proof. These regular words can be described by "primitive terms in normal form" (consult [2, Definitions 57] for their formal definition). Here, we list only the main properties of the set $D(A)$ of all these primitive terms in normal form:

- The set $D(A)$ is decidable (clear by its definition).
- If ν is a regular word with a single $\sim$-equivalence class, then there exists a term $t \in D(A)$ with $\nu \cong |t|$ [2, Lemma 58].
- If $t \in D(A)$, then $|t|$ has a single $\sim$-equivalence class (clear by the definition of $D(A)$).
- If $s, t \in D(A)$ with $|s| \cong |t|$, then $s = t$ [2, Proposition 62].

Let $\nu = (L; \leq_{\text{lex}}, \lambda)$ be a regular word. The equivalence classes with respect to $\sim$ are convex sets (or segments). Hence they can be ordered by

$$[x]_\sim <' [y]_\sim :\Longleftrightarrow x < y \text{ and } x \not\sim y$$

such that $(L/{\sim}; \leq')$ is a linear order. For $X \in L/{\sim}$, the restriction of ν to the equivalence class X is a regular word with a single $\sim$-equivalence class. Hence there exists a unique term $t_X \in D(A)$ with $|t| \cong \nu{\restriction}X$. Define $\lambda' \colon L/{\sim} \to D(A)$ by $X \mapsto t_X$. Then

$$c(\nu) = (L/{\sim}; \leq', \lambda')$$

is an extended word with possibly infinite alphabet.

▶ **Theorem 1.** *The set of rigid regular words ν (given as tuple of finite automata) is decidable.*

Proof. Let $\nu = (L; \leq_{\text{lex}}, \lambda)$ be a regular word given by finite automata that accept L and $\lambda^{-1}(a)$ for $a \in A$ (without loss of generality, we can assume $\varepsilon \notin L$). In a first step, we compute a term t with $|t| \cong \nu$ which is possible by Heilbrunner [15].

Using [2, Theorem 64], we construct a term $c(t)$ over $D(A)$ with $|c(t)| \cong c(|t|)$, in particular, $c(|t|)$ has a finite alphabet. From this term $c(t)$, we can extract the alphabet $D \subseteq D(A)$ of all symbols from $D(A)$ that appear in $c(t)$. Then we observe that $|t|$ has a nontrivial automorphism if and only if

- $c(|t|) = |c(t)|$ has a nontrivial automorphism or
- there exists a $\sim$-equivalence class X such that $|t|{\restriction}X$ has a non-trivial automorphism.

Note that $s \in D$ if and only if there exists a $\sim$-equivalence class X with $|t|{\restriction}X \cong |s|$. Hence the second item holds if and only if there exists $s \in D$ such that $|s|$ has a nontrivial automorphism – but this is the case if and only if s is of the form $u \cdot \omega^* + u \cdot \omega$ or $[u_1, \ldots, u_k]^\eta$. To decide whether $|c(t)|$ has a nontrivial automorphism, we call this process recursively. From [22], we observe that $c^n(|t|)$ is a singleton for some $n \in \mathbb{N}$, hence this recursive procedure stops eventually with $|t|$ a singleton. ◀

▶ **Corollary 2.** *The set of regular languages L such that $(L; \leq_{\text{lex}})$ is rigid, is decidable.*

3.2 Regular universe and $\leq_{\text{lex}}^2$

The situation changes completely when we move from the lexicographic order $\leq_{\text{lex}}$ to the linear order $\leq_{\text{lex}}^2$ since, as we will see, rigidity of $(L;\leq_{\text{lex}}^2)$ is undecidable for regular languages L.

Let $p,q\in\mathbb{N}[\bar{x}]$ be two polynomials with coefficients in $\mathbb{N}$ and variables among $\bar{x}=(x_1,\dots,x_k)$. Then define the scattered linear order

$$\mathcal{L}_{p,q}=\sum_{\bar{x}\in(\mathbb{N}^k;\leq_{\text{lex}})}\Big((\underline{p(\bar{x})}+\delta)\cdot\omega^*+(\underline{q(\bar{x})}+\delta)\cdot\omega\Big)\,.$$

This linear order $\mathcal{L}_{p,q}$ forms an ω-sequence of "blocks" of the form

$$B(m,n)=(\underline{m}+\delta)\cdot\omega^*+(\underline{n}+\delta)\cdot\omega$$

with $m,n\in\mathbb{N}$. Therefore, every automorphism of $\mathcal{L}_{p,q}$ has to map every block onto itself. In other words, $\mathcal{L}_{p,q}$ is rigid if and only if all these blocks are rigid. But $B(m,n)$ is rigid if and only if $m\neq n$. Hence we showed

$$\mathcal{L}_{p,q}\text{ is rigid }\iff\forall\bar{x}\in\mathbb{N}^k\colon p(\bar{x})\neq q(\bar{x})\,.$$

We now prove that $\mathcal{L}_{p,q}$ is word-automatic or, more specifically, we will construct a regular set $L\subseteq\{0,1\}^+\otimes\{0,1\}^+$ such that $\mathcal{L}_{p,q}\cong(L;\leq_{\text{lex}}^2)$ (see Lemma 4 below).

Let $\mathcal{A}=(Q,\iota,\Delta,F)$ be a word automaton over the alphabet Σ and let $w\in\Sigma^+$ be a word. Then $\text{Run}(\mathcal{A},w)$ is the set of all words over Δ of the form

$$(q_0,a_1,q_1,\iota)(q_1,a_2,q_2,\iota)\dots(q_{k-1},a_k,\iota,\iota)$$

with $w=a_1a_2\dots a_k$ and $q_0\in F$. These words encode the accepting runs of the word automaton $\mathcal{A}$ (recall that word automata are special bottom up tree automata which explains the unusual position of the initial and final states in the run). Furthermore, let $\text{Run}(\mathcal{A})=\bigcup_{w\in\Sigma^+}\text{Run}(\mathcal{A},w)$.

▶ **Lemma 3.** *From polynomials $p,q\in\mathbb{N}[x_1,\dots,x_k]$, one can construct an alphabet Σ and a regular language $K\subseteq\Sigma^+\otimes\Sigma^+$ such that $(K;\leq_{\text{lex}}^2)\cong\mathcal{L}_{p,q}$.*

If $\mathcal{L}_{p,q}$ has a non-trivial automorphism, then $(K;\leq_{\text{lex}}^2)$ has a non-trivial automatic automorphism.

Proof. Let p and q be polynomials from $\mathbb{N}[x_1,\dots,x_k]$. For $\bar{x}=(x_1,\dots,x_k)\in\mathbb{N}^k$, set

$$a^{\bar{x}}=a^{x_1}¢a^{x_2}¢\cdots¢a^{x_k}¢\in(a^*¢)^k\,.$$

Then, as in the proof of [24, Lemma 7], one can construct nondeterministic finite automata $\mathcal{A}_p=(Q_p,\iota_p,\Delta_p,F_p)$ and $\mathcal{A}_q=(Q_q,\iota_q,\Delta_q,F_q)$ with $L(\mathcal{A}_p),L(\mathcal{A}_q)\subseteq(a^*¢)^k$, such that, for $\bar{x}\in\mathbb{N}^k$, the NFA $\mathcal{A}_p$ has precisely $p(\bar{x})$ many accepting runs on the word $a^{\bar{x}}$, i.e., $|\text{Run}(\mathcal{A}_p,a^{\bar{x}})|=p(\bar{x})$, and similarly $|\text{Run}(\mathcal{A}_q,a^{\bar{x}})|=q(\bar{x})$. We will assume $\Delta_p\cap\Delta_q=\emptyset$.

Let $\Sigma=\{a,¢,0,1,2,3\}\cup\Delta_p\cup\Delta_q$ and let $K\subseteq\Sigma^+\otimes\Sigma^+$ be the union (for $\bar{x}\in\mathbb{N}^k$) of the languages

$$\Big(a^{\bar{x}}0^+1\otimes(\text{Run}(\mathcal{A}_p,a^{\bar{x}})\cup 32^+3^+2)\Big)\cup\Big(a^{\bar{x}}1^+0\otimes(\text{Run}(\mathcal{A}_q,a^{\bar{x}})\cup 32^+3^+2)\Big)\,.$$

We have to show that the language K is effectively regular. Here, the crucial point is the regularity of

$$\bigcup_{\bar{x}\in\mathbb{N}^k}a^{\bar{x}}0^+1\otimes\text{Run}(\mathcal{A}_p,a^{\bar{x}})=\left[\bigcup_{\bar{x}\in\mathbb{N}^k}a^{\bar{x}}\otimes\text{Run}(\mathcal{A}_p,a^{\bar{x}})\right]\cdot(0^+1\otimes\{\varepsilon\})$$

(this equality holds since $|w| = |W|$ for any $w \in (a^*\cent)^k$ and $W \in \mathrm{Run}(\mathcal{A}_p, w)$). But a word belongs to the language in square brackets if and only if it is the convolution of a word w from the regular language $(a^*\cent)^k$ and a run of the automaton $\mathcal{A}_p$ on this word w, a property that a finite automaton can check easily.

On the alphabet Σ, we fix a linear order $\leq$ such that $\Delta_p \cup \Delta_q < 0 < 1 < 2 < 3 < \cent < a$. Then $(K; \leq_{\mathrm{lex}}^2) \cong \mathcal{L}_{p,q}$.

Now suppose that $\mathcal{L}_{p,q}$ has a non-trivial automorphism. Then, as we saw above, there is $\bar{y} \in \mathbb{N}^k$ such that $p(\bar{y}) = q(\bar{y})$. Then $\mathcal{L}_{p,q}$ contains an interval isomorphic to a $\mathbb{Z}$-sequence of copies of $\underline{p(\bar{y})} + \delta \cong \underline{q(\bar{y})} + \delta$. Moving these blocks in $(K; \leq_{\mathrm{lex}}^2)$ upwards by 1 and fixing everything else in K gives a non-trivial automatic automorphism. ◀

▶ **Lemma 4.** *From polynomials $p, q \in \mathbb{N}[x_1, \dots, x_k]$, one can construct a regular language $L \subseteq \{0,1\}^+ \otimes \{0,1\}^+$ such that $(L; \leq_{\mathrm{lex}}^2) \cong \mathcal{L}_{p,q}$.*

If $\mathcal{L}_{p,q}$ has a non-trivial automorphism, then $(L; \leq_{\mathrm{lex}}^2)$ has a non-trivial automatic automorphism.

Proof. Let $p, q \in \mathbb{N}[x_1, \dots, x_k]$ be polynomials, let Σ be the alphabet and K the language from Lemma 3, and let $(\Sigma; \leq)$ be the sequence $\sigma_1 < \sigma_2 < \cdots < \sigma_\ell$. Furthermore, let g denote the monoid homomorphism from Σ^* to $\{0,1\}^*$ defined by $g(\sigma_i) = 1^i 0^{\ell - i}$ for $1 \leq i \leq \ell$. Now set $L = \{g(u) \otimes g(v) \mid u \otimes v \in K\}$. Then $u \otimes v \mapsto g(u) \otimes g(v)$ is an isomorphism from $(K; \leq_{\mathrm{lex}}^2)$ onto $(L; \leq_{\mathrm{lex}}^2)$. Since all the words $g(\sigma_i)$ have the same length, the language L is also regular. ◀

The set of pairs of polynomials $p, q \in \mathbb{N}[\bar{x}]$ with $p(\bar{y}) \neq q(\bar{y})$ for all $\bar{y} \in \mathbb{N}^k$ is Π_1^0-complete [26]. This allows to prove the following result:

▶ **Theorem 5.** **(i)** *The set of regular languages $L \subseteq \{0,1\}^+ \otimes \{0,1\}^+$ such that $(L; \leq_{\mathrm{lex}}^2)$ is rigid (is rigid and scattered, resp.), is Π_1^0-hard.*

(ii) *The set of regular languages $L \subseteq \{0,1\}^+ \otimes \{0,1\}^+$ such that $(L; \leq_{\mathrm{lex}}^2)$ is automatically rigid (automatically rigid and scattered, resp.) is Π_1^0-hard.*

▶ **Corollary 6.** **(i)** *The set of word-automatic presentations of rigid (rigid and scattered, resp.) linear orders is Π_1^0-hard.*

(ii) *The set of word-automatic presentations of automatically rigid (automatically rigid and scattered, resp.) linear orders is Π_1^0-complete.*

3.3 Context-free universe and $\leq_{\mathrm{lex}}$

Ésik initiated the investigation of linear orders of the form $(L; \leq_{\mathrm{lex}})$ where L is context-free. Density of such a linear order is undecidable [14], the isomorphism problem is Σ_1^1-complete [24] and their rank is bounded by ω^ω [7].

We will show that rigidity of $(L; \leq_{\mathrm{lex}})$ is undecidable for deterministic context-free languages L. The proof uses the linear order $\mathcal{L}_{p,q}$ and constructs a deterministic context-free language L' such that $(L'; \leq_{\mathrm{lex}}) \cong \mathcal{L}_{p,q}$. This construction is a variant of the construction in the proof of Lemma 3.

▶ **Lemma 7.** *From polynomials $p, q \in \mathbb{N}[x_1, \dots, x_k]$, one can construct a deterministic context-free language $L' \subseteq \{0,1\}^+$ such that $(L'; \leq_{\mathrm{lex}}) \cong \mathcal{L}_{p,q}$.*

Proof. Let $p, q \in \mathbb{N}[x_1, \dots, x_k]$ be polynomials, let Σ be the alphabet and let K be the language from Lemma 3. Then set

$$K' = \{u\$v^{rev} \mid u \otimes v \in K\}$$

where v^{rev} is the reversal of the word v. Then, from a deterministic finite automaton $\mathcal{A}$ accepting K^{rev}, one can construct a deterministic pushdown automaton accepting K'. Recall that there is a word u such that $u \otimes 3232 <^2_{\text{lex}} u \otimes 32232$ both belong to K. But we have $u\$2323 >_{\text{lex}} u\$23223$ (the same phenomenon can be observed with words $u \otimes \rho$ where ρ is a run of one of the two weighted automata). In other words, the obvious mapping $u \otimes v \mapsto u\$v^{rev}$ is no isomorphism from $(K; \leq^2_{\text{lex}})$ onto $(K'; \leq_{\text{lex}})$.

Note that the alphabet of K' is $\Sigma' = \{\$\} \cup \Sigma = \{\$, a, ¢, 0, 1, 2, 3\} \cup \Delta_p \cup \Delta_q$. We order Σ' in such a way that $\Delta_p \cup \Delta_q <' 0 <' 1 <' 3 <' 2 <' ¢ <' a <' \$$. Compared to the proof of Lemma 3, the order of 2 and 3 is inverted and \$ is made the new maximal element. The reason for this invertion is that now, we have $(32^+3^+2^{rev}; \leq'_{\text{lex}}) \cong (32^+3^+2; \leq_{\text{lex}}) \cong \delta$. Given this definition and observation, one can show $(K'; \leq'_{\text{lex}}) \cong (K; \leq^2_{\text{lex}})$ which was isomorphic to $\mathcal{L}_{p,q}$. The construction of $L' \subseteq \{0,1\}^+$ then follows the proof of Lemma 4. ◀

Now we obtain, in the same way that we proved Theorem 5, the following result.

▶ **Theorem 8.** *The set of context-free languages $L \subseteq \{0,1\}^+$ such that $(L; \leq_{\text{lex}})$ is rigid (is rigid and scattered, resp.), is Π^0_1-hard.*

4 Isomorphisms and automorphisms of linear orders on tree languages

In this section, we will show that the isomorphism of scattered and tree-automatic linear orders is undecidable. Furthermore, we will prove that the existence of a non-trivial automorphism in this case is Σ^0_2-hard. Both these results use (an improved version of) a theorem by Krob [23] from [10] that we discuss first.

4.1 Weighted automata and Minsky machines

A *weighted automaton* is a tuple $\mathcal{A} = (Q, \Sigma, \iota, \mu, F)$ where Q is the finite set of states, Σ is the alphabet, $\iota \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\mu\colon Q \times \Sigma \times Q \to \{-\infty, 0, 1\}$ is the weight function.

A *run* of $\mathcal{A}$ is a sequence $\rho = (q_0, a_1, q_1) \dots (q_{k-1}, a_k, q_k) \in \Delta^+$ such that $q_0 = \iota$, $\mu(q_{i-1}, a_i, q_i) \in \{0, 1\}$ for all $1 \leq i \leq k$, and $q_k \in F$. Its *label* is the word $a_1 \dots a_k \in \Sigma^+$ and its *weight* $\text{wt}(\rho)$ is the number of indices $i \in \{1, 2, \dots, k\}$ with $\mu(q_{i-1}, a_i, q_i) = 1$. By $\text{Run}(\mathcal{A}, w)$ we denote the set of runs labeled w and $\text{Run}(\mathcal{A})$ denotes the set of all runs of $\mathcal{A}$. The *behaviour* $||\mathcal{A}||$ of $\mathcal{A}$ is the function from Σ^+ to $\mathbb{N} \cup \{-\infty\}$ that maps the word w to the maximal weight of a run with label w.

▶ **Theorem 9** (cf. proof of [10, Theorem 8.6]). *From a Minsky machine (or two-counter automaton) $\mathcal{M}$, one can construct a weighted automaton $\mathcal{A}$ and a regular language $\text{CT}_{\text{reg}} \subseteq (\Sigma \cdot \square)^+$ such that, for any $m \in \mathbb{N}$, the following are equivalent:*

1. *m is not accepted by $\mathcal{M}$.*
2. *$||\mathcal{A}||(u) > \frac{1}{2}|u|$ for all $u \in \text{CT}_{\text{reg}}$ with $m = \max\{n \mid \$\square(a\square)^n \leq_{\text{pref}} u\}$.*

Furthermore, $||\mathcal{A}||(u) \in \mathbb{N}$ for all $u \in \text{CT}_{\text{reg}}$.

From the weighted automaton $\mathcal{A}$, one can then construct (cf. [9, 10]) weighted automata $\mathcal{A}_{\mathcal{M}}$ on the alphabet Σ and $\mathcal{B}_{\mathcal{M}}$ on the alphabet $\Sigma_{\#}^2$ such that

$$\begin{aligned}
||\mathcal{A}_{\mathcal{M}}||(u) &= \max(\lfloor \tfrac{|u|}{2} \rfloor + 1, ||\mathcal{A}||(u)) \text{ and} \\
||\mathcal{B}_{\mathcal{M}}||(x) &= \begin{cases} ||\mathcal{A}||(u) & \text{if } x = u \otimes \$\square(a\square)^m, u \in \mathrm{CT}_{\mathrm{reg}}, \\ & \quad \text{and } m = \max\{n \mid \$\square(a\square)^n \leq_{\mathrm{pref}} u\} \\ ||\mathcal{A}_{\mathcal{M}}||(u) & \text{if } x = u \otimes \$\square(a\square)^m \text{ and} \\ & \quad (u \notin \mathrm{CT}_{\mathrm{reg}} \text{ or } m \neq \max\{n \mid \$\square(a\square)^n \leq_{\mathrm{pref}} u\}) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

for all $u \in \Sigma^+$ and $x \in (\Sigma_{\#}^2)^+$.

For $m \in \mathbb{N}$, we define the function $r_{\mathcal{M},m} \colon \Sigma^+ \to \mathbb{N}$ by $r_{\mathcal{M},m}(u) = ||\mathcal{B}_{\mathcal{M}}||(u \otimes \$\square(a\square)^m$. This is well-defined since, for any $u \in \Sigma^+$ and $m \in \mathbb{N}$, we have $||\mathcal{A}||(u) \in \mathbb{N}$ and therefore also $||\mathcal{B}_{\mathcal{M}}||(u \otimes \$\square(a\square)^m) \in \mathbb{N}$. In other words, we have

$$r_{\mathcal{M},m}(u) = \begin{cases} ||\mathcal{A}||(u) & \text{if } u \in \mathrm{CT}_{\mathrm{reg}} \text{ and } m = \max\{n \mid \$\square(a\square)^n \leq_{\mathrm{pref}} u\} \\ ||\mathcal{A}_{\mathcal{M}}||(u) & \text{otherwise}\,. \end{cases}$$

The following is the central property from this section that we will use in our handling of tree-automatic linear orders.

▶ Proposition 10. For all $m \in \mathbb{N}$, the following are equivalent:

1. m is not accepted by the Minsky machine $\mathcal{M}$.
2. $||\mathcal{A}_{\mathcal{M}}||(u) = r_{\mathcal{M},m}(u)$ holds for all $u \in \Sigma^*$.

▶ Remark. Fix some Minsky machines $\mathcal{M}$ that accepts an undecidable set of natural numbers. From $m \in \mathbb{N}$, one can then construct a weighted automaton $\mathcal{B}$ with $||\mathcal{B}|| = r_{\mathcal{M},m}$. Hence the set of weighted automata $\mathcal{B}$ with $||\mathcal{B}|| = ||\mathcal{A}_{\mathcal{M}}||$ is Π_1^0-hard and therefore undecidable. This strengthens Krob's result stating that the set of pairs $(\mathcal{A}, \mathcal{B})$ of weighted automata with $||\mathcal{A}|| = ||\mathcal{B}||$ is Π_1^0-hard.

4.2 Isomorphism

Note that Krob's result talks about functions $\Sigma^+ \to \mathbb{N}$ while we are interested in linear orders. Therefore, we set

$$\mathcal{L}_r = \sum_{w \in (\Sigma^+; \leq_{\mathrm{llex}})} (\omega^{r(w)+1} + \delta)$$

for a function $r \colon \Sigma^+ \to \mathbb{N}$. Since $(\Sigma^+; \leq_{\mathrm{llex}}) \cong \omega$, this linear order is an ω-sequence of ordinals of the form ω^n with $n \geq 1$, separated by our delimiter δ. Hence it is scattered. Furthermore, for all functions $r, r' \colon \Sigma^+ \to \mathbb{N}$, we obtain

$$\mathcal{L}_r \cong \mathcal{L}_{r'} \iff r = r'\,. \tag{1}$$

The following lemma states in particular that $\mathcal{L}_r$ is tree automatic whenever $r = ||\mathcal{A}||$ for some weighted automaton $\mathcal{A}$.

▶ **Lemma 11.** *From a weighted automaton $\mathcal{A}$, one can compute a regular set of trees $L_{\mathcal{A}}$ such that $(L_{\mathcal{A}}; \leq_{\mathrm{trees}}) \cong \mathcal{L}_{||\mathcal{A}||}$.*

Before we prove this lemma, we show how we can use it to prove that the isomorphism problem of scattered tree-automatic linear orders is undecidable (the proof of Lemma 11 can be found following the proof of Corollary 14).

▶ **Lemma 12.** *From a Minsky machine $\mathcal{M}$ and $m \in \mathbb{N}$, one can compute a regular set of trees L such that $(L; \leq_{\text{trees}}) \cong \mathcal{L}_{r_{\mathcal{M},m}}$.*

Proof. Let $\mathcal{M}$ be a Minsky machine and let $m \in \mathbb{N}$. Let $\mathcal{B}_{\mathcal{M}}$ be the weighted automaton constructed following Theorem 9. Then, from $m \in \mathbb{N}$, we can compute a weighted automaton $\mathcal{B}_{\mathcal{M},m}$ with alphabet Σ such that

$$||\mathcal{B}_{\mathcal{M},m}||(u) = ||\mathcal{B}_{\mathcal{M}}||(u \otimes \$\square(a\square)^m) \text{ for all } u \in \Sigma^+ .$$

But then $||\mathcal{B}_{\mathcal{M},m}|| = r_{\mathcal{M},m}$. By Lemma 11, we can compute, from $m \in \mathbb{N}$, a regular language of trees L such that $(L; \leq_{\text{trees}}) \cong \mathcal{L}_{||\mathcal{B}_{\mathcal{M},m}||} = \mathcal{L}_{r_{\mathcal{M},m}}$. ◀

▶ **Theorem 13.** *There is a scattered linear order $\mathcal{L}$ such that the set of regular tree languages L with $(L; \leq_{\text{trees}}) \cong \mathcal{L}$ is Π^0_1-hard.*

Proof. Let $P \subseteq \mathbb{N}$ be some Π^0_1-complete set. Then there exists a Minsky machine $\mathcal{M}$ that accepts the set $\mathbb{N} \setminus P$. Let $\mathcal{A}_{\mathcal{M}}$ and $\mathcal{B}_{\mathcal{M}}$ be the weighted automata constructed following Theorem 9. Then we get

$$\begin{aligned} m \in P &\iff m \text{ is not accepted by } \mathcal{M} \\ &\overset{\text{Prop. 10}}{\iff} ||\mathcal{A}_{\mathcal{M}}||(u) = r_{\mathcal{M},m}(u) \text{ for all } u \in \Sigma^+ \\ &\iff \mathcal{L}_{||\mathcal{A}_{\mathcal{M}}||} \cong \mathcal{L}_{r_{\mathcal{M},m}} \end{aligned}$$

where the last equivalence follows from (1). Hence, by Lemma 12, we can reduce the Π^0_1-complete set P to the set of regular tree languages L with $(L; \leq_{\text{trees}}) \cong \mathcal{L}_{||\mathcal{A}_{\mathcal{M}}||}$. The theorem therefore holds with $\mathcal{L} = \mathcal{L}_{||\mathcal{A}_{\mathcal{M}}||}$. ◀

Since the linear order $\leq_{\text{trees}}$ is tree-automatic, we immediately obtain

▶ **Corollary 14.** *There is a scattered linear order $\mathcal{L}$ whose set of tree-automatic presentations is Π^0_1-hard.*

One immediately gets that the isomorphism problem for tree-automatic scattered linear orders is Π^0_1-hard. We do not know whether the set of tree-automatic presentations of *scattered* linear orders is decidable. Therefore, the following immediate consequence of Corollary 14 is a bit stronger:

▶ **Corollary 15.** *Let X be a set of pairs of tree-automatic presentations such that, for all tree-automatic presentations P_1 and P_2 of scattered linear orders $\mathcal{L}_1$ and $\mathcal{L}_2$, one has*

$$(P_1, P_2) \in X \iff \mathcal{L}_1 \cong \mathcal{L}_2 .$$

Then X is Π^0_1-hard.

The rest of this section is devoted to the proof of Lemma 11.

Proof of Lemma 11. Let $\mathcal{A} = (Q, \Sigma, \iota, \mu, F)$ be a weighted automaton. We will construct a tree-automatic presentation of the linear order $\mathcal{L}_{||\mathcal{A}||}$.

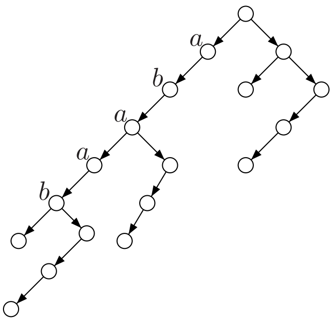
A *run tree of $\mathcal{A}$* is a tree t over the alphabet $\Sigma \uplus \{\$\}$ such that there exist states $\iota = q_0, q_1, \ldots, q_{k-1} \in Q$ and $q_k \in F$ (with $k = \max\{i \mid 0^{i+1} \in \text{dom}(t)\}$) with the following properties (see the tree on the next page with $k = 5$ where we omitted the label \$):

(T1) $11 \in \mathrm{dom}(t) \subseteq 0^* \cup 0^*10^* \cup 110^*$ and $100 \notin \mathrm{dom}(t)$
(T2) $t(0^i) \in \Sigma$ and $\mu(q_{i-1}, t(0^i), q_i) \neq -\infty$ for all $1 \leq i \leq k$
(T3) $0^i1 \in \mathrm{dom}(t)$ implies $1 \leq i \leq k$ and $\mu(q_{i-1}, a_i, q_i) = 1$ or $i = 0$
(T4) $t^{-1}(\$) = \mathrm{dom}(t) \setminus \{0^i \mid 1 \leq i \leq k\}$
Note that every run tree t defines a word over Σ, namely

$$\mathrm{word}(t) = t(0)\, t(00)\, \ldots\, t(0^k)\,.$$

Since $11 \in \mathrm{dom}(t)$, also 10 and therefore 1 belong to $\mathrm{dom}(t)$. Since, consequently, $0 \in \mathrm{dom}(t)$, we have $\mathrm{word}(t) \neq \varepsilon$.

The figure on the right shows a run tree t with $\mathrm{word}(t) = abaab$. The idea is that the "main branch" $\{0, 00, \ldots, 0^k\}$ carries a run ρ of the weighted automaton $\mathcal{A}$. The number of "side branches" starting in some node 0^i1 with $i > 0$ is at most the weight $\mathrm{wt}(\rho)$ of the encoded run. Since these side branches have arbitrary length, the whole run tree stands for an element of $\omega^{\mathrm{wt}(\rho)}$. The "side branch" starting in 11 plays a special role, its length $|\mathrm{dom}(t) \cap 110^+|$ is denoted $n(t)$ (the run tree t on the right satisfies $n(t) = 2$).



We next define, for two trees s and t, the tree $s + t$ by adding a new \$-labeled root and considering s as left subtree of $s + t$ and t as right subtree. In particular, we will need trees of the form $w + t$ with w a word. These trees carry the sequences $\$w$ on $\mathrm{dom}(w + t) \cap 0^*$ and satisfy $(w + t) \restriction_1 \cong t$.

We consider the language $L_{\mathcal{A}} = \{t \mid t \text{ is a run tree}\} \cup \{w\$ + t \mid w \in \Sigma^+, t \in D\}$ where D is the set of trees from page 458 that satisfies $(D; \leq_{\mathrm{trees}}) \cong \delta$. This language $L_{\mathcal{A}}$ is regular.

Note that trees from $L_{\mathcal{A}}$ use the alphabet $\Sigma \cup \{\$\}$ that we order arbitrarily. We will now prove $(L_{\mathcal{A}}; \leq_{\mathrm{trees}}) \cong \mathcal{L}_{||\mathcal{A}||}$.

First let $w \in \Sigma^+$ and $n \in \mathbb{N}$. Then let $\mathcal{I}^1_{w,n}$ denote the restriction of $(L_{\mathcal{A}}; \leq_{\mathrm{trees}})$ to all run trees t with

$$\mathrm{word}(t) = w \text{ and } n(t) = n\,. \tag{2}$$

Note that for any two run trees s and t satisfying (2), we have $s\restriction_{0^*} = t\restriction_{0^*}$ and $s\restriction_1 = t\restriction_1$. Hence $s \leq_{\mathrm{trees}} t$ if and only if there exists $i \geq 1$ with $t\restriction_{0^j1} = s\restriction_{0^j1}$ for all $1 \leq j < i$ and $s\restriction_{0^i1} <_{\mathrm{trees}} t\restriction_{0^i1}$. By (T3), $\mathrm{dom}(t) \cap 0^+1$ contains at most $|w|$ elements. Furthermore note that the trees $t\restriction_{0^i1}$ can be identified with natural numbers (namely with $|\mathrm{dom}(t) \cap 0^i10^*|$). This shows that $\mathcal{I}^1_{w,n}$ can be embedded into $(\mathbb{N}^{|w|}; \leq_{\mathrm{lex}})$ and is therefore well-ordered and at most $\omega^{|w|}$.

Now let $\rho = (q_0, a_1, q_1)(q_1, a_2, q_2) \ldots (q_{k-1}, a_k, q_k) \in \mathrm{Run}(\mathcal{A}, w)$ be a run of the weighted automaton $\mathcal{A}$ on the word $w = a_1 \ldots a_k$. Consider a tuple $(m_1, \ldots, m_k) \in \mathbb{N}^k$ such that $m_i > 0$ implies $\mu(q_{i-1}, a_i, q_i) = 1$. Then there exists a unique run tree t satisfying (2) and $|\mathrm{dom}(t) \cap 0^i10^*| = m_i$ for all $1 \leq i \leq k$. This gives an order-preserving embedding $f_\rho \colon \omega^{\mathrm{wt}(\rho)} \to \mathcal{I}^1_{w,n}$, i.e., we showed $\omega^{\mathrm{wt}(\rho)} \leq \mathcal{I}^1_{w,n}$. Since this holds for abitrary runs $\rho \in \mathrm{Run}(\mathcal{A}, w)$ and since $||\mathcal{A}||(w) = \max\{\mathrm{wt}(\rho) \mid \rho \in \mathrm{Run}(\mathcal{A}, w)\}$, we get $\omega^{||\mathcal{A}||(w)} \leq \mathcal{I}^1_{w,n}$ and therefore $\omega^{||\mathcal{A}||(w)+1} \leq \mathcal{I}^1_{w,n} \cdot \omega$.

By (T2), for every run tree t satisfying (2), there exists at least one run $\rho \in \mathrm{Run}(\mathcal{A}, w)$ such that t is in the image of the embedding f_ρ. Hence $\mathcal{I}^1_{w,n} \leq \bigoplus_{\rho \in \mathrm{Run}(\mathcal{A}, w)} \omega^{\mathrm{wt}(\rho)}$ where

$\bigoplus$ denotes the natural sum of ordinals. We can conclude

$$\begin{aligned}\omega^{||\mathcal{A}||(w)+1} \leq \mathcal{I}^1_{w,n} \cdot \omega \leq \left(\bigoplus_{\rho \in \mathrm{Run}(\mathcal{A},w)} \omega^{\mathrm{wt}(\rho)} \right) \cdot \omega \\ = \omega^{\max\{\mathrm{wt}(\rho) \mid \rho \in \mathrm{Run}(\mathcal{A},w)\}+1} \\ = \omega^{||\mathcal{A}||(w)+1}\end{aligned}$$

and therefore $\mathcal{I}^1_{w,n} \cdot \omega = \omega^{||\mathcal{A}||(w)+1}$.

Next consider the restriction $\mathcal{I}^1_w$ of $(L_\mathcal{A}; \leq_{\mathrm{trees}})$ to the set of run trees t with $\mathrm{word}(t) = w$. Then $n(s) < n(t)$ implies $s <_{\mathrm{trees}} t$. Furthermore, the restriction of $\mathcal{I}^1_w$ to the set of run trees t with $n(t) = n$ equals $\mathcal{I}^1_{w,n}$. Hence $\mathcal{I}^1_w = \sum_{n \in (\mathbb{N};\leq)} \mathcal{I}^1_{w,n} = \mathcal{I}^1_{w,0} \cdot \omega = \omega^{||\mathcal{A}||(w)+1}$ since $\mathcal{I}^1_{w,0} \cong \mathcal{I}^1_{w,n}$ for all $n \geq 0$.

Next consider the restriction $\mathcal{I}^2_w$ of $(L_\mathcal{A}; \leq^2_{\mathrm{trees}})$ to the set of trees $w\$ + D$. Then $\mathcal{I}^2_w \cong \delta$ by what we saw on page 458. Let s be a run tree with $\mathrm{word}(s) = w$ and let $t \in w\$ + D$. Then s and t coincide on 0^* (where they both carry the sequence $\$w\$$). Consider $s{\restriction}_{10^*}$ and $t{\restriction}_{10^*}$. Since s is a run tree, we have $\mathrm{dom}(s) \cap 10^* = \{1, 10\}$ while $t{\restriction}_1 \in D$ implies $\mathrm{dom}(t) \cap 10^* = \{1, 10, 100\}$. Hence $s{\restriction}_1 <_{\mathrm{trees}} t{\restriction}_1$ and therefore $s <_{\mathrm{trees}} t$. Hence, the restriction $\mathcal{I}_w$ of $(L_\mathcal{A}; \leq_{\mathrm{trees}})$ to the set of run trees t with $\mathrm{word}(t) = w$ and the set of trees $w\$ + D$ satisfies $\mathcal{I}_w = \mathcal{I}^1_w + \mathcal{I}^2_w \cong \omega^{||\mathcal{A}||(w)+1} + \delta$.

Finally, let $u, v \in \Sigma^+$. Then $u \leq_{\mathrm{llex}} v$ if and only if $u \leq_{\mathrm{trees}} v$. This implies

$$(L_\mathcal{A}; \leq_{\mathrm{trees}}) = \sum_{w \in (\Sigma^+; \leq_{\mathrm{llex}})} \mathcal{I}_w \cong \sum_{w \in (\Sigma^+; \leq_{\mathrm{llex}})} \omega^{||\mathcal{A}||(w)+1} + \delta = \mathcal{L}_\mathcal{A} \,.$$

◀

4.3 Automorphisms

From Theorem 5, we already know that the existence of a non-trivial automorphism of a word-automatic and scattered linear order is Σ^0_1-hard. Here, we push this lower bound one level higher for tree-automatic scattered linear orders. The order theoretic construction resembles that from Section 3.2, but also uses ideas from the previous section.

Let $\mathcal{M}$ be a Minsky machine, let $\mathcal{A}_\mathcal{M}$ and $\mathcal{B}_\mathcal{M}$ be the weighted automata and, for $m \in \mathbb{N}$, let $r_{\mathcal{M},m}$ be the function defined following Theorem 9. Then we define the linear order

$$\mathcal{L}_\mathcal{M} = \sum_{m \in (\mathbb{N};\leq)} \left(\mathcal{L}_{||\mathcal{A}_\mathcal{M}||} \cdot \omega^* + \mathcal{L}_{r_{\mathcal{M},m}} \cdot \omega \right) .$$

Note that this linear order is rigid if and only if $\mathcal{L}_{||\mathcal{A}_\mathcal{M}||} \not\cong \mathcal{L}_{r_{\mathcal{M},m}}$ for all $m \in \mathbb{N}$. But this is the case if and only if $\mathcal{M}$ accepts all natural numbers m, a Π^0_2-complete problem.

▶ **Lemma 16.** *From a Minsky machine $\mathcal{M}$, one can construct a tree-automatic presentation of the linear order $\mathcal{L}_\mathcal{M}$.*

Proof. Let $\mathcal{M}$ be a Minsky machine, let $\mathcal{A}_\mathcal{M}$ and $\mathcal{B}_\mathcal{M}$ be the weighted automata and let $r_{\mathcal{M},m} \colon \Sigma^+ \to \mathbb{N}$ be the function defined following Theorem 9. Recall that the alphabet of $\mathcal{A}_\mathcal{M}$ is Σ and that of $\mathcal{B}_\mathcal{M}$ is $\Sigma^2_\#$. Recall the notion of a run tree from the proof of Lemma 11 that is based on a weighted automaton. In this proof, we will consider run trees with respect to the weighted automaton $\mathcal{A}_\mathcal{M}$ and with respect to the weighted automaton $\mathcal{B}_\mathcal{M}$. Now recall the definition of the language $L_{\mathcal{A}_\mathcal{M}}$ and $L_{\mathcal{B}_\mathcal{M}}$:

$$\begin{aligned} L_{\mathcal{A}_\mathcal{M}} &= \{t \mid t \text{ is a run tree wrt. } \mathcal{A}_\mathcal{M}\} \cup \{w\$ + t \mid w \in \Sigma^+, t \in D\} \\ L_{\mathcal{B}_\mathcal{M}} &= \{t \mid t \text{ is a run tree wrt. } \mathcal{B}_\mathcal{M}\} \cup \{w\$ + t \mid w \in (\Sigma^2_\#)^+, t \in D\} \end{aligned}$$

Note that these two tree languages are disjoint since the alphabets Σ and $\Sigma^2_\#$ are disjoint. Now define the language

$$\begin{aligned} L_{\mathcal{M}} &= (L_{\mathcal{A}_{\mathcal{M}}} \otimes \$^* \otimes \$\Box(a\Box)^*) \\ &\quad \cup \{t \otimes \$^k \otimes \$\Box(a\Box)^m \mid \; k, m \in \mathbb{N}, t \in L_{\mathcal{B}_{\mathcal{M}}}, \text{ and} \\ &\qquad (t \text{ is a run tree } \Rightarrow \text{word}(t) \in \Sigma^+ \otimes \$\Box(a\Box)^m)\}\,. \end{aligned}$$

The crucial point regarding the regularity of this set is the verification that a tree $t \otimes \$^k \otimes \$\Box(a\Box)^m$ with t a run tree of $\mathcal{B}_{\mathcal{M}}$ belongs to the second set. But this is the case if $t{\restriction}_{0^*} = \$\$\Box(a\Box)^m\$$, a property that a tree automaton can check easily.

On this set, we define the following linear order $\preceq$: $(s\otimes\$^k\otimes\$\Box(a\Box)^m) \preceq (t\otimes\$^\ell\otimes\$\Box(a\Box)^n)$ if and only if one of the following hold

(O1) $m < n$ or
(O2) $m = n$, $s \in L_{\mathcal{A}_{\mathcal{M}}}$, and $t \in L_{\mathcal{B}_{\mathcal{M}}}$, or
(O3) $m = n$, $s, t \in L_{\mathcal{A}_{\mathcal{M}}}$, and $k > \ell$, or
(O4) $m = n$, $s, t \in L_{\mathcal{A}_{\mathcal{M}}}$, $k = \ell$, and $s \leq_{\text{trees}} t$, or
(O5) $m = n$, $s, t \in L_{\mathcal{B}_{\mathcal{M}}}$, and $k < \ell$, or
(O6) $m = n$, $s, t \in L_{\mathcal{B}_{\mathcal{M}}}$, $k = \ell$, and $s \leq_{\text{trees}} t$.

It is clear that this relation is automatic and one can show $(L_{\mathcal{M}}; \preceq) \cong \mathcal{L}_{\mathcal{M}}$. ◀

▶ **Theorem 17.** **(i)** *The set of tree-automatic presentations of rigid (rigid and scattered, resp.) linear orders is Π^0_2-hard.*
(ii) *The set of tree-automatic presentations of automatically rigid linear orders is Π^0_1-complete.*

5 Open questions

The isomorphism and rigidity problems for word-automatic scattered linear orders both belong to Δ^0_ω (cf. [24]), our lower bound Π^0_1 for the rigidity problem leaves quite some room for improvements. Since the rank of a tree-automatic linear order is properly below ω^ω [18, 17], the proof of [24] can be adapted to show that the isomorphism and the rigidity problems for tree-automatic scattered linear orders both belong to $\Sigma^0_{\omega^\omega}$. But we only have the lower bounds Π^0_1 and Π^0_2, resp. Finally, the rigidity problem for arbitrary word or tree-automatic linear orders is in Π^1_1, but also here, we only have the arithmetic lower bound Π^0_1 and Π^0_2, resp.

But the most pressing open question is the isomorphism problem of scattered and word-automatic linear orders.

