

Erich Grädel  
Reinhard Kahle (Eds.)

LNCS 5771

# Computer Science Logic

23rd International Workshop, CSL 2009  
18th Annual Conference of the EACSL  
Coimbra, Portugal, September 2009, Proceedings



Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Erich Grädel Reinhard Kahle (Eds.)

# Computer Science Logic

23rd International Workshop, CSL 2009  
18th Annual Conference of the EACSL  
Coimbra, Portugal, September 7-11, 2009  
Proceedings

Volume Editors

Erich Grädel  
RWTH Aachen  
Mathematische Grundlagen der Informatik  
52056 Aachen, Germany  
E-mail: [graedel@logic.rwth-aachen.de](mailto:graedel@logic.rwth-aachen.de)

Reinhard Kahle  
Universidade Nova de Lisboa  
CENTRIA and Departamento de Matemática  
2829-516 Caparica, Portugal  
E-mail: [kahle@mat.uc.pt](mailto:kahle@mat.uc.pt)

Library of Congress Control Number: 2009933208

CR Subject Classification (1998): F.4, I.2.4, I.2.3, F.3, G.2

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

ISSN 0302-9743  
ISBN-10 3-642-04026-8 Springer Berlin Heidelberg New York  
ISBN-13 978-3-642-04026-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

[springer.com](http://springer.com)

© Springer-Verlag Berlin Heidelberg 2009  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper SPIN: 12746989 06/3180 5 4 3 2 1 0



# Preface

The annual conference of the European Association for Computer Science Logic (EACSL), CSL 2009, was held in Coimbra (Portugal), September 7–11, 2009. The conference series started as a programme of International Workshops on Computer Science Logic, and then at its sixth meeting became the Annual Conference of the EACSL. This conference was the 23rd meeting and 18th EACSL conference; it was organized at the Department of Mathematics, Faculty of Science and Technology, University of Coimbra.

In response to the call for papers, a total of 122 abstracts were submitted to CSL 2009 of which 89 were followed by a full paper. The Programme Committee selected 34 papers for presentation at the conference and publication in these proceedings.

The Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science. The award recipient for 2009 was Jakob Nordström. Citation of the award, abstract of the thesis, and a biographical sketch of the recipient may be found at the end of the proceedings. The award was sponsored for the years 2007–2009 by Logitech S.A.

After the submission deadline we received the sad news that Volker Weber died on the night of April 6–7, 2009. He had submitted a paper to CSL 2009, which ran through the regular referee process and was accepted according to the usual CSL standards. The final version, printed in these proceedings was revised by his supervisor, Thomas Schwentick, taking into account the (few) remarks of the referees. We are grateful to Thomas Schwentick, who wrote an obituary which is added to his paper.

We sincerely thank the Programme Committee and all of the referees for their help in reviewing the papers. We also thank the Organizing Committee, in particular Ana Almeida and Pedro Quaresma from the Department of Mathematics of the University of Coimbra, for their help in the organization of the conference. The conference received support from the following Portuguese Research Centers: CENTRIA, Centre for Artificial Intelligence, New University of Lisbon; CISUC, Centre for Informatics and Systems of the University of Coimbra; CITI, Center for Informatics and Information Technologies, New University of Lisbon; CMAF, Centro de Matemática e Aplicações Fundamentais, University of Lisbon; CMAT, Centre of Mathematics, University of Minho; CMUC, Centre for Mathematics, University of Coimbra; LIACC, Artificial Intelligence and Computer Science Laboratory, University of Porto; and SQIG, Security and Quantum Information Group, Instituto de Telecomunicações, Technical University of Lisbon. Further support was given by the Association for Symbolic Logic,

the Kurt Gödel Society, the Fundação para a Ciência e a Tecnologia (FCT), and the Fundação Luso-Americana. We are grateful to all these institutions for their sponsorship.

September 2009

Erich Grädel  
Reinhard Kahle

# Organization

## Programme Committee

Samson Abramsky, Oxford  
Matthias Baaz, Vienna  
Patricia Bouyer, Cachan  
Andrei Bulatov, Burnaby  
Stephen Cook, Toronto  
Anuj Dawar, Cambridge  
Hugo Gimbert, Bordeaux  
Erich Grädel, Aachen, Co-chair  
Steffen Hölldobler, Dresden  
Gerhard Jäger, Berne  
Reinhard Kahle, Lisbon, Co-chair  
Antonín Kučera, Brno

Benedikt Löwe, Amsterdam  
Simone Martini, Bologna  
Dale Miller, Paris  
Luke Ong, Oxford  
Martin Otto, Darmstadt  
Jean-Francois Raskin, Brussels  
Thomas Schwentick, Dortmund  
Luc Segoufin, Cachan  
Anton Setzer, Swansea  
Helmut Veith, Darmstadt  
Thomas Wilke, Kiel

## Organizing Committee

Ana Almeida, Coimbra, Co-chair  
Sabine Broda, Porto  
José Carlos Espírito Santo, Braga  
Mário Florido, Porto  
Gonçalo Gutierrez, Coimbra

Reinhard Kahle, Lisbon, Co-chair  
Isabel Oitavem, Lisbon  
Pedro Quaresma, Coimbra, Co-chair  
João Rasga, Lisbon  
Carlota Simões, Coimbra

## Additional Referees

Andreas Abel  
Beniamino Accattoli  
Marco Aiello  
Luca Alberucci  
Thorsten Altenkirch  
Bob Atkey  
Albert Atserias  
David Baelde  
Patrick Baillot  
Jiri Barnat  
Michele Basaldella  
Nick Benton

Christoph Benz Müller  
Stefano Berardi  
Benno van den Berg  
Achim Blumensath  
Guillaume Bonfante  
Julian Bradfield  
Franck van Breugel  
Gerhard Brewka  
Christopher Broadbent  
Veronique Bruyere  
Kai Brunnler  
Sam Buss

Xavier Caicedo Ferrer  
Catarina Carvalho  
Pierre Casteran  
Peter Chapman  
Krishnendu Chatterjee  
Hubie Chen  
Thomas Colcombet  
Willem Conradie  
Paolo Coppola  
Ugo Dal Lago  
Victor Dalmau  
Nils Anders Danielsson

Stephane Demri	Vladimir Krupski	Horst Reichel
Derek Dreyer	Kai-Uwe Kühnberger	Hilverd Reker
Arnaud Durand	Clemens Kupke	Christian Retore
Peter Dybjer	Alexander Kurz	Colin Riba
Joerg Endrullis	Francois Lamarche	Chloé Rispal
José Espírito Santo	Francois Laroussinie	Eike Ritter
Berndt Farwer	Soren Lassen	Michael Rusinowitch
Christian Fermüller	Joachim de Lataillade	Marko Samer
Andrzej Filinski	Stephane Lengrand	Ari Saptawijaya
Bernd Finkbeiner	Jérôme Leroux	Alexis Saurin
Pascal Fontaine	Reinhold Letz	Zdenek Sawa
Vojtech Forejt	Paul Levy	Renate Schmidt
Bertram Fronhöfer	Tadeusz Litak	Peter Schmitt
Murdoch Gabbay	Christof Löding	Peter Schneider-Kamp
Didier Galmiche	Markus Lohrey	Ulrich Schöpp
Blaise Genest	John Longley	Peter Schueller
Herman Geuvers	Etienne Lozes	Nicole Schweikardt
Valentin Goranko	Andrea Masini	Olivier Serre
Rajeev Gore	Ralph Matthes	Jakob Grue Simonsen
Bernhard Gramlich	Richard McKinley	Ludwig Staiger
Philippe de Groote	Enrico Moriconi	Lutz Straßburger
Masahiro Hamano	Georg Moser	Thomas Streicher
Lauri Hella	Larry Moss	Jan Strejcek
Hugo Herbelin	Andrzej Murawski	Thomas Studer
Chris Heunen	Koji Nakazawa	Hayo Thielecke
Mathieu Hoyrup	Aleks Nanevski	René Thiemann
Martin Hyland	Robert Nieuwenhuis	Leendert van der Torre
Rosalie Iemhoff	Milad Niqui	Nikos Tzevelekos
Neil Immerman	Jan Obdrzalek	Michael Ummels
Florent Jacquemard	Jaap van Oosten	Alasdair Urquhart
David Janin	Erik Palmgren	Lionel Vaux
Emil Jeřábek	Luca Paolini	Heribert Vollmer
Neil Jones	Mati Pentus	Christoph Wernhard
Yukiyoshi Kameyama	Guy Perrier	Gregory Wheeler
Akitoshi Kawamura	Brigitte Pientka	Freek Wiedijk
Delia Kesner	David Pym	Frank Wolter
Marcus Kracht	Diana Ratiu	Hans Zantema
Steve Kremer	Brian Redmond	