References

1 V. Bárány, E. Grädel, and S. Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*, pages 1–76. Cambridge University Press, 2011.
2 S.L. Bloom and Z. Ésik. The equational theory of regular words. *Information and Computation*, 197:55–89, 2005.
3 A. Blumensath and E. Grädel. Automatic Structures. In *LICS'00*, pages 51–62. IEEE Computer Society Press, 2000.
4 A. Blumensath and E. Grädel. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems*, 37(6):641–674, 2004.
5 G. Braun and L. Strüngmann. Breaking up finite automata presentable torsion-free abelian groups. *International Journal of Algebra and Computation*, 21(8):1463–1472, 2011.

6 G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre, I. *Math. Annalen*, 46:481–512, 1895.

7 A. Carayol and Z. Ésik. The FC-rank of a context-free language. arXiv:1202.6275, February 2012.

8 B. Courcelle. Frontiers of infinite trees. *RAIRO - Theoretical Informatics*, 12(4):319–337, 1978.

9 M. Droste, W. Kuich, and H. Vogler, editors. *Handbook of Weighted Automata.* EATCS Monographs in Theoretical Computer Science. Springer, 2009.

10 M. Droste and D. Kuske. Weighted automata. To appear in the forthcoming handbook AutoMathA, 2012.

11 A. Durand-Gasselin and P. Habermehl. Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree. In *STACS'12*, pages 242–253. Dagstuhl Publishing, 2012.

12 C.C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.*, 98:21–51, 1961.

13 D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson, and W.P. Thurston. *Word Processing In Groups.* Jones and Bartlett Publishers, Boston, 1992.

14 Z. Ésik. An undecidable property of context-free linear orders. *Inform. Processing Letters*, 111(3):107–109, 2011.

15 St. Heilbrunner. An algorithm for the solution of fixed-point equations for infinite words. *RAIRO – Theoretical Informatics*, 14(2):131–141, 1980.

16 B.R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19:331–335, 1982.

17 M. Huschenbett. The rank of tree-automatic linear orderings. arXiv:1204.3048, 2012.

18 S. Jain, B. Khoussainov, Ph. Schlicht, and F. Stephan. Tree-automatic scattered linear orders. Manuscript, 2012.

19 B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logic and Computational Complexity*, Lecture Notes in Comp. Science vol. 960, pages 367–392. Springer, 1995.

20 B. Khoussainov and A. Nerode. Open questions in the theory of automatic structures. *Bulletin of the EATCS*, 94:181–204, 2008.

21 B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: richness and limitations. *Log. Methods in Comput. Sci.*, 3(2), 2007.

22 B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computational Logic*, 6(4):675–700, 2005.

23 D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4(3):405–425, 1994.

24 D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. *Transactions of the AMS*, 2011. Accepted.

25 M. Lohrey and Ch. Mathissen. Isomorphism of regular trees and words. In *ICALP'11*, Lecture Notes in Comp. Science vol. 6756, pages 210–221. Springer, 2011.

26 Y. Matijasevich. *Hilbert's Tenth Problem.* Foundations of Computing Series. MIT Press, 1993.

27 A. Nies. Describing groups. *Bulletin of Symbolic Logic*, 13(3):305–339, 2007.

28 S. Rubin. *Automatic Structures.* PhD thesis, University of Auckland, 2004.

29 S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14:169–209, 2008.

30 W. Thomas. On frontiers of regular trees. *RAIRO – Theoretical Informatics*, 20(4):371–381, 1986.

Undecidable First-Order Theories of Affine Geometries*

Antti Kuusisto[1], Jeremy Meyers[2], and Jonni Virtema[1]

1 **University of Tampere**
`{antti.j.kuusisto, jonni.virtema}@uta.fi`
2 **Stanford University**
`jjmeyers@stanford.edu`

Abstract

Tarski initiated a logic-based approach to formal geometry that studies first-order structures with a ternary *betweenness* relation (β) and a quaternary *equidistance* relation ($\equiv$). Tarski established, inter alia, that the first-order (FO) theory of $(\mathbb{R}^2, \beta, \equiv)$ is decidable. Aiello and van Benthem (2002) conjectured that the FO-theory of expansions of $(\mathbb{R}^2, \beta)$ with unary predicates is decidable. We refute this conjecture by showing that for all $n \geq 2$, the FO-theory of monadic expansions of $(\mathbb{R}^n, \beta)$ is Π^1_1-hard and therefore not even arithmetical. We also define a natural and comprehensive class $\mathcal{C}$ of geometric structures (T, β), where $T \subseteq \mathbb{R}^n$, and show that for each structure $(T, \beta) \in \mathcal{C}$, the FO-theory of the class of monadic expansions of (T, β) is undecidable. We then consider classes of expansions of structures (T, β) with restricted unary predicates, for example finite predicates, and establish a variety of related undecidability results. In addition to decidability questions, we briefly study the expressivity of universal MSO and weak universal MSO over expansions of $(\mathbb{R}^n, \beta)$. While the logics are incomparable in general, over expansions of $(\mathbb{R}^n, \beta)$, formulae of weak universal MSO translate into equivalent formulae of universal MSO.

An extended version of this article can be found on the ArXiv (arXiv:1208.4930v1).

1998 ACM Subject Classification F.4.1 Model theory, Computability theory

Keywords and phrases Tarski's geometry, undecidability, spatial logic, classical logic

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.470

1 Introduction

Decidability of theories of (classes of) structures is a central topic in various different fields of computer science and mathematics, with different motivations and objectives depending on the field in question. In this article we investigate formal theories of *geometry* in the framework introduced by Tarski [21, 22]. The logic-based framework was originally presented in a series of lectures given in Warsaw in the 1920's. The system is based on first-order structures with two predicates: a ternary *betweenness* relation β and a quaternary *equidistance* relation $\equiv$. Within this system, $\beta(u, v, w)$ is interpreted to mean that the point v is between the points u and w, while $xy \equiv uv$ means that the distance from x to y is equal to the distance from u to v. The betweenness relation β can be considered to simulate the action of a ruler, while the equidistance relation $\equiv$ simulates the action of a compass. See [22] for information about the history and development of Tarski's geometry.

Tarski established in [21] that the first-order theory of $(\mathbb{R}^2, \beta, \equiv)$ is decidable. In [1], Aiello and van Benthem pose the question: "*What is the complete monadic* Π^1_1 *theory of*

* This work was partially supported by grant 129761 of the Academy of Finland.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 470–484
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the affine real plane?" By *affine real plane*, the authors refer to the structure $(\mathbb{R}^2, \beta)$. The monadic Π^1_1-theory of $(\mathbb{R}^2, \beta)$ is of course essentially the same as the first-order theory of the class of expansions $(\mathbb{R}^2, \beta, (P_i)_{i \in \mathbb{N}})$ of the the affine real plane $(\mathbb{R}^2, \beta)$ by unary predicates $P_i \subseteq \mathbb{R}^2$. Aiello and van Benthem conjecture that the theory is decidable. Expansions of $(\mathbb{R}^2, \beta)$ with *unary* predicates are especially relevant in investigations related to the geometric structure $(\mathbb{R}^2, \beta)$, since in this context unary predicates correspond to *regions* of the plane $\mathbb{R}^2$.

In this article we study structures of the type of (T, β), where $T \subseteq \mathbb{R}^n$ and β is the canonical Euclidean betweenness predicate restricted to T, see Section 2.3 for the formal definition. Let $E\big((T, \beta)\big)$ denote the class of expansions $(T, \beta, (P_i)_{i \in \mathbb{N}})$ of (T, β) with unary predicates. We identify a significant collection of canonical structures (T, β) with an undecidable first-order theory of $E\big((T, \beta)\big)$. Informally, if there exists a flat two-dimensional region $R \subseteq \mathbb{R}^n$, no matter how small, such that $T \cap R$ is in a certain sense sufficiently dense with respect to R, then the first-order theory of the class $E\big((T, \beta)\big)$ is undecidable. If the related density conditions are satisfied, we say that T *extends linearly in* $2D$, see Section 2.3 for the formal definition. We prove that for any $T \subseteq \mathbb{R}^n$, if T extends linearly in $2D$, then the FO-theory of $E\big((T, \beta)\big)$ is Σ^0_1-hard. In addition, we establish that for all $n \geq 2$, the first-order theory of $E\big((\mathbb{R}^n, \beta)\big)$ is Π^1_1-hard, and therefore not even arithmetical. We thereby refute the conjecture of Aiello and van Benthem from [1]. The results are ultimately based on tiling arguments. The result establishing Π^1_1-hardness relies on the *recurrent tiling problem* of Harel [14]—once again demonstrating the usefulness of Harel's methods.

Our results establish undecidability for a wide range of monadic expansion classes of natural geometric structures (T, β). In addition to $(\mathbb{R}^2, \beta)$, such structures include for example the rational plane $(\mathbb{Q}^2, \beta)$, the real unit cube $([0, 1]^3, \beta)$, and the plane of algebraic reals $(\mathbb{A}^2, \beta)$ — to name a few.

In addition to investigating monadic expansion classes of the type $E\big((T, \beta)\big)$, we also study classes of expansions with *restricted* unary predicates. Let n be a positive integer and let $T \subseteq \mathbb{R}^n$. Let $F\big((T, \beta)\big)$ denote the class of structures $(T, \beta, (P_i)_{i \in \mathbb{N}})$, where the sets P_i are *finite* subsets of T. We establish that if T extends linearly in $2D$, then the first-order theory of $F\big((T, \beta)\big)$ is undecidable. An alternative reading of this result is that the *weak* universal monadic second-order theory of (T, β) is undecidable. We obtain a Π^0_1-hardness result by an argument based on the *periodic torus tiling problem* of Gurevich and Koryakov [12]. The torus tiling argument can easily be adapted to deal with various different kinds of natural classes of expansions of geometric structures (T, β) with restricted unary predicates. These include the classes with unary predicates denoting—for example—polygons, finite unions of closed rectangles, and real algebraic sets (see [8] for the definition).

Our results could turn out useful in investigations concerning logical aspects of spatial databases. It turns out that there is a canonical correspondence between $(\mathbb{R}^2, \beta)$ and $(\mathbb{R}, 0, 1, \cdot, +, <)$, see [13]. See the survey [17] for further details on logical aspects of spatial databases.

The betweenness predicate is also studied in spatial logic [3]. The recent years have witnessed a significant increase in the research on spatially motivated logics. Several interesting systems with varying motivations have been investigated, see for example the articles [1, 4, 5, 15, 16, 18, 20, 23, 24]. See also the surveys [2] and [6] in the Handbook of Spatial Logics [3], and the Ph.D. thesis [11]. Several of the above articles investigate fragments of first-order theories by way of modal logics for affine, projective, and metric geometries. Our results contribute to the understanding of spatially motivated first-order languages, and hence they can be useful in the search for decidable (modal) spatial logics.

In addition to studying issues of decidability, we briefly compare the expressivities of universal monadic second-order logic $\forall$MSO and weak universal monadic second-order logic $\forall$WMSO. It is straightforward to observe that in general, the expressivities of $\forall$MSO and $\forall$WMSO are incomparable in a rather strong sense: $\forall\text{MSO} \not\leq \text{WMSO}$ and $\forall\text{WMSO} \not\leq \text{MSO}$. Here MSO and WMSO denote monadic second-order logic and weak monadic second-order logic, respectively. The result $\forall\text{WMSO} \not\leq \text{MSO}$ follows from already existing results (see [10] for example), and the result $\forall\text{MSO} \not\leq \text{WMSO}$ is more or less trivial to prove. While $\forall$MSO and $\forall$WMSO are incomparable in general, the situation changes when we consider expansions $(\mathbb{R}^n, \beta, (R_i)_{i\in I})$ of the stucture $(\mathbb{R}^n, \beta)$, i.e., structures embedded in the geometric structure $(\mathbb{R}^n, \beta)$. Here $(R_i)_{i\in I}$ is an arbitrary vocabulary and I an arbitrary related index set. We show that over such structures, sentences of $\forall$WMSO translate into equivalent sentences of $\forall$MSO. The proof is based on the Heine-Borel theorem.

The structure of the current article is as follows. In Section 2 we define the central notions needed in the later sections. In Section 3 we compare the expressivities of $\forall$MSO and $\forall$WMSO. In Section 4 we show undecidability of the first-order theory of the class of monadic expansions of any geometric structure (T, β) such that T exends linearly in $2D$. In addition, we show that for $n \geq 2$, the first-order theory of monadic expansions of $(\mathbb{R}^n, \beta)$ is not on any level of the arithmetical hierarchy. In Section 5 we modify the approach in Section 4 and show undecidability of the FO-theory of the class of expansions by finite unary predicates of any geometric structure (T, β) such that T extends linearly in $2D$.

2 Preliminaries

2.1 Interpretations

Let σ and τ be relational vocabularies. Let $\mathcal{A}$ be a nonempty class of σ-structures and $\mathcal{C}$ a nonempty class of τ-structures. Assume that there exists a surjective map F from $\mathcal{C}$ onto $\mathcal{A}$ and a first-order τ-formula $\varphi_{Dom}(x)$ in one free variable, x, such that for each structure $\mathfrak{B} \in \mathcal{C}$, there is a bijection f from the domain of $F(\mathfrak{B})$ to the set

$$\{\ b \in Dom(\mathfrak{B}) \mid \mathfrak{B} \models \varphi_{Dom}(b)\ \}.$$

Assume, furthermore, that for each relation symbol $R \in \sigma$, there is a first-order τ-formula $\varphi_R(x_1, ..., x_{Ar(R)})$ such that we have

$$R^{F(\mathfrak{B})}(a_1, ..., a_{Ar(R)}) \ \Leftrightarrow\ \mathfrak{B} \models \varphi_R\big(f(a_1), ..., f(a_{Ar(R)})\big)$$

for every tuple $(a_1, ..., a_{Ar(R)}) \in (Dom(F(\mathfrak{B})))^{Ar(R)}$. Here $Ar(R)$ is the arity of R. We then say that the class $\mathcal{A}$ is *uniformly first-order interpretable in* $\mathcal{C}$. If $\mathcal{A}$ is a singleton class $\{\mathfrak{A}\}$, we say that $\mathfrak{A}$ is *uniformly first-order interpretable in* $\mathcal{C}$.

Assume that a class of σ-structures $\mathcal{A}$ is uniformly first-order interpretable in a class $\mathcal{C}$ of τ-structures. Let $\mathcal{P}$ be a set of unary relation symbols such that $\mathcal{P} \cap (\sigma \cup \tau) = \emptyset$. Define a map I from the set of first-order $(\sigma \cup \mathcal{P})$-formulae to the set of first-order $(\tau \cup \mathcal{P})$-formulae as follows.

1. If $P \in \mathcal{P}$, then $I(Px) := Px$.
2. If $k \in \mathbb{N}_{\geq 1}$ and $R \in \sigma$ is a k-ary relation symbol, then $I(R(x_1, ..., x_k)) := \varphi_R(x_1, ..., x_k)$, where $\varphi_R(x_1, ..., x_k)$ is the first-order formula for R witnessing the fact that $\mathcal{A}$ is uniformly first-order interpretable in $\mathcal{C}$.
3. $I(x = y) := x = y$.

4. $I(\neg\varphi) := \neg I(\varphi)$.
5. $I(\varphi \wedge \psi) := I(\varphi) \wedge I(\psi)$.
6. $I\big(\exists x\, \psi(x)\big) := \exists x\big(\varphi_{Dom}(x) \wedge I(\psi(x))\big)$.

We call the map I the *$\mathcal{P}$-expansion of a uniform interpretation of $\mathcal{A}$ in $\mathcal{C}$*. When $\mathcal{A}$ and $\mathcal{C}$ are known from the context, we may call I simply a *$\mathcal{P}$-interpretation*. In the case where $\mathcal{P}$ is empty, the map I is a *uniform interpretation of $\mathcal{A}$ in $\mathcal{C}$*.

▶ **Lemma 1.** *Let σ and τ be finite relational vocabularies. Let $\mathcal{A}$ be a class of σ-structures and $\mathcal{C}$ a class of τ-structures. Assume that $\mathcal{A}$ is uniformly first-order interpretable in $\mathcal{C}$. Let $\mathcal{P}$ be a set of unary relation symbols such that $\mathcal{P} \cap (\sigma \cup \tau) = \emptyset$. Let I denote a related $\mathcal{P}$-interpretation. Let φ be a first-order $(\sigma \cup \mathcal{P})$-sentence. The following conditions are equivalent.*

1. *There exists an expansion $\mathfrak{A}^*$ of a structure $\mathfrak{A} \in \mathcal{A}$ to the vocabulary $\sigma \cup \mathcal{P}$ such that $\mathfrak{A}^* \models \varphi$.*
2. *There exists an expansion $\mathfrak{B}^*$ of a structure $\mathfrak{B} \in \mathcal{C}$ to the vocabulary $\tau \cup \mathcal{P}$ such that $\mathfrak{B}^* \models I(\varphi)$.*

Proof. Straightforward. ◀

2.2 Logics and structures

Monadic second order logic, MSO, extends first-order logic with quantification of relation symbols ranging over subsets of the domain of a model. In *universal (existential) monadic second order logic*, ∀MSO (∃MSO), the quantification of monadic relations is restricted to universal (existential) prenex quantification in the beginning of formulae. The logics ∀MSO and ∃MSO are also known as monadic Π^1_1 and monadic Σ^1_1. *Weak monadic second-order logic*, WMSO, is a semantic variant of monadic second-order logic in which the quantified relation symbols range over finite subsets of the domain of a model. The weak variants ∀WMSO and ∃WMSO of ∀MSO and ∃MSO are defined in the obvious way.

Let $\mathcal{L}$ be any fragment of second-order logic. The *$\mathcal{L}$-theory* of a structure $\mathfrak{M}$ of a vocabulary τ is the set of τ-sentences φ of $\mathcal{L}$ such that $\mathfrak{M} \models \varphi$.

Define two binary relations $H, V \subseteq \mathbb{N}^2 \times \mathbb{N}^2$ as follows.

- $H = \{\ \big((i,j),(i+1,j)\big) \mid i,j \in \mathbb{N}\ \}$.
- $V = \{\ \big((i,j),(i,j+1)\big) \mid i,j \in \mathbb{N}\ \}$.

We let $\mathfrak{G}$ denote the structure $(\mathbb{N}^2, H, V)$, and call it the *grid*. The relations H and V are called the *horizontal* and *vertical successor relations* of $\mathfrak{G}$, respectively. A *supergrid* is a structure of the vobabulary $\{H, V\}$ that has $\mathfrak{G}$ as a substructure. We denote the class of supergrids by $\mathcal{G}$.

Let $(\mathfrak{G}, R)$ be the expansion of $\mathfrak{G}$, where $R = \{\ \big((0,i),(0,j)\big) \in \mathbb{N}^2 \times \mathbb{N}^2 \mid i < j\ \}$. We denote the structure $(\mathfrak{G}, R)$ by $\mathfrak{R}$, and call it the *recurrence grid*.

Let m and n be positive integers. Define two binary relations $H_{m,n}, V_{m,n} \subseteq (m \times n)^2$ as follows. (Note that we define $m = \{0, ..., m-1\}$, and analogously for n.)

- $H_{m,n} = H \upharpoonright (m \times n)^2 \cup \{((m-1,i),(0,i)) \mid i < n\}$.
- $V_{m,n} = V \upharpoonright (m \times n)^2 \cup \{((i,n-1),(i,0)) \mid i < m\}$.

We call the structure $(m \times n, H_{m,n}, V_{m,n})$ the $m \times n$ *torus* and denote it by $\mathfrak{T}_{m,n}$. A torus is essentially a finite grid whose east border wraps back to the west border and north border back to the south border.

2.3 Geometric affine betweenness structures

Let $(\mathbb{R}^n, d)$ be the n-dimensional Euclidean space with the canonical metric d. We always assume $n \geq 1$. We define the ternary Euclidean *betweenness* relation β such that $\beta(s,t,u)$ iff $d(s,u) = d(s,t) + d(t,u)$. By β^* we denote the *strict betweenness* relation, i.e., $\beta^*(s,t,u)$ iff $\beta(s,t,u)$ and $s \neq t \neq u$. We say that the points $s,t,u \in \mathbb{R}^n$ are *collinear* if the disjunction $\beta(s,t,u) \vee \beta(s,u,t) \vee \beta(t,s,u)$ holds in $(\mathbb{R}^n, \beta)$. We define the first-order $\{\beta\}$-formula $collinear(x,y,z) := \beta(x,y,z) \vee \beta(x,z,y) \vee \beta(y,x,z)$.

Below we study geometric betweenness structures of the type (T, β_T) where $T \subseteq \mathbb{R}^n$ and $\beta_T = \beta \restriction T$. Here $\beta \restriction T$ is the restriction of the betweenness predicate β of $\mathbb{R}^n$ to the set T. To simplify notation, we usually refer to these structures by (T, β).

Let $T \subseteq \mathbb{R}^n$ and let β be the corresponding betweenness relation. We say that $L \subseteq T$ is a *line in* T if the following conditions hold.

1. There exist points $s,t \in L$ such that $s \neq t$.
2. For all $s,t,u \in L$, the points s,t,u are collinear.
3. Let $s,t \in L$ be points such that $s \neq t$. For all $u \in T$, if $\beta(s,u,t)$ or $\beta(s,t,u)$, then $u \in L$.

Let $T \subseteq \mathbb{R}^n$ and let L_1 and L_2 be lines in T. We say that L_1 and L_2 *intersect* if $L_1 \neq L_2$ and $L_1 \cap L_2 \neq \emptyset$. We say that the lines L_1 and L_2 *intersect in* $\mathbb{R}^n$ if $L_1 \neq L_2$ and $L_1' \cap L_2' \neq \emptyset$, where L_1', L_2' are the lines in $\mathbb{R}^n$ such that $L_1 \subseteq L_1'$ and $L_2 \subseteq L_2'$.

A subset $S \subseteq \mathbb{R}^n$ is an *m-dimensional flat* of $\mathbb{R}^n$, where $0 \leq m \leq n$, if there exists a set of m linearly independent vectors $v_1, \dots, v_m \in \mathbb{R}^n$ and a vector $h \in \mathbb{R}^n$ such that S is the h-translated span of the vectors $v_1, \dots, v_m$, in other words $S = \{u \in \mathbb{R}^n \mid u = h + r_1 v_1 + \dots + r_m v_m,\ r_1, \dots, r_m \in \mathbb{R}\}$. None of the vectors v_i is allowed to be the zero-vector.

A set $U \subseteq \mathbb{R}^n$ is a *linearly regular m-dimensional flat*, where $0 \leq m \leq n$, if the following conditions hold.

1. There exists an m-dimensional flat S such that $U \subseteq S$.
2. There does not exist any $(m-1)$-dimensional flat S such that $U \subseteq S$.
3. U is *linearly complete*, i.e., if L is a line in U and $L' \supseteq L$ the corresponding line in $\mathbb{R}^n$, and if $r \in L'$ is a point in L' and $\epsilon \in \mathbb{R}_+$ a positive real number, then there exists a point $s \in L$ such that $d(s,r) < \epsilon$. Here d is the canonical metric of $\mathbb{R}^n$.
4. U is *linearly closed*, i.e., if L_1 and L_2 are lines in U and L_1 and L_2 intersect in $\mathbb{R}^n$, then the lines L_1 and L_2 intersect. In other words, there exists a point $s \in U$ such that $s \in L_1 \cap L_2$.

A set $T \subseteq \mathbb{R}^n$ *extends linearly in mD*, where $m \leq n$, if there exists a linearly regular m-dimensional flat S, a positive real number $\epsilon \in \mathbb{R}_+$ and a point $x \in S \cap T$ such that $\{\ u \in S \mid d(x,u) < \epsilon\ \} \subseteq T$. It is easy show that for example $\mathbb{Q}^2$ extends linearly in $2D$.

2.4 Tilings

A function $t : 4 \longrightarrow \mathbb{N}$ is called a *tile type*. Define the set TILES $:= \{\ P_t \mid t \text{ is a tile type }\}$ of unary relation symbols. The unary relation symbols in the set TILES are called *tiles*. The numbers $t(i)$ of a tile P_t are the *colours* of P_t. The number $t(0)$ is the *top colour*, $t(1)$ the *right colour*, $t(2)$ the *bottom colour*, and $t(3)$ the *left colour* of P_t.

Let T be a finite nonempty set of tiles. We say that a structure $\mathfrak{A} = (A, V, H)$, where $V, H \subseteq A^2$, is *T-tilable*, if there exists an expansion of $\mathfrak{A}$ to the vocabulary $\{H, V\} \cup \{\ P_t \mid P_t \in T\ \}$ such that the following conditions hold.

1. Each point of A belongs to the extension of exactly one symbol P_t in T.

2. If uHv for some points $u, v \in A$, then the right colour of the tile P_t s.t. $P_t(u)$ is the same as the left colour of the tile $P_{t'}$ such that $P_{t'}(v)$.
3. If uVv for some points $u, v \in A$, then the top colour of the tile P_t s.t. $P_t(u)$ is the same as the bottom colour of the tile $P_{t'}$ such that $P_{t'}(v)$.

Let $t \in T$. We say that the grid $\mathfrak{G}$ is *t-recurrently T-tilable* if there exists an expansion of $\mathfrak{G}$ to the vocabulary $\{H, V\} \cup \{\ P_t \mid t \in T\ \}$ such that the above conditions $1 - 3$ hold, and additionally, there exist infinitely many points $(0, i) \in \mathbb{N}^2$ such that $P_t\big((0, i)\big)$. Intuitively this means that the tile P_t occurs infinitely many times in the leftmost column of the grid $\mathfrak{G}$. Let $\mathcal{F}$ be the set of finite, nonempty sets $T \subseteq$ TILES, and let $\mathcal{H} := \{\ (t, T) \mid T \in \mathcal{F},\ t \in T\ \}$. Define the following languages

$$\begin{aligned}
\mathcal{T} &:= \{\ T \in \mathcal{F} \mid \mathfrak{G} \text{ is } T\text{-tilable}\ \},\\
\mathcal{R} &:= \{\ (t, T) \in \mathcal{H} \mid \mathfrak{G} \text{ is } t\text{-recurrently } T\text{-tilable}\ \},\\
\mathcal{S} &:= \{\ T \in \mathcal{F} \mid \text{ there is a torus } \mathfrak{D} \text{ which is } T\text{-tilable}\ \}.
\end{aligned}$$

The *tiling problem* is the membership problem of the set $\mathcal{T}$ with the input set $\mathcal{F}$. The *recurrent tiling problem* is the membership problem of the set $\mathcal{R}$ with the input set $\mathcal{H}$. The *periodic tiling problem* is the membership problem of $\mathcal{S}$ with the input set $\mathcal{F}$.

▶ **Theorem 2.** *[7] The tiling problem is Π^0_1-complete.*

▶ **Theorem 3.** *[14] The recurrent tiling problem is Σ^1_1-complete.*

▶ **Theorem 4.** *[12] The periodic tiling problem is Σ^0_1-complete.*

▶ **Lemma 5.** *There is a computable function associating each input T to the (periodic) tiling problem with a first-order sentence φ_T of the vocabulary $\tau := \{H, V\} \cup T$ such that for all structures $\mathfrak{A}$ of the vocabulary $\{H, V\}$, the structure $\mathfrak{A}$ is T-tilable iff there exists an expansion $\mathfrak{A}^*$ of $\mathfrak{A}$ to the vocabulary τ such that $\mathfrak{A}^* \models \varphi_T$.*

Proof. Straightforward. ◀

▶ **Lemma 6.** *There is a computable function associating each input (t, T) of the recurrent tiling problem with a first-order sentence $\varphi_{(t,T)}$ of the vocabulary $\tau := \{H, V, R\} \cup T$ such that the grid $\mathfrak{G}$ is t-recurrently T-tilable iff there exists an expansion $\mathfrak{R}^*$ of the recurrence grid $\mathfrak{R}$ to the vocabulary τ such that $\mathfrak{R}^* \models \varphi_{(t,T)}$.*

Proof. Straightforward. ◀

It is easy to see that the grid $\mathfrak{G}$ is T-tilable iff there exists a supergrid $\mathfrak{G}'$ that is T-tilable.

3 Expressivity of universal MSO and weak universal MSO over affine real structures $(\mathbb{R}^n, \beta)$

In this section we investigate the expressive powers of ∀WMSO and ∀MSO. While it is rather easy to conclude that the two logics are incomparable in a rather strong sense (see Proposition 7), when attention is limited to structures $(\mathbb{R}^n, \beta, (R_i)_{i\in I})$ that expand the affine real structure $(\mathbb{R}^n, \beta)$, sentences of ∀WMSO translate into equivalent sentences of ∀MSO.

Let $\mathcal{L}$ and $\mathcal{L}'$ be fragments of second-order logic. We write $\mathcal{L} \leq \mathcal{L}'$, if for every vocabulary σ, any class of σ-structures definable by a σ-sentence of $\mathcal{L}$ is also definable by a σ-sentence of $\mathcal{L}'$. Let τ be a vocabulary such that $\beta \notin \tau$. The class of all expansions of $(\mathbb{R}^n, \beta)$ to the vocabulary $\{\beta\} \cup \tau$ is called the class of *affine real τ-structures*. Such structures can be regarded as τ-structures *embedded* in the geometric structure $(\mathbb{R}^n, \beta)$. We say that $\mathcal{L} \leq \mathcal{L}'$ *over* $(\mathbb{R}^n, \beta)$, if for every vocabulary τ s.t. $\beta \notin \tau$, any subclass definable w.r.t. the class $\mathcal{C}$ of all affine real τ-structures by a sentence of $\mathcal{L}$ is also definable w.r.t. $\mathcal{C}$ by a sentence of $\mathcal{L}'$.

▶ **Proposition 7.** *$\forall$WMSO $\not\leq$ MSO and $\forall$MSO $\not\leq$ WMSO.*

Proof. Finiteness is definable in $\exists$WMSO, and hence infinity is expressible in $\forall$WMSO. Infinity is not expressible in MSO. It is easy to show that $\forall$MSO can separate the structures $(\mathbb{R}, <)$ and $(\mathbb{Q}, <)$, while WMSO cannot. ◀

We then show that $\forall$WMSO $\leq$ $\forall$MSO and WMSO $\leq$ MSO over $(\mathbb{R}^n, \beta)$ for any $n \geq 1$.

▶ **Theorem 8** (Heine-Borel)**.** *A set $S \subseteq \mathbb{R}^n$ is closed and bounded iff every open cover of S has a finite subcover.*

▶ **Theorem 9.** *Let $\mathcal{C}$ be the class of expansions $(\mathbb{R}^n, \beta, P)$ of $(\mathbb{R}^n, \beta)$ with a unary predicate P, and let $\mathcal{F} \subseteq \mathcal{C}$ be the subclass of $\mathcal{C}$ where P is finite. The class $\mathcal{F}$ is first-order definable with respect to $\mathcal{C}$.*

Proof. We shall first establish that a set $T \subseteq \mathbb{R}^n$ is finite iff it is closed, bounded and consists of isolated points of T. Recall that an isolated point u of a set $U \subseteq \mathbb{R}^n$ is a point such that there exists some open ball B such that $B \cap U = \{u\}$.

Assume $T \subseteq \mathbb{R}^n$ is finite. Since T is finite, we can find a minimum distance between points in the set T. Therefore it is clear that each point t in T belongs to some open ball B such that $B \cap T = \{t\}$, and hence T consists of isolated points. Similarly, since T is finite, each point b in the complement of T has some minimum distance to the points of T, and therefore b belongs to some open ball $B \subseteq \mathbb{R}^n \setminus T$. Hence the set T is the complement of the union of open balls B such that $B \subseteq \mathbb{R}^n \setminus T$, and therefore T is closed. Finally, since T is finite, we can find a maximum distance between the points in T, and therefore T is bounded.

Assume then that $T \subseteq \mathbb{R}^n$ is closed, bounded and consists of isolated points of T. Since T consists of isolated points, it has an open cover $\mathcal{C} \subseteq \mathrm{Pow}(\mathbb{R}^n)$ such that each set in $\mathcal{C}$ contains exactly one point $t \in T$. The set $\mathcal{C}$ is an open cover of T, and by the Heine-Borel theorem, there exists a finite subcover $\mathcal{D} \subseteq \mathcal{C}$ of the set T. Since $\mathcal{D}$ is finite and each set in $\mathcal{D}$ contains exactly one point of T, the set T must also be finite.

We then conclude the proof by establishing that there exists a first-order formula $\varphi(P)$ stating that the unary predicate P is closed, bounded and consists of isolated points. We will first define a formula $\mathit{parallel}(x, y, t, k)$ stating that the lines defined by x, y and t, k are parallel in $(\mathbb{R}^n, \beta)$. We define

$$\begin{aligned} \mathit{parallel}(x,y,t,k) := x \neq y \wedge t \neq k \wedge \Big((\mathit{collinear}(x,y,t) \wedge \mathit{collinear}(x,y,k)) \\ \vee \big(\neg \exists z (\mathit{collinear}(x,y,z) \wedge \mathit{collinear}(t,k,z)) \\ \wedge \exists z_1 z_2 (x \neq z_1 \wedge \mathit{collinear}(x,y,z_1) \wedge \mathit{collinear}(x,t,z_2) \wedge \mathit{collinear}(z_1,z_2,k)) \big) \Big). \end{aligned}$$

We will then define first-order $\{\beta\}$-formulae $\mathit{basis}_k(x_0, \ldots, x_k)$ and $\mathit{flat}_k(x_0, \ldots, x_k, z)$ using simultaneous recursion. The first formula states that the vectors corresponding to the pairs (x_0, x_i), $1 \leq i \leq k$, form a basis of a k-dimensional flat. The second formula states the points z are exactly the points in the span of the basis defined by the vectors (x_0, x_i), the origin being x_0. First define $\mathit{basis}_0(x_0) := x_0 = x_0$ and $\mathit{flat}_0(x_0, z) := x_0 = z$. Then define flat_k

and $basis_k$ recursively in the following way.

$$
\begin{aligned}
&basis_k(x_0,\dots,x_k) := basis_{k-1}(x_0,\dots,x_{k-1}) \wedge \neg flat_{k-1}(x_0,\dots,x_{k-1},x_k),\\
&flat_k(x_0,\dots,x_k,z) := basis_k(x_0,\dots,x_k)\\
&\quad \wedge \exists y_0,\dots,y_k\Big(y_0 = x_0 \wedge y_k = z \wedge \bigwedge_{i\leq k-1} \big(y_i = y_{i+1} \vee parallel(x_0,x_{i+1},y_i,y_{i+1})\big)\Big).
\end{aligned}
$$

We then define a first-order $\{\beta, P\}$-formula $sepr(x,P)$ asserting that x belongs to an open ball B such that each point in $B \setminus \{x\}$ belongs to the complement of P. The idea is to state that there exist $n+1$ points $x_0,\dots,x_n$ that form an *n-dimensional triangle* around x, and every point contained in the triangle (with x being a possible exception) belongs to the complement of P. Every open ball in $\mathbb{R}^n$ is contained in some n-dimensional triangle in $\mathbb{R}^n$ and vice versa. We will recursively define first-order formulae $opentriangle_k(x_0,\dots,x_k,z)$ stating that z is properly inside a k-dimensional triangle defined by $x_0,\dots,x_k$. First define $opentriangle_1(x_0,x_1,z) := \beta^*(x_0,z,x_1)$, and then define

$$
\begin{aligned}
&opentriangle_k(x_0,\dots,x_k,z) := basis_k(x_0,\dots,x_k)\\
&\quad \wedge \exists y\big(opentriangle_{k-1}(x_0,\dots,x_{k-1},y) \wedge \beta^*(y,z,x_k)\big).
\end{aligned}
$$

We are now ready to define $sepr(x,P)$. Let

$$
\begin{aligned}
sepr(x,P) := \exists x_0,\dots,x_n\Big(&opentriangle_n(x_0,\dots,x_n,x)\\
&\wedge \forall y\big((opentriangle_n(x_0,\dots,x_n,y) \wedge y \neq x) \to \neg Py\big)\Big).
\end{aligned}
$$

Now, the sentence $\varphi_1 := \forall x\big(\neg Px \ \to \ sepr(x,P)\big)$ states that each point in the complement of P is contained in an open ball $B \subseteq \mathbb{R}^n \setminus P$. The sentence therefore states that the complement of P is a union of open balls. Since the set of unions of open balls is exactly the same as the set of open sets, the sentence states that P is closed.

The sentence $\varphi_2 := \forall x\big(Px \ \to \ sepr(x,P)\big)$ clearly states that P consists of isolated points.

Finally, in order to state that P is bounded, we define a formula asserting that there exist points $x_0,\dots,x_n$ that form an n-dimensional triangle around P.

$$
\varphi_3 := \exists x_0,\dots,x_n\Big(basis_n(x_0,\dots,x_n) \wedge \forall y\big(Py \to opentriangle_n(x_0,\dots,x_n,y)\big)\Big)
$$

The conjunction $\varphi_1 \wedge \varphi_2 \wedge \varphi_3$ states that P is finite. ◀

▶ **Corollary 10.** *Limit attention to expansions of* $(\mathbb{R}^n, \beta)$. *Sentences of* $\forall$WMSO *translate into equivalent sentences of* $\forall$MSO*, and sentences of* WMSO *into equivalent sentences of* MSO*.*

4 Undecidable theories of geometric structures with an affine betweenness relation

In this section we prove that the universal monadic second-order theory of any geometric structure (T,β) that extends linearly in $2D$ is undecidable. In addition we show that the universal monadic second-order theories of structures $(\mathbb{R}^n,\beta)$ with $n \geq 2$ are highly undecidable. In fact, we show that the theories of structures extending linearly in $2D$ are Σ^0_1-hard, while the theories of structures $(\mathbb{R}^n,\beta)$ with $n \geq 2$ are Π^1_1-hard—and therefore not even arithmetical. We establish the results by a reduction from the (recurrent) tiling problem to the problem of deciding whether a particular $\{\beta\}$-sentence of monadic Σ^1_1 is *satisfied* by (T, β) (respectively, $(\mathbb{R}^n, \beta)$). The argument is based on interpreting supergrids in corresponding $\{\beta\}$-structures.

4.1 Lines and sequences

Let $T \subseteq \mathbb{R}^n$. Let L be a line in T. Any nonempty subset Q of L is called a *sequence* in T. Let $E \subseteq T$ and $s, t \in T$. If $s \neq t$ and if $u \in E$ for all points $u \in T$ such that $\beta^*(s, u, t)$, we say that the points s and t are *linearly E-connected* (in (T, β)). If there exists a point $v \in T \setminus E$ such that $\beta^*(s, v, t)$, we say that s and t are *linearly disconnected with respect to E* (in (T, β)).

▶ **Definition 11.** Let Q be a sequence in $T \subseteq \mathbb{R}^n$. Suppose that for each $s, t \in Q$ such that $s \neq t$, there exists a point $u \in T \setminus \{s\}$ such that
1. $\beta(s, u, t)$ and
2. $\forall r \in T \ \big(\beta^*(s, r, u) \to r \notin Q\big)$, i.e., the points s and u are linearly $(T \setminus Q)$-connected.

Then we call Q a *discretely spaced sequence in T.*

▶ **Definition 12.** Let Q be a discretely spaced sequence in $T \subseteq \mathbb{R}^n$. Assume that there exists a point $s \in Q$ such that for each point $u \in Q$, there exists a point $v \in Q \setminus \{u\}$ such that $\beta(s, u, v)$. Then we call the sequence Q a *discretely infinite sequence in T*. The point s is called a *base point* of Q.

▶ **Definition 13.** Let Q be a sequence in $T \subseteq \mathbb{R}^n$. Let $s \in Q$ be a point such that there do not exist points $u, v \in Q \setminus \{s\}$ such that $\beta(u, s, v)$. Then we call Q a *sequence in T with a zero.* The point s is a *zero-point of* Q. Notice that Q may have up to two zero-points.

It is easy to see that a discretely infinite sequence has at most one zero point.

▶ **Definition 14.** Let Q be a discretely infinite sequence in $T \subseteq \mathbb{R}^n$ with a zero. Assume that for each $r \in T$ such that there exist points $s, u \in Q \setminus \{r\}$ with $\beta(s, r, u)$, there also exist points $s', u' \in Q \setminus \{r\}$ such that
1. $\beta(s', r, u')$ and
2. $\forall v \in T \setminus \{r\} \ \big(\beta^*(s', v, u') \to v \notin Q\big)$.

Then we call Q an *ω-like sequence in T* (cf. Lemma 17).

▶ **Lemma 15.** *Let P be a unary relation symbol. There is a first-order sentence $\varphi_\omega(P)$ of the vocabulary $\{\beta, P\}$ such that for every $T \subseteq \mathbb{R}^n$ and for every expansion (T, β, P) of (T, β), we have $(T, \beta, P) \models \varphi_\omega(P)$ if and only if the interpretation of P is an ω-like sequence in T.*

Proof. Straightforward. ◀

▶ **Definition 16.** Let P be a sequence in $T \subseteq \mathbb{R}^n$ and $s, t \in P$. The points s, t are called *adjacent* with respect to P, if the points are linearly $(T \setminus P)$-connected. Let $E \subseteq P \times P$ be the set of pairs (u, v) such that
1. u and v are adjacent with respect to P, and
2. $\beta(z, u, v)$ for some zero point z of P.

We call E the *successor relation* of P.

We let *succ* denote the successor relation of $\mathbb{N}$, i.e., $succ := \{\ (i, j) \in \mathbb{N} \times \mathbb{N} \mid i + 1 = j\ \}$.

▶ **Lemma 17.** *Let P be an ω-like sequence in $T \subseteq \mathbb{R}^n$ and E the successor relation of P. There is an embedding from $(\mathbb{N}, succ)$ into (P, E) such that $0 \in \mathbb{N}$ maps to the zero point of P. If $T = \mathbb{R}^n$, then $(\mathbb{N}, succ)$ is isomorphic to (P, E).*

Proof. We denote by i_0 the unique zero point of P. Since P is a discretely infinite sequence, it has a base point. Clearly i_0 has to be the only base point of P. It is straightforward to establish that since P is an ω-like sequence with the base point i_0, there exists a sequence $(a_i)_{i\in\mathbb{N}}$ of points $a_i \in P$ such that $i_0 = a_0$ and a_{i+1} is the unique E-successor of a_i for all $i \in \mathbb{N}$. Define the function $h : \mathbb{N} \to P$ such that $h(i) = a_i$ for all $i \in \mathbb{N}$. It is easy to see that h is an embedding of $(\mathbb{N}, succ)$ into (P, E).

Assume then that $T = \mathbb{R}^n$. We shall show that the function $h : \mathbb{N} \longrightarrow P$ is a surjection. Let d denote the canonical metric of $\mathbb{R}$, and let d_R be the restriction of the canonical metric of $\mathbb{R}^n$ to the line R in $\mathbb{R}^n$ such that $P \subseteq R$. Let $g : \mathbb{R} \longrightarrow R$ be the isometry from $(\mathbb{R}, d)$ to (R, d_R) such that $g(0) = i_0 = h(0)$ and such that for all $r \in ran(h)$, we have $\beta\big(i_0, g(1), r\big)$ or $\beta\big(i_0, r, g(1)\big)$. Let $(R, \leq^R)$ be the structure, where $\leq^R = \{\, (u, v) \in R \times R \mid g^{-1}(u) \leq^{\mathbb{R}} g^{-1}(v) \,\}$. If $ran(h)$ is not bounded from above w.r.t. $\leq^R$, then h must be a surjection. Therefore assume that $ran(h)$ is bounded above. By the Dedekind completeness of the reals, there exists a least upper bound $s \in R$ of $ran(h)$ w.r.t. $\leq^R$. Notice that since h is an embedding of $(\mathbb{N}, succ)$ into (P, E), we have $s \notin ran(h)$. Due to the definition of E, it is sufficient to show that $\{\, t \in P \mid s \leq^R t \,\} = \emptyset$ in order to conclude that h maps onto P.

Assume that the least upper bound s belongs to the set P. Since P is a discretely spaced sequence, there is a point $u \in \mathbb{R}^n \setminus \{s\}$ such that $\beta(s, u, i_0)$ and $\forall r \in \mathbb{R}^n \big(\beta^*(s, r, u) \to r \notin P\big)$. Now $u <^R s$ and the points u and s are linearly $(\mathbb{R}^n \setminus P)$-connected, implying that s cannot be the least upper bound of $ran(h)$. This is a contradiction. Therefore $s \notin P$.

Assume, ad absurdum, that there exists a point $t \in P$ such that $\beta(i_0, s, t)$. Now, since P is an ω-like sequence, there exists points $u', v' \in P \setminus \{s\}$ such that $\beta(u', s, v')$ and $\forall r \in \mathbb{R}^n \big(\beta^*(u', r, v') \to r \notin P\big)$. We have $\beta(s, u', i_0)$ or $\beta(s, v', i_0)$. Assume, by symmetry, that $\beta(s, u', i_0)$. Now $u' <^R s$, and the points u' and s are linearly $(\mathbb{R}^n \setminus P)$-connected. Hence, since $s \notin ran(h)$, we conclude that s is not the least upper bound of $ran(h)$. This is a contradiction. ◀

4.2 Geometric structures (T, β) with an undecidable monadic Π^1_1-theory

Let Q be an ω-like sequence in $T \subseteq \mathbb{R}^n$ and let q_0 be the unique zero point of Q. Assume there exists a point $q_e \in T \setminus Q$ such that $\beta(q_0, q, q_e)$ holds for all $q \in Q$. We call $Q \cup \{q_e\}$ *an ω-like sequence with an endpoint in T*. The point q_e is the *endpoint of* $Q \cup \{q_e\}$. Notice that the endpoint q_e is the only point x in $Q \cup \{q_e\}$ such that the following conditions hold.

1. There does not exist points $s, t \in Q \cup \{q_e\}$ such that $\beta^*(s, x, t)$.
2. $\forall yz \in Q \cup \{q_e\}\Big(\beta^*(x, y, z) \to \exists v \in Q \cup \{q_e\}\big(\beta^*(x, v, y)\big)\Big)$.

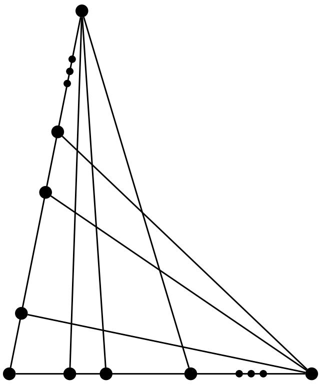


Figure 1 Illustration of how the grid is interpreted in a Cartesian frame.

▶ **Definition 18.** Let P and Q be ω-like sequences with an endpoint in $T \subseteq \mathbb{R}^n$. Let p_e and q_e be the endpoints of P and Q, respectively. Assume that the following conditions hold.

1. There exists a point $z \in P \cap Q$ such that z is the zero-point of both $P \setminus \{p_e\}$ and $Q \setminus \{q_e\}$.
2. There exists lines L_P and L_Q in T such that $L_P \neq L_Q$, $P \subseteq L_P$ and $Q \subseteq L_Q$.
3. For each point $p \in P \setminus \{p_e\}$ and $q \in Q \setminus \{q_e\}$, the unique lines L_p and L_q in T such that $p, q_e \in L_p$ and $q, p_e \in L_q$ intersect.

We call the structure (T, β, P, Q) a *Cartesian frame.*

▶ **Lemma 19.** *Let $T \subseteq \mathbb{R}^n$, $n \geq 2$, and let $\mathcal{C}$ be the class of all expansions (T, β, P, Q) of (T, β) by unary relations P and Q. The class of Cartesian frames with the domain T is definable with respect to $\mathcal{C}$ by a first-order sentence.*

Proof. Straightforward by virtue of Lemma 15. ◀

▶ **Lemma 20.** *Let $T \subseteq \mathbb{R}^n$, $n \geq 2$. Let $\mathcal{C}$ be the class of Cartesian frames with the domain T, and assume that $\mathcal{C}$ is nonempty. Let $\mathcal{G}$ be the class of supergrids and $\mathfrak{G}$ the grid. There is a class $\mathcal{A} \subseteq \mathcal{G}$ that is uniformly first-order interpretable in the class $\mathcal{C}$, and furthermore, $\mathfrak{G} \in \mathcal{A}$.*

Proof. Let $\mathfrak{C} = (T, \beta, P, Q)$ be a Cartesian frame. Let $p_e \in P$ and $q_e \in Q$ be the endpoints of P an Q, respectively. We shall interpret a supergrid $\mathfrak{G}_\mathfrak{C}$ in the Cartesian frame $\mathfrak{C}$. The domain of the interpretation of $\mathfrak{G}_\mathfrak{C}$ in $\mathfrak{C}$ will be the set of points where the lines that connect the points of $P \setminus \{p_e\}$ to q_e and the lines that connect the points of $Q \setminus \{q_e\}$ to p_e intersect.

First let us define the following formula which states in $\mathfrak{C}$ that x is the endpoint of P.

$$end_P(P, Q, x) \;:=\; Px \wedge \neg Qx \wedge \neg\exists y \exists z\big(Py \wedge Pz \wedge \beta^*(y, x, z)\big)$$

In the following, we let atomic expressions of the type $x \neq p_e$ and $\beta^*(x, y, q_e)$ abbreviate corresponding first-order formulae $\exists z\big(end_P(P, Q, z) \wedge x \neq z\big)$ and $\exists z\big(end_Q(Q, P, z) \wedge \beta^*(x, y, z)\big)$ of the vocabulary $\{\beta, P, Q\}$ of $\mathfrak{C}$. We define

$$\begin{aligned}
\varphi_{Dom}(u) \;&:= u \neq p_e \wedge u \neq q_e \\
&\quad \wedge \Big(Pu \vee Qu \vee\ \exists xy\big(Px \wedge x \neq p_e \wedge Qy \wedge y \neq q_e \wedge \beta(x, u, q_e) \wedge \beta(y, u, p_e)\big)\Big), \\
\varphi_H(u, v) \;&:= \exists x\big(Qx \,\wedge\, \beta(x, u, v) \,\wedge\, \beta^*(u, v, p_e)\big) \wedge \forall r\big(\,\beta^*(u, r, v) \,\rightarrow\, \neg\varphi_{Dom}(r)\,\big), \\
\varphi_V(u, v) \;&:= \exists x\big(Px \,\wedge\, \beta(x, u, v) \,\wedge\, \beta^*(u, v, q_e)\big) \wedge \forall r\big(\,\beta^*(u, r, v) \,\rightarrow\, \neg\varphi_{Dom}(r)\,\big).
\end{aligned}$$

Call $D_\mathfrak{C} := \{\ r \in T \mid \mathfrak{C} \models \varphi_{Dom}(r)\ \}$ and define the structure $\mathfrak{D}_\mathfrak{C} = (D_\mathfrak{C}, H^{\mathfrak{D}_\mathfrak{C}}, V^{\mathfrak{D}_\mathfrak{C}})$, where

$$H^{\mathfrak{D}_\mathfrak{C}} \;:=\; \{\ (s, t) \in D_\mathfrak{C} \times D_\mathfrak{C} \mid \mathfrak{C} \models \varphi_H(s, t)\ \},$$

and analogously for $V^{\mathfrak{D}_\mathfrak{C}}$. By Lemma 17, it is easy to see that there exists an injection f from the domain of the grid $\mathfrak{G} = (G, H, V)$ to $D_\mathfrak{C}$ such that the following three conditions hold for all $u, v \in G$.

1. $(u, v) \in H \;\Leftrightarrow\; \varphi_H\big(f(u), f(v)\big)$,
2. $(u, v) \in V \;\Leftrightarrow\; \varphi_V\big(f(u), f(v)\big)$.

Hence there is a supergrid $\mathfrak{G}_\mathfrak{C} = (G_\mathfrak{C}, H, V)$ such that there exists an isomorphism f from $G_\mathfrak{C}$ to $D_\mathfrak{G}$ such that the above two conditions hold.

Let $\mathcal{A} := \{\ \mathfrak{G}_\mathfrak{C} \in \mathcal{G} \mid \mathfrak{C}$ is a Cartesian frame with the domain $T\ \}$. Clearly $\mathfrak{G} \in \mathcal{A}$, and furthermore, $\mathcal{A}$ is uniformly first-order interpretable in the class of Cartesian frames with the domain T. ◀

▶ **Lemma 21.** *Let $n \geq 2$ be an integer. The recurrence grid $\mathfrak{R}$ is uniformly first-order interpretable in the class of Cartesian frames with the domain $\mathbb{R}^n$.*

Proof. Straightforward by Lemma 17 and the proof of Lemma 20. ◀

▶ **Theorem 22.** *Let $T \subseteq \mathbb{R}^n$ be a set and let β be the corresponding betweenness relation. Assume that T extends linearly in $2D$. The monadic Π^1_1-theory of (T, β) is Σ^0_1-hard.*

Proof. Since T extends linearly in $2D$, we have $n \geq 2$. Let $\sigma = \{H, V\}$ be the vocabulary of supergrids, and let $\tau = \{\beta, X, Y\}$ be the vocabulary of Cartesian frames. By Lemma 19, there exists a first-order τ-sentence that defines the class of Cartesian frames with the domain T with respect to the class of all expansions of (T, β) to the vocabulary τ. Let φ_{Cf} denote such a sentence.

By Lemma 5, there is a computable function that associates each input S to the tiling problem with a first-order $\sigma \cup S$-sentence φ_S such that a structure $\mathfrak{A}$ of the vocabulary σ is S-tilable if and only if there is an expansion $\mathfrak{A}^*$ of the structure $\mathfrak{A}$ to the vocabulary $\sigma \cup S$ such that $\mathfrak{A}^* \models \varphi_S$.

Since T extends linearly in $2D$, the class of Cartesian frames with the domain T is nonempty. By Lemma 20 there is a class of supergrids $\mathcal{A}$ such that $\mathfrak{G} \in \mathcal{A}$ and $\mathcal{A}$ is uniformly first-order interpretable in the class of Cartesian frames with the domain T. Therefore there exists a uniform interpretation I' of $\mathcal{A}$ in the class of Cartesian frames with the domain T. Let S be a finite nonempty set of tiles. Note that S is by definition a set of proposition symbols P_t, where t is a tile type. Let I be the S-expansion of the uniform interpretation I' of $\mathcal{A}$ in the class of Cartesian frames with the domain T.

Define $\psi_S := \exists X \, \exists Y \, (\exists P_t)_{P_t \in S} \big(\varphi_{Cf} \, \wedge \, I(\varphi_S \,) \big)$. We will prove that for each input S to the tiling problem, we have $(T, \beta) \models \psi_S$ if and only if the grid $\mathfrak{G}$ is S-tilable. Thereby we establish that there exists a computable reduction from the *complement problem* of the tiling problem to the membership problem of the monadic Π^1_1-theory of (T, β). Since the tiling problem is Π^0_1-complete, its complement problem is Σ^0_1-complete.[1]

Let S be an input to the tiling problem. Assume first that there exists an S-tiling of the grid $\mathfrak{G}$. Therefore there exists an expansion $\mathfrak{G}^*$ of the grid $\mathfrak{G}$ to the vocabulary $\{H, V\} \cup S$ such that $\mathfrak{G}^* \models \varphi_S$. Hence, by Lemma 1 and since $\mathfrak{G} \in \mathcal{A}$, there exists a Cartesian frame $\mathfrak{C}$ with the domain T such that for some expansion $\mathfrak{C}^*$ of $\mathfrak{C}$ to the vocabulary $\{\beta, X, Y\} \cup S$, we have $\mathfrak{C}^* \models I(\varphi_S)$. On the other hand, since $\mathfrak{C}$ is a Cartesian frame, we have $\mathfrak{C}^* \models \varphi_{Cf}$. Therefore $\mathfrak{C}^* \models \varphi_{Cf} \wedge I(\varphi_S)$, and hence $(T, \beta) \models \psi_S$.

For the converse, assume that $(T, \beta) \models \psi_S$. Therefore there exists an expansion $\mathfrak{B}^*$ of (T, β) to the vocabulary $\{\beta, X, Y\} \cup S$ such that we have $\mathfrak{B}^* \models \varphi_{Cf} \, \wedge \, I(\varphi_S)$. Since $\mathfrak{B}^* \models \varphi_{Cf}$, the $\{\beta, X, Y\}$-reduct of $\mathfrak{B}^*$ is a Cartesian frame with the domain T. Therefore, we conclude by Lemma 1 that $\mathfrak{A}^* \models \varphi_S$ for some expansion $\mathfrak{A}^*$ of some supergrid $\mathfrak{A} \in \mathcal{A}$ to the vocabulary $\{H, V\} \cup S$. Thus there exists a supergrid that S-tilable. Hence the grid $\mathfrak{G}$ is S-tilable. ◀

▶ **Corollary 23.** *Let $T \subseteq \mathbb{R}^n$ be such that T extends linearly in $2D$. Let $\mathcal{C}$ be the class of expansions $(T, \beta, (P_i)_{i \in \mathbb{N}})$ of (T, β) with arbitrary unary predicates. The first-order theory of $\mathcal{C}$ is undecidable.*

We note that T extending linearly in $1D$ is not a sufficient condition for undecidability of the monadic Π^1_1-theory of (T, β). The monadic Π^1_1-theory of $(\mathbb{R}, \beta)$ is decidable; this follows trivially from the known result that the monadic Π^1_1-theory $(\mathbb{R}, \leq)$ is decidable, see [9]. Also

[1] It is of course a well-known triviality that the complement $\overline{A}$ of a problem A is Σ^0_1-hard if A is Π^0_1-hard. Choose an arbitrary problem $B \in \Sigma^0_1$. By definition $\overline{B} \in \Pi^0_1$. By the hardness of A, there is a computable reduction f such that $x \in \overline{B} \Leftrightarrow f(x) \in A$, whence $x \in B \Leftrightarrow f(x) \in \overline{A}$.

the monadic Π^1_1-theory of $(\mathbb{Q}, \beta)$ is decidable since the MSO theory of $(\mathbb{Q}, \leq)$ is decidable [19].

▶ **Theorem 24.** *Let $n \geq 2$ be an integer. The monadic Π^1_1-theory of the structure $(\mathbb{R}^n, \beta)$ is Π^1_1-hard.*

Proof. The proof is essentially the same as the proof of Theorem 22. The main difference is that we use Lemma 21 and interpret the recurrence grid $\mathfrak{R}$ instead of a class of supergrids and hence obtain a reduction from the recurring tiling problem instead of the ordinary tiling problem. Thereby we establish Π^1_1-hardness instead of Σ^0_1-hardness. Due to the recurrence condition of the recurrent tiling problem, the result of Lemma 17 that there is an isomorphism from $(\mathbb{N}, succ)$ to (P, E)—rather than an embedding—is essential. ◀

▶ **Corollary 25.** *Let $n \geq 2$ be an integer. Let $\mathcal{C}$ be the class of expansions $(\mathbb{R}^n, \beta, (P_i)_{i \in \mathbb{N}})$ of $(\mathbb{R}^n, \beta)$ with arbitrary unary predicates. The first-order theory of $\mathcal{C}$ is not on any level of the arithmetical hierarchy.*

5 Geometric structures (T, β) with an undecidable weak monadic Π^1_1-theory

In this section we prove that the weak universal monadic second-order theory of any structure (T, β) such that T extends linearly in $2D$ is undecidable. In fact, we show that any such theory is Π^0_1-hard. We establish this by a reduction from the periodic tiling problem to the problem of deciding truth of $\{\beta\}$-sentences of weak monadic Σ^1_1 in (T, β). The argument is based on interpreting tori in (T, β). Most notions used in this section are inherited either directly or with minor modification from Section 4.

Let Q be a subset of $T \subseteq \mathbb{R}^n$. We say that Q is a *finite sequence* in T if Q is a finite nonempty set and the points in Q are all collinear.

▶ **Definition 26.** Let $T \subseteq \mathbb{R}^n$ and let β be the corresponding betweenness relation. Let P and Q be finite sequences in T such that the following conditions hold.

1. $P \cap Q = \{a_0\}$, where a_0 is a zero point of both P and Q.
2. P and Q are non-singleton sequences.
3. There exists lines L_P, L_Q in T such that $L_P \neq L_Q$, $P \subseteq L_P$ and $Q \subseteq L_Q$.

We call the structure (T, β, P, Q) *a finite Cartesian frame* with the domain T.

▶ **Lemma 27.** *Let $T \subseteq \mathbb{R}^n$, $n \geq 2$. Let $\mathcal{C}$ be the class of all expansions (T, β, P, Q) of (T, β) by finite unary relations P and Q. The class of finite Cartesian frames with the domain T is definable with respect to $\mathcal{C}$ by a first-order sentence.*

Proof. Straightforward. ◀

▶ **Lemma 28.** *Let $T \subseteq \mathbb{R}^n$, $n \geq 2$. Assume that T extends linearly in $2D$. The class of tori is uniformly first-order interpretable in the class of finite Cartesian frames with the domain T.*

Proof. The proof is similar to that of Lemma 20. ◀

▶ **Theorem 29.** *Let $T \subseteq \mathbb{R}^n$ and let β be the corresponding betweenness relation. Assume that T extends linearly in $2D$. The weak monadic Π^1_1-theory of (T, β) is Π^0_1-hard.*

Proof. The proof is based on the above two lemmas and is analogous to the proof of Theorem 22. ◀

▶ **Corollary 30.** *Let $T \subseteq \mathbb{R}^n$ be a set such that T extends linearly in $2D$. Let $\mathcal{C}$ be the class of expansions $(T, \beta, (P_i)_{i\in\mathbb{N}})$ of (T, β) with finite unary predicates. The first-order theory of $\mathcal{C}$ is undecidable.*

6 Conclusions

We have studied first-order theories of geometric structures (T, β), $T \subseteq \mathbb{R}^n$, expanded with (finite) unary predicates. We have established that for $n \geq 2$, the first-order theory of the class of all expansions of $(\mathbb{R}^n, \beta)$ with arbitrary unary predicates is highly undecidable (Π^1_1-hard). This refutes a conjecture from the article [1] of Aiello and van Benthem. In addition, we have established the following for any geometric structure (T, β) that extends linearly in $2D$.

1. The first-order theory of the class of expansions of (T, β) with arbitary unary predicates is Σ^0_1-hard.
2. The first-order theory of the class of expansions of (T, β) with finite unary predicates is Π^0_1-hard.

Geometric structures that extend linearly in $2D$ include, for example, the rational plane $(\mathbb{Q}^2, \beta)$ and the real unit rectangle $([0, 1]^2, \beta)$, to name a few.

The techniques used in the proofs can be easily modified to yield undecidability of first-order theories of a significant variety of natural restricted expansion classes of the affine real plane $(\mathbb{R}^2, \beta)$, such as those with unary predicates denoting polygons, finite unions of closed rectangles, and real algebraic sets, for example. Such classes could be interesting from the point of view of applications.

In addition to studying issues of decidability, we briefly compared the expressivities of universal monadic second-order logic and weak universal monadic second-order logic. While the two are incomparable in general, we established that over any class of expansions of $(\mathbb{R}^n, \beta)$, it is no longer the case. We showed that finiteness of a unary predicate is definable by a first-order sentence, and hence obtained translations from ∀WMSO into ∀MSO and from WMSO into MSO.

Our original objective to study weak monadic second order logic over $(\mathbb{R}^n, \beta)$ was to identify decidable logics of space with distinguished regions. Due to the ubiquitous applicability of the tiling methods, this pursuit gave way to identifying several undecidable theories of geometry. Hence we shall look elsewhere in order to identify well behaved natural decidable logics of space. Possible interesting directions include considering natural fragments of first-order logic over expansions of $(\mathbb{R}^n, \beta)$, and also other geometries. Related results could provide insight, for example, in the background theory of modal spatial logics.

References

1 M. Aiello and J. van Benthem. A Modal Walk through Space. *Journal of Applied Non-Classical Logics* 12(3-4):319-363, Hermes, 2002.

2 M. Aiello, I. Pratt-Hartmann, and J. van Benthem. What is Spatial Logic. In Marco Aiello, Ian Pratt-Hartmann and Johan van Benthem, *Handbook of Spatial Logics*, Springer, 2007.

3 M. Aiello, I. Pratt-Hartmann and J. van Benthem. Handbook of Spatial Logics. Springer, 2007.

4 P. Balbiani, L. Farinas del Cerro, T. Tinchev, and D. Vakarelov. Modal Logics for Incidence Geometries, *Journal of Logic and Computation*, 7(1), 59-78, 1997.

5 P. Balbiani and V. Goranko. Modal logics for parallelism, orthogonality, and affine geometries, *Journal of Applied Non-Classical Logics*, 12,365-397, 2002.

6 P. Balbiani, V. Goranko, R. Kellerman and D. Vakarelov. Logical Theories for Fragments of Elementary Geometry. In *Handbook of Spatial Logics*. Springer. 343-428, 2007.

7 R. Berger. The undecidability of the domino problem. *Mem. Amer. Math. Soc.*, 66, 1966.

8 J. Bochnak, M. Coste and M. Roy. *Real Algebraic Geometry*, Springer, 1998.

9 J. P. Burgess and Y. Gurevich. The Decision Problem for Linear Temporal Logic. *Notre Dame Journal of Formal Logic*, vol. 26, no. 2, 1985.

10 B. ten Cate and A. Facchini. Characterizing EF over Infnite Trees and Modal Logic on Transitive Graphs. *Proceedings of the MFCS*, 2011.

11 A. Griffiths. Computational Properties of Spatial Logics in the Real Plane. PhD thesis, University of Manchester, 2008.

12 Y. Gurevich and I. O. Koryakov. Remarks on Berger's paper on the domino problem. *Siberian Mathematical Journal 13*, 319-321, 1972.

13 M. Gyssens, J. Van den Bussche and D. Van Gucht. Complete Geometric Query Languages. *Journal of Computer and System Sciences 58*, 483-511, 1999.

14 D. Harel. Recurring Dominoes: Making the Highly Undecidable Highly Understandable. *Annals of Discrete Mathematics 24*, 51-72, 1985.

15 I. Hodkinson and A. Hussain. The modal logic of affine planes is not finitely axiomatisable. *Journal of Symbolic Logic* 73(3), 940-952, 2008.

16 R. Kontchakov, I. Pratt-Hartmann, F. Wolter and M. Zakharyaschev. Spatial logics with connectedness predicates. *Logical Methods in Computer Science*, 6(3), 2010.

17 B. Kujpers and J. Van den Bussche. Logical aspects of spatial database theory. In *Finite and Algorithmic Model Theory*, London Mathematical Society Lecture Notes Series 379, Cambridge University Press, 2011.

18 Y. Nenov and I. Pratt-Hartmann. On the Computability of Region-Based Euclidean Logics. In Proceedings of *19th EACSL Annual Conferences on Computer Science Logic*, 2010.

19 M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. of the Amer. Math. Soc.* 141, 1-35, 1969.

20 M. Sheremet, F. Wolter and M. Zakharyaschev. A modal logic framework for reasoning about comparative distances and topology. *Ann. Pure Appl. Logic*, 161(4):534-559, 2010.

21 A. Tarski. A decision method for elementary algebra and geometry. RAND Corporation, Santa Monica, 1948.

22 A. Tarski and S. Givant. Tarski's System of Geometry. *Bull. Symbolic Logic* 5(2), 1999.

23 T. Tinchev and D. Vakarelov. Logics of Space with Connectedness Predicates: Complete Axiomatizations. In *Proceedings of Advances in Modal Logic 8 (AiML)*, 434-453, 2010.

24 Y. Venema. Points, Lines and Diamonds: a Two-sorted Modal Logic for Projective Planes. *Journal of Logic and Computation*, 9(5) 601-621, 1999.

Towards CERes in intuitionistic logic

Alexander Leitsch, Giselle Reis, and Bruno Woltzenlogel Paleo

Theory and Logic Group, Institut für Computersprachen
Technische Universität Wien, Vienna, Austria
`{leitsch, giselle, bruno}@logic.at`

Abstract

Cut-elimination, introduced by Gentzen, plays an important role in automating the analysis of mathematical proofs. The removal of cuts corresponds to the elimination of intermediate statements (lemmas), resulting in an analytic proof. **CERes** is a method of cut-elimination by resolution that relies on global proof transformations, in contrast to reductive methods, which use local proof-rewriting transformations. By avoiding redundant operations, it obtains a speed-up over Gentzen's traditional method (and its variations). **CERes** has been successfully implemented and applied to mathematical proofs, and it is fully developed for classical logic (first and higher order), multi-valued logics and Gödel logic. But when it comes to mathematical proofs, intuitionistic logic also plays an important role due to its constructive characteristics and computational interpretation.

This paper presents current developments on adapting the **CERes** method to intuitionistic sequent calculus **LJ**. First of all, we briefly describe the **CERes** method for classical logic and the problems that arise when extending the method to intuitionistic logic. Then, we present the solutions found for the mentioned problems for the subclass $\mathbf{LJ}^-$ (the class of intuitionistic proofs of an end-sequent containing no strong quantifiers and no formula on the right). In addition, we explain, with an example, some ideas for improving the method and covering a bigger fragment of **LJ** proofs. Finally, we summarize the results and point the direction for future research.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases cut-elimination, resolution, **LJ**

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.485

1 Introduction

Proof analysis is an essential part of mathematical activity, since it leads often to better proofs and occasionally to the discovery of important new mathematical concepts that allow the structuring of existing arguments [15]. Abstract notions such as *groups* and *probability*, for instance, are clear examples of concepts that are undoubtfully useful for organizing common patterns of mathematical reasoning.

The elimination of unnecessary lemmas from a proof is a prominent example of a technique for obtaining potentially simpler (or at least different) proofs. When a constructive mathematical proof is formalized in the sequent calculus **LJ**, lemmas correspond to cuts.

$$\frac{\Gamma \vdash A \quad \Gamma', A \vdash C}{\Gamma, \Gamma' \vdash C}\ cut$$

A proof without cuts has the subformula property: all formulas on the proof are (instances of) subformulas of end-sequent formulas. Consequently, cut-free proofs of a theorem will use only the theorem's theory itself. The main result on cut-elimination - the *Hauptsatz* - was proven by Gentzen [8, 9] in 1935. It states that the cut rule is admissible for the sequent calculi **LK** and **LJ**, for classical and intuitionistic logics respectively. Gentzen's proof of the *Hauptsatz* actually contains an algorithm for removing the cuts from a proof. Therefore, cut-elimination, seen as a method to remove lemmas from formalized proofs, is one of the most important techniques for automating proof analysis. Gentzen's method and some of its variants are often referred to as *reductive cut-elimination*, because they are based on local proof-rewriting rules that gradually reduce the grade and rank of cuts.

The method **CERes** [5] (cut elimination by resolution) is an alternative to reductive cut-elimination, and it is proven to display a non-elementary speed up over the latter. **CERes** was first developed for first order classical logic, and then extended to second and higher order logic [12, 11]. It has also been adapted to multi-valued logics [6] and Gödel logic [1]. The method has been implemented[1] and applied successfully to proofs of moderate size, such as the tape proof [2] and the lattice proof [13], in fully automatic mode. Also, Fürstenberg's proof of the infinitude of primes was successfully transformed, semi-automatically, into Euclid's argument of prime construction using **CERes** [3].

Intuitionistic logic, in contrast to classical logic, is based on a natural proof semantics [10] which is reflected in the rules of natural deduction. Consequently, from an intuitionistic proof of $A \vee B$, one can actually obtain a proof of one of the disjuncts, and from an intuitionistic proof of $\exists x.P(x)$, one can obtain a *witness* a such that $P(a)$ is true. This is not always the case in classical proofs. For this reason, intuitionistic logic is often referred to as a constructive logic. This is particularly useful in mathematics when one wants not only to guarantee the existence of a solution but to actually find it. This constructivism also makes intuitionistic logic more suitable for modeling computations, since constructive proofs can be directly related to algorithms.

The importance of intuitionistic logic for mathematics and computer science is the main motivation for extending the **CERes** method to **LJ**. This paper presents the results obtained so far, while pursuing this goal. More specifically, we present the **CERes** method for a subclass of **LJ** proofs, namely, proofs with end sequents having no strong quantifiers and no formula on the right side. This class represents proofs by contradiction in **LJ**. Observe that a proof of the end sequent $\Gamma \vdash F$ can be transformed into a proof by contradiction by applying the $\neg_l$ rule and obtaining $\Gamma, \neg F \vdash$ as an end-sequent.

The paper is organized as follows: Section 2 briefly describes the **CERes** method for classical logic and the problems that arise when extending the method to intuitionistic logic; Section 3 presents the solutions found for the mentioned problems and shows the new revised method applied to an example; Section 4 explains some ideas for improving the method and covering a bigger fragment of **LJ** proofs; and finally, Section 5 summarizes the results and points the direction for future research.

2 CERes in LK

The **CERes** method for classical logic is based on the computation of three structures from an **LK** proof φ: a characteristic clause set $CL(\varphi)$, a resolution refutation of this set and proof projections of φ w.r.t the elements in $CL(\varphi)$. By merging instances of the projections and the resolution refutation properly, one obtains a proof with only atomic cuts (ACNF - atomic cut normal form) of the same end sequent of φ. These three structures are informally explained in the subsections below. A more detailed and precise definition of **CERes** for **LK** is available in [7].

[1] http://code.google.com/p/gapt/

The remaining atomic cuts on the final proof are inessential [16], and, since we use standard axioms ($A \vdash A$, where A is atomic), these can be eliminated using reductive cut-elimination.

2.1 Characteristic clause set

The characteristic clause set is computed by removing from φ all the rules that operate on end-sequent ancestors and the end-sequent ancestors themselves (including the end-sequent). After that, what is left is a derivation of the empty sequent from a set of axioms. These axioms contain only cut-ancestors and they compose the characteristic clause set. It is important to note that some branches of φ might be merged during this procedure, if they resulted from the application of a binary rule on an end-sequent ancestor. Consider the sub-derivation of a proof below, in which cut ancestors are marked with $\star$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(a)^\star \vdash P(a)}\,I \quad \dfrac{}{Q(a) \vdash Q(a)^\star}\,I}{P(a)^\star, P(a) \to Q(a) \vdash Q(a)^\star}\,\to_l}{P(a) \to Q(a) \vdash (P(a) \to Q(a))^\star}\,\to_r}{P(a) \to Q(a) \vdash \exists x.(P(x) \to Q(x))^\star}\,\exists_r \quad \dfrac{\dfrac{\dfrac{\dfrac{}{P(b)^\star \vdash P(b)}\,I \quad \dfrac{}{Q(b) \vdash Q(b)^\star}\,I}{P(b)^\star, P(b) \to Q(b) \vdash Q(b)^\star}\,\to_l}{P(b) \to Q(b) \vdash (P(b) \to Q(b))^\star}\,\to_r}{P(b) \to Q(b) \vdash \exists x.(P(x) \to Q(x))^\star}\,\exists_r}{(P(a) \to Q(a)) \lor (P(b) \to Q(b)) \vdash \exists x.(P(x) \to Q(x))^\star}\,\lor_l$$

By removing all inferences on end-sequent ancestors, we obtain the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P(a)^\star, P(b)^\star \vdash Q(a)^\star, Q(b)^\star}\,I}{P(a)^\star \vdash Q(a)^\star, (P(b) \to Q(b))^\star}\,\to_r}{\vdash (P(a) \to Q(a))^\star, (P(b) \to Q(b))^\star}\,\to_r}{\vdash (P(a) \to Q(a))^\star, \exists x.(P(x) \to Q(x))^\star}\,\exists_r}{\vdash \exists x.(P(x) \to Q(x))^\star, \exists x.(P(x) \to Q(x))^\star}\,\exists_r$$

In this case, the sequent $P(a), P(b) \vdash Q(a), Q(b)$ would be in the characteristic clause set.

▶ Remark. Observe that, in classical logic, this is not a problem, but in intuitionistic logic (**LJ** - Figure 1) this is not a well-formed sequent since it has more than one formula on the right side. Note also that the original derivation could easily be part of an **LJ** proof, but the transformed derivation contains non-intuitionistic sequents. Thus, sequents of the characteristic clause set might be classical, even if we start with an intuitionistic proof. As they will be part of the final proof, it is not desirable that they are classical, because we expect to obtain a cut-free proof in **LJ**.

2.2 Resolution refutation

By the transformation exemplified above, there exists an **LK** derivation of the empty sequent from the clauses of $CL(\varphi)$. Since **LK** is sound, the set $CL(\varphi)$ is unsatisfiable. And since the resolution calculus is complete, there exists a resolution refutation of $CL(\varphi)$.

The resolution refutation, which can be obtained with a resolution theorem prover, is used as a basis for the final proof, and can be seen as its skeleton. The resolution steps will correspond to the atomic cuts.

2.3 Projections

The projections are derivations of a sequent from $CL(\varphi)$ merged with the end-sequent. Dually to what was done for the characteristic clause set, they are constructed by removing the rules

$$\dfrac{}{A \vdash A}\,[\mathrm{I}] \qquad \dfrac{\Gamma_1 \vdash P \quad \Gamma_2, P \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\,[\mathrm{Cut}]$$

$$\dfrac{\Gamma \vdash P}{\Gamma, \neg P \vdash}\,[\neg_l] \qquad \dfrac{\Gamma, P \vdash}{\Gamma \vdash \neg P}\,[\neg_r]$$

$$\dfrac{P_i, \Gamma \vdash C}{P_1 \wedge P_2, \Gamma \vdash C}\,[\wedge_{li}] \qquad \dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}\,[\wedge_r]$$

$$\dfrac{P, \Gamma \vdash C \quad Q, \Gamma \vdash C}{P \vee Q, \Gamma \vdash C}\,[\vee_l] \qquad \dfrac{\Gamma \vdash P_i}{\Gamma \vdash P_1 \vee P_2}\,[\vee_{ri}]$$

$$\dfrac{\Gamma_1 \vdash P \quad Q, \Gamma_2 \vdash C}{P \to Q, \Gamma_1, \Gamma_2 \vdash C}\,[\to_l] \qquad \dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \to Q}\,[\to_r]$$

$$\dfrac{P\{x \leftarrow \alpha\}, \Gamma \vdash C}{\exists x.P, \Gamma \vdash C}\,[\exists_l] \qquad \dfrac{\Gamma \vdash P\{x \leftarrow t\}}{\Gamma \vdash \exists x.P}\,[\exists_r]$$

$$\dfrac{P\{x \leftarrow t\}, \Gamma \vdash C}{\forall x.P, \Gamma \vdash C}\,[\forall_l] \qquad \dfrac{\Gamma \vdash P\{x \leftarrow \alpha\}}{\Gamma \vdash \forall x.P}\,[\forall_r]$$

$$\dfrac{P, P, \Gamma \vdash C}{P, \Gamma \vdash C}\,[C_l] \qquad \dfrac{\Gamma \vdash C}{P, \Gamma \vdash C}\,[W_l] \qquad \dfrac{\Gamma \vdash}{\Gamma \vdash P}\,[W_r]$$

Figure 1 LJ: Sequent calculus for intuitionistic logic. It is assumed that α is a variable not contained in P, Γ or C and t does not contain variables bound in P.

applied to cut-ancestors. Each sequent (clause) in $CL(\varphi)$ will generate a projection, possibly with variables that can later be instantiated to form the final proof. Using the same example as before, the projection corresponding to the clause $P(a), P(b) \vdash Q(a), Q(b)$ is the following:

$$\dfrac{\dfrac{\dfrac{}{P(a)^\star \vdash P(a)}\,I \quad \dfrac{}{Q(a) \vdash Q(a)^\star}\,I}{P(a)^\star, P(a) \to Q(a) \vdash Q(a)^\star}\,\to_l \quad \dfrac{\dfrac{}{P(b)^\star \vdash P(b)}\,I \quad \dfrac{}{Q(b) \vdash Q(b)^\star}\,I}{P(b)^\star, P(b) \to Q(b) \vdash Q(b)^\star}\,\to_l}{P(a)^\star, P(b)^\star, (P(a) \to Q(a)) \vee (P(b) \to Q(b)) \vdash Q(a)^\star, Q(b)^\star}\,\vee_l$$

▶ Remark. Note that, once again, the resulting derivation is classical. Since these will be directly used on the final proof, it is also a problem that should be solved if we expect the output of **CERes** to be intuitionistic when applied to **LJ** proofs.

Even if the resulting projections were intuitionistic, they are merged with the resolution refutation of $CL(\varphi)$, and if two sequents with one formula on the right side are merged, the resulting sequent will have two formulas on the right side and will then be classical.

3 CERes in LJ

The problems described in Section 2 were addressed and solved for a subclass of **LJ**, namely **LJ**$^-$ (Definition 2). The resulting **iCERes** method is presented in this section.

▶ **Definition 1** (Strong and weak quantifiers)**.** Let F be a formula. If $\forall x$ occurs positively (negatively) in F, then $\forall x$ is called a *strong (weak) quantifier*. If $\exists x$ occurs positively (negatively) in F, then $\exists x$ is called a *weak (strong) quantifier*. Let $A_1, ..., A_n \vdash B_1, ..., B_m$ be a sequent. A quantifier is called strong (weak) on this sequent if it is strong (weak) on the corresponding formula $A_1 \wedge ... \wedge A_n \rightarrow B_1 \vee ... \vee B_m$.

▶ **Definition 2** (**LJ**$^-$)**.** **LJ**$^-$ is the set of **LJ** proofs of end-sequents with no formula on the right side and no strong quantifiers.

Note that, in principle, the condition of absence of formulas on the right side of the end-sequent, can be satisfied by simply applying the $\neg_l$ inference rule at the bottom of the proof, in order to negate and shift to the left the formula that occurred in the right side. The other requirement, the absence of strong quantifiers, can be achieved by using methods of skolemization of **LJ** proofs [4]. A more detailed discussion of the implications of using these transformations is postponed to Section 4.

This class contains proofs by contradiction in **LJ**, which are exactly those proofs of $\Gamma \vdash F$ transformed into a proof of $\Gamma, \neg F \vdash$ with the application of a $\neg_l$ rule. It is also worth mentioning that **LJ**$^-$ is a "nontrivial" class of proofs, in the sense that there exist sequents of the form $\Gamma \vdash$ which are provable classically but not intuitionistically (e.g. $\neg\exists x.\forall y.(P(x) \rightarrow P(y)) \vdash$).

Although the problem was solved only for a subclass of **LJ** proofs, all definitions and proofs on this section are valid for full **LJ**, with the exception of Theorem 16.

▶ **Definition 3** (Intuitionistic Clause)**.** An *intuitionistic clause* is a sequent composed only of atoms or negated atoms and with the right hand side containing at most one formula.

▶ **Definition 4** (Intuitionistic Clause Set with Negations)**.** The *intuitionistic characteristic clause set* is built analogously to the usual characteristic clause set, except that all the formulas on the right hand side are negated and added to the left hand side:

- If ν is an axiom, then $CL^I(\nu)$ is the set containing the sub-sequent composed only of the formulas that are cut-ancestors, such that all the formulas that would appear on the right-hand side are negated and added to the left-hand side.
- If ν is the result of the application of a unary rule on μ, then $CL^I(\nu) = CL^I(\mu)$
- If ν is the result of the application of a binary rule on μ_1 and μ_2, we have to distinguish two cases:
  - If the rule is applied to ancestors of a cut formula, then $CL^I(\nu) = CL^I(\mu_1) \cup CL^I(\mu_2)$.
  - If the rule is applied to ancestors of the end-sequent, then $CL^I(\nu) = CL^I(\mu_1) \times CL^I(\mu_2)$.

  Where[2]:
  $$CL^I(\mu_1) \times CL^I(\mu_2) = \{C \circ D | C \in CL^I(\mu_1), D \in CL^I(\mu_2)\}$$

Note that since the formulas on the right hand side are moved to the left hand side already on the axioms, the clauses always have the right side empty. This guarantees that we always have intuitionistic sequents and no conflicts arise while merging.

▶ **Theorem 5** (Refutability of the Intuitionistic Clause Set)**.** *The intuitionistic clause set is* ***LJ****-refutable.*

Proof. Let φ be an **LJ** proof and $CL^I(\varphi)$ be its intuitionistic clause set built according to Definition 4 and $CL(\varphi)$ be its classical clause set obtained by the classical version of **CERes**. For each clause $C_i = A^i_1, ..., A^i_{n_i} \vdash B^i_1, ..., B^i_{m_i}$ of the classical clause set, we build the closed formula $F_i = \forall\overline{x}.\neg(A^i_1 \wedge ... \wedge A^i_{n_i} \wedge \neg B^i_1 \wedge ... \wedge \neg B^i_{m_i})$.

[2] The operation $\circ$ represents the merging of sequents, i.e., $(\Gamma \vdash \Delta) \circ (\Gamma' \vdash \Delta') = \Gamma, \Gamma' \vdash \Delta, \Delta'$.