# Table of Contents

## Invited Talks

Algebra for Tree Languages (Abstract) . . . . .	1
<i>Mikołaj Bojańczyk</i>	
Forcing and Type Theory (Abstract) . . . . .	2
<i>Thierry Coquand</i>	
Functional Interpretations of Intuitionistic Linear Logic . . . . .	3
<i>Gilda Ferreira and Paulo Oliva</i>	
Fixed-Point Definability and Polynomial Time (Extended Abstract) . . . .	20
<i>Martin Grohe</i>	

## Special Invited Talk to Commemorate the Centenary of Stephen Cole Kleene

Kleene's Amazing Second Recursion Theorem (Extended Abstract) . . . . .	24
<i>Yiannis N. Moschovakis</i>	

## Contributed Papers

Typed Applicative Structures and Normalization by Evaluation for System $F^\omega$ . . . . .	40
<i>Andreas Abel</i>	
Jumping Boxes: Representing Lambda-Calculus Boxes by Jumps . . . . .	55
<i>Beniamino Accattoli and Stefano Guerrini</i>	
Tree-Width for First Order Formulae . . . . .	71
<i>Isolde Adler and Mark Weyer</i>	
Algorithmic Analysis of Array-Accessing Programs . . . . .	86
<i>Rajeev Alur, Pavol Černý, and Scott Weinstein</i>	
Decidable Relationships between Consistency Notions for Constraint Satisfaction Problems . . . . .	102
<i>Albert Atserias and Mark Weyer</i>	
Cardinality Quantifiers in MLO over Trees . . . . .	117
<i>Vince Bárány, Lukasz Kaiser, and Alexander Rabinovich</i>	
From Coinductive Proofs to Exact Real Arithmetic . . . . .	132
<i>Ulrich Berger</i>	

On the Relation between Sized-Types Based Termination and Semantic Labelling . . . . .	147
<i>Frédéric Blanqui and Cody Roux</i>	
Expanding the Realm of Systematic Proof Theory . . . . .	163
<i>Agata Ciabattoni, Lutz Straßburger, and Kazushige Terui</i>	
EXPTIME Tableaux for the Coalgebraic $\mu$ -Calculus . . . . .	179
<i>Corina Cîrstea, Clemens Kupke, and Dirk Pattinson</i>	
On the Word Problem for $\Sigma\pi$ -Categories, and the Properties of Two-Way Communication (Extended Abstract) . . . . .	194
<i>Robin Cockett and Luigi Santocanale</i>	
Intersection, Universally Quantified, and Reference Types . . . . .	209
<i>Mariangiola Dezani-Ciancaglini, Paola Giannini, and Simona Ronchi Della Rocca</i>	
Linear Game Automata: Decidable Hierarchy Problems for Stripped-Down Alternating Tree Automata . . . . .	225
<i>Jacques Duparc, Alessandro Facchini, and Filip Murlak</i>	
Enriching an Effect Calculus with Linear Types . . . . .	240
<i>Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson</i>	
Degrees of Undecidability in Term Rewriting . . . . .	255
<i>Jörg Endrullis, Herman Geuvers, and Hans Zantema</i>	
Upper Bounds on Stream I/O Using Semantic Interpretations . . . . .	271
<i>Marco Gaboardi and Romain Péchoux</i>	
Craig Interpolation for Linear Temporal Languages . . . . .	287
<i>Amélie Gheerbrant and Balder ten Cate</i>	
On Model Checking Boolean BI . . . . .	302
<i>Heng Guo, Hanpin Wang, Zhongyuan Xu, and Yongzhi Cao</i>	
Efficient Type-Checking for Amortised Heap-Space Analysis . . . . .	317
<i>Martin Hofmann and Dulma Rodriguez</i>	
Deciding the Inductive Validity of $\forall\exists^*$ Queries . . . . .	332
<i>Matthias Horbach and Christoph Weidenbach</i>	
On the Parameterised Intractability of Monadic Second-Order Logic . . . . .	348
<i>Stephan Kreutzer</i>	
Automatic Structures of Bounded Degree Revisited . . . . .	364
<i>Dietrich Kuske and Markus Lohrey</i>	
Nondeterminism and Observable Sequentiality . . . . .	379
<i>James Laird</i>	