By previous results, summarized in Section 2, we know that there is an **LK** refutation ψ of $CL(\varphi)$:

$$\dfrac{\begin{array}{ccc}\underline{C_1} & & \underline{C_k}\\ \vdots & \dots & \vdots\end{array}}{\vdash}$$

By merging each formula F_i to its corresponding clause C_i and propagating it down the refutation, we obtain an LK proof ψ_1 from the formulas $F_i, A^i_1, ..., A^i_{n_i} \vdash B^i_1, ..., B^i_{m_i}$ of the end-sequent $F_1, ..., F_k \vdash$:

$$\dfrac{\begin{array}{ccc}\varphi_1 & & \varphi_k\\ \underline{F_1 \circ C_1} & & \underline{F_k \circ C_k}\\ \vdots & \dots & \vdots\end{array}}{F_1, ..., F_k \vdash}$$

where each φ_i is a derivation of $F_i \circ C_i$ from tautological axioms. We can transform the proof ψ_1 into a proof ψ_2 of $\vdash \neg(F_1 \wedge ... \wedge F_k)$:

$$\dfrac{\dfrac{\dfrac{\psi_1}{F_1, ..., F_k \vdash}}{\overline{F_1 \wedge ... \wedge F_k \vdash}}\ \wedge_l \times (k-1)}{\vdash \neg(F_1 \wedge ... \wedge F_k)}\ \neg_r$$

Since the axioms of this proof are tautological, we can transform this into an **LJ** proof ψ_3 via the following negative translation [14]:

$$\begin{array}{rl}
A & \rightarrow \neg\neg A^*\\
A^* & \rightarrow A \text{ (if A is an atom)}\\
(\neg A)^* & \rightarrow \neg A^* \text{ (if A is an atom)}\\
(A \Box B)^* & \rightarrow (A^* \Box B^*), \Box \in \{\wedge, \vee, \Rightarrow\}\\
(\exists x.A)^* & \rightarrow \exists x.A^*\\
(\forall x.A)^* & \rightarrow \forall x.\neg\neg A^*
\end{array}$$

The end-sequent of ψ_3 is $\vdash \neg(\tilde{F_1} \wedge ... \wedge \tilde{F_k})$, where each $\tilde{F_i}$ is the negative translation of F_i. Note that $\vdash \neg\neg\neg A$ is **LJ**-equivalent to $\vdash \neg A$, so there is still only one negation on this end-sequent.

From the proof ψ_3, we can construct the proof ψ_4:

$$\dfrac{\dfrac{\psi_3}{\vdash \neg(\tilde{F_1} \wedge ... \wedge \tilde{F_k})} \qquad \dfrac{\dfrac{\begin{array}{ccc}\Xi_1 & & \Xi_n\\ \vdash \tilde{F_1} & ... & \vdash \tilde{F_k}\end{array}}{\overline{\vdash \tilde{F_1} \wedge ... \wedge \tilde{F_k}}}\ \wedge_r \times k}{\neg(\tilde{F_1} \wedge ... \wedge \tilde{F_k}) \vdash}\ \neg_l}{\vdash}\ cut$$

Note that the end-sequent of each derivation Ξ_i is of the form:

$$\vdash \neg\neg\forall x_1 .... \neg\neg\forall x_r.\neg\neg\neg(A^i_1 \wedge ... \wedge A^i_{n_i} \wedge \neg B^i_1 \wedge ... \wedge \neg B^i_{m_i})$$

And each Ξ_i is:

$$\begin{array}{cl}
A_1^i, ..., A_{n_i}^i, \neg B_1^i, ..., \neg B_{m_i}^i \vdash & \\
\hline\hline
A_1^i \wedge ... \wedge A_{n_i}^i \wedge \neg B_1^i \wedge ... \wedge \neg B_{m_i}^i \vdash & \wedge_l \times (n_i + m_i - 1) \\
\hline\hline
\vdash \neg\neg\neg(A_1^i \wedge ... \wedge A_{n_i}^i \wedge \neg B_1^i \wedge ... \wedge \neg B_{m_i}^i) & \neg_r, \neg_l, \neg_r \\
\hline
\vdots & \\
\hline\hline
\vdash \neg\neg\forall x_2 .... \neg\neg\forall x_r . \neg\neg\neg(A_1^i \wedge ... \wedge A_{n_i}^i \wedge \neg B_1^i \wedge ... \wedge \neg B_{m_i}^i) & \forall_r, \neg_l, \neg_r \\
\hline\hline
\vdash \neg\neg\forall x_1 .... \neg\neg\forall x_r . \neg\neg\neg(A_1^i \wedge ... \wedge A_{n_i}^i \wedge \neg B_1^i \wedge ... \wedge \neg B_{m_i}^i) & \forall_r, \neg_l, \neg_r
\end{array}$$

So, we obtain an **LJ**-refutation of the clauses $A_1^i, ..., A_{n_i}^i, \neg B_1^i, ..., \neg B_{m_i}^i \vdash$ for every i, which are exactly the elements of $CL^I(\varphi)$.

◀

▶ **Definition 6** ($\mathbf{R}^\neg$)**.** The $\mathbf{R}^\neg$ calculus is a resolution calculus with the following rules:

$$\dfrac{\Gamma \vdash A \quad \Gamma', A' \vdash \Delta}{\Gamma\sigma, \Gamma'\sigma \vdash \Delta\sigma}\, R \qquad \dfrac{\Gamma \vdash \neg A \quad \Gamma', \neg A' \vdash \Delta}{\Gamma\sigma, \Gamma'\sigma \vdash \Delta\sigma}\, R\neg$$

$$\dfrac{A, A' \vdash \Delta}{A\sigma \vdash \Delta\sigma}\, C \qquad \dfrac{\neg A, \neg A' \vdash \Delta}{\neg A\sigma \vdash \Delta\sigma}\, C\neg$$

$$\dfrac{\Gamma, A \vdash}{\Gamma \vdash \neg A}\, \neg_r \qquad \dfrac{\Gamma \vdash A}{\Gamma, \neg A \vdash}\, \neg_l$$

Where Δ is a multi-set with at most one formula[3] and σ is the most general unifier of A and A'.

The choice of a modified resolution calculus is justified by the fact that a proof in this calculus will be used as a part of the final **LJ** proof. In fact, any calculus for intuitionistic logic could be used for the proof search itself, but then we would need a translation of the corresponding proof object into an **LJ**-proof to use in this method.

▶ **Lemma 7.** *If φ is an **LJ**-refutation of a set of intuitionistic clauses (Definition 3) $\mathcal{S}$ and φ' is a normal form of φ with respect to reductive cut-elimination, then any cut-formula in φ' is either an atom or a negated atom.*

Proof. Assume, for the sake of contradiction, that φ' contains a cut whose cut-formula F is neither an atom nor a negated atom. Since the axioms of φ' contain only atoms or negated atoms, it must be the case that the left and right occurrences of F in this cut are introduced, respectively, by inferences ρ_l and ρ_r occurring somewhere in φ'. Two cases can be distinguished:

1. Both ρ_l and ρ_r occur immediately above the cut: in this case, either a grade reduction rule can be applied, if both ρ_l and ρ_r are logical inferences, or a reduction over weakening, if at least one of them is a weakening.
2. At least one of ρ_l and ρ_r does not occur immediately above the cut: in this case, a rank reduction rule can be applied.

In both cases, the assumption contradicts the fact that φ' is in normal form. Therefore, it must be the case that all cut-formulas in φ' are either atoms or negated atoms. ◀

[3] Throughout the paper, Δ stands as a multi-set with at most one formula.

▶ **Lemma 8.** *If φ is an **LJ**-refutation of a set of intuitionistic clauses $\mathcal{S}$ and φ' is a normal form of φ with respect to reductive cut-elimination, then the only inference rules used in φ' are $\neg_l$, $\neg_r$, cut and left contraction.*

Proof. Assume, for the sake of contradiction, that there is an inference ρ in φ' that is neither a $\neg_l$, nor a $\neg_r$, nor a *cut* inference, nor a left contraction, and let F be its main formula. Since φ' is an **LJ**-refutation, its end-sequent is empty. Hence, F must be the ancestor of a cut-formula, and since F is neither an atom nor a negated atom, its descendant cut-formula is also neither an atom nor a negated atom. However, this contradicts Lemma 7, according to which any cut-formula in φ' must be either an atom or a negated atom. Therefore, inferences that are neither $\neg_l$, nor $\neg_r$, nor *cut*, nor left contraction cannot occur in φ'. ◀

▶ Remark. All logical inferences that are neither $\neg_l$ nor $\neg_r$ disappear when φ is rewritten to φ' due to grade reduction rules. This is exemplified below for the conjunction case:

$$\dfrac{\dfrac{\overset{\varphi_1}{\Gamma_1 \vdash A} \quad \overset{\varphi_2}{\Gamma_2 \vdash B}}{\Gamma_1, \Gamma_2 \vdash A \wedge B}\wedge_r \quad \dfrac{\overset{\varphi_3}{\Gamma_3, A, B \vdash \Delta}}{\Gamma_3, A \wedge B \vdash \Delta}\wedge_l}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta}\mathit{cut} \quad \Rightarrow \quad \dfrac{\overset{\varphi_1}{\Gamma_1 \vdash A} \quad \dfrac{\overset{\varphi_2}{\Gamma_2 \vdash B} \quad \overset{\varphi_3}{\Gamma_3, A, B \vdash \Delta}}{\Gamma_2, \Gamma_3, A \vdash \Delta}\mathit{cut}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta}\mathit{cut}$$

The same cannot be done with negation inferences. Observe that, as usual, the grade reduction for negation requires the cut-formulas to be introduced by $\neg_l$ and $\neg_r$:

$$\dfrac{\dfrac{\overset{\varphi_1}{\Gamma_1, A \vdash}}{\Gamma_1 \vdash \neg A}\neg_r \quad \dfrac{\overset{\varphi_2}{\Gamma_2 \vdash A}}{\Gamma_2, \neg A \vdash}\neg_l}{\Gamma_1, \Gamma_2 \vdash}\mathit{cut} \quad \Rightarrow \quad \dfrac{\overset{\varphi_1}{\Gamma_1, A \vdash} \quad \overset{\varphi_2}{\Gamma_2 \vdash A}}{\Gamma_1, \Gamma_2 \vdash}\mathit{cut}$$

However, since the intuitionistic clause can have negated atoms, it may be the case that, (at least) one of the cut-formulas was directly introduced by an axiom, as shown in the example proof below:

$$\dfrac{\dfrac{\overset{\varphi_1}{\Gamma_1, A \vdash}}{\Gamma_1 \vdash \neg A}\neg_r \quad \dfrac{}{\Gamma_2, \neg A \vdash}}{\Gamma_1, \Gamma_2 \vdash}\mathit{cut}$$

In such cases, the grade reduction rule for negation cannot be applied, and hence the negation inference and the cut with a negated atomic formula remain.

▶ **Lemma 9.** *If φ is an **LJ**-refutation of an unsatisfiable set of intuitionistic clauses $\mathcal{S}$ and φ' is a normal form of φ with respect to reductive cut-elimination, then the axioms of φ' are instances of the clauses of $\mathcal{S}$.*

Proof. On applying the rewriting rules for cut-elimination, the initial sequents are not altered, except for the quantifier grade reduction rules, shown below:

$$\dfrac{\dfrac{\overset{\varphi_1}{\Gamma_1 \vdash F(\alpha)}}{\Gamma_1 \vdash \forall x.F(x)}\forall_r \quad \dfrac{\overset{\varphi_2}{\Gamma_2, F(t) \vdash \Delta}}{\Gamma_2, \forall x.F(x) \vdash \Delta}\forall_l}{\Gamma_1, \Gamma_2 \vdash \Delta}\mathit{cut} \quad \Rightarrow \quad \dfrac{\overset{\varphi_1\{\alpha \leftarrow t\}}{\Gamma_1 \vdash F(t)} \quad \overset{\varphi_2}{\Gamma_2, F(t) \vdash \Delta}}{\Gamma_1, \Gamma_2 \vdash \Delta}\mathit{cut}$$

$$\dfrac{\dfrac{\overset{\varphi_1}{\Gamma_1 \vdash F(t)}}{\Gamma_1 \vdash \exists x.F(x)}\,\exists_r \qquad \dfrac{\overset{\varphi_2}{\Gamma_2, F(\alpha) \vdash \Delta}}{\Gamma_2, \exists x.F(x) \vdash \Delta}\,\exists_l}{\Gamma_1, \Gamma_2 \vdash \Delta}\,cut \quad \Rightarrow \quad \dfrac{\overset{\varphi_1}{\Gamma_1 \vdash F(t)} \qquad \overset{\varphi_2\{\alpha \leftarrow t\}}{\Gamma_2, F(t) \vdash \Delta}}{\Gamma_1, \Gamma_2 \vdash \Delta}\,cut$$

In order to eliminate the quantifier of the cut formula, the instantiated version of the formulas must be used. But this imposes no problem, since we can apply the substitution $\sigma = \{\alpha \leftarrow t\}$ on the proof.

If X is an axiom clause in φ_2, $X\{\alpha \leftarrow t\}$ will be an axiom clause in $\varphi_2\{\alpha \leftarrow t\}$. Finally, φ' will have axioms that are, in fact, instances of the clauses in $\mathcal{S}$. ◀

Next, we prove the completeness of the $R^{\neg}$ resolution calculus. In order to do that, we need the lifting lemma for this calculus. Intuitively, this lemma guarantees that if there is a resolution of instantiated terms, it is possible to transform ("lift") this into a resolution of the same terms with variables and a substitution.

▶ **Definition 10.** Let X and Y be clauses, then $X \leq_s Y$ iff there exists a substitution Θ with $X\Theta = Y$.

▶ **Lemma 11** (Lifting)**.** *Let C and D be clauses with $C \leq_s C'$ and $D \leq_s D'$. Assume that C' and D' have a $R^{\neg}$-resolvent E'. Then, there exists a $R^{\neg}$-resolvent E of C and D such that $E \leq_s E'$.*

The proof of Lemma 11 is analogous to the one for the ordinary resolution calculus and will not be described here.

▶ **Theorem 12** (Completeness of $\mathbf{R}^{\neg}$)**.** *Let $\mathcal{S}$ be an **LJ**-refutable set of intuitionistic clauses. Then $\mathcal{S}$ is $R^{\neg}$-refutable.*

Proof. Let φ be an **LJ**-refutation of $\mathcal{S}$. By applying Gentzen's proof-rewriting rules for cut-elimination exhaustively, φ is rewritten to a normal form φ', whose existence is guaranteed by the fact that Gentzen's proof-rewriting system is terminating (see Gentzen's Hauptsatz [8, 9]). By Lemmas 7 and 8, φ' has only $\neg_l$, $\neg_r$, *cut* and left contraction inferences. As these inference rules correspond, respectively, to the rules $\neg_l$, $\neg_r$, $\{R, R\neg\}$ and $\{C, C\neg\}$ (without unification) of the $\mathbf{R}^{\neg}$ calculus, φ' can be immediately converted to a ground $\mathbf{R}^{\neg}$-refutation δ. By Lemma 9, the axioms of φ' and hence also of δ are instances of the clauses in $\mathcal{S}$. Therefore, by the lifting lemma (Lemma 11), δ can be lifted into an $\mathbf{R}^{\neg}$-refutation δ^* of $\mathcal{S}$. ◀

Due to the way the intuitionistic clause set is constructed, all the clauses have no formula on the right hand side. This means that the rule $\neg_l$ can be dropped from $\mathbf{R}^{\neg}$ and the clause sets used in our scenario will still be refutable. Also, the resolution rule on non-negated atoms could also be eliminated in our case, since we could always replace any (non-negated) resolution by negation inferences and negated resolution.

▶ **Definition 13** (Intuitionistic Projection)**.** An *intuitionistic projection* is built analogously to a usual projection, except that all the formulas on the right side are negated and added to the left side.

Let φ be a proof in **LJ** and $C \in CL^I(\varphi)$. Then the **LJ**-proof $\varphi(C)$ is called an *intuitionistic projection* and it is build inductively on the number of inferences of φ. Let ν be a node in φ and $\varphi_\nu(C)$ the projection for clause C until node ν:

1. ν is a leaf: then $\varphi_\nu(C)$ is the derivation consisting of applying a negation rule ($\neg_l$) to the atoms which are cut-ancestors in order to shift them from the right to the left side (if there is a cut-ancestor on the right).
2. ν is the result of a unary rule ξ applied to μ:

 2.a. ξ operates on a cut ancestor: $\varphi_\nu(C) = \varphi_\mu(C)$

 2.b. ξ operates on an end sequent ancestor: $\varphi_\nu(C)$ is $\varphi_\mu(C)$ plus the application of ξ to its end-sequent

3. ν is the result of a binary rule ξ applied to μ_1 and μ_2:

 3.a. ξ operates on a cut ancestor: $\varphi_\nu(C)$ is $\varphi_{\mu_i}(C)$ (i depends on which branch C is coming from) plus some weakenings to obtain formulas that were used in the other branch.

 3.b. ξ operates on an end sequent ancestor: $\varphi_\nu(C)$ is the result of applying ξ to the end-sequents of $\varphi_{\mu_1}(C)$ and $\varphi_{\mu_2}(C)$.

▶ **Definition 14** (NACNF)**.** A proof is said to be in negated atomic cut normal form (NACNF) when all the cuts are on atoms or negated atoms.

▶ **Definition 15** (iCERes)**.** Let φ be a proof in **LJ** of a sequent S, $CL^I(\varphi)$ its intuitionistic clause set (Definition 4) and $\pi_1, ..., \pi_n$ the intuitionistic projections (Definition 13) of the clauses of $CL^I(\varphi)$. By Theorems 5 and 12, there exists a grounded refutation φ^* of $CL^I(\varphi)$. We define **iCERes** as the procedure of computing the elements $CL^I(\varphi)$, $\pi_1, ..., \pi_n$, and φ^* from φ and then merging (instances of) $\pi_1, ..., \pi_n$ with φ^* in the following way:

Let C_i be the clause of a leaf in φ^*. Then, C_i is replaced by the projection π_i (with the proper substitution of variables), which is actually a derivation of $C_i \circ S$. Moreover, the formulas of S are propagated down the refutation.

▶ **Theorem 16.** *Let φ be an proof in **LJ**$^-$ (Definition 2). Then **iCERes**, applied to φ, produces an intuitionistic negated atomic cut normal form.*

Proof. From Definition 15, we can observe that the result of applying **iCERes** to an **LJ**-proof consists of the resolution refutation in $R^\neg$ merged with the projections. These last elements have no cuts and are derivations in **LJ** by definition. The refutation has resolution rules on atoms and negated atoms, which will be the cuts on the final proof. Since the projections have no formula on the right side of their end sequents, and the resolution sequents have no more than one formula on the right side of each sequent, the final proof is an **LJ**-proof of an end-sequent equal to the one of φ up to some contractions on the left. ◀

3.1 Example

In order to illustrate the **iCERes** method, we will apply it to the following **LJ**$^-$ proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\genfrac{}{}{1.5pt}{0}{\dfrac{\dfrac{}{P\alpha^\star \vdash P\alpha}\,I \quad \dfrac{\dfrac{}{Pf\alpha \vdash Pf\alpha}\,I \quad \dfrac{}{Pf^2\alpha \vdash Pf^2\alpha^\star}\,I}{Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash Pf^2\alpha^\star}\,\to_l}{P\alpha^\star, P\alpha \to Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash Pf^2\alpha^\star}\,\to_l}{P\alpha^\star, \forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash Pf^2\alpha^\star}\,\forall_l}{\forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash (P\alpha \to Pf^2\alpha)^\star}\,\to_r}{\forall x.(Px \to Pfx) \vdash (P\alpha \to Pf^2\alpha)^\star}\,c_l}{\forall x.(Px \to Pfx) \vdash \forall x.(Px \to Pf^2x)^\star}\,\forall_r \quad \dfrac{\dfrac{\genfrac{}{}{1.5pt}{0}{\dfrac{\dfrac{}{Pa \vdash Pa^\star}\,I \quad \dfrac{\dfrac{}{Pf^2a^\star \vdash Pf^2a^\star}\,I \quad \dfrac{}{Pf^4a^\star \vdash Pf^4a}\,I}{Pf^2a^\star, (Pf^2a \to Pf^4a)^\star \vdash Pf^4a}\,\to_l}{Pa, (Pa \to Pf^2a)^\star, (Pf^2a \to Pf^4a)^\star \vdash Pf^4a}\,\to_l}{Pa, \forall x.(Px \to Pf^2x)^\star, \forall x.(Px \to Pf^2x)^\star \vdash Pf^4a}\,\forall_l}{Pa, \forall x.(Px \to Pf^2x)^\star \vdash Pf^4a}\,c_l}{Pa, \forall x.(Px \to Pf^2x)^\star \vdash \exists z.Pf^4z}\,\exists_r}{\dfrac{Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}{Pa, \forall x.(Px \to Pfx), \neg\exists z.Pf^4z \vdash}\,\neg_l}\,cut$$

Note that the cut formulas and cut ancestors are superscribed with $\star$. By removing the rules applied on end-sequent ancestors and merging the branches as was described on Definition 4, the intuitionistic clause set obtained is:

$$CL^I(\varphi) = \{P\alpha, \neg Pf^2\alpha \vdash \ ; \ \neg Pa \vdash \ ; \ Pf^4a \vdash\}$$

As was proved previously, there is a resolution refutation of this clause set on $R^\neg$:

$$\dfrac{\dfrac{\dfrac{P\alpha, \neg Pf^2\alpha \vdash}{\neg Pf^2\alpha \vdash \neg P\alpha}\,{\scriptstyle \neg_r} \qquad \neg Pa \vdash}{\neg Pf^2a \vdash}\,{\scriptstyle R^\neg\{\alpha \leftarrow a\}} \qquad \dfrac{\dfrac{\dfrac{Pf^4a \vdash}{\vdash \neg Pf^4a}\,{\scriptstyle \neg_r} \qquad P\alpha, \neg Pf^2\alpha \vdash}{Pf^2a \vdash}\,{\scriptstyle R^\neg\{\alpha \leftarrow f^2a\}}}{\vdash \neg Pf^2a}\,{\scriptstyle \neg_r}}{\vdash}\,{\scriptstyle R^\neg}$$

The projections of the three clauses of CL^I are:

$\pi_1[\alpha]$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P\alpha \vdash P\alpha}\,{\scriptstyle I} \qquad \dfrac{\dfrac{}{Pf\alpha \vdash Pf\alpha}\,{\scriptstyle I} \qquad \dfrac{\dfrac{}{Pf^2\alpha \vdash Pf^2\alpha}\,{\scriptstyle I}}{\neg Pf^2\alpha, Pf^2\alpha \vdash}\,{\scriptstyle \neg_l}}{Pf\alpha, \neg Pf^2\alpha, Pf\alpha \to Pf^2\alpha \vdash}\,{\scriptstyle \to_l}}{P\alpha, \neg Pf^2\alpha, P\alpha \to Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash}\,{\scriptstyle \to_l}}{P\alpha, \neg Pf^2\alpha, \forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash}\,{\scriptstyle \forall_l \times 2}}{P\alpha, \neg Pf^2\alpha, \forall x.(Px \to Pfx) \vdash}\,{\scriptstyle c_l}}{P\alpha, \neg Pf^2\alpha, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,{\scriptstyle w \times 2}}{P\alpha, \neg Pf^2\alpha, Pa, \forall x.(Px \to Pfx), \neg\exists z.Pf^4z \vdash}\,{\scriptstyle \neg_l}$$

π_2 :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Pa \vdash Pa}\,{\scriptstyle I}}{\neg Pa, Pa \vdash}\,{\scriptstyle \neg_l}}{\neg Pa, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,{\scriptstyle w \times 2}}{\neg Pa, Pa, \forall x.(Px \to Pfx), \neg\exists z.Pf^4z \vdash}\,{\scriptstyle \neg_l}$$

π_3 :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Pf^4a \vdash Pf^4a}\,{\scriptstyle I}}{Pf^4a \vdash \exists z.Pf^4z}\,{\scriptstyle \exists_r}}{Pf^4a, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,{\scriptstyle w \times 2}}{Pf^4a, Pa, \forall x.(Px \to Pfx), \neg\exists z.Pf^4z \vdash}\,{\scriptstyle \neg_l}$$

By merging the appropriate instances of the projections on the resolution refutation, we obtain the final proof, depicted in Figure 3. The projections are colored accordingly. The projection π_1 used on the left side had α replaced with a and the one used on the right side had α replaced with f^2a. Note that this proof is in NACNF, containing only cuts on atoms or negated atoms, and it is still a proof in **LJ**.

4 On the possibility of extending iCERes to a larger class of proofs

On the example of Section 3.1, the last application of the rule $\neg_l$ was used in order to make the end-sequent fulfill the condition of not having formulas on the right. This is a simple operation, but, as we mentioned before, it is not trivial how to transform the final proof into a proof of the sequent where the shifted formula is on the right side. In this section we analyse a possible solution to deal with end-sequents without this restriction.

Applying **iCERes** to a proof in **LJ** where the end-sequent has a formula on the right will yield intuitionistic projections and a refutation in the $R^\neg$ calculus. But when these elements are put together, it might be the case that some classical sequents appear (in the sense of having more than one formula on the right). Nevertheless, it seems to be often the case that applying reductive cut-elimination on this final **LK** proof and removing the cuts on atoms and negated atoms will again result in an **LJ** proof. We present here an example to illustrate this.

The proof used is the same as the one on Section 3.1, except for the last application of $\neg_l$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P\alpha^\star \vdash P\alpha}\,I \quad \dfrac{\dfrac{}{Pf\alpha \vdash Pf\alpha}\,I \quad \dfrac{}{Pf^2\alpha \vdash Pf^2\alpha^\star}\,I}{Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash Pf^2\alpha^\star}\,\to_l}{P\alpha^\star, P\alpha \to Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash Pf^2\alpha^\star}\,\to_l}{P\alpha^\star, \forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash Pf^2\alpha^\star}\,\forall_l}{\forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash (P\alpha \to Pf^2\alpha)^\star}\,\to_r}{\dfrac{\forall x.(Px \to Pfx) \vdash (P\alpha \to Pf^2\alpha)^\star}{\forall x.(Px \to Pfx) \vdash \forall x.(Px \to Pf^2x)^\star}\,\forall_r}\,c_l \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Pa \vdash Pa^\star}\,I \quad \dfrac{\dfrac{}{Pf^2a^\star \vdash Pf^2a^\star}\,I \quad \dfrac{}{Pf^4a^\star \vdash Pf^4a}\,I}{Pf^2a^\star, (Pf^2a \to Pf^4a)^\star \vdash Pf^4a}\,\to_l}{Pa, (Pa \to Pf^2a)^\star, (Pf^2a \to Pf^4a)^\star \vdash Pf^4a}\,\to_l}{Pa, \forall x.(Px \to Pf^2x)^\star, \forall x.(Px \to Pf^2x)^\star \vdash Pf^4a}\,\forall_l}{\dfrac{Pa, \forall x.(Px \to Pf^2x)^\star \vdash Pf^4a}{Pa, \forall x.(Px \to Pf^2x)^\star \vdash \exists z.Pf^4z}\,\exists_r}\,c_l}{Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,cut$$

The intuitionistic characteristic clause set is (the same as before):

$$CL^I = \{P\alpha, \neg Pf^2\alpha \vdash \;\; ; \;\; \neg Pa \vdash \;\; ; \;\; Pf^4a \vdash\}$$

The projections for each element of CL^I are:

$\pi_1[\alpha]$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{P\alpha \vdash P\alpha}\,I \quad \dfrac{\dfrac{}{Pf\alpha \vdash Pf\alpha}\,I \quad \dfrac{\dfrac{}{Pf^2\alpha \vdash Pf^2\alpha}\,I}{\neg Pf^2\alpha, Pf^2\alpha \vdash}\,\neg_l}{Pf\alpha, \neg Pf^2\alpha, Pf\alpha \to Pf^2\alpha \vdash}\,\to_l}{P\alpha, \neg Pf^2\alpha, P\alpha \to Pf\alpha, Pf\alpha \to Pf^2\alpha \vdash}\,\to_l}{P\alpha, \neg Pf^2\alpha, \forall x.(Px \to Pfx), \forall x.(Px \to Pfx) \vdash}\,\forall_l \times 2}{\dfrac{P\alpha, \neg Pf^2\alpha, \forall x.(Px \to Pfx) \vdash}{P\alpha, \neg Pf^2\alpha, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,w \times 2}\,c_l$$

π_2 :

$$\dfrac{\dfrac{\dfrac{}{Pa \vdash Pa}\,I}{\neg Pa, Pa \vdash}\,\neg_l}{\neg Pa, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,w \times 2$$

π_3 :

$$\dfrac{\dfrac{\dfrac{}{Pf^4a \vdash Pf^4a}\,I}{Pf^4a \vdash \exists z.Pf^4z}\,\exists_r}{Pf^4a, Pa, \forall x.(Px \to Pfx) \vdash \exists z.Pf^4z}\,w \times 2$$

The resolution refutation is (the same as before):

$$\dfrac{\dfrac{\dfrac{P\alpha, \neg Pf^2\alpha \vdash}{\neg Pf^2\alpha \vdash \neg P\alpha}\,\neg_r \quad \neg Pa \vdash}{\neg Pf^2a \vdash}\,R^\neg\{\alpha \leftarrow a\} \quad \dfrac{\dfrac{\dfrac{Pf^4a \vdash}{\vdash \neg Pf^4a}\,\neg_r \quad P\alpha, \neg Pf^2\alpha \vdash}{Pf^2a \vdash}\,R^\neg\{\alpha \leftarrow f^2a\}}{\vdash \neg Pf^2a}\,\neg_r}{\vdash}\,R^\neg$$

$$
\begin{prooftree}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$Pa \vdash Pa$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$Pfa \vdash Pfa$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$Pf^2a \vdash Pf^2a$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$Pf^3a \vdash Pf^3a$}
\AxiomC{}
\RightLabel{I}
\UnaryInfC{$Pf^4a \vdash Pf^4a$}
\RightLabel{$\exists_r$}
\UnaryInfC{$Pf^4a \vdash \exists z.Pf^4z$}
\RightLabel{$\rightarrow_l$}
\BinaryInfC{$Pa, Pf^3a, Pf^3a \rightarrow Pf^4a \vdash \exists z.Pf^4z$}
\RightLabel{$\rightarrow_l$}
\BinaryInfC{$Pf^2a, Pa, Pf^2a \rightarrow Pf^3a, Pf^3a \rightarrow Pf^4a \vdash \exists z.Pf^4z$}
\RightLabel{$\forall_l \times 2$}
\UnaryInfC{$Pf^2a, Pa, \forall x.(Px \rightarrow Pfx), \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\RightLabel{c_l}
\UnaryInfC{$Pf^2a, Pa, \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\RightLabel{$\rightarrow_l$}
\BinaryInfC{$Pa, Pfa, Pfa \rightarrow Pf^2a, \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\RightLabel{$\rightarrow_l$}
\BinaryInfC{$Pa, Pa, Pa \rightarrow Pfa, Pfa \rightarrow Pf^2a, \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\RightLabel{$\forall_l \times 2$}
\UnaryInfC{$Pa, Pa, \forall x.(Px \rightarrow Pfx), \forall x.(Px \rightarrow Pfx), \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\RightLabel{$c_l \times 3$}
\UnaryInfC{$Pa, \forall x.(Px \rightarrow Pfx) \vdash \exists z.Pf^4z$}
\end{prooftree}
$$

Figure 2 Proof obtained after eliminating the atomic cuts.

Given the projections and grounded resolution, the final proof is depicted in Figure 4. Note that, since the end-sequent had a formula on the right side, the final proof is not in **LJ**, but in **LK**. Even though the refutation and projections were intuitionistic, when they are put together some sequents end up having more than one formula on the right.

But if reductive cut-elimination is applied to the proof in Figure 4 and all useless weakenings and contractions are removed, the result is again a proof in **LJ**, depicted in Figure 2.

The second condition, the absence of strong quantifiers in the end-sequent, can be satisfied by removing strong quantifiers with methods of skolemization of **LJ**-proofs [4]. It is important to note, however, that the end-sequent of a final proof φ obtained by applying **iCERes** to an **LJ**-proof that has been skolemized will also contain skolem terms. Depending on the application, it might be desirable to transform φ into a proof of the original non-skolemized end-sequent. Unfortunately, it is not yet clear how to perform such a deskolemization.

5 Conclusions and future work

This paper presents a method of cut-elimination by resolution for a fragment of **LJ**. In order to develop this method, it was necessary to define intuitionistic clause sets, intuitionistic projections and a new resolution calculus. It was proved that this calculus is complete with respect to **LJ** on clause logic. Applying the **CERes** method (previously developed for classical logic) with these new definitions on an **LJ** proof, such that its end-sequent does not contain a formula on the right side, will yield an intuitionistic proof with only atomic or negated atomic cuts.

It is important to observe that, for any **LJ**-proof, the projections and refutation of the clause set, as defined in this paper, are both intuitionistic. But when these elements are assembled together to compose the final proof, a classical sequent might occur. This is why the current method has to be restricted to a fragment of **LJ** for which this undesirable effect cannot occur. Interestingly, this fragment corresponds to the class of intuitionistic proofs by contradiction.

Immediate future work will consist of extending **iCERes** to larger classes of (and hopefully all) **LJ**-proofs. It seems likely that this will require more sophisticated ways of assembling projections and refutations.

In addition to extending **iCERes** to larger classes of proofs, we also intend to eventually apply it to a real mathematical proof, as we have done with the classical **CERes** for Fürstenberg's proof of the infinitude of primes.

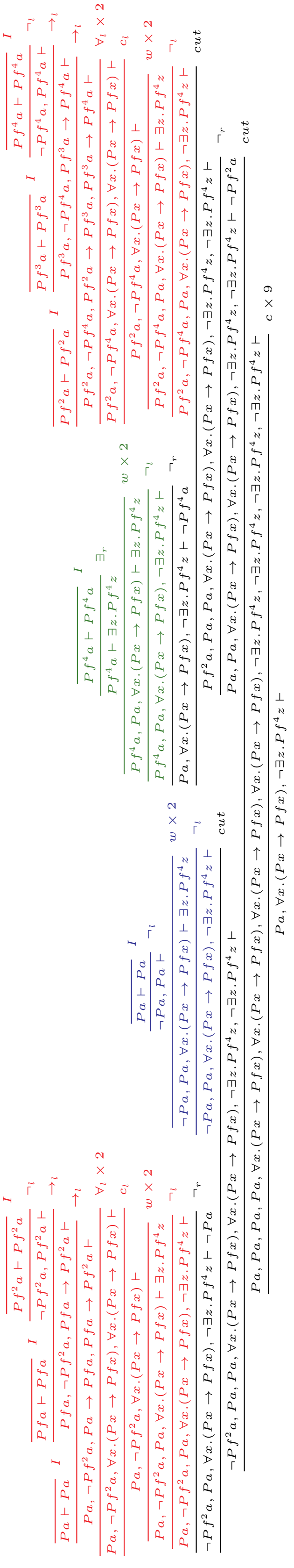


Figure 3 Final proof after applying the method to an LJ-proof of an end-sequent with no formula on the right.

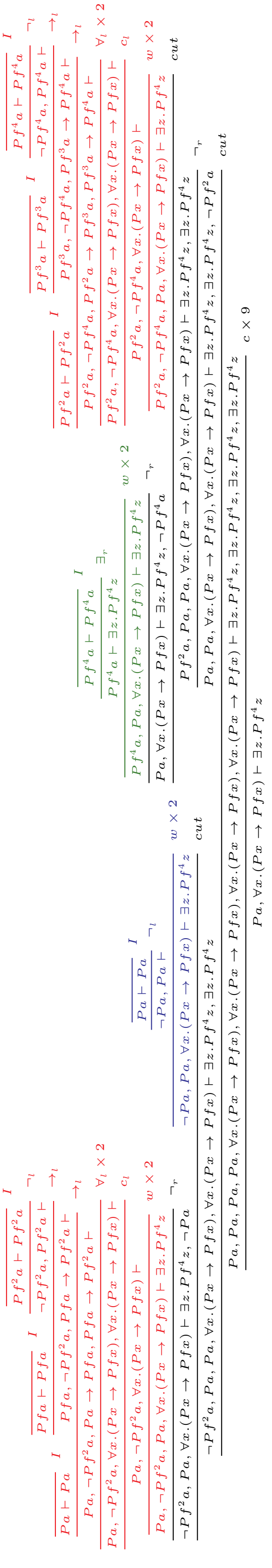


Figure 4 Final proof after applying the method to an LJ-proof of an end-sequent with a formula on the right.

References

1 Matthias Baaz, Agata Ciabattoni, and Christian G. Fermüller. Cut elimination for first order Gödel logic by hyperclause resolution. In *LPAR 2008*, volume 5330 of *LNCS*, pages 451–466, 2008.

2 Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Cut-elimination: Experiments with ceres. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR) 2004*, volume 3452 of *Lecture Notes in Computer Science*, pages 481–495. Springer, 2005.

3 Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Ceres: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403:160–175, 2008.

4 Matthias Baaz and Rosalie Iemhoff. On skolemization in constructive theories. *Journal of Symbolic Logic*, 73(3):969–998, 2008.

5 Matthias Baaz and Alexander Leitsch. Cut-elimination and redundancy-elimination by resolution. *Journal of Symbolic Computation*, 29(2):149–176, 2000.

6 Matthias Baaz and Alexander Leitsch. CERES in many-valued logics. In *Proceedings of LPAR 2004*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 1–20. Springer, 2005.

7 Matthias Baaz and Alexander Leitsch. *Methods of Cut-Elimination*, volume 34 of *Trends in Logic*. Springer, 2011.

8 Gerhard Gentzen. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift*, 39(1):176–210, dec 1935.

9 Gerhard Gentzen. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift*, 39(1):405–431, dec 1935.

10 J.Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types*. Cambridge University Press New York, 1989.

11 Stefan Hetzl, Alexander Leitsch, and Daniel Weller. Ceres in higher-order logic. *Annals of Pure and Applied Logic*, 162(12):1001–1034, 2011.

12 Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. CERES in second-order logic. Technical report, Vienna University of Technology, 2008.

13 Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo. Herbrand sequent extraction. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *Intelligent Computer Mathematics*, volume 5144 of *Lecture Notes in Computer Science*, pages 462–477. Springer Berlin, 2008.

14 Ulrich Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer Monographs in Mathematics. Springer Verlag, 2008.

15 G. Polya. *Mathematics and plausible reasoning, 2 vols.* Princeton, 2 edition, 1968. vol 1. Induction and analogy in mathematics; vol 2. Patterns of plausible inference.

16 Gaisi Takeuti. *Proof Theory*. North-Holland/American Elsevier, 1975.

Variants of Collapsible Pushdown Systems

Paweł Parys[*]

University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland
parys@mimuw.edu.pl

Abstract

We analyze the relationship between three ways of generating trees using collapsible pushdown systems (CPS's): using deterministic CPS's, nondeterministic CPS's, and deterministic word-accepting CPS's. We prove that (for each level of the CPS and each input alphabet) the three classes of trees are equal. The nontrivial translations increase $n-1$ times exponentially the size of the level-n CPS. The same results stay true if we restrict ourselves to higher-order pushdown systems without collapse. As a second contribution we prove that the hierarchy of word languages recognized by nondeterministic CPS's is infinite. This is a consequence of a lemma which bounds the length of the shortest accepting run. It also implies that the hierarchy of ε-closures of configuration graphs is infinite (which was already known). As a side effect we obtain a new algorithm for the reachability problem for CPS's; it has the same complexity as previously known algorithms.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases collapsible pushdown systems, determinization, infinite hierarchy, shrinking lemma, reachability

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.500

1 Introduction

Already in the 70's, Maslov ([14, 15]) generalized the concept of pushdown automata to higher-order pushdown automata and studied such devices as acceptors of string languages. In the last decade, renewed interest in these automata has arisen. They are now studied also as generators of graphs and trees. Knapik et al. [12] showed that the class of trees generated by deterministic level-n pushdown systems coincides with the class of trees generated by *safe* level-n recursion schemes,[1] and Caucal [8] gave another characterization: trees on level $n+1$ are obtained from trees on level n by an MSO-interpretation of a graph, followed by application of unfolding. Carayol and Wöhrle [7] studied the ε-closures of configuration graphs of level-n pushdown systems and proved that these graphs are exactly the graphs in the n-th level of the Caucal hierarchy. Driven by the question whether safety implies a semantical restriction to recursion schemes (which was recently proven [17]), Hague et al. [10] extended the model of level-n pushdown systems to level-n collapsible pushdown systems (n-CPS's) by introducing a new stack operation called collapse. They showed that the trees generated by such systems coincide exactly with the class of trees generated by all higher-order recursion schemes and this correspondence is level-by-level.

We compare three ways of generating trees using CPS's of some level n. We consider here edge-labelled, unranked, and unordered trees. In the classical definition of a tree-generating CPS we take a deterministic CPS. Moreover it is typically assumed that every nonempty

[*] The author is partially supported by the Polish Ministry of Science grant nr N N206 567840.
[1] Safety is a syntactic restriction on the recursion scheme.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 500–515
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

run (from some reachable configuration) using only ε-transitions can be extended into a run reading also some letter.[2] We call such CPS strongly deterministic (see Definition 2.11). To generate a tree, we unfold the configuration graph of the strongly deterministic CPS $\mathcal{S}$ into a tree, and we contract all ε-labelled edges. We can also say that we take the ε-closure of the configuration graph, and then we unfold it into a tree. Denote this tree $T(\mathcal{S})$, and the class of all such trees as $Trees^n_D$. Notice that in such a tree each node has, for each letter, at most one outgoing edge labelled by that letter. Trees having this property will be called deterministic.

But we can do the same (i.e. take the ε-closure of the configuration graph, and then unfold it into a tree) for nondeterministic CPS's. In general such trees can be very strange, e.g. can contain a node which has infinitely many successors. We restrict ourselves only to CPS's generating deterministic trees. This class of trees will be called $Trees^n_N$. Notice that whether a tree is deterministic is only a property of the tree, not the system generating it; of course nondeterministic systems can also generate deterministic trees (the configuration graphs can contain big parts consisting of only ε-transitions, containing many loops and nondeterministic choices, but they are whole contracted to one node in the ε-closure). Nevertheless, we prove that every such tree can be also generated by a strongly deterministic system of the same level. In other words, there is an easy syntactic condition saying whether a tree generated by a CPS can be also generated by a strongly deterministic CPS.