A Decidable Spatial Logic with Cone-Shaped Cardinal Directions . . . . .	394
<i>Angelo Montanari, Gabriele Puppis, and Pietro Sala</i>	
Focalisation and Classical Realisability . . . . .	409
<i>Guillaume Munch-Maccagnoni</i>	
Decidable Extensions of Church's Problem . . . . .	424
<i>Alexander Rabinovich</i>	
Nested Hoare Triples and Frame Rules for Higher-Order Store . . . . .	440
<i>Jan Schwinghammer, Lars Birkedal, Bernhard Reus, and Hongseok Yang</i>	
A Complete Characterization of Observational Equivalence in Polymorphic $\lambda$ -Calculus with General References . . . . .	455
<i>Eijiro Sumii</i>	
Non-Commutative First-Order Sequent Calculus . . . . .	470
<i>Makoto Tatsuta</i>	
Model Checking FO(R) over One-Counter Processes and beyond . . . . .	485
<i>Anthony Widjaja To</i>	
Confluence of Pure Differential Nets with Promotion . . . . .	500
<i>Paolo Trinquilli</i>	
Decision Problems for Nash Equilibria in Stochastic Games . . . . .	515
<i>Michael Ummels and Dominik Wojtczak</i>	
On the Complexity of Branching-Time Logics . . . . .	530
<i>Volker Weber</i>	
Nominal Domain Theory for Concurrency . . . . .	546
<i>David Turner and Glynn Winskel</i>	
<b>Appendix</b>	
The Ackermann Award 2009 . . . . .	561
<i>Johann A. Makowsky and Alexander Razborov</i>	
<b>Author Index</b> . . . . .	567

# Algebra for Tree Languages<sup>\*</sup>

Mikołaj Bojańczyk

Institute of Informatics, Warsaw University, Poland  
bojan@mimuw.edu.pl

There are at least as many interesting classes of regular tree languages as there are of regular word languages. However, much less is known about the former ones. In particular, very few effective characterizations of tree language classes are known. Since for words most known characterizations are obtained using algebra, it seems to be a good idea to look for an algebra for tree languages. I will talk about one such attempt, which is called forest algebra. (Other frameworks in the literature include pre-clones of Ésik and Weil or tree algebra of Wilke. Another approach is to forget about algebra and study the structure of a tree automaton.)

In the algebraic theory of word languages, elements of a monoid or semigroup are used to represent words. In a forest algebra, there are two sorts. The first sort represents forests, and the second sort represents contexts (forests with a hole). The operations of forest algebra include concatenation of forests (concatenating a forest of  $n$  trees with a forest of  $m$  trees gives a forest with  $n + m$  trees) and composition of contexts (composing a context where the hole is at depth  $n$  with a context where the hole is at depth  $m$  gives a context where the hole is at depth  $n + m$ ).

I will talk about some recent work, which tries to relate the algebraic structure of a forest algebra with the type of tree languages that this algebra recognizes. A very simple example of this kind says that if the algebra has commutative concatenation, then the languages it recognizes are invariant under swapping siblings. Of course, the theory contains some non-trivial results.

A big open problem in the field is finding an effective characterization of first-order logic. One of the reasons why forest algebra was developed is the hope that there is some structural property of forest algebra – something like aperiodicity for monoids – that corresponds to first-order definability of a tree language.

---

<sup>\*</sup> Joint work with: Luc Segoufin, Howard Straubing and Igor Walukiewicz.



# Forcing and Type Theory

Thierry Coquand

Department of Computer Science and Engineering,  
Göteborg University and Chalmers University of Technology, Sweden  
coquand@chalmers.se

We present the technique of *forcing* from a constructive point of view. The connections with proof theory [1] and constructive mathematics are not so surprising given that Cohen’s original discovery of forcing was “motivated by an attempt to prove analysis consistent” and the idea that statements which seemed to involve infinities “could be reduced to pieces of finite informations” [9]. The interest of combining forcing and realizability was pointed out in the proof of Goodman’s Theorem [5].

We explain first how forcing allows to explain non effective objects (such as non principal ultrafilters, or well-ordering of the reals) in term of classical logic and dependent choices [7]. The combination of classical logic and dependent choices have a computational interpretation [3,4,6]. However, the associated computational behaviour is difficult to describe. We show then how to describe in type theory the addition of one Cohen real. We make explicit the computations involved in such an extension. One application, similar to the ones described in the reference [2], is the uniform continuity of functionals on Cantor space definable in type theory. By iterating this extension, we get a computational interpretation of universal quantification over Cantor space.