▶ **Theorem 1.1.** *Let $\mathcal{S}$ be an n-CPS, such that $T(\mathcal{S})$ is deterministic. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$. System $\mathcal{S}'$ has size $n-1$ times exponential in the size of $\mathcal{S}$, and can be computed in such time.*

Determinization results are always important, because deterministic automata are, colloquially speaking, simpler and have better properties. For example a strongly deterministic system can be easily simulated: having a configuration one can just run the system to see which letters can be read next. For general (nondeterministic) systems this is much more difficult, as by principle there can be arbitrarily long runs reading just one letter, so we do not know when to stop. Other reason is that it is easier to prove that a tree is not generated by a deterministic system, than that a graph is not generated by any (nondeterministic) system. To be more precise, consider our result [11] showing that the hierarchy of graphs generated by CPS's is strict, i.e. that for each n there is a graph generated by an $(n+1)$-CPS (i.e. which is the ε-closure of its configuration graph) which is not generated by any n-CPS. We perform there a pumping construction which says that in a long enough run there is a fragment which (in some sense) can be repeated arbitrarily many times, and this leads to a contradiction. The problem is when this fragment does not read any letter; then repeating it does not change the path in the ε-closure. To avoid such situation a very sophisticated analysis is performed, which causes a technical complication of the proof. However using the result of this paper we could just show that the unfolding of this graph is not generated by a deterministic system of level n, which would significantly simplify the proof. For deterministic systems it is enough to perform the pumping in a simple form: repeating arbitrarily many times a fragment not reading a letter is impossible in a deterministic system, so automatically by repeating some fragment of a run we obtain longer and longer paths in the ε-closure.

[2] Sometimes this assumption is dropped (and then in the generated tree all branches of ε-labelled edges which do not lead to a non-ε edge are replaced by a $\perp$-labelled edge). Because our definition is more restrictive, our results become stronger.

We also consider CPS's as word acceptors (such CPS's are given together with a set of final states). Let us recall that no determinization is possible for word languages: it is known that nondeterministic systems recognize more languages than deterministic ones (even if we are allowed to increase the level). However we show that every deterministic system can be converted into a strongly deterministic one, in which additionally from each reachable configuration there exists an accepting run.

▶ **Theorem 1.2.** *Let $\mathcal{S}$ be a deterministic n-CPS with final states F, used as word acceptor. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ with final states F' such that it recognizes the same language as $(\mathcal{S}, F)$, and from every reachable configuration of $\mathcal{S}'$ there is an accepting run (unless the language is empty). System $\mathcal{S}'$ has size $n-1$ times exponential in the size of $\mathcal{S}$, and can be computed in such time.*

This can be equivalently stated for trees. Let $Pref(\mathcal{S}, F)$ be the tree whose nodes are all prefixes of words accepted by the CPS $\mathcal{S}$ with final states F, and let $Trees_L^n$ be the class of all such trees. If from every reachable configuration of $\mathcal{S}$ there is an accepting run, we have $Pref(\mathcal{S}, F) = T(\mathcal{S})$. Thus thanks to the above theorem we have $Trees_L^n \subseteq Trees_D^n$. For the opposite implication it is enough to assume that every state is final (then every path of $T(\mathcal{S})$ is accepted). Thus we obtain equality of the three classes of trees.

▶ **Corollary 1.3.** *For each $n \in \mathbb{N}$ we have $Trees_D^n = Trees_N^n = Trees_L^n$.*

All our results still hold if we restrict ourselves to systems without collapse (i.e. then the resulting systems also do not use collapse). Let us notice that Theorem 1.2 can be also easily deduced (but probably without the bound on the size of the new automaton) from a theorem [4] saying that collapsible pushdown systems are closed under logical reflection. We however believe that our proof is simpler (and it gives also Theorem 1.1 which cannot be obtained using logical reflection).

As a second contribution we prove that the hierarchy of word languages recognized by (nondeterministic) CPS's is infinite.

▶ **Theorem 1.4.** *For each $n \in \mathbb{N}$ there is a language recognized by a pushdown system (without collapse) of level $2n+1$ which is not recognized by any n-CPS.*

Our example language from Theorem 1.4 can be also used to show that the hierarchies of graphs (i.e. ε-closures of configuration graphs) and of trees are infinite. However we already know [11] that these two hierarchies are not only infinite but in fact strict, i.e. that for each n there is a tree generated by a level-$(n+1)$ pushdown system without collapse which is not generated by any n-CPS, and similarly for ε-closures of configuration graphs. The strictness of the hierarchy of word languages recognized by higher-order pushdown systems without collapse follows from the Damm's paper [9]. Similar results for the hierarchy of word languages recognized by CPS's were missing so far (this is stated as an open problem in [5]).

The strictness results for tree and graph hierarchies are obtained using a pumping lemma. This lemma essentially says that if an automaton has a very long run, then it also has arbitrarily long runs. Such arbitrarily long runs can be then transformed into arbitrarily long paths in the trees or graphs. However for the word languages hierarchy such lemma is useless: the problem is that in a nondeterministic system such longer and longer runs can all read the same word. We instead need a shrinking lemma, which would allow to shorten a run. We prove the following lemma, which is good for this purpose.

▶ **Theorem 1.5.** *Let $\mathcal{S}$ be an n-CPS with set of states Q and stack alphabet Γ, given together with a set of accepting states. Assume that there exists an accepting run of $\mathcal{S}$. Set $\exp_0(i) = i$ and $\exp_{k+1}(i) = 2^{\exp_k(i)}$. Then there exists an accepting run of $\mathcal{S}$ of length at most $\exp_{2n-1}(8|Q|^2|\Gamma|)$.*

As a third contribution we obtain two new algorithms which check whether the language recognized by a CPS is nonempty (one can equivalently talk about reachability of a configuration having the state in a given set). One algorithm is extremely simple: it is enough to check if there exists a run from the initial configuration to a configuration with accepting state, whose length is bounded by the threshold from Theorem 1.5. Its complexity is however $(2n-1)$-NEXPTIME. In Section 4 we present another algorithm for the emptiness problem, whose complexity is $(n-1)$-EXPTIME.

The author is unaware of any result dealing with the emptiness problem directly. Of course the emptiness problem is a special case of the μ-calculus model-checking problem. A direct solution of this problem uses parity games over configuration graphs of the systems [10]. For such games one can (in a nontrivial way) reduce the level of the system by one, increasing its size exponentially. This approach gives n-EXPTIME complexity for μ-calculus model checking (and in fact the problem is n-EXPTIME complete); however it is not difficult to see that if we just consider emptiness, the complexity can be lowered to $(n-1)$-EXPTIME (as the first reduction of level gives then only a polynomial blowup). There are several other approaches to μ-calculus model checking [16, 13, 18], which use equivalent characterizations of collapsible pushdown systems by recursion schemes or Krivine machines. In contrast, our algorithm analyzes directly possible behaviors of the system, works only for the emptiness problem, and is simple.

Let us remark that an algorithm for the emptiness problem allows us also to check whether a given word is accepted by a system, also in $(n-1)$-EXPTIME: it is enough to take the product of the given system with a finite automaton accepting only the one given word.

Organization The strategy for proving our results is as follows. In Section 2 we give all necessary definitions. We begin the proof by Section 3, where we show how Theorem 1.4 follows from Theorem 1.5. Then in Section 4 we define so-called types of stacks, which talk about possible kinds of runs starting in a given configuration. We also present there (in Subsection 4.1) the algorithm for the emptiness problem. Next, in Section 5 we say that the pushdown systems can itself calculate the type of its current configuration. This already gives us Theorem 1.2. Finally in Section 6 we say that the systems can not only calculate the types, but also use them to choose some unique run (from some class of runs). This gives us the determinization described by Theorem 1.1. As a side effect, in Subsection 6.1, we obtain a bound on the length of runs described by types, which gives us Theorem 1.5.

2 Preliminaries

Collapsible pushdown systems of level n (in the following n is always assumed to be a fixed natural number) are an extension of pushdown systems where we replace the stack by an n-fold nested stack structure. For the manipulation of the higher-order stack, we have a push and a pop operation for each stack level $1 \leq i \leq n$ and a collapse operation. When a new symbol is pushed onto the stack, we attach a copy of the stack to this symbol and the collapse operation may replace the current stack with the stack attached to the topmost symbol again, i.e. in some sense the collapse operation allows to jump back to the stack where the current topmost symbol was created for the first time.

▶ **Definition 2.1.** Given a number n (the level of the system) and stack alphabet Γ, we define the set of stacks as the smallest set satisfying the following.

- If $s_1, s_2, \dots, s_m$ are $(k-1)$-stacks, where $1 \leq k \leq n$, then the sequence $[s_1, s_2, \dots, s_m]$ is a k-stack. This includes the empty sequence ($m = 0$).
- If s^k is a k-stack, where $1 \leq k \leq n$, and $\alpha \in \Gamma$, then (α, k, s^k) is a 0-stack.

For a k-stack s^k and a $(k-1)$-stack s^{k-1} we write $s^k : s^{k-1}$ to denote the k-stack obtained by appending s^{k-1} on the end of s^k. We write $s^2 : s^1 : s^0$ for $s^2 : (s^1 : s^0)$.

Let us remark that in the classical definition the stacks are defined differently: they are not nested, the 0-stack does not store the linked k-stack, just stores a natural number. While performing the collapse operation, this number is used to find the stack pointed by the link. Note that this is only a syntactical difference. Let us emphasize however that this modification is essential to obtain a correct definition of "types" below: Our k-stack already contains all stacks in the links, so looking at a k-stack we have the complete information about it, and we can summarize it using a type from a finite set. On the other hand the original k-stack has arbitrarily many numbers pointing to some stacks "outside", and for each of this numbers it is important how the target of the pointer looks like; thus already the interface with the external world is arbitrarily big.

▶ **Definition 2.2.** We define stack operations as follows. We decompose a stack s of level n into its topmost stacks $s^n : s^{n-1} : \cdots : s^0$. We have $\mathsf{pop}^i(s) := s^n : \cdots : s^{i+1} : s^i$, where $1 \leq i \leq n$; the result is undefined if s^i is empty. For $2 \leq i \leq n$ we have $\mathsf{push}^i(s) := s^n : \cdots : s^{i+1} : (s^i : \cdots : s^0) : s^{i-1} : \cdots : s^0$. The level-1 push is $\mathsf{push}^1_{\alpha,k}$ for $\alpha \in \Gamma$, $1 \leq k \leq n$ which is defined by $\mathsf{push}^1_{\alpha,k}(s) := s^n : \cdots : s^2 : (s^1 : s^0) : (\alpha, k, s^k)$. The collapse operation col^i (where $1 \leq i \leq n$) is defined if the topmost 0-stack is (a, i, t^i), and t^i is not empty. Then it is $\mathsf{col}^i(s) := s^n : \cdots : s^{i+1} : t^i$. Otherwise the collapse operation is undefined.

Pushdown systems only use n-stacks that can be created from the initial n-stack using the stack operations. We call those n-stacks *pushdown stores (pds)*.

▶ **Definition 2.3.** The *initial n-stack* $\perp_n$ is the n-stack which contains only one 0-stack, which is $(\perp, 1, s^1)$, where $\perp \in \Gamma$ is a special symbol, and s^1 is the empty 1-stack.[3] Some n-stack s is a *pushdown store* (or *pds*), if there is a finite sequence of stack operations that creates s from $\perp_n$.

▶ **Definition 2.4.** A *collapsible pushdown system of level n* (an n-CPS) is a tuple $\mathcal{S} = (\Gamma, A, Q, q_I, \perp, \Delta)$, where Γ is a finite stack alphabet containing the initial stack symbol $\perp$, A is a finite input alphabet, Q is a finite set of states, $q_I \in Q$ is an initial state, and $\Delta \subseteq Q \times \Gamma \times (A \cup \{\varepsilon\}) \times Q \times OP^n_\Gamma$ is a transition relation where OP^n_Γ denotes the set of stack operations. When saying that a transition goes from $(q, \alpha) \in Q \times \Gamma$ we mean that it has q and α on the first two coordinates. A *configuration* is a pair (q, s) for a state q and an pds s. The *initial configuration* is $(q_I, \perp_n)$.

Below our convention is that, whenever we have a system $\mathcal{S}$, we assume that Q is its set of states, A its input alphabet, and Γ its stack alphabet. The *size* of an n-CPS is the number of its transitions. Let us emphasize that, according to our definition, a configuration of a pushdown system contains a pds, not an arbitrary n-stack.

[3] The choice of 1 is arbitrary, it can be any number from 1 to n.

▶ **Definition 2.5.** A *run* R of length m, from c_0 to c_m, is a sequence $c_0 \vdash^{a_1} c_1 \vdash^{a_2} \cdots \vdash^{a_m} c_m$ where $c_i = (q_i, s_i)$ are configurations and $a_i \in A \cup \{\varepsilon\}$ are such that, for $1 \leq i \leq m$, there exists a transition $(q_{i-1}, \alpha_{i-1}, a_i, q_i, op)$ such that the topmost stack symbol of s_{i-1} is α_{i-1} and $s_i = op(s_{i-1})$.

▶ **Definition 2.6.** A *labelled transition system* (LTS) over alphabet A is an edge-labelled directed graph with a distinguished initial node. More formally, it is $(G, \mathit{init}, (E_a)_{a\in A})$, where G is a set of nodes, $\mathit{init} \in G$, and $E_a \subseteq G \times G$ for each $a \in A$. A *configuration graph* of a CPS $\mathcal{S}$ is an LTS over alphabet $A \cup \{\varepsilon\}$, which contains all reachable configurations of $\mathcal{S}$ as nodes, and where $(c, d) \in E_a$ when there is a run $c \vdash^a d$ of length 1.

By a *tree* we always mean an LTS which is a tree, i.e. in which every node is reachable from the initial node by exactly one path.

▶ **Definition 2.7.** The *ε-closure* of an LTS $\mathcal{G} = (G, \mathit{init}, (E_a)_{a\in A\cup\{\varepsilon\}})$ over alphabet $A \cup \{\varepsilon\}$ is the LTS $(G', \mathit{init}, (E'_a)_{a\in A})$ where

$$G' := \{\mathit{init}\} \cup \{d \in G : \exists c \in G\ (c, d) \in E_a \text{ for some } a \in A\}$$

and two nodes c, d are connected by E'_a if there is a path in $\mathcal{G}$ from c to d whose last edge is labelled by a, and all earlier edges are labelled by ε. The *unfolding* of an LTS $\mathcal{G} = (G, \mathit{init}, (E_a)_{a\in A})$ is a tree $(G', \mathit{init}, (E'_a)_{a\in A})$ where G' contains all paths of $\mathcal{G}$ starting in its initial node, and paths $p_1 = c_1c_2 \dots c_k$ and $p_2 = c_1c_2 \dots c_kc_{k+1}$ for $(c_k, c_{k+1}) \in E_a$ are connected by E'_a.

We denote the unfolding of the ε-closure of the configuration graph of an CPS $\mathcal{S}$ by $T(\mathcal{S})$.

▶ **Definition 2.8.** The word read by a run $c_0 \vdash^{a_1} c_1 \vdash^{a_2} \cdots \vdash^{a_m} c_m$ is obtained from $a_1a_2 \dots a_m$ by dropping all appearances of ε. We say that a word w is accepted by a CPS $\mathcal{S}$ given together with a set of final states $F \subseteq Q$, if there exists a run of $\mathcal{S}$ reading w from the initial configuration to a configuration whose state is final. The language recognized by $(\mathcal{S}, F)$, denoted $L(\mathcal{S}, F)$, is the set of all words accepted by $(\mathcal{S}, F)$. The prefix tree of $L(\mathcal{S}, F)$, denoted $\mathit{Pref}(\mathcal{S}, F)$, is $(G, \varepsilon, (E_a)_{a\in A})$ where G contains all prefixes of words from $L(\mathcal{S}, F)$, and words w and wa (for $a \in A$) are connected by E_a.

▶ **Definition 2.9.** A system $\mathcal{S} = (\Gamma, A, Q, q_I, \perp, \Delta)$ is called *deterministic* if Δ is a partial function $\Delta\colon Q \times \Gamma \times (A \cup \{\varepsilon\}) \to Q \times OP^n_\Gamma$, and additionally from each $(q, \alpha) \in Q \times \Gamma$ we either have only an ε-transition, or only letter-labelled transitions.

▶ **Definition 2.10.** Let $(G, \mathit{init}, (E_a)_{a\in A})$ be an LTS. We say that it is *deterministic* if for each node c and each $a \in A$ there is at most one d such that $(c, d) \in E_a$.

Notice that if a system $\mathcal{S}$ is deterministic, then also the tree $T(\mathcal{S})$ is deterministic; however the opposite implication does not hold.

▶ **Definition 2.11.** We say that a deterministic CPS $\mathcal{S}$ is *strongly deterministic* if for some set of *dead states* $Q_{die} \subseteq Q$ it holds

- in each reachable configuration with state q and topmost stack symbol α each transition from (q, α) can be applied (i.e. we do not try to perform pop or col when it is impossible), and
- there are no transitions from (q, α) for $q \in Q_{die}$ and any α, and

- if c is a reachable configuration whose state is not a dead state, there is a run from c having some non-ε-transitions.[4]

3 The hierarchy is infinite

In this section we prove that the hierarchy of word languages is infinite (Theorem 1.4) basing on Theorem 1.5. We keep the notation $\exp_k$ from the statement of Theorem 1.5. For each $n \in \mathbb{N}$ consider the language

$$L_n = \{1^k 0^{\exp_n(k)} : k \in \mathbb{N}\}.$$

It is known that L_n can be recognized by a level-$(n+1)$ pushdown system (without collapse); see e.g. [2], example on pages 6-7, where a very similar pushdown system is constructed. Thus L_{2n} can be recognized by a level-$(2n+1)$ pushdown system.

Assume now that there exists an n-CPS $\mathcal{S}$, which recognizes L_{2n}. Let Q be its set of states, and Γ its stack alphabet. Choose k such that $\exp_{2n}(k) > \exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. We construct a system $\mathcal{R}$ which accepts only one word $w = 1^k 0^{\exp_{2n}(k)}$, i.e. the only word from L_{2n} which has k letters 1. Such system can have the same stack alphabet as $\mathcal{S}$, and $(k+1)|Q|$ states. Indeed, we make $k+1$ copies of the set of states of $\mathcal{S}$. System $\mathcal{R}$ works like $\mathcal{S}$, but whenever it reads the 1 letter, it passes to the next copy of the set of states. If it is in the last copy, no more 1 letters can be read. And only the states from the last copy (those which were accepting in $\mathcal{S}$) are accepting. This way $\mathcal{R}$ accepts all words from the language of $\mathcal{S}$ which contain k letters 1, which is what we want.

By Theorem 1.5 $\mathcal{R}$ has an accepting run of length at most $\exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. However this run reads $|w|$ letters, so it has length at least $|w| \geq \exp_{2n}(k)$. This is a contradiction, as $\exp_{2n}(k) > \exp_{2n-1}(8(k+1)^2|Q|^2|\Gamma|)$. This finishes the proof of Theorem 1.4.

As a side remark we observe that the same proof works for the hierarchy of ε-closures of configuration graphs. Let G_n be the graph whose nodes are

$$\{1^k 0^m : m \leq \exp_n(k),\ \ k \in \mathbb{N}\} \cup \{1^k 0^{\exp_n(k)} \sharp : k \in \mathbb{N}\},$$

and a node w is connected with node wa by an edge labelled a (where $a \in \{0, 1, \sharp\}$). We know that graph G_{2n} is the ε-closure of the configuration graph of a level-$(2n+1)$ pushdown system (similar to the one recognizing L_{2n}). On the other hand, G_{2n} cannot be the ε-closure of the configuration graph of a level-n pushdown system. If such system exists, it can be easily transformed into a word-accepting n-CPS recognizing L_{2n}; it is enough to stop in an accepting state instead of reading $\sharp$. The same can be done for trees.

4 Types of stacks

For the rest of this section we fix some n-CPS $\mathcal{S}$. The aim of this section is to assign to any k-stack s^k a type $\mathsf{type}(s^k)$ that determines existence of some runs starting in a stack with the topmost k-stack s^k. In order to obtain Theorem 1.1 we are interested in runs in which every transition except the last is labelled by ε, and the last is labelled by a letter from A. Additionally we want to know whether it can be further extended to read some next letter

[4] Thus there are no infinite runs reading no letters, and only configurations just after reading a letter can have no successors; whether this is the case is determined by the state.

(as otherwise we have to hold the system immediately to ensure strong determinism). For this reason we fix a morphism $\varphi\colon (A\cup\{\varepsilon\})^* \to M$ into a finite monoid M, which[5] applied to a word w tells us for each $a \in A$ whether $w \in \varepsilon^* a$ and whether $w \in \varepsilon^* a \varepsilon^* A$. The morphism φ will be applied to the word created from labels of a run, i.e. to the word read by a run, but including all epsilons; for simplicity of notation we just say that we apply φ to a run.

The idea of defining types is present also in [11] (and in [17] for automata without collapse). The novelty is that we bound here (see Subsection 6.1) the length of runs implied by the type (the previous proof instead of giving such bound was using a nontrivial induction). Moreover we define the types more carefully than in [11], so that their number is exponentially smaller (and hence our algorithm for the emptiness problem is $(n-1)$-EXPTIME, not n-EXPTIME). On the other hand the types in [11] are much more general, as we characterize here only accepting runs, while there much more sophisticated kinds of runs were characterized; however the new proof can be easily generalized to those other kinds of runs.

The type of s^k will be a set of *run descriptors* which come from a set $\mathcal{T}^k$ that will be defined inductively from $k = n$ to $k = 0$. A typical element of $\mathcal{T}^k$ has the form

$$\sigma = (q, \Psi^n, \Psi^{n-1}, \dots, \Psi^{k+1}, m, q')$$

where $q, q' \in Q$ are states of $\mathcal{S}$, Ψ^i are some types of i-stacks, and $m \in M$. Let us explain the intended meaning of such a tuple. We want to have $\sigma \in \mathsf{type}(s^k)$ if and only if for all stacks $t^n, t^{n-1}, \dots, t^{k+1}$ where $\Psi^i \subseteq \mathsf{type}(t^i)$ there is a run evaluating to m under φ, from configuration $(q, t^n : t^{n-1} : \dots : t^{k+1} : s^k)$ to a configuration with state q'. In other words, if we put σ into $\mathsf{type}(s^k)$ we make the following claim. If for each $k+1 \le i \le n$ we take an i-stack t^i that satisfies the claims made by Ψ^i, then we will find a run evaluating to m that starts in state q and the stack obtained by putting s^k on top of the sequence of $t^n : \dots : t^{k+1}$, and ends in state q'. Let us mention that in order to obtain Theorem 1.1 we need not to keep the state q'.

We first introduce the set $\mathcal{T}^k$ of possible run descriptors of level k (the possible types of k-stacks are elements of $\mathcal{P}(\mathcal{T}^k)$). We write $\mathcal{P}(X)$ for the power set of X, and $\mathcal{P}_{\le 1}(X)$ for $\{Y \in \mathcal{P}(X) : |Y| \le 1\}$.

▶ **Definition 4.1.** We define $\mathcal{T}^n = Q \times M \times Q$, and inductively for $0 \le k \le n-1$:

$$\mathcal{T}^k = Q \times \mathcal{P}_{\le 1}(\mathcal{T}^n) \times \big(\mathcal{P}(\mathcal{T}^{n-1}) \times \mathcal{P}(\mathcal{T}^{n-2}) \times \dots \times \mathcal{P}(\mathcal{T}^{k+1})\big) \times M \times Q.$$

Notice that the n-th level is treated differently: we take $\mathcal{P}_{\le 1}(\mathcal{T}^n)$ instead of $\mathcal{P}(\mathcal{T}^n)$. This is possible because an n-stack can be used only once (cannot be copied), so we need just one run descriptor. The purpose for restricting to $\mathcal{P}_{\le 1}(\mathcal{T}^n)$ is to decrease exponentially the number of all run descriptors; for all other reasons a definition with $\mathcal{P}(\mathcal{T}^n)$ would be also good.

Next, we state the intended meaning of run descriptors and types.

▶ **Lemma 4.2.** *Let $0 \le l \le n$, and let $c = (q, s^n : s^{n-1} : \dots : s^l)$ be a configuration. The following two conditions are equivalent:*

1. *there exists a run from c which evaluates to m under φ and ends in state q',*
2. $\mathsf{type}(s^l)$ *contains a run descriptor $(q, \Psi^n, \Psi^{n-1}, \dots, \Psi^{l+1}, m, q')$ such that $\Psi^i \subseteq$* $\mathsf{type}(s^i)$ *for $l+1 \le i \le n$.*

[5] The results about types hold for any morphism; this morphism is just what we need for Theorem 1.1.

Let us remark that in Section 6 we strengthen the “2⇒1” implication of the above lemma: we not only say that the run exists, but we present a deterministic CPS which reconstructs such run (arbitrarily choosing one of them), and we also bound the length of this run.

Now we come to the definition of types. We first define how types can be composed. The intention of the next definition is that when Ψ^k is the type of a k-stack s^k, and Ψ^{k-1} is the type of a $(k-1)$-stack s^{k-1}, then $\Psi^k : \Psi^{k-1}$ is the type of $s^k : s^{k-1}$.

▶ **Definition 4.3.** Let $1 \leq k \leq n$, let Ψ^k be a subset of $\mathcal{T}^k$, and Ψ^{k-1} a subset of $\mathcal{T}^{k-1}$. Their *composition*, $\Psi^k : \Psi^{k-1}$, is a subset of $\mathcal{T}^k$ containing all run descriptors $(q, \Sigma^n, \Sigma^{n-1}, \dots, \Sigma^{k+1}, m, q')$ such that in Ψ^{k-1} there is a run descriptor $(q, \Sigma^n, \Sigma^{n-1}, \dots, \Sigma^{k+1}, \Sigma^k, m, q')$ for which $\Sigma^k \subseteq \Psi^k$.

Like for stacks, we write $\Psi^2 : \Psi^1 : \Psi^0$ for $\Psi^2 : (\Psi^1 : \Psi^0)$. Notice that the composition of types is monotone: if $\Psi^k \subseteq \Phi^k$ and $\Psi^{k-1} \subseteq \Phi^{k-1}$, then also $\Psi^k : \Psi^{k-1} \subseteq \Phi^k : \Phi^{k-1}$.

Next, we are going to define a function cons. In fact, cons is defined as a fixpoint of a sequence $(\mathsf{cons}_z)_{z\in\mathbb{N}}$. For each $z \in \mathbb{N}$, each stack symbol $\alpha \in \Gamma$, each number $1 \leq K \leq n$, and each $\Sigma^K \subseteq \mathcal{T}^K$ we define a set $\mathsf{cons}_z(\alpha, K, \Sigma^K) \subseteq \mathcal{T}^0$. The intention is that if a run descriptor $(q, \Psi^n, \Psi^{n-1}, \dots, \Psi^1, m, q')$ is in the set $\mathsf{cons}_z(\alpha, K, \Sigma^K)$, then there exists a run evaluating to m, ending in state q, and starting in a configuration $(q, s^n : s^{n-1} : \cdots : s^0)$ such that Ψ^i is contained in the type of s^i and s^0 has symbol α and carries a link of level K to a stack of type Σ^K. When we enter the fixpoint $\mathsf{cons}(a, K, \Sigma^K)$ we will be able to replace the “if-then” by an “if and only if”. For the “and only if” part, we need to know the complete type; when we reach the fixpoint cons of the functions cons_z, it will compute the complete type.

▶ **Definition 4.4.** Let $z \in \mathbb{N}$, let $\alpha \in \Gamma$, let $1 \leq K \leq n$, and let $\Sigma^K \subseteq \mathcal{T}^K$. For $z = 0$ we define $\mathsf{cons}_z(\alpha, K, \Sigma^K) = \emptyset$. For $z > 0$ assume that cons_{z-1} is already defined. We define $\mathsf{cons}_z(\alpha, K, \Sigma^K)$ as the set containing all run descriptors $(q_0, \Psi^n, \Psi^{n-1}, \dots, \Psi^1, m', q')$ such that

1. $q' = q_0$ and $m' = \mathbf{1}_M$ is the identity element of M, or
2. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{pop}^k)$, $m' = \varphi(a)m$, and in Ψ^k there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \dots, \Phi^{k+1}, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $k+1 \leq i \leq n$, or
3. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{col}^K)$, $m' = \varphi(a)m$, and in Σ^K there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \dots, \Phi^{K+1}, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $K+1 \leq i \leq n$, or
4. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{push}^1_{\beta,k})$, $m' = \varphi(a)m$, and in $\mathsf{cons}_{z-1}(\beta, k, \Psi^k)$ there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \dots, \Phi^1, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $2 \leq i \leq n$, and $\Phi^1 \subseteq \Psi^1 : \mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$, or
5. $\mathcal{S}$ has a transition $(q_0, \alpha, a, q_1, \mathsf{push}^k)$ where $k \geq 2$, $m' = \varphi(a)m$, and in $\mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$ there is a run descriptor $(q_1, \Phi^n, \Phi^{n-1}, \dots, \Phi^1, m, q')$ such that $\Phi^i \subseteq \Psi^i$ for $1 \leq i \leq k-1$ and for $k+1 \leq i \leq n$, and $\Phi^k \subseteq \Psi^k : \Psi^{k-1} : \cdots : \Psi^1 : \mathsf{cons}_{z-1}(\alpha, K, \Sigma^K)$.

Notice that the sequence cons_z is monotone with respect to both z and Σ^K: for $\Sigma^K \subseteq \Sigma'^K$ and each $z \in \mathbb{N}$ we have $\mathsf{cons}_z(\alpha, K, \Sigma^K) \subseteq \mathsf{cons}_{z+1}(\alpha, K, \Sigma'^K)$. Independent of $z \in \mathbb{N}$, the domain and range of cons_z are fixed finite sets whence there is some $z_\infty \in \mathbb{N}$ such that $\mathsf{cons}_{z_\infty} = \mathsf{cons}_{z_\infty - 1}$. This fixpoint is denoted as cons. Next, we define types of stacks of arbitrary level.

▶ **Definition 4.5.** We define $\mathsf{type}(s^k)$ for each k-stack s^k (for $0 \leq k \leq n$) by induction on the structure of s^k. Assume that $k = 0$ and $s^k = (\alpha, K, t^K)$ where $\alpha \in \Gamma$, $1 \leq K \leq n$, and t^K is a K-stack such that $\mathsf{type}(t^K)$ is already defined. In this case we set $\mathsf{type}(s^k) = \mathsf{cons}\left(\alpha, K, \mathsf{type}(t^K)\right)$. Otherwise $k \geq 1$; if s^k is empty, we set $\mathsf{type}(s^k) = \emptyset$. Finally, assume that $k \geq 1$ and $s^k = t^k : t^{k-1}$ such that $\mathsf{type}(t^k)$ and $\mathsf{type}(t^{k-1})$ are already defined. In this case we set $\mathsf{type}(s^k) = \mathsf{type}(t^k) : \mathsf{type}(t^{k-1})$.

Observe that $\mathsf{type}(s^k : s^{k-1} : \cdots : s^l) = \mathsf{type}(s^k) : \mathsf{type}(s^{k-1}) : \cdots : \mathsf{type}(s^l)$. From Definition 4.3 it follows that we have $(q, \Sigma^n, \Sigma^{n-1}, \dots, \Sigma^{k+1}, m, q') \in \Psi^k : \Psi^{k-1} : \cdots : \Psi^l$ if and only if in Ψ^l there is a run descriptor $(q, \Sigma^n, \Sigma^{n-1}, \dots, \Sigma^{l+1}, m, q')$ such that $\Sigma^i \subseteq \Psi^i$ for $l+1 \leq i \leq k$. It follows that the second condition of Lemma 4.2 for one value of l immediately implies this condition for all other values of l.

The proof of the "1⇒2" implication of Lemma 4.2 is almost a straightforward induction on the length of the run (we prove it for $l = 0$); the proof is in fact slightly complicated because we are using $\mathcal{P}_{\leq 1}(\mathcal{T}^n)$ instead of $\mathcal{P}(\mathcal{T}^n)$ in the definition of type, but these are just technical complications. The opposite implication follows from [11], and also follows from the facts proved later in this paper: we not only prove that the run (from item 1 of the lemma) exists, but we construct a deterministic CPS which simulates some (arbitrarily chosen) such run.

Next, let us calculate the number of run descriptors.

▶ Proposition 4.6.

$$\sum_{k=1}^{n} |\mathcal{T}^n| \leq |\mathcal{T}^0| \leq \frac{1}{2} \exp_{n-1}(4|Q|^2).$$

4.1 The algorithm

Let us now describe the algorithm for the emptiness problem. By Lemma 4.2 there is an accepting run from the initial configuration if and only if $(q_I, \emptyset, \emptyset, \dots, \emptyset, m, q') \in \mathsf{cons}(\perp, 1, \emptyset)$ for any m and any accepting state q', where q_I is the initial state and $\perp$ the initial stack symbol (recall that the type of an empty stack is empty). Thus it is enough to calculate the sets $\mathsf{cons}(\alpha, K, \Sigma^K)$ for all values of α, K, Σ^K. We do that directly from Definition 4.4. Notice that it can be done in time polynomial in the size of $\mathcal{T}^0$ and in the number of triples (α, K, Σ^K), which by Proposition 4.6 are $n-1$ times exponential in the number of states, and polynomial in the size of the stack alphabet.

Let us remark that in the same way we can check whether from any configuration (q, s) there is an accepting run: it is enough to calculate $\mathsf{type}(s)$ (the algorithm is $n-1$ times exponential in the number of states, and polynomial in the size of the stack alphabet and in the size of s).

5 Computing the types

In this section we show that a CPS can at each moment maintain the type of its current stack. In fact the same can be done for any homomorphism, defined as follows. A *stack algebra of level n* $\mathcal{A} = (A^0, A^1, \dots, A^n)$ over a stack alphabet Γ is an algebra which has $n+1$ sorts (of level $0, 1, \dots, n$) and for each $1 \leq k \leq n$ operations $\mathsf{empty}^k : A^k$, $\mathsf{comp}^k : A^k \times A^{k-1} \to A^k$ (which we denote using the colon symbol), and $\mathsf{cons}(\alpha, k, \cdot) : A^k \to A^0$ for each $\alpha \in \Gamma$. A typical stack algebra of level n is the algebra of all stacks: in the sort of level k we have stacks of level k; empty^k returns an empty stack of level k, $\mathsf{comp}^k(s^k, s^{k-1}) = s^k : s^{k-1}$ puts stack s^{k-1} on top of stack s^k, and $\mathsf{cons}(\alpha, k, s^k)$ constructs the 0-stack with label α and a link to s^k. Notice that it is in fact the free algebra (without generators). But the set of all run descriptors is also a stack algebra of level n: $\mathcal{P}(\mathcal{T}^k)$ is the sort of level k, and the empty^k operation returns the empty set. Moreover type is the unique homomorphism between these two algebras.

Let $\mathcal{A} = (A^0, A^1, \dots, A^n)$ be a finite stack algebra of level n (letter A is also used to denote the input alphabet), and f the unique homomorphism from the algebra of stacks to $\mathcal{A}$. We define f-driven n-CPS's as an extension of n-CPS's which work as follows. The transitions of an *f-driven n-CPS* are elements of

$$Q \times A^n \times A^{n-1} \times \dots \times A^1 \times \Gamma \times \Big(\bigcup_{k=1}^{n} \{k\} \times A^k \Big) \times A \times Q \times OP^n_\Gamma.$$

From a configuration $(q, s^n : s^{n-1} : \dots : s^1 : (\alpha, k, t^k))$ we can perform a transition

$$(q, f(s^n), f(s^{n-1}), \dots, f(s^1), \alpha, k, f(t^k), a, q', op),$$

which reads letter a, changes state to q', and performs operation op on the stack. In other words, the set of transitions which can be applied to a configuration depend not only on the state and the topmost stack symbol, but also on the values of f on all stacks s^i (the topmost i-stack with removed its topmost $(i-1)$-stack) and on the stack t^k (the stack to which we have a link in the topmost 0-stack). The *size* of an f-driven n-CPS is the number of its transitions.

We have the following lemma (which is very similar to Theorem 3 in [4]). It says that a CPS can maintain the values of f applied to its current stack, so its transitions may depend on these values (like it is for an f-driven CPS). The proof is not difficult: we just have to keep in the topmost symbol of each substack (of any level) the value of f applied to this substack.

▶ **Lemma 5.1.** *Let $\mathcal{S}$ be a strongly deterministic f-driven n-CPS. Then there exists a strongly deterministic n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$. For every $F \subseteq Q$ we have $L(\mathcal{S}, F) = L(\mathcal{S}, F)$ (in particular $\mathcal{S}'$ contains the states of $\mathcal{S}$); if from every reachable configuration of $(\mathcal{S}, F)$ there is an accepting run, the same holds for $(\mathcal{S}', F)$. The size of $\mathcal{S}'$ is linear in the size of $\mathcal{S}$, and $\mathcal{S}'$ can be computed in such time.*

The proof of Theorem 1.2 becomes now straightforward. Notice that, for an n-stack $s = s^n : s^{n-1} : \dots : s^1 : (\alpha, k, t^k)$, it is enough to know α and k and $\mathsf{type}(t^k)$ and $\mathsf{type}(s^i)$ for all $1 \leq i \leq n$ in order to determine $\mathsf{type}(op(s))$ for any stack operation op (for example $\mathsf{type}(\mathsf{pop}^j(s)) = \mathsf{type}(s^n) : \mathsf{type}(s^{n-1}) : \dots : \mathsf{type}(s^j)$, etc.). Moreover $\mathsf{type}(op(s))$ determines if there is an accepting run from configuration $(q, op(s))$ (by Lemma 4.2 taken for $l = n$ such run exists if and only if $(q, m, q') \in \mathsf{type}(op(s))$ for some m and some $q' \in F$).[6] Denote by $good(q, op)$ the set of those tuples

$$(\mathsf{type}(s^n), \mathsf{type}(s^{n-1}), \dots, \mathsf{type}(s^1), \alpha, k, \mathsf{type}(t^k))$$

for which there is an accepting run from configuration $(q, op(s))$. Then from $(\mathcal{S}, F)$ we construct a type-driven n-CPS $(\mathcal{S}', F)$, in which from every reachable configuration there is an accepting run, as follows. For each transition (q, α, a, q', op) of $\mathcal{S}$, to $\mathcal{S}'$ we add those transitions

$$(q, \Psi^n, \Psi^{n-1}, \dots, \Psi^1, \alpha, k, \Sigma^k, a, q', op)$$

for which $(\Psi^n, \Psi^{n-1}, \dots, \Psi^1, \alpha, k, \Sigma^k) \in good(q', op)$. So in $\mathcal{S}'$ a transition will be performed only if it leads to a configuration from which there is an accepting run. Moreover $(\mathcal{S}', F)$

[6] This is particular means that op can be applied to s: if $op = \mathsf{pop}^k$ (or col^k) would result in an empty k-stack, $\mathsf{type}(op(s))$ would be empty.

accepts the same words as $(\mathcal{S}, F)$ since we do not remove transitions of accepting runs. By applying Lemma 5.1 we obtain from $(\mathcal{S}', F)$ a deterministic n-CPS having the same property.[7] Let us calculate the size (the number of transitions) of $\mathcal{S}'$ (hence of the resulting n-CPS). Notice that $|\mathcal{T}^n|$ and $|\mathcal{T}^{n-1}|$ are polynomial in the size of $\mathcal{S}$, and $|\mathcal{T}^i| = |\mathcal{T}^{i+1}| \cdot 2^{|\mathcal{T}^{i+1}|}$ for $i < n-1$ is $n-1-i$ times exponential in the size of $\mathcal{S}$. In particular $|\mathcal{T}^1|$ is the greatest and is $n-2$ times exponential in the size of $\mathcal{S}$. The transitions of $\mathcal{S}'$ are defined for subsets of $\mathcal{T}^i$, hence their number is at most $n-1$ times exponential in the size of $\mathcal{S}$.

6 Reconstructing a run

In this section we sketch how Theorem 1.1 can be proved. Fix some n-CPS $\mathcal{S}$ (with states Q), for which $T(\mathcal{S})$ is deterministic. As a first step we would like to construct a deterministic type-driven n-CPS $\mathcal{R}$ which would uniquely choose some run of $\mathcal{S}$ in the following sense. System $\mathcal{R}$ has the same stack alphabet as $\mathcal{S}$, and for each state $q \in Q$ of $\mathcal{S}$, system $\mathcal{R}$ also has the state q, as well a state $\mathsf{start}_{q,a}$ for each $a \in A$. Assume that $\mathcal{S}$ has a run whose labels form a word from $\varepsilon^* a$ (for some $a \in A$), starting in a reachable configuration $c = (q, s)$, and ending in a configuration $d = (q', s')$ (by determinism of $T(\mathcal{S})$, there is at most one such d for given a and c). Then in $\mathcal{R}$ there is a run from $(\mathsf{start}_{q,a}, s)$ to d using only ε-transitions, and not using states from Q (except its last configuration, which is d).

Assume we have such $\mathcal{R}$.[8] Then we can construct a strongly deterministic type-driven n-CPS $\mathcal{S}'$ such that $T(\mathcal{S}) = T(\mathcal{S}')$ as follows. Let $\mathcal{G}$ be the ε-closure of the configuration graph of $\mathcal{S}$, and G its set of nodes; in other words G contains the initial configuration and all configurations reachable by a run which ends by a non-ε-transition. Let also $H \subseteq G$ be its subsets containing only those of them, from which there exists a run containing some non-ε-transition (i.e. a run to another configuration from G). The set of configurations of $\mathcal{S}'$ which have state from Q are exactly those from H. In particular the initial configuration of $\mathcal{S}$ and $\mathcal{S}'$ is the same. All non-ε-transitions will be going from elements of H (i.e. configurations with state from Q). For each such configuration $c = (q, s)$, type determines the labels of edges outgoing from c in $\mathcal{G}$. Indeed, there is an edge from c to some $d \in G$ labelled by some $a \in A$ if and only if there is a run from c whose labels form a word from $\varepsilon^* a$; thus if and only if $(q, m, q') \in \mathsf{type}(s)$ for any element m which is the image of a word from $\varepsilon^* a$ and for any q'. Notice also that configuration d is unique for given c and a, by determinism of $T(\mathcal{S})$. Moreover, type determines whether $d \in H$: this is the case when there is a run from c whose labels form a word from $\varepsilon^* a \varepsilon^* A$; thus when $(q, m, q') \in \mathsf{type}(s)$ for any element m which the image of a word from $\varepsilon^* a \varepsilon^* A$ and for any q'. In such situation we add an a-labelled transition to the state $\mathsf{start}_{q,a}$ which does not change the stack (e.g. we make a push and then a pop). Then the system will simulate deterministically some run to d, making only ε-transitions (this is already realized by $\mathcal{R}$). Otherwise (for $d \in G \setminus H$), we just make an a-labelled transition to a dead state q_{die}, from which there are no more transitions. This way $T(\mathcal{S}) = T(\mathcal{S}')$, and $\mathcal{S}'$ is strongly deterministic.

It remains to construct $\mathcal{R}$. Let $c = (q, s^n : s^{n-1} : \cdots : s^0)$ be a configuration such that in $\mathsf{type}(s^0)$ we have a run descriptor $\sigma = (q, \Psi^n, \Psi^{n-1}, \dots, \Psi^1, m, q')$ such that $\Psi^i \subseteq \mathsf{type}(s^i)$ for $1 \leq i \leq n$. Lemma 4.2 says that then there exists a run from c which evaluates to m under φ and ends in state q'. But how to simulate it using a type-driven CPS? It is easy to

[7] In order to obtain strong determinism, an additional step has to be performed.

[8] In the actual construction $\mathcal{R}$ is slightly more complicated (we keep some additional information on the stack).

perform some first step of such run; we just need to follow a transition used in Definition 4.4 to add σ to $\mathsf{type}(s^0)$. Moreover this leads to a configuration which again satisfies the same condition, so we can repeat the same. When we reach a configuration in which the first point of Definition 4.4 is used, we are done: we have finished the run.

The problem is that by such construction we can obtain an infinite run (e.g. a loop). The reason why the "2⇒1" implication of Lemma 4.2 holds is that cons is defined as the smallest fixpoint, not any fixpoint. According to the definition of cons_z, after a push operation it is enough to use cons_{z-1} both for the 0-stack which was topmost till now, and for the new topmost 0-stack. Thus we should keep the maximal allowed value of z with each 0-stack, and decrease this value for each push operation. Then for calculating the type of a bigger stack we should only use this cons_{z-1}, not whole cons.

There is also a technical difficulty that the values of z stored with each symbol have to be reset everywhere to z_∞ (recall that z_∞ is a number for which $\mathsf{cons} = \mathsf{cons}_{z_\infty}$) when we finally reach a configuration from G, and we want to run the simulation again. Of course we cannot do this explicitly, but we can just put some marker on the stack, which denotes that the value of z is z_∞ everywhere below. Then above the marker we work as previously, and we assume that bellow the marker all symbols have z_∞ on its second coordinate. More precisely, we need a separate kind of marker for each level: in each k-stack we will mark the bottommost $(k-1)$-stack in which the z values are actual; we also need another kind of marker to denote the 0-stacks in which the z values in the link are not actual.

▶ **Example 6.1.** Consider the 1-CPS having the following transitions.

$$\begin{array}{lll}(\perp, q_1, a, q_2, \mathsf{push}^1_{x,1}) & (x, q_3, \varepsilon, q_1, \mathsf{push}^1_{x,1}) & (x, q_1, \varepsilon, q_1, \mathsf{pop}^1) \\ (x, q_2, \varepsilon, q_3, \mathsf{push}^1_{x,1}) & (x, q_3, \varepsilon, q_2, \mathsf{pop}^1) & \end{array}$$

Then the types contain the following run descriptors (not all are shown).

$$\begin{array}{ll}(q_1, \emptyset, \varepsilon^* a, q_2) \in \mathsf{cons}(\perp, 1, \emptyset) & (q_2, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_3(x, 1, \emptyset) \\ \tau = (q_1, \varepsilon^* a, q_2) \in \mathsf{type}([\perp]) & \sigma = (q_2, \varepsilon^* a, q_2) \in \mathsf{type}([\perp x]) \\ (q_1, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_1(x, 1, \emptyset) & (q_3, \{\tau, \sigma\}, \varepsilon^* a, q_2) \in \mathsf{cons}_1(x, 1, \emptyset) \\ (q_3, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_2(x, 1, \emptyset) & \end{array}$$

The key decision to take is in configuration $(q_3, [\perp xx])$. By just choosing the run descriptor $(q_3, \{\tau, \sigma\}, \varepsilon^* a, q_2)$ from cons_z for the smallest z ($z = 1$) we make pop^1 leading to $(q_2, [\perp x])$. Now the only possibility is to use $(q_2, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_3(x, 1, \emptyset)$, and to perform $\mathsf{push}^1_{x,1}$. If we just went to $(q_3, [\perp xx])$, we would fall into a loop. Also restricting z used in this configuration (for the new topmost x) does not help. We should instead go to a configuration $(q_3, [\perp x'x'])$, where x' is a version of x which implies that only $z \leq 2$ can be used. Then $\sigma \notin \mathsf{type}([\perp x'])$, so we cannot perform the pop^1 transition again. We have to use $(q_3, \{\tau\}, \varepsilon^* a, q_2) \in \mathsf{cons}_2(x, 1, \emptyset)$, thus we perform $\mathsf{push}^1_{x,1}$ and we leave the loop.

6.1 Bound on the run length

Next we notice that the method described above not only guarantees that the obtained run is finite, but also gives a concrete bound on its length. Let us repeat that in the system $\mathcal{R}$, which deterministically simulates one run of $\mathcal{S}$, we keep with each stack symbol a natural number z (which is between 0 and z_∞). Whenever we perform some push^k operation, this value is decreased in the topmost 0-stack, both in the original $(k-1)$-stack and in its copy. We want to argue that every run working like that will terminate. In order to obtain that we define potential of a stack.

▶ **Definition 6.2.** Let $0 \le k \le n$, and let s^k be a k-stack over alphabet $\Gamma \times \{0, 1, \dots, z_\infty\}$. We define a natural number $\mathsf{pt}(s^k)$ (the *potential* of s^k) by induction on the structure of s^k.

- When $k = 0$ and $s^k = ((\alpha, z), K, t^K)$, we take $\mathsf{pt}(s^k) = 2z$.
- When $k \ge 1$ and s^k is empty, we take $\mathsf{pt}(s^k) = 0$.
- When $k \ge 1$ and $s^k = t^k : t^{k-1}$, we take

$$\mathsf{pt}(s^k) = \mathsf{pt}(t^k) + 2^{\mathsf{pt}(t^{k-1})}.$$

It is not difficult to see that the potential is decreased by each operation (assuming that the push operations behave as described above). Thus the length of the run reconstructed by $\mathcal{R}$ is bounded by the potential of the starting configuration. If we forget about the system $\mathcal{R}$, we obtain the following strengthened version of the ‘2⇒1” implication of Lemma 4.2 (for $l = n$).

▶ **Lemma 6.3.** *Let $c = (q, s)$ be a configuration of $\mathcal{S}$, and let $\widetilde{s}$ be the stack over alphabet $\Gamma \times \{0, 1, \dots, z_\infty\}$ obtained from s by appending z_∞ to each stack symbol. Assume that $(q, m, q') \in \mathsf{type}(s)$. Then there exists a run from c which evaluates to m under φ, ends in state q', and has length at most $\mathsf{pt}(\widetilde{s})$.*

Let us conclude by a proof of Theorem 1.5. By assumption some accepting run exists from the initial configuration (q_I, s_I). By Lemma 4.2 this means that $(q_I, m, q') \in \mathsf{type}(s_I)$ for some $m \in M$ and some accepting state q'. Using the above lemma we obtain back an accepting run, but now its length is bounded by the potential of the initial configuration. Notice that the potential is n times exponential in z_∞. Moreover z_∞ is $n-1$ times exponential in the number of states of the system, which follows from Proposition 4.6. Thus we obtain an accepting run of length $2n - 1$ times exponential in the size of the system; a precise calculation gives the value written in Theorem 1.5.

7 Conclusions

Let us mention that the methods presented in this paper can be used in a much more general context. In [11] we define types of stacks which characterize a lot of interesting kinds of runs, called top^k-non-erasing runs, pumping runs, k-returns, k-colreturns; additionally a general definition is given, which allows to consider also other classes of runs having some properties. To be concrete: we can consider for example a class of runs such that the topmost k-stack at the end is equal to the topmost k-stack at the beginning, and is indeed obtained as its copy. It is almost immediate to generalize the results of this paper to all these classes of runs:

- The number of run descriptors for these more general types is $n - 1$ times exponential.
- The type function is still a homomorphism of stack algebras, so it can be computed by the CPS, as in Section 5.
- We can deterministically reconstruct runs of that kind, like in Section 6.
- The length of such runs can be bounded like in Lemma 6.3.

Observe that most of these classes of runs cannot (at least in any natural way) be defined using μ-calculus in the configuration graph.

Techniques similar to ours were used independently in a recent paper [3]. Moreover some of our results can be deduced from another independent work [6].

Future work. One open question is whether the hierarchy of word languages recognized by collapsible pushdown systems is strict (if every two its levels are different). Notice that we only show that the hierarchy is infinite (that there are infinitely many different levels). One possible way to obtain the strictness would be to show that if two levels coincide, then all higher levels have to coincide (however we do not see any easy reason for that). A second way would be to improve the bounds given in our proof.

It is also an open problem whether these languages are context-sensitive.

A second direction would be to extend our approach of testing emptiness to the μ-calculus model checking problem (and obtain a simple algorithm for that problem, which uses collapsible pushdown systems directly).

Another interesting question is about the relation between the classes of word languages recognized by collapsible and non-collapsible pushdown systems. It is known that on level 2 these classes coincide [1] (unlike for trees and graphs); nevertheless the conjecture is that for higher levels these classes are different.

References

1 K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In V. Sassone, editor, *FoSSaCS*, volume 3441 of *Lecture Notes in Computer Science*, pages 490–504. Springer, 2005.

2 A. Blumensath. On the structure of graphs in the Caucal hierarchy. *Theor. Comput. Sci.*, 400(1-3):19–45, 2008.

3 C. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. To appear in ICALP, 2012.

4 C. H. Broadbent, A. Carayol, C.-H. L. Ong, and O. Serre. Recursion schemes and logical reflection. In *LICS*, pages 120–129. IEEE Computer Society, 2010.

5 C. H. Broadbent and C.-H. L. Ong. On global model checking trees generated by higher-order recursion schemes. In L. de Alfaro, editor, *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009.

6 A. Carayol and O. Serre. Collapsible pushdown automata and labeled recursion schemes. Equivalence, safety and effective selection. To appear in LICS, 2012.

7 A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In P. K. Pandya and J. Radhakrishnan, editors, *FSTTCS*, volume 2914 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2003.

8 D. Caucal. On infinite terms having a decidable monadic theory. In K. Diks and W. Rytter, editors, *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002.

9 W. Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982.

10 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.

11 A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. *CoRR*, abs/1201.3250, 2012.

12 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

13 N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. E. Nikoletseas, and W. Thomas, editors, *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2009.

14 A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15:1170–1174, 1974.

15 A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.

16 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, pages 81–90. IEEE Computer Society, 2006.

17 P. Parys. On the significance of the collapse operation. Accepted to LICS 2012, 2012.

18 S. Salvati and I. Walukiewicz. Krivine machines and higher-order schemes. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (2)*, volume 6756 of *Lecture Notes in Computer Science*, pages 162–173. Springer, 2011.

A Proof of Kamp's theorem

Alexander Rabinovich

The Blavatnik School of Computer Science, Tel Aviv University
rabinoa@post.tau.ac.il

Abstract

We provide a simple proof of Kamp's theorem.

1998 ACM Subject Classification F.4.1 Temporal Logic

Keywords and phrases Temporal Logic, Monadic Logic, Expressive Completeness

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.516

1 Introduction

Temporal Logic (*TL*) introduced to Computer Science by Pnueli in [8] is a convenient framework for reasoning about "reactive" systems. This has made temporal logics a popular subject in the Computer Science community, enjoying extensive research in the past 30 years. In *TL* we describe basic system properties by *atomic propositions* that hold at some points in time, but not at others. More complex properties are expressed by formulas built from the atoms using Boolean connectives and *Modalities* (temporal connectives): A k-place modality M transforms statements $\varphi_1, \dots, \varphi_k$ possibly on 'past' or 'future' points to a statement $M(\varphi_1, \dots, \varphi_k)$ on the 'present' point t_0. The rule to determine the truth of a statement $M(\varphi_1, \dots, \varphi_k)$ at t_0 is called a *truth table* of M. The choice of particular modalities with their truth tables yields different temporal logics. A temporal logic with modalities $M_1, \dots, M_k$ is denoted by $TL(M_1, \dots, M_k)$.

The simplest example is the one place modality $\Diamond X$ saying: "X holds some time in the future." Its truth table is formalized by $\varphi_{\Diamond}(t_0, X) := (\exists t > t_0)X(t)$. This is a formula of the First-Order Monadic Logic of Order (*FOMLO*) - a fundamental formalism in Mathematical Logic where formulas are built using atomic propositions $P(t)$, atomic relations between elements $t_1 = t_2$, $t_1 < t_2$, Boolean connectives and first-order quantifiers $\exists t$ and $\forall t$. Two more natural modalities are the modalities Until ("*Until*") and Since ("*Since*"). $X\mathsf{Until}Y$ means that X will hold from now until a time in the future when Y will hold. $X\mathsf{Since}Y$ means that Y was true at some point of time in the past and since that point X was true until (not necessarily including) now. Both modalities have truth tables in *FOMLO*. Most modalities used in the literature are defined by such *FOMLO* truth tables, and as a result, every temporal formula translates directly into an equivalent *FOMLO* formula. Thus, the different temporal logics may be considered as a convenient way to use fragments of *FOMLO*. *FOMLO* can also serve as a yardstick by which one is able to check the strength of temporal logics: A temporal logic is *expressively complete* for a fragment L of *FOMLO* if every formula of L with a single free variable t_0 is equivalent to a temporal formula.

Actually, the notion of expressive completeness refers to a temporal logic and to a model (or a class of models), since the question whether two formulas are equivalent depends on the domain over which they are evaluated. Any (partially) ordered set with monadic predicates is a model for *TL* and *FOMLO*, but the main, *canonical*, linear time intended models are the non-negative integers $\langle \mathbb{N}, < \rangle$ for discrete time and the reals $\langle \mathbb{R}, < \rangle$ for continuous time.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 516–527
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Kamp's theorem [7] states that the temporal logic with modalities Until and Since is expressively complete for *FOMLO* over the above two linear time canonical[1] models.

> This seminal theorem initiated the whole study of expressive completeness, and it remains one of the most interesting and distinctive results in temporal logic; very few, if any, similar 'modal' results exist. Several alternative proofs of it and stronger results have appeared; none of them are trivial (at least to most people) [6].

The objective of this paper is to provide a simple proof of Kamp's theorem.

The rest of the paper is organized as follows: In Section 2 we recall the definitions of the monadic logic, the temporal logics and state Kamp's theorem. Section 3 introduces formulas in a normal form and states their simple properties. In Section 4 we prove Kamp's theorem. The proof of one proposition is postponed to Section 5. Section 6 comments on the previous proofs of Kamp's theorem.

2 Preliminaries

In this section we recall the definitions of the first-order monadic logic of order, the temporal logics and state Kamp's theorem.

Fix a set Σ of *atoms*. We use $P, Q, R, S \dots$ to denote members of Σ. The syntax and semantics of both logics are defined below with respect to such Σ.

2.1 First-Order Monadic Logic of Order

Syntax. In the context of *FOMLO*, the atoms of Σ are referred to (and used) as *unary predicate symbols*. Formulas are built using these symbols, plus two binary relation symbols: $<$ and $=$, and a set of first-order variables (denoted: $x, y, z, \dots$). Formulas are defined by the grammar:

$$atomic ::= \quad x < y \ \mid \ x = y \ \mid \ P(x) \qquad \text{(where } \ P \in \Sigma)$$

$$\varphi ::= \quad atomic \ \mid \ \neg\varphi_1 \ \mid \ \varphi_1 \vee \varphi_2 \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \exists x \varphi_1 \ \mid \ \forall x \varphi_1$$

We will also use the standard abbreviated notation for ***bounded quantifiers***, e.g., $(\exists x)_{>z}(\dots)$ denotes $\exists x((x > z) \wedge (\dots))$, and $(\forall x)^{<z}(\dots)$ denotes $\forall x((x < z) \rightarrow (\dots))$, and $(\forall x)^{<z_2}_{>z_1}(\dots)$ denotes $\forall x((z_1 < x < z_2) \rightarrow (\dots))$, etc.

Semantics. Formulas are interpreted over *labeled linear orders* which are called *chains*. A Σ-***chain*** is a triplet $\mathcal{M} = (T, <, \mathcal{I})$ where T is a set - the *domain* of the chain, $<$ is a linear order relation on T, and $\mathcal{I} : \Sigma \rightarrow \mathcal{P}(T)$ is the *interpretation* of Σ (where $\mathcal{P}$ is the powerset notation). We use the standard notation $\mathcal{M}, t_1, t_2, \dots t_n \models \varphi(x_1, x_2, \dots x_n)$ to indicate that the formula φ with free variables among $x_1, \dots, x_n$ is satisfiable in $\mathcal{M}$ when x_i are interpreted as elements t_i of $\mathcal{M}$. For atomic $P(x)$ this is defined by: $\mathcal{M}, t \models P(x)$ iff $t \in \mathcal{I}(P)$; the semantics of $<, =, \neg, \wedge, \vee, \exists$ and $\forall$ is defined in a standard way.

2.2 TL(Until, Since) Temporal Logic

In this section we recall the syntax and semantics of a temporal logic with ***strict***-Until and ***strict***-Since modalities, denoted by TL(Until, Since).

[1] the technical notion which unifies $\langle \mathbb{N}, <\rangle$ and $\langle \mathbb{R}, <\rangle$ is Dedekind completeness.

In the context of temporal logics, the atoms of Σ are used as *atomic propositions* (also called *propositional atoms*). Formulas of $TL(\mathsf{Until}, \mathsf{Since})$ are built using these atoms, Boolean connectives and ***strict***-Until and ***strict***-Since binary modalities. The formulas are defined by the grammar:

$$F ::= \text{ True } \mid P \mid \neg F_1 \mid F_1 \vee F_2 \mid F_1 \wedge F_2 \mid F_1 \mathsf{Until} F_2 \mid F_1 \mathsf{Since} F_2,$$

where $P \in \Sigma$.

Semantics. Formulas are interpreted at *time-points* (or *moments*) in chains (elements of the domain). The semantics of $TL(\mathsf{Until}, \mathsf{Since})$ formulas is defined inductively: Given a chain $\mathcal{M} = (T, <, \mathcal{I})$ and $t \in T$, define when a formula F *holds* in $\mathcal{M}$ at t - denoted $\mathcal{M}, t \models F$:

- $\mathcal{M}, t \models P$ iff $t \in \mathcal{I}(P)$, for any propositional atom P.
- $\mathcal{M}, t \models F \vee G$ iff $\mathcal{M}, t \models F$ or $\mathcal{M}, t \models G$; similarly for $\wedge$ and $\neg$.
- $\mathcal{M}, t \models F_1 \mathsf{Until} F_2$ iff there is $t' > t$ such that $\mathcal{M}, t' \models F_2$ and $\mathcal{M}, t_1 \models F_1$ for all $t_1 \in (t, t')$.
- $\mathcal{M}, t \models F_1 \mathsf{Since} F_2$ iff there is $t' < t$ such that $\mathcal{M}, t' \models F_2$ and $\mathcal{M}, t_1 \models F_1$ for all $t_1 \in (t', t)$.

We will use standard abbreviations. As usual $\Box F$ (respectively, $\overleftarrow{\Box} F$) is an abbreviation for $\neg(\text{True}\mathsf{Until}(\neg F))$ (respectively, $\neg(\text{True}\mathsf{Since}(\neg F)))$, and $\mathsf{K}^+(F)$ (respectively, $\mathsf{K}^-(F)$) is an abbreviation for $\neg((\neg F)\mathsf{Until}\text{True})$ (respectively, $\neg((\neg F)\mathsf{Since}\text{True})$).

1. $\Box F$ (respectively, $\overleftarrow{\Box} F$) holds at t iff F holds everywhere after (respectively, before) t.
2. $\mathsf{K}^-(F)$ holds at a moment t iff $t = \sup(\{t' \mid t' < t \text{ and } F \text{ holds at } t'\})$.
3. $\mathsf{K}^+(F)$ holds at a moment t iff $t = \inf(\{t' \mid t' > t \text{ and } F \text{ holds at } t'\})$.

Note that $\mathsf{K}^+(\text{True})$ (respectively, $\mathsf{K}^-(\text{True})$) holds at t in $\mathcal{M}$ if t is a right limit (respectively, a left limit) point of the underlining order. In particular, both $\mathsf{K}^+(\text{True})$ and $\mathsf{K}^-(\text{True})$ are equivalent to False in the chains over $(\mathbb{N}, <)$,

2.3 Kamp's Theorem

Equivalence between temporal and monadic formulas is naturally defined: F is equivalent to $\varphi(x)$ over a class $\mathcal{C}$ of structures iff for any $\mathcal{M} \in \mathcal{C}$ and $t \in \mathcal{M}$: $\mathcal{M}, t \models F \Leftrightarrow \mathcal{M}, t \models \varphi(x)$. If $\mathcal{C}$ is the class of all chains, we will say that F is equivalent to φ.

A linear order $(T, <)$ is *Dedekind complete* if for every non-empty subset S of T, if S has a lower bound in T then it has a greatest lower bound, written $\inf(S)$, and if S has an upper bound in T then it has a least upper bound, written $\sup(S)$. The canonical linear time models $(\mathbb{N}, <)$ and $(\mathbb{R}, <)$ are Dedekind complete, while the order of the rationals is not Dedekind complete. A chain is Dedekind complete if its underlying linear order is Dedekind complete.

The fundamental theorem of Kamp's states that $TL(\mathsf{Until}, \mathsf{Since})$ is expressively equivalent to *FOMLO* over Dedekind complete chains.

▶ Theorem 2.1 (Kamp [7]). **1.** Given any $TL(\mathsf{Until}, \mathsf{Since})$ formula A there is a *FOMLO* formula $\varphi_A(x)$ which is equivalent to A over all chains.

2. Given any *FOMLO* formula $\varphi(x)$ with one free variable, there is a $TL(\mathsf{Until}, \mathsf{Since})$ formula which is equivalent to φ over Dedekind complete chains.

The meaning preserving translation from $TL(\mathsf{Until}, \mathsf{Since})$ to *FOMLO* is easily obtained by structural induction. The contribution of our paper is a proof of Theorem 2.1 (2). The proof is constructive. An algorithm which for every *FOMLO* formula $\varphi(x)$ constructs a $TL(\mathsf{Until}, \mathsf{Since})$ formula which is equivalent to φ over Dedekind complete chains is easily extracted from our proof. However, this algorithms is not efficient in the sense of complexity theory. This is unavoidable because there is a non-elementary succinctness gap between $FOMLO$ and $TL(\mathsf{Until}, \mathsf{Since})$ even over the class of finite chains, i.e., for every $m, n \in \mathbb{N}$ there is a $FOMLO$ formula $\varphi(x_0)$ of size $|\varphi| > n$ which is not equivalent (even over finite chains) to any $TL(\mathsf{Until}, \mathsf{Since})$ formula of size $\leq \exp(m, |\varphi|)$, where the m-iterated exponential function $\exp(m, n)$ is defined by induction on m so that $\exp(1, n) = 2^n$, and $\exp(m+1, n) = 2^{\exp(m,n)}$.

3 $\overrightarrow{\exists}\forall$ formulas

First, we introduce $\overrightarrow{\exists}\forall$ formulas which are instances of the Decomposition formulas of [3].

▶ **Definition 3.1** ($\overrightarrow{\exists}\forall$-formulas)**.** Let Σ be a set of monadic predicate names. An $\overrightarrow{\exists}\forall$-formula over Σ is a formula of the form:

$$
\begin{aligned}
\psi(z_0, \ldots, z_m) := \exists x_n \ldots \exists x_1 \exists x_0 & \\
& \left(\bigwedge_{k=0}^{m} z_k = x_{i_k}\right) \wedge (x_n > x_{n-1} > \cdots > x_1 > x_0) && \text{“ordering of } x_i \text{ and } z_j\text{”} \\
& \wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) && \text{“Each } \alpha_j \text{ holds at } x_j\text{”} \\
& \wedge \bigwedge_{j=1}^{n} [(\forall y)^{<x_j}_{>x_{j-1}} \beta_j(y)] && \text{“Each } \beta_j \text{ holds along } (x_{j-1}, x_j)\text{”} \\
& \wedge (\forall y)_{>x_n} \beta_{n+1}(y) && \text{“}\beta_{n+1} \text{ holds everywhere after } x_n\text{”} \\
& \wedge (\forall y)^{<x_0} \beta_0(y) && \text{“}\beta_0 \text{ holds everywhere before } x_0\text{”}
\end{aligned}
$$

with a prefix of $n+1$ existential quantifiers and with all α_j, β_j quantifier free formulas with one variable over Σ. (ψ has $m+1$ free variables $z_0, \ldots, z_m$ and $m+1 \leq n+1$ existential quantifiers are dummy and are introduced just in order to simplify notations.)

It is clear that

▶ **Lemma 3.2.** **1.** Conjunction of $\overrightarrow{\exists}\forall$-formulas is equivalent to a disjunction of $\overrightarrow{\exists}\forall$-formulas.
2. Every $\overrightarrow{\exists}\forall$-formula is equivalent to a conjunction of $\overrightarrow{\exists}\forall$-formulas with at most two free variables.
3. For every $\overrightarrow{\exists}\forall$-formula φ the formula $\exists x\varphi$ is equivalent to a $\overrightarrow{\exists}\forall$-formula.

▶ **Definition 3.3** ($\vee\overrightarrow{\exists}\forall$-formulas)**.** A formula is a $\vee\overrightarrow{\exists}\forall$ formula if it is equivalent to a disjunction of $\overrightarrow{\exists}\forall$-formulas.

▶ **Lemma 3.4** (closure properties)**.** The set of $\vee\overrightarrow{\exists}\forall$ formulas is closed under disjunction, conjunction, and existential quantification.

Proof. By (1) and (3) of Lemma 3.2, and distributivity of $\exists$ over $\vee$. ◀

The set of $\vee\overrightarrow{\exists}\forall$ formulas is not closed under negation[2]. However, we show later (see Proposition 4.3) that the negation of a $\vee\overrightarrow{\exists}\forall$ formula is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula in the expansion of the chains by all $TL(\mathsf{Until}, \mathsf{Since})$ definable predicates.

The $\vee\overrightarrow{\exists}\forall$ formulas with one free variable can be easily translated to $TL(\mathsf{Until}, \mathsf{Since})$.

[2] The truth table of $P\mathsf{Until}Q$ is an $\overrightarrow{\exists}\forall$ formula $(\exists x')_{>x}(Q(x') \wedge (\forall y)^{<x'}_{>x} P(y))$, yet we can prove that its negation is not equivalent to any $\vee\overrightarrow{\exists}\forall$ formula.

▶ Proposition 3.5 (From $\vee\overrightarrow{\exists}\forall$-formulas to $TL(\mathsf{Until}, \mathsf{Since})$ formulas). Every $\vee\overrightarrow{\exists}\forall$-formula with one free variable is equivalent to a $TL(\mathsf{Until}, \mathsf{Since})$ formula.

Proof. By a simple formalization we show that every $\overrightarrow{\exists}\forall$-formula with one free variable is equivalent to a $TL(\mathsf{Until}, \mathsf{Since})$ formula. This immediately implies the proposition.

Let $\psi(z_0)$ be an $\overrightarrow{\exists}\forall$-formula

$$\exists x_n \dots \exists x_1 \exists x_0 z_0 = x_k \wedge \ (x_n > x_{n-1} > \dots > x_1 > x_0) \wedge \ \bigwedge_{j=0}^{n} \alpha_j(x_j)$$

$$\wedge \ \bigwedge_{j=1}^{n} (\forall y)^{<x_j}_{>x_{j-1}} \beta_j(y) \wedge \ (\forall y)^{<x_0} \beta_0(y) \wedge \ (\forall y)_{>x_n} \beta_{n+1}(y)$$

Let A_i and B_i be temporal formulas equivalent to α_i and β_i (A_i and B_i do not even use Until and Since modalities). It is easy to see that ψ is equivalent to the conjunction of

$$A_k \wedge (B_{k+1}\mathsf{Until}(A_{k+1} \wedge (B_{k+2}\mathsf{Until} \cdots (A_{n-1} \wedge (B_n\mathsf{Until}(A_n \wedge \Box B_{n+1})) \cdots))$$

and

$$A_k \wedge (B_{k-1}\mathsf{Since}(A_{k-1} \wedge (B_{k-2}\mathsf{Since}(\cdots A_1 \wedge (B_1\mathsf{Since}(A_0 \wedge \overleftarrow{\Box} B_0)) \cdots))$$ ◀

4 Proof of Kamp's theorem

The next definition plays a major role in the proof of both Kamp's and Stavi's theorems [3].

▶ Definition 4.1. Let $\mathcal{M}$ be a Σ chain. We denote by $\mathcal{E}[\Sigma]$ the set of unary predicate names $\Sigma \cup \{A \ \mid A$ is an $TL(\mathsf{Until}, \mathsf{Since})$-formula over Σ $\}$. The canonical $TL(\mathsf{Until}, \mathsf{Since})$-expansion of $\mathcal{M}$ is an expansion of $\mathcal{M}$ to an $\mathcal{E}[\Sigma]$-chain, where each predicate name $A \in \mathcal{E}[\Sigma]$ is interpreted as $\{a \in \mathcal{M} \mid \mathcal{M}, a \models A\}$[3]. We say that first-order formulas in the signature $\mathcal{E}[\Sigma] \cup \{<\}$ are equivalent over $\mathcal{M}$ (respectively, over a class of Σ-chains $\mathcal{C}$) if they are equivalent in the canonical expansion of $\mathcal{M}$ (in the canonical expansion of every $\mathcal{M} \in \mathcal{C}$).

Note that if A is a $TL(\mathsf{Until}, \mathsf{Since})$ formula over $\mathcal{E}[\Sigma]$ predicates, then it is equivalent to a $TL(\mathsf{Until}, \mathsf{Since})$ formula over Σ, and hence to an atomic formula in the canonical $TL(\mathsf{Until}, \mathsf{Since})$-expansions.

From now on we say that "formulas are equivalent in a chain $\mathcal{M}$" instead of "formulas are equivalent in the canonical $TL(\mathsf{Until}, \mathsf{Since})$-expansion of $\mathcal{M}$." The $\overrightarrow{\exists}\forall$ and $\vee\overrightarrow{\exists}\forall$ formulas are defined as previously, but now they can use as atoms $TL(\mathsf{Until}, \mathsf{Since})$ definable predicates.

It is clear that all the results stated above hold for this modified notion of $\vee\overrightarrow{\exists}\forall$ formulas. In particular, every $\vee\overrightarrow{\exists}\forall$ formula with one free variable is equivalent to an $TL(\mathsf{Until}, \mathsf{Since})$ formula, and the set of $\vee\overrightarrow{\exists}\forall$ formulas is closed under conjunction, disjunction and existential quantification. However, now the set of $\vee\overrightarrow{\exists}\forall$ formulas is also closed under negation, due to the next proposition whose proof is postponed to Sect. 5.

▶ Proposition 4.2. (Closure under negation) The negation of $\overrightarrow{\exists}\forall$-formulas with at most two free variables is equivalent over Dedekind complete chains to a disjunction of $\overrightarrow{\exists}\forall$-formulas.

As a consequence we obtain

[3] We often use "$a \in \mathcal{M}$" instead of "a is an element of the domain of $\mathcal{M}$"

▶ Proposition 4.3. Every first-order formula is equivalent over Dedekind complete chains to a disjunction of $\overrightarrow{\exists}\forall$-formulas.

Proof. We proceed by structural induction.

Atomic It is clear that every atomic formula is equivalent to a disjunction of (even quantifier free) $\overrightarrow{\exists}\forall$-formulas.

Disjunction - immediate.

Negation If φ is an $\overrightarrow{\exists}\forall$-formula, then by Lemma 3.2(2) it is equivalent to a conjunction of $\overrightarrow{\exists}\forall$ formulas with at most two free variables. Hence, $\neg\varphi$ is equivalent to a disjunction of $\neg\psi_i$ where ψ_i are $\overrightarrow{\exists}\forall$-formulas with at most two free variables. By Proposition 4.2, $\neg\psi_i$ is equivalent to a disjunction of $\overrightarrow{\exists}\forall$ formulas γ_i^j. Hence, $\neg\varphi$ is equivalent to a disjunction $\bigvee_i \bigvee_j \gamma_i^j$ of $\overrightarrow{\exists}\forall$ formulas.

If φ is a disjunction of $\overrightarrow{\exists}\forall$ formulas φ_i, then $\neg\varphi$ is equivalent to the conjunction of $\neg\varphi_i$. By the above, $\neg\varphi_i$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula. Since, $\vee\overrightarrow{\exists}\forall$ formulas are closed under conjunction (Lemma 3.4), we obtain that $\neg\varphi$ is equivalent to a disjunction of $\overrightarrow{\exists}\forall$ formulas.

$\exists$-quantifier For $\exists$-quantifier, the claim follows from Lemma 3.4. ◀

Now, we are ready to prove Kamp's Theorem:

▶ Theorem 4.4. For every *FOMLO* formula $\varphi(x)$ with one free variable, there is a $TL(\mathsf{Until}, \mathsf{Since})$ formula which is equivalent to φ over Dedekind complete chains.

Proof. By Proposition 4.3, $\varphi(x)$ is equivalent over Dedekind complete chains to a disjunction of $\overrightarrow{\exists}\forall$ formulas $\varphi_i(x)$. By Proposition 3.5, $\varphi_i(x)$ is equivalent to a $TL(\mathsf{Until}, \mathsf{Since})$ formula. Hence, $\varphi(x)$ is equivalent over Dedekind complete chains to a $TL(\mathsf{Until}, \mathsf{Since})$ formula. ◀

This completes our proof of Kamp's theorem except Proposition 4.2 which is proved in the next section.

5 Proof of Proposition 4.2

Let $\psi(z_0, z_1)$ be an $\overrightarrow{\exists}\forall$-formula

$$\exists x_n \ldots \exists x_1 \exists x_0 [z_0 = x_m \wedge z_1 = x_k \wedge \ (x_0 < x_1 < \cdots < x_{n-1} < x_n) \wedge \bigwedge_{j=0}^{n} \alpha_j(x_j)$$

$$\wedge \bigwedge_{j=1}^{n} (\forall y)_{>x_{j-1}}^{<x_j} \beta_j(y) \wedge \ (\forall y)^{<x_0} \beta_0(y) \wedge \ (\forall y)_{>x_n} \beta_{n+1}(y)]$$

We consider two cases. In the first case $k = m$, i.e., $z_0 = z_1$ and in the second $k \neq m$.

If $k = m$, then ψ is equivalent to $z_0 = z_1 \wedge \psi'(z_0)$, where ψ' is an $\overrightarrow{\exists}\forall$-formula. By Proposition 3.5, ψ' is equivalent to an $TL(\mathsf{Until}, \mathsf{Since})$ formula A'. Therefore, ψ is equivalent to an $\overrightarrow{\exists}\forall$-formula $\exists x_0[z_0 = x_0 \wedge z_1 = x_0 \wedge A'(x_0)]$.

If $k \neq m$, w.l.o.g. we assume that $m < k$. Hence, ψ is equivalent to a conjunction of

1. $\psi_0(z_0)$ defined as:
   $\exists x_0 \ldots \exists x_{m-1} \exists x_m [z_0 = x_m \wedge \ (x_0 < x_1 < \cdots < x_m) \wedge \ \bigwedge_{j=0}^{m} \alpha_j(x_j)$
   $$\wedge \bigwedge_{j=1}^{m} (\forall y)_{>x_{j-1}}^{<x_j} \beta_j(y) \wedge \ (\forall y)^{<x_0} \beta_0(y)$$

2. $\psi_1(z_1)$ defined as:
$\exists x_k \dots \exists x_{k+1} \exists x_n [z_1 = x_k \wedge \ (x_k < x_{k+1} < \cdots < x_n) \wedge \ \bigwedge_{j=k}^{n} \alpha_j(x_j)$

$$\wedge \bigwedge_{j=k+1}^{n} (\forall y)_{>x_{j-1}}^{<x_j} \beta_j(y) \wedge \ (\forall y)_{>x_n} \beta_{n+1}(y)]$$

3. $\varphi(z_0, z_1)$ defined as:
$\exists x_m \dots \exists x_k [(z_0 = x_m < x_{m+1} < \cdots < x_k = z_1) \wedge \ \bigwedge_{j=m}^{k} \alpha_j(x_j)$

$$\wedge \bigwedge_{j=m+1}^{k} (\forall y)_{>x_{j-1}}^{<x_j} \beta_j(y)$$

The first two formulas are $\overrightarrow{\exists}\forall$-formulas with one free variable. Therefore, (by Proposition 3.5) they are equivalent to a $TL(\mathsf{Until}, \mathsf{Since})$ formulas (in the signature $\mathcal{E}[\Sigma]$). Hence, their negations are equivalent (over the canonical expansions) to atomic (and hence to $\overrightarrow{\exists}\forall$) formulas.

Therefore, it is sufficient to show that the negation of the third formula is equivalent over Dedekind complete chains to a disjunction of $\overrightarrow{\exists}\forall$-formulas. This is stated in the following lemma:

▶ Lemma 5.1. The negation of any formula of the form

$$\exists x_0 \dots \exists x_n [(z_0 = x_0 < \cdots < x_n = z_1) \wedge \bigwedge_{j=0}^{n} \alpha_j(x_j) \wedge \bigwedge_{j=1}^{n} (\forall y)_{>x_{j-1}}^{<x_j} \beta_j(y)] \tag{1}$$

where α_i, β_i are quantifier free, is equivalent (over Dedekind complete chains) to a disjunction of $\overrightarrow{\exists}\forall$-formulas.

In the rest of this section we prove Lemma 5.1. Our proof is organized as follows. In Lemma 5.3 we prove an instance of Lemma 5.1 where α_0, α_n and all β_i are equivalent to True. Then we derive a more general instance (Corollary 5.4) where β_n is equivalent to true. Finally we prove the full version of Lemma 5.1.

First, we introduce some helpful notations.

▶ Notations 5.2. We use the abbreviated notation $[\alpha_0, \beta_1 \dots, \alpha_{n-1}, \beta_n \alpha_n](z_0, z_1)$ for the $\overrightarrow{\exists}\forall$-formula as in (1).

In this notation Lemma 5.1 can be rephrased as $\neg[\alpha_0, \beta_1 \dots, \alpha_{n-1}, \beta_n \alpha_n](z_0, z_1)$ is equivalent (over Dedekind complete chains) to a $\vee\overrightarrow{\exists}\forall$ formula.

We start with the instance of Lemma 5.1 where all β_i are True.

▶ Lemma 5.3. $\neg \exists x_1 \dots \exists x_n \, (z_0 < x_1 < \cdots < x_n < z_1) \wedge \bigwedge_{i=1}^{n} P_i(x_i)$ is equivalent over Dedekind complete chains to a $\vee\overrightarrow{\exists}\forall$ formula $O_n(P_1, \dots, P_n, z_0, z_1)$.

Proof. We proceed by induction.

Basis: $\neg(\exists x_1)_{>z_0}^{<z_1} P_1(x_1)$ is equivalent to $(\forall y)_{>z_0}^{<z_1} \neg P_1(y)$.

Inductive step: $n \mapsto n+1$. We assume that a $\vee\overrightarrow{\exists}\forall$ formula O_n was defined and construct a $\vee\overrightarrow{\exists}\forall$ formula O_{n+1}.

Observe that if the interval (z_0, z_1) is non-empty, then one of the following cases holds:
Case 1 There is no occurrence of P_1 in (z_0, z_1), i.e. $(\forall y)_{>z_0}^{<z_1} \neg P_1(y)$.

In this case $O_{n+1}(P_1, \dots, P_{n+1}, z_0, z_1)$ should be equivalent to True.

Case 2 If case 1 does not hold then let $r_0 = \inf\{z \in (z_0, z_1) \mid P_1(z)\}$ (such r_0 exists by Dedekind completeness. Note that $r_0 = z_0$ iff $\mathsf{K}^+(P_1)(z_0)$. If $r_0 > z_0$ then $r_0 \in (z_0, z_1)$ and r_0 is definable by the following $\vee\overrightarrow{\exists}\forall$ formula:

$$\begin{aligned} INF(z_0, r_0, z_1, P_1) :=& z_0 < r_0 < z_1 \wedge (\forall y)_{>z_0}^{<r_0} \neg P_1(y) \wedge \\ & \wedge (P_1(r_0) \vee \mathsf{K}^+(P_1)(r_0)) \end{aligned} \tag{2}$$

Subcase $r_0 = z_0$
In this subcase $O_n(P_2, \dots, P_n, z_0, z_1)$ and $O_{n+1}(P_1, \dots, P_{n+1}, z_0, z_1)$ should be equivalent.

Subcase $r_0 \in (z_0, z_1)$
In this subcase $O_n(P_2, \dots, P_n, r_0, z_1)$ and $O_{n+1}(P_1, \dots, P_{n+1}, z_0, z_1)$ should be equivalent.

Hence, $O_{n+1}(P_1, \dots, P_{n+1}, z_0, z_1)$ can be defined as the disjunction of "(z_0, z_1) is empty" and the following formulas:

1. $(\forall y)_{>z_0}^{<z_1} \neg P_1((y)$
2. $\mathsf{K}^+(P_1)(z_0) \wedge O_n(P_2, \dots, P_n, z_0, z_1)$
3. $(\exists r_0)_{>z_0}^{<z_1} \big(INF(z_0, r_0, z_1, P_1) \wedge O_n(P_2, \dots, P_n, r_0, z_1)\big)$

The first formula is a $\vee\overrightarrow{\exists}\forall$ formula. By the inductive assumptions O_n is a $\vee\overrightarrow{\exists}\forall$ formula. $\mathsf{K}^+(P_1)(z_0)$ is an atomic (and hence a $\vee\overrightarrow{\exists}\forall$) formula in the canonical expansion, and $INF(z_0, r_0, z_1, P_1)$ is a $\vee\overrightarrow{\exists}\forall$ formula. Since $\vee\overrightarrow{\exists}\forall$ formulas are closed under conjunction, disjunction and the existential quantification, we conclude that O_{n+1} is a $\vee\overrightarrow{\exists}\forall$ formula. ◀

As a consequence we obtain

▶ Corollary 5.4. **1.** $\neg(\exists z)_{>z_0}^{<z_1}[\alpha_0, \beta_1, \alpha_1, \beta_2, \dots, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z)$ is equivalent over Dedekind complete chains to a $\vee\overrightarrow{\exists}\forall$ formula.