## References

1. Avigad, J.: Forcing in proof theory. *Bulletin of Symbolic Logic* 10(3), 305–333 (2004)
2. Beeson, M.: Foundations of constructive mathematics. *Metamathematical studies*. In: *Ergebnisse der Mathematik und ihrer Grenzgebiete (3) [Results in Mathematics and Related Areas (3)]*, vol. 6, Springer, Berlin (1985)
3. Berardi, S., Bezem, M., Coquand, T.: On the Computational Content of the Axiom of Choice. *J. Symb. Log.* 63(2), 600–622 (1998)
4. Berger, U., Oliva, P.: Modified bar recursion. *Mathematical Structures in Computer Science* 16(2), 163–183 (2006)
5. Goodman, N.: Relativized realizability in intuitionistic arithmetic of all finite types. *J. Symbolic Logic* 43(1), 23–44 (1978)
6. Krivine, J.L.: Dependent choice, ‘quote’ and the clock. *Th. Comp. Sc.* 308, 259–276 (2003)
7. Krivine, J.L.: Structures de réalisabilité, RAM et ultrafiltre sur  $\mathbb{N}$  (to appear, 2009)
8. Martin-Löf, P.: Constructive mathematics and computer programming. In: *Logic, methodology and philosophy of science, VI* (Hannover, 1979). *Stud. Logic Found. Math.*, vol. 104, pp. 153–175. North-Holland, Amsterdam (1982)
9. Platek, R.: Kreisel, Generalized Recursion Theory, Stanford and Me. In: Odifreddi, P. (ed.) *Kreiseliana, About and Around Georg Kreisel* (1996)

# Functional Interpretations of Intuitionistic Linear Logic

Gilda Ferreira and Paulo Oliva

School of Electronic Engineering and Computer Science  
Queen Mary, University of London

**Abstract.** We present three functional interpretations of intuitionistic linear logic and show how these correspond to well-known functional interpretations of intuitionistic logic via embeddings of  $IL^\omega$  into  $ILL^\omega$ . The main difference from previous work of the second author is that in intuitionistic linear logic the interpretations of  $!A$  are simpler (at the cost of an asymmetric interpretation of pure  $ILL^\omega$ ) and simultaneous quantifiers are no longer needed for the characterisation of the interpretations.

## 1 Introduction

This paper presents a family of functional interpretations of intuitionistic linear logic  $ILL^\omega$ , starting from a single functional interpretation of pure (exponential-free)  $ILL^\omega$ , followed by three possible interpretations of  $!A$ .

The second author [7,8,9,10] has recently shown how different functional interpretations of intuitionistic logic can be factored into a uniform family of interpretations of classical linear logic combined with Girard's standard embedding  $(\cdot)^*$  of intuitionistic logic into linear logic. In the symmetric context of *classical linear logic* each formula  $A$  is associated with a simultaneous one-move two-player game  $|A|_y^x$ . Intuitively, the two players, say Eloise and Abelard, must pick their moves  $x$  and  $y$  simultaneously and Eloise wins if and only if  $|A|_y^x$  holds. The symmetric nature of the game implies that (proof-theoretically) the formula  $A$  was interpreted as the formula

$$\exists_y^x |A|_y^x$$

where  $\exists_y^x A$  is a simple form of branching quantifier – termed simultaneous quantifier. Following this game-theoretic reading, the different interpretations of the modality  $!A$  are all of the following form: First, it (always) turns a symmetric game into an asymmetric one, where Eloise plays first, giving Abelard the advantage of playing second. In the symmetric context, this asymmetric game can be modelled by allowing Abelard to play a function  $f$  which calculates his move from a given Eloise move  $x$ . But also, the game  $!A$  gives a second (non-canonical) advantage to Abelard, by allowing him to play a *set of moves*, rather than a single move. The idea being that he wins the game  $!A$  if any move  $y \in f x$  is winning with respect to Eloise's move  $x$ , i.e.  $\neg |A|_y^x$ . Formally

$$|!A|_f^x \equiv \forall y \in f x |A|_y^x.$$

Therefore, the game  $!A$  always introduces a break of symmetric, but it leaves open *what kind of sets* Abelard is allowed to play. What the second author has shown is that if only

singleton sets are allowed the resulting interpretation corresponds to Gödel’s dialectica interpretation [149]; if finite sets are allowed then it corresponds to the Diller-Nahm variant of the dialectica interpretation [210]; and if these sets are actually the whole set of moves then it corresponds to Kreisel’s modified realizability interpretation [68].

In this paper we show that in the context of *intuitionistic linear logic* every formula can be interpreted as a game where Eloise plays first and Abelard plays second, being the branching quantifiers no longer needed. In other words, Abelard’s advantage of playing second, which was limited to the game  $!A$  in classical linear logic, is ubiquitous in intuitionistic linear logic. In this way, the game-theoretic interpretation of the modality  $!A$  is simply to lift the moves of Abelard from a single move to a set of moves. Formally,

$$!A|_a^x \equiv \forall y \in a |A|_y^x.$$

Hence, by working in the context of  $\text{ILL}^\omega$ , we can fully separate the canonical part of the interpretation (pure intuitionistic linear logic), where all interpretations coincide, and the non-canonical part where each choice of “sets of moves” gives rise to a different functional interpretation.