2. $\neg(\exists z)_{>z_0}^{<z_1}[\alpha_0, \beta_1, \alpha_1, \beta_2, \dots, \alpha_{n-1}, \beta_n, \alpha_n](z, z_1)$ is equivalent over Dedekind complete chains to a $\vee\overrightarrow{\exists}\forall$ formula.

Proof. (1) Define

$$\begin{aligned} F_n &:= \alpha_n \\ F_{i-1} &:= \alpha_{i-1} \wedge \beta_i \mathsf{Until} F_i \qquad \text{for } i = 1, \dots, n \end{aligned}$$

Observe that there is $z \in (z_0, z_1)$ such that $[\alpha_0, \beta_1, \alpha_1, \beta_2, \dots, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z)$ iff $F_0(z_0)$ and there is an increasing sequence $x_1 < \dots < x_n$ in an open interval (z_0, z_1) such that $F_i(x_i)$ for $i = 1, \dots, n$. Indeed, the direction $\Rightarrow$ is trivial. The direction $\Leftarrow$ is easily proved by induction.

The *basis* is trivial.

Inductive step: $n \mapsto n+1$. Assume $F_0(z_0)$ holds and that (z_0, z_1) contains an increasing sequence $x_1 < \dots < x_{n+1}$ such that $F_i(x_i)$ for $i = 1, \dots, n+1$. By the inductive assumption there is $y_1 \in (z_0, x_{n+1})$ such that

$$[\alpha_0, \beta_1, \alpha_1, \beta_2, \dots, \beta_{n-1}\alpha_{n-1}, \beta_n, (\alpha_n \wedge \beta_{n+1} \mathsf{Until} \alpha_{n+1})](z_0, y_1).$$

In particular, y_1 satisfies $(\alpha_n \wedge \beta_{n+1} \mathsf{Until} \alpha_{n+1})$. Hence, there is $y_2 > y_1$ such that y_2 satisfies α_{n+1} and β_{n+1} holds along (y_1, y_2).

If $y_2 \le x_{n+1}$ then the required $z \in (z_0, z_1)$ equals to y_2, and we are done. Otherwise, $x_{n+1} < y_2$. Therefore, $x_{n+1} \in (y_1, y_2)$ and β_{n+1} holds along (y_1, x_{n+1}). Hence, the required z equals to x_{n+1}.

From the above observation and Lemma 5.3, it follows that $\neg F_0(z_0) \vee O_n(F_1, \dots, F_n, z_0, z_1)$ is a $\vee\overrightarrow{\exists}\forall$ formula that is equivalent to $\neg(\exists z)^{<z_1}_{>z_0}[\alpha_0, \beta_1, \alpha_1, \beta_2, \dots, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z)$.

(2) is the mirror image of (1) and is proved similarly. ◀

Now we are ready to prove Lemma 5.1, i.e.,

$$\neg[\alpha_0, \beta_1 \dots, \beta_{n-1}, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z_1) \text{ is equivalent over Dedekind complete chains to a } \vee\overrightarrow{\exists}\forall \text{ formula.}$$

If the interval (z_0, z_1) is empty then the assertion is immediate. We assume that (z_0, z_1) is non-empty. Hence, at least one of the following cases holds:

Case 1 $\neg\alpha_0(z_0)$ or $\neg\alpha_n(z_1)$ or $\neg(\beta_1 \mathsf{Until} \alpha_1)(z_0)$ or $\neg(\beta_n \mathsf{Since} \alpha_{n-1})(z_1)$.

Case 2 $\alpha_0(z_0)$, and β_1 holds along (z_0, z_1).

Case 3

1. $\alpha_0(z_0) \wedge (\beta_1 \mathsf{Until} \alpha_1)(z_0)$, and
2. there is $x \in (z_0, z_1)$ such that $\neg\beta_1(x)$.

For each of these cases we construct a $\vee\overrightarrow{\exists}\forall$ formula Cond_i which describes it (i.e., Case i holds iff Cond_i holds) and show that if Cond_i holds, then $\neg[\alpha_0, \beta_1 \dots, \beta_{n-1}, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z_1)$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula Form_i. Hence, $\neg[\alpha_0, \beta_1 \dots, \beta_{n-1}, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z_1)$ is equivalent to $\vee_i[\mathit{Cond}_i \wedge \mathit{Form}_i]$ which is a $\vee\overrightarrow{\exists}\forall$ formula.

Case 1 This case is already explicitly described by the $\vee\overrightarrow{\exists}\forall$ formula (in the canonical expansion). In this case $\neg[\alpha_0, \beta_1 \dots, \beta_{n-1}, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z_1)$ is equivalent to True.

Case 2 This case is described by a $\vee\overrightarrow{\exists}\forall$ formula $\alpha_0(z_0) \wedge (\forall z)^{<z_1}_{>z_0}\beta_1$. In this case $\neg[\alpha_0, \beta_1 \dots, \beta_{n-1}, \alpha_{n-1}, \beta_n, \alpha_n](z_0, z_1)$ is equivalent to "there is no $z \in (z_0, z_1)$ such that $[\alpha_1, \beta_2 \dots, \beta_n, \alpha_n](z, z_1)$." By Corollary 5.4 this is expressible by a $\vee\overrightarrow{\exists}\forall$ formula.

Case 3 The first condition of Case 3 is already explicitly described by a $\vee\overrightarrow{\exists}\forall$ formula. When the first condition holds, then the second condition is equivalent to "there is (a unique) $r_0 \in (z_0, z_1)$ such that $r_0 = \inf\{z \in (z_0, z_1) \mid \neg\beta_1(z)\}$" (If $\beta_1 \mathsf{Until} \alpha_1$ holds at z_0 and there is $x \in (z_0, z_1)$ such that $\neg\beta_1(x)$, then such r_0 exists because we deal with Dedekind complete chains.) This r_0 is definable by the following $\vee\overrightarrow{\exists}\forall$ formula, i.e., it is a unique z which satisfies it[4]:

$$\mathit{INF}^{\neg\beta_1}(z_0, z, z_1) := z_0 < z < z_1 \wedge (\forall y)^{<z}_{>z_0}\beta_1(y) \wedge (\neg\beta_1(z) \vee \mathbf{K}^+(\neg\beta_1)(z)) \tag{3}$$

Hence, Case 3 is described by $\alpha_0(z_0) \wedge (\beta_1 \mathsf{Until} \alpha_1)(z_0) \wedge (\exists z)^{<z_1}_{>z_0}\mathit{INF}^{\neg\beta_1}(z_0, z, z_1)$. Since the set of $\vee\overrightarrow{\exists}\forall$ formulas is closed under conjunction, disjunction and $\exists$, this case is described by a $\vee\overrightarrow{\exists}\forall$ formula.

It is sufficient to show that $(\exists z)^{<z_1}_{>z_0}\mathit{INF}^{\neg\beta_1}(z) \wedge \neg[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1)$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula.

We prove this by induction on n.

The *basis* is trivial.

Inductive step $n \mapsto n+1$.

[4] We will use only existence and will not use uniqueness.

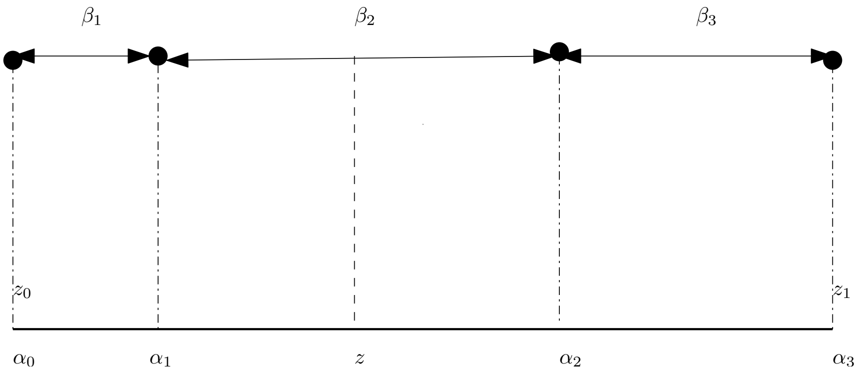


Figure 1 $B_2(z_0, z, z_1) := [\alpha_0, \beta_1, \alpha_1, \beta_2, \beta_2](z_0, z) \wedge [\beta_2, \beta_2, \alpha_2, \beta_3, \alpha_3](z, z_1)$.

Define:

$$\begin{aligned}
A_i^-(z_0, z) :=& [\alpha_0, \beta_1, \dots, \beta_i, \alpha_i](z_0, z) && i = 1, \dots, n\\
A_i^+(z, z_1) :=& [\alpha_i, \beta_{i+1}, \dots \beta_{n+1}\alpha_{n+1}](z, z_1) && i = 1, \dots, n\\
A_i(z_0, z, z_1) :=& A_i^-(z_0, z) \wedge A_i^+(z, z_1) && i = 1, \dots, n\\
B_i^-(z_0, z) :=& [\alpha_0\beta_1, \dots, \beta_{i-1}, \alpha_{i-1}, \beta_i, \beta_i](z_0, z) && i = 1, \dots, n+1\\
B_i^+(z, z_1) :=& [\beta_i, \beta_i, \alpha_i\beta_{i+1}\alpha_{i+1}, \dots, \beta_{n+1}, \alpha_{n+1}](z, z_1) && i = 1, \dots, n+1\\
B_i(z_0, z, z_1) :=& B_i^-(z_0, z) \wedge B_i^+(z, z_1) && i = 1, \dots, n+1
\end{aligned}$$

If the interval (z_0, z_1) is non-empty, these definitions imply

$$[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1) \Leftrightarrow (\forall z)^{<z_1}_{>z_0}\big(\bigvee_{i=1}^{n} A_i \vee \bigvee_{i=1}^{n+1} B_i\big)$$

$$[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1) \Leftrightarrow (\exists z)^{<z_1}_{>z_0}\big(\bigvee_{i=1}^{n} A_i \vee \bigvee_{i=1}^{n+1} B_i\big)$$

Hence, for every φ

$$(\exists z)^{<z_1}_{>z_0}\varphi(z) \wedge \neg[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1)$$

is equivalent to

$$(\exists z)^{<z_1}_{>z_0}\big(\varphi(z) \wedge \bigwedge_{i=1}^{n} \neg A_i \wedge \bigwedge_{i=1}^{n+1} \neg B_i\big)$$

In particular,

$$(\exists z)^{<z_1}_{>z_0} \mathit{INF}^{\neg\beta_1}(z) \wedge \neg[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1)$$

is equivalent to

$$(\exists z)^{<z_1}_{>z_0}\big(\mathit{INF}^{\neg\beta_1}(z) \wedge \bigwedge_{i=1}^{n} \neg A_i \wedge \bigwedge_{i=1}^{n+1} \neg B_i\big),$$

where $\mathit{INF}^{\neg\beta_1}(z)$ was defined in equation (3).

By the inductive assumption

(a) $\neg A_i$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula for $i = 1, \dots, n$.

(b) $\neg B_i$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula for $i = 2, \dots, n$.

Recall $B_1 := B_1^- \wedge B_1^+$ and $B_{n+1} := B_{n+1}^- \wedge B_{n+1}^+$.

(c) $\neg B_1^-$ and $\neg B_{n+1}^+$ are equivalent to $\vee\overrightarrow{\exists}\forall$ formulas, by the induction basis.

(d) $\mathit{INF}^{\neg\beta_1}(z) \wedge \neg B_1^+(z, z_1)$ is equivalent to $\mathit{INF}^{\neg\beta_1}(z)$, because if $\mathit{INF}^{\neg\beta_1}(z)$, then for no $x > z$, β_1 holds along $[z, x)$.

(e) $\mathit{INF}^{\neg\beta_1}(z) \wedge \neg B_{n+1}^-(z_0, z)$ is equivalent to $\mathit{INF}^{\neg\beta_1}(z) \wedge$ ("β_1 holds on (z_0, z)" $\wedge$ $\neg B_{n+1}^-(z_0, z)$). Since, by case 2, "β_1 holds on (z_0, z)" $\wedge\ \neg B_{n+1}^-(z_0, z)$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula, and $\mathit{INF}^{\neg\beta_1}(z)$ is a $\vee\overrightarrow{\exists}\forall$ formula, we conclude that $\mathit{INF}^{\neg\beta_1}(z) \wedge \neg B_{n+1}^-(z_0, z)$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula.

Since the set of $\vee\overrightarrow{\exists}\forall$ formulas is closed under conjunction, disjunction and $\exists$, by (a)-(e) we obtain that $(\exists z)^{<z_1}_{>z_0}\big(\mathit{INF}^{\neg\beta_1}(z) \wedge \bigwedge_{i=1}^{n} \neg A_i \wedge \bigwedge_{i=1}^{n+1} \neg B_i\big)$ is equivalent to a $\vee\overrightarrow{\exists}\forall$ formula. Therefore, $(\exists z)^{<z_1}_{>z_0}\mathit{INF}^{\neg\beta_1}(z) \wedge \neg[\alpha_0, \beta_1, \alpha_1, \dots, \beta_{n+1}, \alpha_{n+1}](z_0, z_1)$ is also a $\vee\overrightarrow{\exists}\forall$ formula.

This completes our proof of Lemma 5.1 and of Proposition 4.2.

6 Related Works

Kamp's theorem was proved in

1. Kamp's thesis [7] (proof > 100pages).
2. Outlined by Gabbay, Pnueli, Stavi and Shelah [3] (Sect. 2) for $\mathbb{N}$ and stated that it can be extended to Dedekind complete orders using game arguments.
3. Was proved by Gabbay [1] by separation arguments for $\mathbb{N}$, and extended to Dedekind complete order in [2].
4. Was proved by Hodkinson [4] by game arguments and simplified in [5] (unpublished).

A temporal logic has the *separation* property if its formulas can be equivalently rewritten as a boolean combination of formulas, each of which depends only on the past, present or future. The separation property was introduced by Gabbay [1], and surprisingly, a temporal logic which can express $\Box$ and $\overleftarrow{\Box}$ has the separation property (over a class $\mathcal{C}$ of structures) iff it is expressively complete for *FOMLO* over $\mathcal{C}$.

The separation proof for $\mathit{TL}(\mathsf{Until}, \mathsf{Since})$ over $\mathbb{N}$ is manageable; however, over the real (and over Dedekind complete) chains it contains many rules and transformations and is not easy to follow. Hodkinson and Reynolds [6] write:

> The proofs of theorems 18 and 19 [Kamp's theorem over naturals and over reals, respectively] are direct, showing that each formula can be separated. They are tough and tougher, respectively. Nonetheless, they are effective, and so, whilst not quite providing an algorithm to determine if a set of connectives is expressively complete, they do suggest a potential way of telling in practice whether a given set of connectives is expressively complete – in Gabbay's words, *try to separate and see where you get stuck!*

The game arguments are easier to grasp, but they use complicated inductive assertions. The proof in [5] proceeds roughly as follows. Let $\mathcal{L}_r$ be the set of $\mathit{TL}(\mathsf{Until}, \mathsf{Since})$ formulas of nesting depth at most r. A formula of the form: $\exists\bar{x}\forall y\chi(\bar{x}, y, \bar{z})$ where $\bar{x}$ is an n-tuple of variables and χ is a quantifier free formula over $\{<, =\}$ and $\mathcal{L}_r$-definable monadic predicates is called $\langle n, r\rangle$-decomposition formula. The main inductive assertion is proved by "unusual back-and-forth games" and can be rephrased in logical terms as there is a function f :

$\mathbb{N} \to \mathbb{N}$ such that for every $n, r \in \mathbb{N}$, the negation of positive Boolean combinations $\langle n, r\rangle$-decomposition formula is equivalent to a positive Boolean combination of $\langle f(n), (n+r)\rangle$-decomposition formulas.

Our proof is inspired by [3] and [5]; however, it avoids games, and it separates general logical equivalences and temporal arguments.

Many temporal formalisms studied in computer science concern only future formulas - whose truth value at any moment is determined by what happens from that moment on. A formula (temporal, or monadic with a single free first-order variable) F is (*semantically*) *future* if for every chain $\mathcal{M}$ and moment $t_0 \in \mathcal{M}$:

$$\mathcal{M}, t_0 \models F \text{ iff } \mathcal{M}|_{\geq t_0}, t_0 \models F,$$

where $\mathcal{M}|_{\geq t_0}$ is the subchain of $\mathcal{M}$ over the interval $[t_0, \infty)$. For example, $P\mathsf{Until}Q$ and $\mathsf{K}^+(P)$ are future formulas, while $P\mathsf{Since}Q$ and $\mathsf{K}^-(P)$ are not future ones.

It was shown in [3] that over the *discrete* chains Kamp's theorem holds also for *future formulas* of *FOMLO*:

▶ Theorem 6.1. Every future *FOMLO* formula is equivalent over discrete orders (Natural, Integer, finite) to a $TL(\mathsf{Until})$ formula.

Theorems 6.1 can be easily obtained from our proof of Kamp's theorem.

The temporal logic with the modalities Until and Since is not expressively complete for *FOMLO* over the rationals. Stavi introduced two additional modalities Until^s and Since^s and proved that $TL(\mathsf{Until}, \mathsf{Since}, \mathsf{Until}^s, \mathsf{Since}^s)$ is expressively complete for *FOMLO* over all linear orders [2]. In the full version of this paper we prove Stavi's theorem. The proof is similar to our proof of Kamp's theorem; however, it treats some additional cases related to gaps in orders, and replaces $\overrightarrow{\exists}\forall$-formulas by slightly more general formulas.

Acknowledgments I am very grateful to Yoram Hirshfeld for numerous insightful discussions, and to the anonymous referees for their helpful suggestions.

References

1 D. Gabbay. Expressive functional completeness in tense logic (preliminary report). In U. Monnich, editor, Aspects of Philosophical Logic, pages 91-117. Reidel, Dordrecht, 1981.

2 D. Gabbay, I. Hodkinson, and M. Reynolds. Temporal logic: Mathematical Foundations and Computational Aspects. *Oxford University Press, 1994.*

3 D. Gabbay, A. Pnueli, S. Shelah and J. Stavi. On the Temporal Analysis of Fairness. *In POPL 1980, pp. 163-173, 1980. Information and Computation, 187(2003), 196-208.*

4 I. Hodkinson. Expressive completeness of Until and Since over dedekind complete linear time. *Modal logic and process algebra, ed. A. Ponse, M. de Rijke, Y. Venema, CSLI Lecture Notes 53, 1995.*

5 I. Hodkinson. Notes on games in temporal logic. *Lecture notes for LUATCS meting, Johannesburg, Dec 1999.* http://www.doc.ic.ac.uk/~imh/index.html

6 I. Hodkinson and M. Reynolds. Temporal Logic Chapter 11 (pp. 655-720) in Handbook of Modal Logic, Patrick Blackburn, Johan van Benthem, and Frank Wolter, eds., Elsevier Science, 2006.

7 H. W. Kamp. Tense logic and the theory of linear order. *Phd thesis, University of California, Los Angeles, 1968.*

8 A. Pnueli (1977). The temporal logic of programs. In *Proc. IEEE 18th Annu. Symp. on Found. Comput. Sci.*, pages 46–57, New York, 1977.

Commutative Data Automata*

Zhilin Wu

State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences,
P.O. Box 8718, # 4 Zhongguancun South 4th Street, Beijing, 100190, China
wuzl@ios.ac.cn

Abstract

Formalisms over infinite alphabets have recently received much focus in the community of theoretical computer science. Data automaton is a formal model for words over an infinite alphabet, that is, the product of a finite set of labels and an infinite set of data values, proposed by Bojanczyk, Muscholl, Schwentick et. al. in 2006. A data automaton consists of two parts, a nondeterministic letter-to-letter transducer, and a class condition specified by a finite automaton, which acts as a condition on each subword of the outputs of the transducer in corresponding to a maximal set of positions with the same data value. It is open whether the nonemptiness of data automata can be decided with elementary complexity, since this problem is equivalent to the reachability of Petri nets. Very recently, a restriction of data automata with elementary complexity, called weak data automata, was proposed by Kara, Schwentick and Tan and its nonemptiness problem was shown to be in 2-NEXPTIME. In weak data automata, the class conditions are specified by some simple constraints on the number of occurrences of labels occurring in every class. The aim of this paper is to demonstrate that the commutativity of class conditions is the genuine reason accounting for the elementary complexity of weak data automata. For this purpose, we define and investigate commutative data automata, which are data automata with class conditions restricted to commutative regular languages. We show that while the expressive power of commutative data automata is strictly stronger than that of weak data automata, the nonemptiness problem of this model can still be decided with elementary complexity, more precisely, in 3-NEXPTIME. In addition, we extend the results to data ω-words and prove that the nonemptiness of commutative Büchi data automata can be decided in 4-NEXPTIME. We also provide logical characterizations for commutative (Büchi) data automata, similar to those for weak (Büchi) data automata.

1998 ACM Subject Classification F.1.1, F4

Keywords and phrases Data Automata, Commutative regular languages, Presburger arithmetic, Existential Monadic Second-order logic, Büchi automata

Digital Object Identifier 10.4230/LIPIcs.CSL.2012.528

1 Introduction

With the momentums from the XML document processing and the verification of computer programs, formalisms over infinite alphabets have been intensively investigated in recent years. In the database community, XML documents are usually represented by trees, where the nodes can have tags together with several attributes e.g. identifiers. While the tags are from a finite set, the attributes may take values from some infinite domains. On the other hand, in the verification community, take concurrent systems as an example, if there is an unbounded number of processes in the system, then the behavior of the global system consists of the sequences of observed events attached with the process identifiers.

* Supported by the National Natural Science Foundation of China under Grant No. 61100062.

Computer Science Logic 2012 (CSL'12).
Editors: Patrick Cégielski, Arnaud Durand; pp. 528–542
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

With these motivations, researchers in the two communities have investigated various formalisms over infinite alphabets, to name a few, register automata ([11]), pebble automata ([14]), data automata ([1]), XPath with data values ([9, 8]), LTL with freeze quantifiers ([6]), as well as two-variable logic interpreted on words or trees over infinite alphabets ([1, 3]). A survey on this topic can be found in [16].

By infinite alphabet, we mean $\Sigma \times \mathbb{D}$, with Σ a finite set of tags (labels) and $\mathbb{D}$ an infinite data domain. Words and trees over the alphabet $\Sigma \times \mathbb{D}$ are called data words and data trees.

The model of data automata was introduced by Bojanczyk, Muscholl, Schwentick, et. al. in [1] to prove the decidability of two-variable logic over data words. A data automaton $\mathscr{D}$ over data words consists of two parts, a nondeterministic letter-to-letter transducer $\mathscr{A}$, and a class condition specified by a finite automaton $\mathscr{B}$ over the output alphabet of $\mathscr{A}$, which acts as a condition on the subsequence of the outputs of $\mathscr{A}$ in every class, namely, every maximal set of positions with the same data value. By a reduction to the reachability problem of Petri nets (also called multicounter machines), the nonemptiness of data automata was shown to be decidable. On the other hand, data automata are also powerful enough to simulate Petri nets easily. Since it is a well-known open problem whether the complexity of the reachability problem for Petri nets is elementary, it is also not known whether the complexity of the nonemptiness of data automata is elementary.

Aiming at lowering the complexity of data automata, a restriction of data automata, called weak data automata, was introduced very recently by Kara, Schwentick, and Tan ([12]). In weak data automata, the class conditions are replaced by some simple constraints on the number of occurrences of labels occurring in every class. The nonemptiness of weak data automata can be decided with elementary complexity, more precisely, in 2-NEXPTIME.

By comparing data automata with weak data automata, we notice that to simulate Petri nets in data automata, the ability to express the property $L_{a<b}$, "for every occurrence of a, there is an occurrence of b on the right with the same data value", is crucial; on the other hand, as shown in [12], $L_{a<b}$ is not expressible in weak data automata. It is a simple observation that $L_{a<b}$ is a non-commutative language while the class conditions of weak data automata are commutative. This suggests that the commutativity of class conditions might be the *genuine* reason accounting for the elementary complexity of weak data automata. With this observation, we are motivated to define and investigate commutative data automata, which are data automata with class conditions restricted to commutative regular languages. We would like to see that the nonemptiness of commutative data automata can still be decided with elementary complexity, even though they have stronger class conditions than weak data automata. This is indeed the case, as we will show in this paper.

More specifically, the contributions of this paper consist of the following three aspects.

1. At first, we investigate the expressibility of commutative data automata. We show that the expressive power of commutative data automata lies strictly between data automata and weak data automata. In addition, commutative data automata are closed under intersection and union, but not under complementation. We also present a logical characterization of commutative data automata, similar to that for weak data automata.
2. The nonemptiness of commutative data automata can be decided in 3-NEXPTIME, which is the main result of this paper.
3. At last, we extend the results to data ω-words. We define commutative Büchi data automata and prove that the nonemptiness of commutative Büchi data automata can be decided in 4-NEXPTIME.

The main ideas of most of the proofs in this paper come from those for weak data automata ([12, 5]). Nevertheless, some proof steps become much more involved as a result of the stronger class conditions in commutative data automata.

Related work.

> Several variants of data automata have been investigated. Bojanczyk and Lasota proposed an (undecidable) extension of data automata, called class automata, to capture the full XPath with data values over data trees; in addition, they established the correspondences of various class conditions of class automata over data words with the various models of counter machines ([2]). We continued this line of research by introducing another decidable extension of data automata over data words and establishing the correspondence with priority multicounter machines ([19]). There is another model, called class counting automata, relevant to this paper. Class counting automata over data words was proposed by Manuel and Ramanujan in [13]. In class counting automata, each data value is assigned a counter; and in each transition step, if the value of the counter corresponding to the current data value satisfies some constraint, then the value of the counter is updated according to a prescribed instruction; a run is accepting if a final state is reached in the end. The nonemptiness of class counting automata was shown to be EXPSPACE-complete. Nevertheless, the expressive power of class counting automata is relatively weak, for instance, the property "Each data value occurs exactly twice" cannot be expressed by class counting automata, while this property can be easily expressed by commutative data automata. Commutative regular languages have been investigated by many researchers. Pin presented a counting characterization of the expressibility of commutative regular languages ([15]). Gomez and Alvarez investigated how commutative regular languages can be learned from positive and negative examples ([10]). Chrobak and To proposed polynomial time algorithms to obtain regular expressions from nondeterministic finite automata over unary alphabets ([4, 18]).

The rest of this paper is organized as follows. Definitions are given in the next section. Then in Section 3, the expressibility of commutative data automata is investigated. Section 4 includes the main result of this paper, a 3-NEXPTIME algorithm for the nonemptiness of commutative data automata. Finally the results are extended to data ω-words in Section 5. The missing proofs can be found in the full version of this paper (http://lcs.ios.ac.cn/∼wuzl/pub/cda-wu-12.pdf).

2 Preliminaries

Let Σ be a finite alphabet. A finite word over Σ is an element of Σ^* and an ω-word over Σ is an element of Σ^ω.

2.1 Presburger formulas and Commutative regular languages

Existential Presburger formulas (EP formulas) over a variable set X are formulas of the form $\exists \bar{x} \varphi$, where φ is a quantifier-free Presburger formula, i.e. a Boolean combination of atomic formulas of the form $t \geq c$, or $t \leq c$, or $t = c$, or $t \equiv r \bmod p$, where $c, r, p \in \mathbb{N}$, $p \geq 2, 0 \leq r < p$ and t is a term defined by $t := c \mid cx \mid t_1 + t_2 \mid t_1 - t_2$, where $c \in \mathbb{N}, x \in X$.

Suppose $\Sigma = \{\sigma_1, \dots, \sigma_k\}$ and $v \in \Sigma^*$. The *Parikh image* of v, denoted by $\mathsf{Parikh}(v)$, is a k-tuple $(\#_{\sigma_1}(v), \dots, \#_{\sigma_k}(v))$, where for each $i : 1 \leq i \leq k$, $\#_{\sigma_i}(v)$ is the number of occurrences of σ_i in v. Let $V_\Sigma = \{x_{\sigma_1}, \dots, x_{\sigma_k}\}$ and φ be an EP formula with free variables from V_Σ. The word v is said to satisfy φ, denoted by $v \models \varphi$, iff $\varphi[\mathsf{Parikh}(v)]$ holds. The *language defined by* φ, denoted by $\mathcal{L}(\varphi)$, is the set of words $v \in \Sigma^*$ such that $v \models \varphi$.

A *Presburger automaton* over the alphabet Σ is a binary tuple $(\mathscr{A}, \varphi)$, where $\mathscr{A}$ is a finite automaton over the alphabet Σ and φ is an EP formula with free variables from V_Σ. A word $v \in \Sigma^*$ is accepted by a Presburger automaton $(\mathscr{A}, \varphi)$ iff v is accepted by $\mathscr{A}$ and at the same time $v \models \varphi$. The language accepted by a Presburger automaton $(\mathscr{A}, \varphi)$, denoted by $\mathcal{L}((\mathscr{A}, \varphi))$, is the set of words accepted by $(\mathscr{A}, \varphi)$.

▶ **Theorem 1** ([17]). *The nonemptiness of Presburger automata can be decided in NP.*

Let L be a language over the alphabet Σ. Then L is *commutative* iff for any $\sigma_1, \sigma_2 \in \Sigma$ and $u, v \in \Sigma^*$, $u\sigma_1\sigma_2 v \in L$ iff $u\sigma_2\sigma_1 v \in L$. Commutative regular languages have a characterization in quantifier-free simple Presburger formulas defined in the following.

Quantifier-free simple Presburger formulas (QFSP formulas) over a variable set X are Boolean combinations of atomic formulas of the form $x_1 + \cdots + x_n \leq c$, or $x_1 + \cdots + x_n \geq c$, or $x_1 + \cdots + x_n = c$, or $x_1 + \cdots + x_n \equiv r \bmod p$, where $x_1, \ldots, x_n \in X$, $c, r, p \in \mathbb{N}$, $p \geq 2$, and $0 \leq r < p$.

Let $V_\Sigma = \{x_{\sigma_1}, \ldots, x_{\sigma_k}\}$ and φ be a QFSP formula over the variable set V_Σ. Similar to EP formulas, we can define $\mathcal{L}(\varphi)$, the language defined by φ.

For a set of variables $\{x_1, \ldots, x_k\}$, we use the notation $\varphi(x_1, \ldots, x_k)$ to denote an EP or QFSP formula φ with the free variables from $\{x_1, \ldots, x_k\}$.

▶ Proposition 2 ([15]). *Let L be a language over the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$. Then L is a commutative regular language iff L is defined by a QFSP formula $\varphi(x_{\sigma_1}, \ldots, x_{\sigma_k})$.*

The *size* of an EP or QFSP formula φ, denoted by $|\varphi|$, is defined as the length of a binary encoding of φ (where the constants c, r and p are encoded in binary).

▶ Proposition 3. *Let $\varphi(x_1, \ldots, x_k)$ be a QFSP formula. Then there exists an exponential-time algorithm to transform φ into a QFSP formula $\bigvee\limits_{i:1 \leq i \leq m} \varphi_i$ of size $2^{O(k|\varphi|)}$ such that there is $p_0 : 2 \leq p_0 \leq 2^{|\varphi|}$ satisfying that*

- *each φ_i is of the form $\bigwedge\limits_{1 \leq j \leq k} \varphi_{i,j}$;*
- *for each $j : 1 \leq j \leq k$, $\varphi_{i,j}$ is equal to $x_j = c_{i,j}$ or $x_j \geq p_0 \wedge x_j \equiv r_{i,j} \bmod p_0$ for $c_{i,j}, r_{ij} : 0 \leq c_{i,j}, r_{i,j} < p_0$;*
- *in addition, those φ_i's are mutually exclusive.*

For a QFSP formula $\varphi(x_1, \ldots, x_k)$, the number p_0 and the QFSP formula $\bigvee\limits_{i:1 \leq i \leq m} \varphi_i$ in Proposition 3 are called respectively the *normalization* number and the *normal form* of φ.

▶ Remark. A weaker form of Proposition 3 was proved by Ehrenfeucht and Rozenberg in [7]. But they did not give the complexity bound.

2.2 Data words, two-variable logic and data automata

Let Σ be a finite alphabet and $\mathbb{D}$ be an infinite set of data values. A *data word* over Σ is an element of $(\Sigma \times \mathbb{D})^*$ and a *data ω-word* is an element of $(\Sigma \times \mathbb{D})^\omega$. Let $\sigma \in \Sigma$, a position in a data word or a data ω-word is called a *σ-position* if the position is labelled by σ.

Given a data (finite or ω) word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\ldots$, the *projection* of w to the finite alphabet Σ, denoted by $\mathsf{Proj}(w)$, is the (finite or ω) word $\sigma_1\sigma_2\ldots$. Let X be a set of positions in a word w, we use $w|_X$ to denote the restriction of w to the positions in X. Similarly, $w|_X$ can be defined for ω-words, data words and data ω-words.

Let $FO(+1, \sim, \Sigma)$ denote the first-order logic with the following atomic formulas, $\sigma(x)$ (where $\sigma \in \Sigma$), $x = y$, $x + 1 = y$, and $x \sim y$. Two positions x, y satisfy $x + 1 = y$ if y is the successor of the position x, and two positions satisfy $x \sim y$ if they have the same data value. Let $FO^2(+1, \sim, \Sigma)$ denote the two-variable fragment of $FO(+1, \sim, \Sigma)$. In addition, let $EMSO^2(+1, \sim, \Sigma)$ denote the extension of $FO^2(+1, \sim, \Sigma)$ by existential monadic second-order quantifiers in front of the $FO^2(+1, \sim, \Sigma)$ formulas. The data language defined by an $EMSO^2(+1, \sim, \Sigma)$ sentence φ, denoted by $\mathcal{L}(\varphi)$, is the set of data words satisfying φ.

A *class* of a data (finite or ω) word w is a maximal nonempty set of positions in w with the same data value. Given a class X of a data word w, the *class string* of w corresponding to X is $\mathsf{Proj}(w|_X)$, the projection of $w|_X$.

Let $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots\binom{\sigma_n}{d_n}$ be a data word and φ be a QFSP formula over the variable set V_Σ. Then w is said to *satisfy the class condition* φ, denoted by $w \models_c \varphi$, if for each class X of w, $\mathsf{Proj}(w|_X) \models \varphi$.

Given a data (finite or ω) word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots$, the *profile word* of w, denoted by $\mathsf{Profile}(w)$, is a word $(\sigma_1, s_1)(\sigma_2, s_2)\dots$ over the alphabet $\Sigma \times \{\bot, \top\}$ such that for every $i \geq 1$, we have $s_i = \top$ (resp. $s_i = \bot$) iff $d_i = d_{i+1}$ (resp. $d_i \neq d_{i+1}$), moreover, if w is finite and of length n, then $s_n = \bot$. A data (finite or ω) word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots$ is called *locally different* if for every $i \geq 1$, it holds $d_i \neq d_{i+1}$.

▶ **Definition 4.** A *data automaton* (DA) $\mathscr{D}$ is a tuple $(\mathscr{A}, \mathscr{B})$, where $\mathscr{A} = (Q_1, \Sigma \times \{\bot, \top\}, \Gamma, \delta_1, q_{0,1}, F_1)$ is a nondeterministic letter-to-letter transducer with the input alphabet $\Sigma \times \{\bot, \top\}$ and the output alphabet Γ, and $\mathscr{B} = (Q_2, \Gamma, \delta_2, q_{0,2}, F_2)$ is a finite automaton over the alphabet Γ.

A data word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots\binom{\sigma_n}{d_n}$ is *accepted* by a data automaton $\mathscr{D} = (\mathscr{A}, \mathscr{B})$ iff there is an accepting run of $\mathscr{A}$ over $\mathsf{Profile}(w)$ which produces a word $\gamma_1 \dots \gamma_n$ such that for each class X of $w' = \binom{\gamma_1}{d_1}\binom{\gamma_2}{d_2}\dots\binom{\gamma_n}{d_n}$, the class string $\mathsf{Proj}(w'|_X)$ is accepted by $\mathscr{B}$.

The *data language* defined by a data automaton $\mathscr{D}$, denoted by $\mathcal{L}(\mathscr{D})$, is the set of data words accepted by $\mathscr{D}$.

▶ **Definition 5** ([12])**.** A *weak data automaton* (WDA) is a tuple $(\mathscr{A}, \mathscr{C})$ such that $\mathscr{A} = (Q, \Sigma \times \{\bot, \top\}, \Gamma, \delta, q_0, F)$ is a letter-to-letter transducer and the class condition $\mathscr{C}$ is specified by a collection of

- key constraints of the form $\mathsf{Key}(\gamma)$ (where $\gamma \in \Gamma$), interpreted as "every two γ-positions have different data values",
- inclusion constraints of the form $D(\gamma) \subseteq \bigcup_{\gamma' \in R} D(\gamma')$ (where $\gamma \in \Gamma, R \subseteq \Gamma$), interpreted as "for every data value occurring in a γ-position, there is $\gamma' \in R$ such that the data value also occurs in a γ'-position",
- and denial constraints of the form $D(\gamma) \cap D(\gamma') = \emptyset$ (where $\gamma, \gamma' \in \Gamma$), interpreted as "no data value occurs in both a γ-position and a γ'-position".

A data word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots\binom{\sigma_n}{d_n}$ is accepted by a weak data automaton $\mathscr{D} = (\mathscr{A}, \mathscr{C})$ iff there is an accepting run of $\mathscr{A}$ over $\mathsf{Profile}(w)$ which produces a word $\gamma_1 \dots \gamma_n$ such that the data word $w' = \binom{\gamma_1}{d_1}\binom{\gamma_2}{d_2}\dots\binom{\gamma_n}{d_n}$ satisfies all the constraints in $\mathscr{C}$.

▶ **Definition 6.** A *commutative data automaton* (CDA) $\mathscr{D}$ is a tuple $(\mathscr{A}, \varphi)$ such that $\mathscr{A} = (Q, \Sigma \times \{\bot, \top\}, \Gamma, \delta, q_0, F)$ is a letter-to-letter transducer and φ is a QFSP formula over the variable set V_Γ.

A data word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots\binom{\sigma_n}{d_n}$ is accepted by a commutative data automaton $\mathscr{D} = (\mathscr{A}, \varphi)$ iff there is an accepting run of $\mathscr{A}$ over $\mathsf{Profile}(w)$ which produces a word $\gamma_1 \dots \gamma_n$ such that the data word $w' = \binom{\gamma_1}{d_1}\binom{\gamma_2}{d_2}\dots\binom{\gamma_n}{d_n}$ satisfies that $w' \models_c \varphi$.

▶ Remark. We choose to define the class conditions of commutative data automata by QFSP formulas, instead of finite automata with commutative transition relations. The main purpose of this choice is to ease the extension of the results to data ω-words (c.f. Section 5). In addition, in the definition of commutative data automata, we choose the input alphabet of the transducer to be $\Sigma \times \{\bot, \top\}$, instead of Σ. It seems for us that this choice strictly increases the expressive power of commutative data automata, but we admit that we do not know how to prove it at present. ◀

3 Expressiveness

In this section, we first show that the expressibility of CDA lies strictly between WDA and DA, then we discuss the closure properties of CDA and provide a logical characterization of CDA.

▶ **Theorem 7.** *WDA $<$ CDA $<$ DA.*

Proof.

CDA $<$ DA.

It was shown in [12] that the language "for every occurrence of a, there is an occurrence of b on the right with the same data value" cannot be expressed in WDA. The same proof can be applied to show that the language is not expressible in CDA. On the other hand, it is easy to see that the language can be defined by a DA.

WDA $<$ CDA.

From any WDA $(\mathscr{A}, \mathscr{C})$, an equivalent CDA $(\mathscr{A}, \varphi_{\mathscr{C}})$ can be constructed such that $\varphi_{\mathscr{C}} := \bigwedge\limits_{C \in \mathscr{C}} \varphi_C$, where φ_C is defined as follows,

- if C is of the form $\mathsf{Key}(\gamma)$, then $\varphi_C := x_\gamma \leq 1$,
- if C is of the form $D(\gamma) \subseteq \bigcup\limits_{\gamma' \in R} D(\gamma')$, then $\varphi_C := x_\gamma \geq 1 \rightarrow \sum\limits_{\gamma' \in R} x_{\gamma'} \geq 1$,
- if C is of the form $D(\gamma) \cap D(\gamma') = \emptyset$, then $\varphi_C := x_\gamma \geq 1 \rightarrow x_{\gamma'} = 0$.

For the strictness of the inclusion, it is easy to observe that the language "In each class of the data word, the letter a occurs an even number of times" is expressible in CDA. By some pumping argument, we can show that the language is not expressible in WDA. ◀

▶ Remark. According to the above reduction of WDA to CDA, we would like to say that in some sense, CDA = WDA + Modulo constraints in class conditions.

▶ **Theorem 8.** *CDAs are closed under union and intersection, but not closed under complementation.*

In the following, we define $EMSO^2_{\#}(+1, \sim, \Sigma)$, a counting extension of $EMSO^2(+1, \sim, \Sigma)$, and show that it is expressively equivalent to CDA.

The logic $EMSO^2_{\#}(+1, \sim, \Sigma)$ includes all the formulas of the form $\exists R_1 \dots R_l(\varphi \wedge \forall x \psi)$ (where $R_1, \dots, R_l$ are unary predicates), such that $\varphi \in FO^2(+1, \sim, \Sigma, R_1, \dots, R_l)$ and ψ is a Boolean combination of atomic formulas of the form $\sum\limits_{\tau \in \Delta} \#_{x \sim y \wedge \tau(y)}(y) \geq c$, or $\sum\limits_{\tau \in \Delta} \#_{x \sim y \wedge \tau(y)}(y) \leq c$, or $\sum\limits_{\tau \in \Delta} \#_{x \sim y \wedge \tau(y)}(y) = c$, or $\sum\limits_{\tau \in \Delta} \#_{x \sim y \wedge \tau(y)}(y) \equiv r \bmod p$, satisfying that $\Delta \subseteq \Sigma \times 2^{\{R_1, \dots, R_l\}}$, $c \in \mathbb{N}$, $p \geq 2$, $0 \leq r < p$, and if $\tau = (\sigma, \mathcal{R})$, then $\tau(y) = \sigma(y) \wedge \bigwedge\limits_{i: 1 \leq i \leq l} \eta_{R_i}(y)$, where $\eta_{R_i}(y) = R_i(y)$ if $R_i \in \mathcal{R}$, and $\eta_{R_i}(y) = \neg R_i(y)$ otherwise.