As we shall see, the functional interpretation of *pure intuitionistic linear logic* coincides with Gödel’s dialectica interpretation of *intuitionistic logic*, reading  $\multimap$ ,  $\otimes$  and  $\oplus$  as  $\rightarrow$ ,  $\wedge$  and  $\vee$ , respectively. This is so, because the dialectica interpretation identifies the games  $A$  and  $!A$ . The connection between Gödel’s dialectica interpretation and intuitionistic linear logic was first studied by de Paiva [11]. One can view our work here as a proof-theoretic reading of de Paiva’s category-theoretic work, together with an extension linking the “dialectica” interpretation of intuitionistic linear logic also with Kreisel’s modified realizability.

The main contributions of the paper are as follows: In Section 2 we present the basic interpretation of pure intuitionistic linear logic. In the same section we outline which principles are needed for the characterisation of the interpretation (Subsection 2.1). Section 3 describes three different interpretations of the modality  $!A$ . This is followed (Section 4) by a description of how each of these choices corresponds to the three best-known functional interpretations of intuitionistic logic.

## 1.1 Intuitionistic Linear Logic

Intuitionistic linear logic can be viewed as a fragment of Girard’s linear logic [3] which is sufficient for embedding intuitionistic logic into the linear context. We will make use of the formulation of intuitionistic linear logic shown in Tables 1 and 2. Our system is denoted by  $\text{ILL}^\omega$  since we work in the language of all finite types.

The finite types are inductively defined in the usual way:  $i$  is a finite type and if  $\rho$  and  $\sigma$  are finite types then  $\rho \rightarrow \sigma$  is a finite type. Our language has a constant of type  $i$  (to ensure that all types are inhabited by a closed term) and variables  $x^\rho$  for each finite type  $\rho$ . We assume that the terms of  $\text{ILL}^\omega$  contain all typed  $\lambda$ -terms, i.e. constants and variables are terms and if  $t^\sigma$  and  $s^{\rho \rightarrow \sigma}$  are terms then  $(\lambda x^\rho. t^\sigma)^{\rho \rightarrow \sigma}$  and  $(s^{\rho \rightarrow \sigma} t^\rho)^\sigma$  are also terms.

The atomic formulas of  $\text{ILL}^\omega$  are denoted by  $A_{\text{at}}$  (the linear logic constant  $0$  is an atomic formula) and if  $A$  and  $B$  are formulas, then  $A \otimes B$ ,  $A \& B$ ,  $A \oplus B$ ,  $A \multimap B$ ,  $!A$ ,  $\forall xA(x)$  and  $\exists xA(x)$  are also formulas. In this paper we will also work with a subsystem

**Table 1.** Intuitionistic Linear Logic (connectives)

$\frac{}{P \vdash P} \text{ (id)}$	$\frac{}{\Gamma, 0 \vdash A}$	
$\frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B} \text{ (cut)}$	$\frac{\Gamma \vdash A}{\pi\{\Gamma\} \vdash A} \text{ (per)}$	
$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B} \text{ (\otimes R)}$	$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C} \text{ (\otimes L)}$	
$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \text{ (\multimap R)}$	$\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C} \text{ (\multimap L)}$	
$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \& B} \text{ (\& R)}$	$\frac{\Gamma, A \vdash B}{\Gamma, A \& C \vdash B} \text{ (\& L)}$	$\frac{\Gamma, B \vdash C}{\Gamma, A \& B \vdash C} \text{ (\& L)}$
$\frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} \text{ (\oplus R)}$	$\frac{\Gamma \vdash B}{\Gamma \vdash A \oplus B} \text{ (\oplus R)}$	$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} \text{ (\oplus L)}$

of  $\text{ILL}^\omega$ , dubbed  $\text{ILL}_r^\omega$ , where a restriction is assumed on the  $\&\text{R}$ -rule: it is applied just with contexts of the form  $!\Gamma$ . In subsequent chapters we will see the necessity of this technical restriction. Note, however, that both systems  $\text{ILL}^\omega$  and  $\text{ILL}_r^\omega$  are strong enough to capture intuitionistic logic  $\text{IL}^\omega$  into the linear context, as precised in the following proposition.

**Proposition 1** ([3]). *Define two translations of  $\text{IL}^\omega$  into  $\text{ILL}^\omega$  inductively as follows:*

$$\begin{array}{ll}
 A_{\text{at}}^* & := A_{\text{at}} & A_{\text{at}}^\circ & := !A_{\text{at}}, \quad \text{if } A_{\text{at}} \neq \perp \\
 \perp^* & := 0 & \perp^\circ & := 0 \\
 (A \wedge B)^* & := A^* \& B^* & (A \wedge B)^\circ & := A^\circ \otimes B^\circ \\
 (A \vee B)^* & := !A^* \oplus !B^* & (A \vee B)^\circ & := A^\circ \oplus B^\circ \\
 (A \rightarrow B)^* & := !A^* \multimap B^* & (A \rightarrow B)^\circ & := !(A^\circ \multimap B^\circ) \\
 (\forall xA)^* & := \forall xA^* & (\forall xA)^\circ & := !\forall xA^\circ \\
 (\exists xA)^* & := \exists x!A^* & (\exists xA)^\circ & := \exists xA^\circ
 \end{array}$$

*If  $A$  is provable in  $\text{IL}^\omega$  then  $A^*$  and  $A^\circ$  are provable in  $\text{ILL}_r^\omega$  (and hence also in  $\text{ILL}^\omega$ ). Moreover, it is easy to check that  $A^\circ \multimap !A^*$ .*

**Proof.** It is already known that if  $\Gamma \vdash_{\text{IL}^\omega} A$  then  $!\Gamma^* \vdash_{\text{ILL}^\omega} A^*$ . The result with  $\text{ILL}^\omega$  replaced by  $\text{ILL}_r^\omega$  just require our attention in the rule  $\&\text{R}$ . The result for  $A^\circ$  follows immediately from the fact that in  $\text{ILL}_r^\omega$  we can prove  $A^\circ \multimap !A^*$ .  $\square$

**Table 2.** Intuitionistic Linear Logic (quantifiers and modality)

$\frac{\Gamma \vdash A}{\Gamma \vdash \forall x A} (\forall R)$	$\frac{\Gamma, A[t/x] \vdash B}{\Gamma, \forall x A \vdash B} (\forall L)$		
$\frac{\Gamma \vdash A[t/x]}{\Gamma \vdash \exists x A} (\exists R)$	$\frac{\Gamma, A \vdash B}{\Gamma, \exists x A \vdash B} (\exists L)$		
$\frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B} (\text{con})$	$\frac{\Gamma \vdash B}{\Gamma, !A \vdash B} (\text{wkn})$	$\frac{! \Gamma \vdash A}{! \Gamma \vdash !A} (!R)$	$\frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B} (!L)$

## 1.2 Verifying System

As we will show in the next sections, the three functional interpretations we present interpret the formula  $A \oplus B$  via a sort of flagged disjoint union, i.e. a boolean and a witness for either  $A$  or  $B$ . Therefore, in the verifying system, which we shall denote by  $\text{ILL}_b^\omega$ , we consider that the language also contains the booleans  $b$  as base type, with two boolean constants true and false (T, F), boolean variables, an equality relation  $=^b$  between two terms of boolean type and a constant of type  $b \rightarrow \rho \rightarrow \rho$  that should be seen as a conditional  $\lambda$ -term  $z(t, q)$  that reduces to  $t$  or  $q$  depending on whether  $z^b$  reduces to true or false.  $\text{ILL}_b^\omega$  is assumed to contain the following axioms for equality:

1.  $!(x =^b x)$
2.  $!(x =^b y) \multimap !(y =^b x)$
3.  $!(x =^b y) \otimes !(y =^b z) \multimap !(x =^b z)$
4.  $!(x =^b y) \otimes A[x/w] \multimap A[y/w]$ .

We would also like to ensure that true and false are distinct and that there are no other elements of boolean type

5.  $!(T =^b F) \multimap 0$
6.  $!(z =^b T) \oplus !(z =^b F)$ .

The axioms for the conditional  $\lambda$ -term are as follows

7.  $A[T(t, q)/w] \multimap A[t/w]$  and  $A[F(t, q)/w] \multimap A[q/w]$ .