The semantics of $EMSO^2(+1, \sim, \Sigma)$ formulas can be extended naturally to $EMSO^2_{\#}(+1, \sim, \Sigma)$ formulas by interpreting formulas $\forall x \psi$ as the counting constraints for each class. Let's take the formula $\forall x \left(\#_{x \sim y \wedge \tau(y)}(y) \geq c\right)$ as an example: Given a data word w over the alphabet $\Sigma \times 2^{\{R_1, \dots, R_l\}}$, $w \models \forall x \left(\#_{x \sim y \wedge \tau(y)}(y) \geq c\right)$ iff for each class X of w, the number of τ-positions in X is at least c.

▶ **Theorem 9.** *$EMSO^2_{\#}(+1, \sim, \Sigma)$ and CDA are expressively equivalent.*

- *Given an $EMSO^2_{\#}(+1, \sim, \Sigma)$ formula $\exists R_1 \dots R_l(\varphi \wedge \forall x \psi)$, a CDA $\mathscr{D} = (\mathscr{A}, \varphi')$ of doubly exponential size can be constructed such that $\mathcal{L}(\mathscr{D}) = \mathcal{L}(\exists R_1 \dots R_l(\varphi \wedge \forall x \psi))$. In addition, the size of the output alphabet of $\mathscr{A}$ is at most exponential over the size of $\exists R_1 \dots R_l(\varphi \wedge \forall x \psi)$.*
- *Given a CDA $\mathscr{D} = (\mathscr{A}, \varphi)$, an $EMSO^2_{\#}(+1, \sim, \Sigma)$ formula φ' of polynomial size can be constructed such that $\mathcal{L}(\mathscr{D}) = \mathcal{L}(\varphi')$.*

4 The nonemptiness problem of CDA

In this section, we prove the main result of this paper.

▶ **Theorem 10.** *The nonemptiness of CDA can be decided in 3-NEXPTIME.*

The rest of this section is devoted to the proof of Theorem 10. Although the structure of the proof is similar to that for WDA in [12, 5], the proofs of several lemmas become more complicated.

Through this section, let $\mathscr{D} = (\mathscr{A}, \varphi)$ be a commutative data automaton such that $\mathscr{A} = (Q, \Sigma \times \{\bot, \top\}, \Gamma, \delta, q_0, F)$ and φ is a QFSP formula over the variable set V_Γ.

Because we are concerned with the nonemptiness problem, without loss of generality, we can assume that $\mathscr{A} = (Q, \Gamma \times \{\bot, \top\}, \delta, q_0, F)$ is just a finite automaton over the alphabet $\Gamma \times \{\bot, \top\}$. Then the nonemptiness of $\mathscr{D}$ is reduced to the following problem.

PROBLEM:	NONEMPTINESS-PROFILE
INPUT:	A finite automaton $\mathscr{A} = (Q, \Gamma \times \{\bot, \top\}, \delta, q_0, F)$ and a QFSP formula φ over V_Γ
QUESTION:	Is there a data word w over Γ such that $\mathsf{Profile}(w)$ is accepted by $\mathscr{A}$ and $w \models_c \varphi$?

The outline of the proof goes as follows.

- At first, a finite automaton $\mathscr{A}'$ of exponential size over the alphabet Γ', and a QFSP formula φ' in the normal form of doubly exponential size over the variable set $V_{\Gamma'}$, are constructed from $\mathscr{D} = (\mathscr{A}, \varphi)$ such that the problem of NONEMPTINESS-PROFILE is reduced to the following problem,

  > "is there a *locally different* data word w over the alphabet Γ' such that $\mathsf{Proj}(w)$ is accepted by $\mathscr{A}'$ and $w \models_c \varphi'$?"

  We would like to point out that the finite automaton $\mathscr{A}'$ runs directly on the *projections* of data words, instead of the profile words of them.

  Let's call this problem NONEMPTINESS-LOCALLY-DIFFERENT, which is formally defined as follows.

PROBLEM:	NONEMPTINESS-LOCALLY-DIFFERENT
INPUT:	A finite automaton $\mathscr{A} = (Q, \Gamma, \delta, q_0, F)$ and a QFSP formula φ over V_Γ in the normal form
QUESTION:	Is there a locally different data word w over Γ such that $\mathsf{Proj}(w)$ is accepted by $\mathscr{A}$ and $w \models_c \varphi$?

- Then a 2-NEXPTIME algorithm is presented to solve the problem of NONEMPTINESS-LOCALLY-DIFFERENT.

From the above description of the proof outline, it is evident that NONEMPTINESS-PROFILE can be decided in 4-NEXPTIME. By a finer analysis, the complexity can be shown in 3-NEXPTIME.

Since the reduction of the problem of NONEMPTINESS-PROFILE to the problem of NONEMPTINESS-LOCALLY-DIFFERENT completely mimics that for WDA in [12, 5], it is omitted here due to the lack of space.

In the rest of this section, we will focus on the problem of NONEMPTINESS-LOCALLY-DIFFERENT. Before presenting an algorithm to solve the problem, we will state and prove two lemmas.

4.1 Two lemmas

We first introduce some notations.

▶ **Definition 11.** *Let* $\varphi = \bigvee\limits_{1 \leq i \leq m} \varphi_i$ *be a QFSP formula in the normal form over the variable set* V_Γ*,* p_0 *be the normalization number of* φ*, and for every* $i : 1 \leq i \leq m$*,* $\varphi_i = \bigwedge\limits_{\gamma \in \Gamma} \varphi_{i,\gamma}$*, where* $\varphi_{i,\gamma}$ *is either* $x_\gamma = c_{i,\gamma}$ *or* $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$ *for some* $c_{i,\gamma}, r_{i,\gamma} : 0 \leq c_{i,\gamma}, r_{i,\gamma} < p_0$*. Then for each* $\gamma \in \Gamma$*, define two subsets of* $\{1, \ldots, m\}$*, denoted by* $I_E(\varphi, \gamma)$ *and* $I_M(\varphi, \gamma)$*, as follows: For every* $i : 1 \leq i \leq m$*,*

- $i \in I_E(\varphi, \gamma)$ *iff* $\varphi_{i,\gamma}$ *is* $x_\gamma = c_{i,\gamma}$ *and* $c_{i,\gamma} > 0$*,*
- $i \in I_M(\varphi, \gamma)$ *iff* $\varphi_{i,\gamma}$ *is* $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$*.*

Note that if $i \notin I_E(\varphi, \gamma) \cup I_M(\varphi, \gamma)$, then it holds that $\varphi_{i,\gamma}$ is $x_\gamma = c_{i,\gamma}$ and $c_{i,\gamma} = 0$.

▶ **Definition 12.** *Let* w *be a data word over* Γ *and* $\varphi = \bigvee\limits_{1 \leq i \leq m} \varphi_i$ *be a QFSP formula over the variable set* V_Γ *in the normal form. If* $w \models_c \varphi$*, then for each data value* d *occurring in* w*, there is a unique* $i : 1 \leq i \leq m$ *such that* $\mathsf{Proj}(w|_X) \models \varphi_i$*, where* X *is the class of* w *corresponding to* d*. This unique number* i *is called* the index of the class condition φ *for* d*, denoted by* $\mathsf{idx}_\varphi(d)$*.*

We are ready to state and prove the two lemmas.

▶ **Lemma 13.** *For every QFSP formula* $\varphi = \bigvee\limits_{1 \leq i \leq m} \varphi_i$ *over the variable set* V_Γ *in the normal form, there is an EP formula* $\psi = \exists y_1 \ldots \exists y_m \psi'$ *of polynomial size such that for each word* v *over* Γ*,* $v \models \psi$ *iff there is a data word* w *such that* $\mathsf{Proj}(w) = v$ *and* $w \models_c \varphi$*.*

Proof. Suppose φ is a QFSP formula in the normal form over the variable set V_Γ with the normalization number p_0. Then $\varphi = \bigvee\limits_{i : 1 \leq i \leq m} \varphi_i$, where φ_i is of the form $\bigwedge\limits_{\gamma \in \Gamma} \varphi_{i,\gamma}$ such that $\varphi_{i,\gamma}$ is equal to $x_\gamma = c_{i,\gamma}$ or $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$ for some $c_{i,\gamma}, r_{i,\gamma} : 0 \leq c_{i,\gamma}, r_{i,\gamma} < p_0$. In addition, those φ_i's are mutually exclusive.

Let $\psi = \exists y_1 \ldots \exists y_m \psi'$ such that ψ' is a conjunction of the quantifier-free Presburger formulas ψ'_1, ψ'_2, and ψ'_3, where

1. $\psi'_1 := \bigwedge\limits_{\gamma \in \Gamma} \left(x_\gamma - \left(\sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} y_i \right) - \left(\sum\limits_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) y_i \right) \geq 0 \right)$,
2. $\psi'_2 := \bigwedge\limits_{\gamma \in \Gamma} \left(\left(\bigwedge\limits_{i \in I_M(\varphi,\gamma)} y_i = 0 \right) \rightarrow x_\gamma - \left(\sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} y_i \right) = 0 \right)$,
3. $\psi'_3 := \bigwedge\limits_{\gamma \in \Gamma} \left(x_\gamma - \left(\sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} y_i + \sum\limits_{i \in I_M(\varphi,\gamma)} r_{i,\gamma} y_i \right) \equiv 0 \bmod p_0 \right)$.

Intuitively,

- $y_1, \ldots, y_m$ represent the numbers of classes satisfying respectively $\varphi_1, \ldots, \varphi_m$,
- the formula ψ'_1 specifies the lower bound of γ-positions for every $\gamma \in \Gamma$, which is the sum of the lower bounds of γ-positions in all classes, more precisely, $c_{i,\gamma}$ for $i \in I_E(\varphi, \gamma)$ or $p_0 + r_{i,\gamma}$ for $i \in I_M(\varphi, \gamma)$,
- the formula ψ'_2 specifies that for each $\gamma \in \Gamma$, if there are no classes in which modular constraints for γ are required (this is specified by the condition $y_i = 0$ for every $i \in I_M(\varphi, \gamma)$), then the number of γ-positions is equal to the sum of $c_{i,\gamma}$ for $i \in I_E(\varphi, \gamma)$,
- the formula ψ'_3 says that for every $\gamma \in \Gamma$, the number of γ-positions, subtracting the lower bound specified in ψ'_1, should be equal to zero modulo p_0.

Let $\overline{y}$ denote the tuple $y_1, \ldots, y_m$ in the following.

"If" part:

Suppose there is a data word w such that $\mathsf{Proj}(w) = v$ and $w \models_c \varphi$, namely, for each class X in w, $\mathsf{Proj}(w|_X) \models \varphi$.

For each $i : 1 \leq i \leq m$, let D_i be the set of data values d occurring in w such that $\mathsf{idx}_\varphi(d) = i$. Note that $(D_i)_{1 \leq i \leq m}$ forms a partition of the set of all the data values occurring in w. In addition, let $k_i = |D_i|$ for each $i : 1 \leq i \leq m$. We also use $\overline{k}$ to denote the tuple $k_1, \ldots, k_m$.

It is sufficient to verify that $v \models \psi'[\overline{y} \leftarrow \overline{k}]$ in order to show $v \models \psi$.

Let's exemplify the argument by demonstrating that $v \models \psi'_2[\overline{y} \leftarrow \overline{k}]$.

Suppose $k_i = 0$ for each $i \in I_M(\varphi, \gamma)$. Then $D_i = \emptyset$ for each $i \in I_M(\varphi, \gamma)$. We want to show that $\#_\gamma(v) = \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i$.

For each data value $d \in D_i$ such that $i \notin I_M(\varphi, \gamma)$,

- if $i \in I_E(\varphi, \gamma)$, i.e. $\varphi_{i,\gamma}$ is equal to $x_\gamma = c_{i,\gamma}$ and $c_{i,\gamma} > 0$, then the letter γ occurs exactly $c_{i,\gamma}$ times in the class of w corresponding to d;
- if $i \notin I_E(\varphi, \gamma) \cup I_M(\varphi, \gamma)$, i.e. $\varphi_{i,\gamma}$ is equal to $x_\gamma = c_{i,\gamma}$ and $c_{i,\gamma} = 0$, then the letter γ does not occur in the class of w corresponding to d.

Since $(D_i)_{1 \leq i \leq m}$ is a partition of the set of all the data values occurring in w, it follows that $\#_\gamma(v) = \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i$. So $v \models \psi'_2[\overline{y} \leftarrow \overline{k}]$.

"Only if" part:

Suppose $v \models \psi$. Then there are numbers $\overline{k} = k_1, \ldots, k_m$ such that $v \models \psi'[\overline{y} \leftarrow \overline{k}]$.

Let $K = k_1 + \cdots + k_m$. Define a function $\xi : \{1, \ldots, K\} \to \{1, \ldots, m\}$ such that $|\xi^{-1}(i)| = k_i$ for each $i : 1 \leq i \leq m$.

In the following, we assign the data values from $\{1, \ldots, K\}$ to the positions in v to get a data word w such that $w \models_c \varphi$, namely, for each class X of w, $\mathsf{Proj}(w|_X) \models \varphi$.

From the fact that $v \models \psi'_1[\overline{y} \leftarrow \overline{k}]$, we know that for each $\gamma \in \Gamma$,

$$\#_\gamma(v) \geq \sum_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i \; + \sum_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) k_i.$$

We assign the data values in $\{1, \ldots, K\}$ to the positions in v through the following two-step procedure.

Step 1 For every $\gamma \in \Gamma$ and every $i : 1 \leq i \leq m$, assign the data values in $\xi^{-1}(i)$ to the γ-positions in v such that each data value in $\xi^{-1}(i)$ is assigned to exactly $(c_{i,\gamma})$ γ-positions if $i \in I_E(\varphi, \gamma)$, and is assigned to exactly $(p_0 + r_{i,\gamma})$ γ-positions if $i \in I_M(\varphi, \gamma)$.

Step 2 For every $\gamma \in \Gamma$ such that there is $i \in I_M(\varphi, \gamma)$ satisfying that $k_i > 0$, select such an index i and a data value from $\xi^{-1}(i)$, denoted by d_γ, and assign d_γ to all the γ-positions which have not been assigned data values after Step 1.

Now all the positions of v have been assigned data values from $\{1, \ldots, K\}$, let w be the resulting data word.

For every $\gamma \in \Gamma$, if there are still γ-positions that have not been assigned data values after Step 1, then $\#_\gamma(v) > \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i \; + \sum\limits_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) k_i \geq \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i$. From the fact that $v \models \psi'_2[\overline{y} \leftarrow \overline{k}]$, it follows that there is $i \in I_M(\varphi, \gamma)$ such that $k_i > 0$. So a data value d_γ can be selected and assigned to all the pending γ-positions in Step 2.

It remains to show that $w \models_c \varphi$. It is sufficient to prove that for every $i : 1 \leq i \leq m$ and every data value $d \in \xi^{-1}(i)$, $\mathsf{Proj}(w|_X) \models \varphi_i$, where X is the class of w corresponding to d. Because $\mathsf{Proj}(w|_X) = v|_X$ and $\varphi_i = \bigwedge\limits_{\gamma \in \Gamma} \varphi_{i,\gamma}$, it is equivalent to show that for every

$i : 1 \leq i \leq m$, $d \in \xi^{-1}(i)$, and $\gamma \in \Gamma$, we have $v|_X \models \varphi_{i,\gamma}$, where X is the class of w corresponding to d.

Suppose $i : 1 \leq i \leq m$, $d \in \xi^{-1}(i)$, and $\gamma \in \Gamma$. Let X be the class of w corresponding to d. In the following, we show that $v|_X \models \varphi_{i,\gamma}$.

From the data value assignment procedure, we know that there are still $\left(\#_\gamma(v) - \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i - \sum\limits_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) k_i\right)$ γ-positions which have not been assigned data values after Step 1. Because $v \models \psi'_3[\overline{y} \leftarrow \overline{k}]$, it follows that $\#_\gamma(v) - \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i - \sum\limits_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) k_i \equiv 0 \bmod p_0$. So there is $t_\gamma \in \mathbb{N}$ such that $\#_\gamma(v) - \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i - \sum\limits_{i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}) k_i = t_\gamma p_0$.

We distinguish between the following three cases.

Case $i \in I_E(\varphi, \gamma)$. Then $\varphi_{i,\gamma}$ is $x_\gamma = c_{i,\gamma}$ and $c_{i,\gamma} > 0$. From the data value assignment procedure, we know that each data value in $\xi^{-1}(i)$, including d, has been assigned to exactly $(c_{i,\gamma})$ γ-positions. This implies that $\#_\gamma(v|_X) = c_{i,\gamma}$. So $v|_X \models \varphi_{i,\gamma}$.

Case $i \in I_M(\varphi, \gamma)$. Then $\varphi_{i,\gamma}$ is $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$. From the data value assignment procedure, we know that the data value d is assigned to $(p_0 + r_{i,\gamma})$ γ-positions if $d \neq d_\gamma$, and assigned to $(p_0 + r_{i,\gamma} + t_\gamma p_0)$ γ-positions otherwise. Therefore, $\#_\gamma(v|_X) = p_0 + r_{i,\gamma}$ or $p_0 + r_{i,\gamma} + t_\gamma p_0$. It follows that $v|_X \models \varphi_{i,\gamma}$.

Case $i \notin I_E(\varphi, \gamma) \cup I_M(\varphi, \gamma)$. Then $\varphi_{i,\gamma}$ is $x_\gamma = c_{i,\gamma}$ and $c_{i,\gamma} = 0$. From the data value assignment procedure, we know that each data value in $\xi^{-1}(i)$, including d, has not been assigned to any γ-position in v. Therefore, $\#_\gamma(v|_X) = 0$ and $v|_X \models \varphi_{i,\gamma}$. ◀

▶ **Definition 14.** *Let $\varphi = \bigvee\limits_{1 \leq i \leq m} \varphi_i$ be a QFSP formula in the normal form over the variable set V_Γ with the normalization number p_0 such that for each $i : 1 \leq i \leq m$, $\varphi_i = \bigwedge\limits_{\gamma \in \Gamma} \varphi_{i,\gamma}$, where $\varphi_{i,\gamma}$ is $x_\gamma = c_{i,\gamma}$ or $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$ for $0 \leq c_{i,\gamma}, r_{i,\gamma} < p_0$. Moreover, for each $i : 1 \leq i \leq m$, let*

$$h_i = \sum_{\gamma : i \in I_E(\varphi,\gamma)} c_{i,\gamma} + \sum_{\gamma : i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma}).$$

Let w be a data word over the alphabet Γ, then w is said to satisfy the class condition φ with "many" data values if $w \models_c \varphi$ and for each $i : 1 \leq i \leq m$, either $k_i = 0$ or $k_i \geq \max(2p_0 + 1,\ 2h_i + 3)$, where k_i is the number of data values d occurring in w such that $\mathsf{idx}_\varphi(d) = i$.

Let $v \in \Gamma^*$ and $\psi = \exists y_1 \ldots \exists y_m \psi'$ be the EP formula obtained from φ as in Lemma 13. Then v is said to satisfy ψ with "large" numbers if there are a tuple of numbers $\overline{k} = k_1, \ldots, k_m$ such that $v \models \psi'[\overline{y} \leftarrow \overline{k}]$ and for each $i : 1 \leq i \leq m$, either $k_i = 0$ or $k_i \geq \max(2p_0 + 1,\ 2h_i + 3)$.

▶ **Lemma 15.** *Let $\varphi = \bigvee\limits_{1 \leq i \leq m} \bigwedge\limits_{\gamma \in \Gamma} \varphi_{i,\gamma}$ be a QFSP formula in the normal form with the normalization number p_0. Moreover, let $\psi = \exists y_1 \ldots \exists y_m \psi'$ be the EP formula obtained from φ as stated in Lemma 13. Then for any $v \in \Gamma^*$, $v \models \psi$ with large numbers iff there is a locally different data word w such that $\mathsf{Proj}(w) = v$ and $w \models_c \varphi$ with many data values.*

Proof. "If" part: Obvious.

"Only if" part:

Suppose v satisfies ψ with large numbers, i.e. there are numbers $\overline{k} = k_1, \ldots, k_m$ such that $v \models \psi'[\overline{y} \leftarrow \overline{k}]$ and for each $i : 1 \leq i \leq m$, either $k_i = 0$ or $k_i \geq \max(2p_0 + 1, 2h_i + 3)$.

Let $K = k_1 + \cdots + k_m$. Define a function $\xi : \{1, \ldots, K\} \to \{1, \ldots, m\}$ such that $|\xi^{-1}(i)| = k_i$ for each $i : 1 \leq i \leq m$.

As in the proof of Lemma 13, we assign data values in $\{1, \ldots, K\}$ to the positions of v to get a desired data word w. The assignment procedure is divided into two steps, Step 1 and 2.

Step 1:

The same as Step 1 of the data value assignment procedure in the proof of the "Only if" part of Lemma 13.

After Step 1, we get a partial data word where some positions still have no data values. Let's assign a special data value, say $\sharp$, to all those positions without data values, then we get a data word $w_1 = \binom{\gamma_1}{d_1}\binom{\gamma_2}{d_2}\cdots\binom{\gamma_n}{d_n}$.

In w_1, there may exist positions j such that $d_j = d_{j+1}$ and $d_j, d_{j+1} \neq \sharp$. Let's call these positions as *conflicting* positions of w_1.

Claim. The data word w_1 can be turned into a data word w_1'' such that w_1'' contains no conflicting positions, w_1 and w_1'' have the same set of data values (including $\sharp$), and for each $\gamma \in \Gamma$ and each class X of w_1, $\#_\gamma(w_1|_X) = \#_\gamma(w_1''|_X)$.

Proof of the claim.

Let j be a conflicting position of w_1, $a = \gamma_j$, and $i : 1 \leq i \leq m$ such that $d_j \in \xi^{-1}(i)$. From Step 1, we know that d_j occurs exactly $h_i = \sum\limits_{\gamma : i \in I_E(\varphi,\gamma)} c_{i,\gamma} + \sum\limits_{\gamma : i \in I_M(\gamma)} (p_0 + r_{i,\gamma})$ times in w_1. It follows that there are at most $2h_i$ positions adjacent to a position with the data value d_j. On the other hand, we have that $k_i \geq \max(2p_0 + 1, 2h_i + 3)$ and for each data value in $d \in \xi^{-1}(i)$, there is at least one occurrence of a with the data value d. It follows that there are (at least) *three* positions j_1', j_2', j_3' such that $d_{j_1'}, d_{j_2'}, d_{j_3'} \in \xi^{-1}(i)$, $d_{j_1'}, d_{j_2'}, d_{j_3'}$ are pairwise distinct, $\gamma_{j_1'} = \gamma_{j_2'} = \gamma_{j_3'} = a$, and $d_{j_1'-1}, d_{j_2'-1}, d_{j_3'-1}, d_{j_1'+1}, d_{j_2'+1}, d_{j_3'+1} \neq d_j$. From this, we deduce that there is a position j' such that $\gamma_{j'} = a$, $d_{j'} \in \xi^{-1}(i)$, $d_{j'} \neq d_{j-1}, d_j$, and $d_j \neq d_{j'-1}, d_{j'+1}$. Because $d_{j'} \neq d_{j-1}, d_{j+1}$ ($d_{j+1} = d_j$ since j is conflicting) and $d_j \neq d_{j'-1}, d_{j'+1}$, we can swap the data value d_j in the position j and the data value $d_{j'}$ in the position j' to make the two positions j and j' non-conflicting. Let w_1' be the data word after the swapping. It follows that w_1' has less conflicting positions than w_1.

Continue like this, we finally get a data word w_1'' without conflicting positions. ◀

Now we return to the proof of the lemma.

From the claim, we know that a data word w_1'' containing no conflicting positions can be obtained from w_1. But w_1'' may still contain the special data value $\sharp$. If this is the case, then from the description of Step 1, we know that there exists at least one $\gamma \in \Gamma$ such that $\#_\gamma(v) > \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i + \sum\limits_{i \in I_M(\gamma)} (p_0 + r_{i,\gamma}) k_i$.

Let $\gamma \in \Gamma$ such that $\#_\gamma(v) > \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i + \sum\limits_{i \in I_M(\gamma)} (p_0 + r_{i,\gamma}) k_i$.

From $v \models \psi_3'[\overline{y} \leftarrow \overline{k}]$, it follows that $\#_\gamma(v) - \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i - \sum\limits_{i \in I_M(\gamma)} (p_0 + r_{i,\gamma}) k_i \equiv 0 \bmod p_0$. So there is $t_\gamma \geq 1$ such that $\#_\gamma(v) - \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i - \sum\limits_{i \in I_M(\gamma)} (p_0 + r_{i,\gamma}) k_i = t_\gamma p_0$. Therefore, there are $(t_\gamma p_0)$ γ-positions in w_1 of the data value $\sharp$. Because w_1 and w_1'' have the same set of positions of the data value $\sharp$, it follows that there are also $(t_\gamma p_0)$ γ-positions in w_1'' of the data value $\sharp$. Let $j_{\gamma,1} < \cdots < j_{\gamma,t_\gamma p_0}$ be a list of all such γ-positions in w_1''.

On the other hand, because $v \models \psi_2'[\overline{y} \leftarrow \overline{k}]$ and $\#_\gamma(v) > \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i$, it follows that there is $i \in I_M(\varphi, \gamma)$ such that $k_i > 0$. Let i_γ be such an index i. Then from the assumption that v satisfies ψ with large numbers, we know that $k_{i_\gamma} \geq \max(2p_0 + 1, 2h_{i_\gamma} + 3)$.

Step 2:

For each $\gamma \in \Gamma$ such that $\#_\gamma(v) > \sum\limits_{i \in I_E(\varphi,\gamma)} c_{i,\gamma} k_i + \sum\limits_{i \in I_M(\gamma)} (p_0 + r_{i,\gamma}) k_i$, assign the data values from $\{1, \dots, K\}$ to the γ-positions with the data value $\sharp$ in w_1'' as follows. We distinguish between the following two cases.

- *Case $t_\gamma \geq 2$.*
  Initially set $s := 1$. Repeat following procedure until $s > t_\gamma$.

  Let $J = \{j_{\gamma,s}, j_{\gamma,t_\gamma+s}, \dots, j_{\gamma,t_\gamma(p_0-1)+s}\}$ and J' be the set of all positions adjacent to a position in J. In addition, let D be the set of data values (except $\sharp$) occurring in the positions belonging to J'. Because $|J| = p_0$, we have $|D| \leq 2p_0$. On the other hand, $k_{i_\gamma} \geq 2p_0 + 1$, it follows that there is $d \in \xi^{-1}(i_\gamma) \setminus D$.
  Assign the data value d to every position in J. Then we still get a non-conflicting data word, since all the positions in J are not adjacent to each other.
  Set $s := s + 1$.

- *Case $t_\gamma = 1$.*
  Let $J = \{j_{\gamma,1}, j_{\gamma,2}, \dots, j_{\gamma,p_0}\}$ and J' be the set of all positions adjacent to a position in J. In addition, let D be the set of data values (except $\sharp$) occurring in the positions belonging to J'. Because $|J| = p_0$, we have $|D| \leq 2p_0$. On the other hand, $k_{i_\gamma} \geq 2p_0 + 1$, it follows that there is $d \in \xi^{-1}(i_\gamma) \setminus D$.
  Because $i_\gamma \in I_M(\varphi, \gamma)$, each data value in $\xi^{-1}(i_\gamma)$ has been assigned to exactly $(p_0 + r_{i_\gamma,\gamma})$ γ-positions in Step 1. During Step 1, we can do the assignments in a way so that all the positions in J, i.e. the p_0 γ-positions without data value, are not adjacent to each other. Therefore, we can assign the data value d to every position in J and still get a non-conflicting data word.

Let w be the resulting data word after the two steps of data value assignments. Then w is locally different. Similar to the proof of the "Only if" part of Lemma 13, we can show that for each $i : 1 \leq i \leq m$ and each data value $d \in \xi^{-1}(i)$, $\mathsf{Proj}(w|_X) \models_c \varphi_i$, where X is the class of w corresponding to d. From this, it follows that for each $i : 1 \leq i \leq m$, the number of data values in w such that $i \in \mathsf{idx}_\varphi(d)$ is equal to k_i. Since for each $i : 1 \leq i \leq m$, either $k_i = 0$ or $k_i \geq \max(2p_0 + 1, 2h_i + 3)$, we conclude that $w \models_c \varphi$ with many data values. ◀

4.2 Algorithm for NONEMPTINESS-LOCALLY-DIFFERENT

We first give an algorithm for the following problem.

PROBLEM:	NONEMPTINESS-LOCALLY-DIFFERENT-MANY
INPUT:	A finite automaton $\mathscr{A} = (Q, \Gamma, \delta, q_0, F)$ and a QFSP formula φ over V_Γ in the normal form
QUESTION:	is there a locally different data word w over the alphabet Γ such that $\mathsf{Proj}(w)$ is accepted by $\mathscr{A}$ and $w \models_c \varphi$ with many data values?

From Lemma 15, it follows that NONEMPTINESS-LOCALLY-DIFFERENT-MANY can be solved by the following algorithm.

Suppose the normalization number of φ is p_0 and $\varphi = \bigvee\limits_{i:1\leq i\leq m} \bigwedge\limits_{\gamma\in\Gamma} \varphi_{i,\gamma}$ such that each $\varphi_{i,\gamma}$ is either $x_\gamma = c_{i,\gamma}$ or $x_\gamma \geq p_0 \wedge x_\gamma \equiv r_{i,\gamma} \bmod p_0$ for some $c_{i,\gamma}, r_{i,\gamma} : 0 \leq c_{i,\gamma}, r_{i,\gamma} < p_0$. Let $\psi = \exists y_1 \dots \exists y_m \psi'$ be the existential Presburger formula obtained *from φ as stated in Lemma 13. For every $i : 1 \leq i \leq m$, let $h_i = \sum\limits_{\gamma : i \in I_E(\varphi,\gamma)} c_{i,\gamma} + \sum\limits_{\gamma : i \in I_M(\varphi,\gamma)} (p_0 + r_{i,\gamma})$.*

1. *Construct the following EP formula ψ_g,*

$$\psi_g := \exists y_1 \ldots \exists y_m \left(\psi' \wedge \bigwedge_{1 \leq i \leq m} (y_i = 0 \vee (y_i \geq 2p_0 + 1 \wedge y_i \geq 2h_i + 3)) \right).$$

2. *Decide the nonemptiness of the Presburger automaton $(\mathscr{A}, \psi_g)$.*

Now we consider the problem of NONEMPTINESS-LOCALLY-DIFFERENT.

For a data word $w \in (\Gamma \times \mathbb{D})^*$, if $w \models_c \varphi$, then for each $i : 1 \leq i \leq m$, let k_i be the number of data values d occurring in w such that $\mathsf{idx}_\varphi(d) = i$. For each $i : 1 \leq i \leq m$ such that $k_i < \max(2p_0 + 1, 2h_i + 3)$, if we take the k_i data values d such that $\mathsf{idx}_\varphi(d) = i$ as constants, then the problem of NONEMPTINESS-LOCALLY-DIFFERENT can be solved similar to the problem of NONEMPTINESS-LOCALLY-DIFFERENT-MANY. More specifically, the algorithm goes as follows.

1. *Guess a set $J \subseteq \{1, \ldots, m\}$ and sets of constants D_j's.*
   a) *Guess a set $J \subseteq \{1, \ldots, m\}$.*
   b) *For each $j \in J$, guess an integer $s_j < \max(2p_0 + 1, 2h_i + 3)$.*
   c) *For each $j \in J$, fix a set $D_j = \{\alpha^j_1, \ldots, \alpha^j_{s_j}\}$ of constants such that D_j's are mutually disjoint and $D_j \cap \mathbb{D} = \emptyset$. Let $D_J = \cup_{j \in J} D_j$.*
2. *Construct an automaton $\mathscr{A}'$ over the alphabet $\Gamma \cup \Gamma \times D_J$ from $(\mathscr{A}, \varphi)$ such that $\mathscr{A}'$ accepts a word $v = \lambda_1 \ldots \lambda_n \in (\Gamma \cup \Gamma \times D_J)^*$ iff the following conditions hold.*
   - *A symbol (γ, d) appears in v iff there exists $j \in J$ such that $j \in I_E(\varphi,\gamma) \cup I_M(\varphi,\gamma)$ and $d \in D_j$.*
   - *Let $u = \gamma_1 \ldots \gamma_n \in \Gamma^*$ such that*

$$\gamma_i = \begin{cases} \lambda_i & \text{if } \lambda_i \in \Gamma, \\ \gamma & \text{if } \lambda_i = (\gamma, d) \in \Gamma \times D_J. \end{cases}$$

   Then u is accepted by $\mathscr{A}$.
   - *For any $i : 1 \leq i < n$, if $\lambda_i = (\gamma, d)$ and $\lambda_{i+1} = (\gamma', d')$, then $d \neq d'$.*
   - *For any $j \in J$ and any $\gamma \in \Gamma$, the following holds: If $j \in I_E(\varphi,\gamma)$, then for each $d \in D_j$, the letter (γ, d) occurs exactly $c_{j,\gamma}$ times in v. If $j \in I_M(\varphi,\gamma)$, then for each $d \in D_j$, the number of occurrences of the letter (γ, d) is at least p_0 and equal to $r_{i,\gamma}$ modulo p_0.*
3. *Construct the following EP formula $\psi_{g,J}$,*

$$\psi_{g,J} = \exists y_1 \ldots \exists y_m \left(\psi' \wedge \bigwedge_{i \in J} y_i = 0 \wedge \bigwedge_{i \notin J} (y_i \geq 2p_0 + 1 \wedge y_i \geq 2h_i + 3) \right).$$

   Note that $\psi_{g,J}$ is an EP formula with free variables from V_Γ, and contains no variables $x_{(\gamma,d)}$ with $(\gamma, d) \in \Gamma \times D_J$.
4. *Decide the nonemptiness of the Presburger automaton $(\mathscr{A}', \psi_{g,J})$.*

The proof of the correctness of the above algorithm for NONEMPTINESS-LOCALLY-DIFFERENT follows the same line as the proof for SAT-LOCALLY-DIFFERENT in [5].

5 Commutative Büchi data automata

In this section, we consider data automata with commutative class conditions over data ω-words.

Let $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$ with the linear order ($<$) and the addition ($+$) operation of $\mathbb{N}$ extended in a natural way, i.e. $n < \omega$ for any $n \in \mathbb{N}$, and $\omega + n = \omega$ for any $n \in \mathbb{N}_\omega$.

The definition of the Parikh images of finite words can be easily extended to ω-words: Given an ω-word v over an alphabet $\Gamma = \{\gamma_1, \dots, \gamma_l\}$, $\mathsf{Parikh}(v) = (\#_{\gamma_1}(v), \dots, \#_{\gamma_l}(v))$, where for each $i : 1 \leq i \leq l$, $\#_{\gamma_i}(v)$ is still the number of occurrences of γ_i in v, in particular, if γ_i occurs infinitely many times in v, then $\#_{\gamma_i}(v) = \omega$.

Similar to QFSP formulas, we define ω-QFSP formulas over a variable set X as follows.

The syntax of ω-QFSP formulas is the same as QFSP formulas, except that the atomic formulas can also be of the form $x = \omega$ (where $x \in X$).

The ω-QFSP formulas are interpreted on $\mathbb{N}_\omega$: Let $\pi : X \to \mathbb{N}_\omega$, then the atomic ω-QFSP formulas are interpreted as follows,

- $\pi \models x_1 + \dots + x_n$ op c if $\pi(x_1) + \dots + \pi(x_n)$ op c, where op $\in \{\leq, \geq, =\}$,
- $\pi \models x_1 + \dots + x_n \equiv r \bmod p$ if $\pi(x_1) + \dots + \pi(x_n) < \omega$ and $\pi(x_1) + \dots + \pi(x_n) \equiv r \bmod p$,
- $\pi \models x = \omega$ if $\pi(x) = \omega$.

In addition, the Boolean operators are interpreted in a standard way.

Similar to Proposition 3, there is a normal form for ω-QFSP formulas.

▶ Proposition 16. *Let $\varphi(x_1, \dots, x_k)$ be a ω-QFSP formula. Then there exists an exponential-time algorithm to transform φ into a ω-QFSP formula $\bigvee\limits_{i:1\leq i\leq m} \varphi_i$ of size $2^{O(k|\varphi|)}$ such that there is $p_0 : 2 \leq p_0 \leq 2^{|\varphi|}$ satisfying that*

- *each φ_i is of the form $\bigwedge\limits_{1\leq j\leq k} \varphi_{i,j}$;*
- *for each $j : 1 \leq j \leq k$, $\varphi_{i,j}$ is equal to $x_j = c_{i,j}$ or $x_j \geq p_0 \wedge x_j \equiv r_{i,j} \bmod p_0$, or $x_j = \omega$ for $c_{i,j}, r_{ij} : 0 \leq c_{i,j}, r_{i,j} < p_0$;*
- *in addition, those φ_i's are mutually exclusive.*

Let w be a data ω-word over an alphabet Γ and φ be a ω-QFSP formula over the variable set V_Γ, then the definition of $w \models_c \varphi$, i.e. w satisfies the class condition φ, is a natural extension of that for data words.

A *commutative Büchi data automaton* (CBDA) is a binary tuple $(\mathscr{A}, \varphi)$, where $\mathscr{A} = (Q, \Sigma \times \{\bot, \top\}, \Gamma, \delta, q_0, F)$ is a Büchi letter-to-letter transducer and φ is a ω-QFSP formula over the variable set V_Γ.

A CBDA $(\mathscr{A}, \varphi)$ accepts a data ω-word $w = \binom{\sigma_1}{d_1}\binom{\sigma_2}{d_2}\dots$ if there is an accepting run of $\mathscr{A}$ over $\mathsf{Profile}(w)$ which produces an ω-word $\gamma_1\gamma_2\dots$ such that the data ω-word $w' = \binom{\gamma_1}{d_1}\binom{\gamma_2}{d_2}\dots\dots$ satisfies that $w' \models_c \varphi$.

Similar to the logic $EMSO^2_\#(+1, \sim, \Sigma)$ in Section 3, we define the logic $E_\infty MSO^2_\#(+1, \sim, \Sigma)$ as follows: It includes all the formulas $\exists_\infty R_1 \dots \exists_\infty R_k \exists S_1 \dots \exists S_l (\varphi \wedge \forall x \psi)$, where $\varphi \in FO^2(+1, \sim, \Sigma, R_1, \dots, R_k, S_1, \dots, S_l)$ and ψ is the same as the ψ in $EMSO^2_\#(+1, \sim, \Sigma)$ formulas, except that the atomic formulas in ψ can be also of the form $\#_{x\sim y \wedge \tau(y)}(y) = \omega$.

The semantics of $E_\infty MSO^2(+1, \sim, \Sigma)$ formulas are defined similar to $EMSO^2_\#(+1, \sim, \Sigma)$ formulas, except that the unary relation symbols $R_1, \dots, R_k$ are restricted to bind to infinite sets and $\#_{x\sim y\wedge\tau(y)}(y) = \omega$ are interpreted as the fact that the symbol τ appears infinitely many times in the class that contains the position x.

Similar to CDA, we also have the following logical characterization of CBDA.

▶ **Theorem 17.** *$E_\infty MSO^2_\#(+1, \sim, \Sigma)$ and CBDA are expressively equivalent.*

The proof of Theorem 17 is similar to that for WBDA in [12].

▶ **Theorem 18.** *The nonemptiness of CBDA can be decided in 4-NEXPTIME.*

The proof of Theorem 18 is by a nondeterminstic exponential time reduction to the nonemptiness of CDA on data words.

Acknowledgements. The author thanks Anca Muscholl for the comments and suggestions on this work. The author also thanks anonymous referees for their valuable comments and suggestions to improve the quality of this paper.

References

1 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Logic*, 12(4):27:1–27:26, 2011.

2 M. Bojanczyk and S. Lasota. An extension of data automata that captures xpath. *Logic. Method. in Comput. Sci.*, 8(1), 2012.

3 M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):1–48, 2009.

4 M Chrobak. Finite automata and unary languages. *Theor. Comput. Sci.*, 47(2):149–158, 1986.

5 C. David, L. Libkin, and T. Tan. On the satisfiability of two-variable logic over data words. In *LPAR'10*, pages 248–262, 2010.

6 S. Demri and R. Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Logic*, 10(3):16:1–16:30, 2009.

7 A. Ehrenfeucht and G. Rozenberg. Commutative linear languages. Technical Report CU-CS-209-81, Department of Computer Science, University of Colorado, 1981.

8 D. Figueira. Alternating register automata on finite data words and trees. *Logic. Method. in Comput. Sci.*, 8(1), 2012.

9 D. Figueira. Satisfiability of downward XPath with data equality tests. In *PODS*, pages 197–206, 2009.

10 A. C. Gómez and G. I. Alvarez. Learning commutative regular languages. In *ICGI*, pages 71–83, 2008.

11 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, November 1994.

12 A. Kara, T. Schwentick, and T. Tan. Feasible automata for two-variable logic with successor on data words. In *LATA*, pages 351–362, 2012. A long version can be found at http://arxiv.org/abs/1110.1221.

13 A. Manuel and R. Ramanujam. Class counting automata on datawords. *Int. J. Found. Comput. Sci.*, 22(4):863–882, 2011.

14 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004.

15 J. E. Pin. *Varieties of formal languages.* Plenum Publishers, 1986.

16 L. Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL, LNCS 4207*, pages 41–57, 2006.

17 H. Seidl, T. Schwentick, and A. Muscholl. Counting in trees. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, pages 575–612, 2008.

18 A. W. To. Unary finite automata vs. arithmetic progressions. *Inf. Process. Lett.*, 109(17):1010–1014, 2009.

19 Z. Wu. A decidable extension of data automata. In *GandALF*, pages 116–130, 2011.