For simplicity, we use the following abbreviation:

$$A \diamond_z B := (!(z =^b T) \multimap A) \& (!(z =^b F) \multimap B).$$

**Lemma 1.** *The following are derivable in  $\text{ILL}_b^\omega$*

$$(i) \frac{\vdash A[T] \quad \vdash A[F]}{\vdash A[z]}$$

$$(ii) !(T =^b T) \multimap A \vdash A \text{ and } !(F =^b F) \multimap A \vdash A$$

$$(iii) A \vdash !(T =^b F) \multimap B$$

$$(iv) A \diamond_T B \multimap A \text{ and } A \diamond_F B \multimap B$$

$$(v) !A \diamond_z !B \multimap !(A \diamond_z !B).$$

**Proof.** Assertion (i) can be derived from axioms 4. and 6.; (ii) follows easily from axiom 1.; (iii) can be deduced from axiom 5. and the forward implications in (iv) follow immediately from item (ii) and the inverse implications can easily be deduced using (iii). The direct implication in assertion (v) can be derived using assertions (i) and (iv), being the other implication trivial.  $\square$

## 2 A Basic Interpretation of Pure $\text{ILL}^\omega$

In this section we present a basic functional interpretation of pure (without the exponential !A) intuitionistic linear logic, and prove its soundness. In the next section we then consider different extensions of this interpretation to full intuitionistic linear logic,  $\text{ILL}^\omega$ .

**Definition 1 (Basic functional interpretation of pure  $\text{ILL}^\omega$ ).** *For each formula  $A$  of pure  $\text{ILL}^\omega$ , let us associate a new formula  $|A|_y^x$ , with two fresh sets of free-variables  $x$  and  $y$ , inductively as follows: For atomic formula  $A_{\text{at}}$  we let  $|A_{\text{at}}| := A_{\text{at}}$ . Assume the interpretations of  $A$  and  $B$  have already been defined as  $|A|_y^x$  and  $|B|_w^v$ , we then define*

$$|A \multimap B|_{x,w}^{f,g} := |A|_{fxw}^x \multimap |B|_w^{gx}$$

$$|A \otimes B|_{y,w}^{x,v} := |A|_y^x \otimes |B|_w^v$$

$$|A \& B|_{y,w,z}^{x,v} := |A|_y^x \diamond_z |B|_w^v$$

$$|A \oplus B|_{y,w}^{x,v,z} := |A|_y^x \diamond_z |B|_w^v$$

$$|\exists z A(z)|_y^{x,z} := |A(z)|_y^x$$

$$|\forall z A(z)|_{y,z}^f := |A(z)|_y^f.$$

Intuitively, the meaning of  $A$  is reduced to the existence of an object  $x$  such that  $\forall y |A|_y^x$ . The  $x$ 's are called *witnesses* and the  $y$ 's *challenges*. Note that, contrary to the interpretation of classical linear logic [710], the functional interpretation of intuitionistic linear logic is no longer symmetric. In terms of games, the interpretation above can be seen as associating to each formula  $A$  a one-move two-player *sequential* game  $|A|_y^x$ . In this game, Eloise starts by playing a move  $x$  followed by Abelard playing a move  $y$ . Eloise wins if  $|A|_y^x$  holds, otherwise Abelard wins.

Those familiar with the dialectica interpretation might find it puzzling that linear implication  $A \multimap B$  is interpreted above precisely as the intuitionistic implication in Gödel's dialectica interpretation, even though we claim this is the canonical part of the interpretation, which, depending on the interpretation of  $!A$ , can correspond to modified realizability as well. Again, in terms of games, this is explained by the fact that in the game  $A \multimap B$  Eloise plays first in the game  $B$ , but she plays second in the game  $A$ . In order to circumvent this discrepancy with our general rule that Eloise always plays first, we allow Eloise's move  $f$  in game  $A$  to depend on Abelard's move  $x$ . In this way, although she plays first in both games, it is as if she is playing second in game  $A$ , since her move is a function which might depend on Abelard's move.

**Theorem 1 (Soundness).** *Let  $A_0, \dots, A_n, B$  be formulas of pure  $\text{ILL}^\omega$ , with  $z$  as the only free-variables. If*

$$A_0(z), \dots, A_n(z) \vdash B(z)$$

*is provable in pure  $\text{ILL}^\omega$  then terms  $a_0, \dots, a_n, b$  can be extracted from this proof such that*

$$|A_0(z)|_{a_0}^{x_0}, \dots, |A_n(z)|_{a_n}^{x_n} \vdash |B(z)|_w^b$$

*is provable in  $\text{ILL}_\delta^\omega$ , where  $\text{FV}(a_i) \subseteq \{z, x_0, \dots, x_n, w\}$  and  $\text{FV}(b) \subseteq \{z, x_0, \dots, x_n\}$ .*

**Proof.** By induction on the derivation of  $A_0(z), \dots, A_n(z) \vdash B$ . The axioms are trivial since the interpretation does not change atomic formulas and every type is inhabited. The permutation rule is also immediate. Let us consider a few cases:

*Cut*

$$\frac{\frac{| \Gamma |_\gamma^u \vdash | A |_y^{a_0} \quad [a_1[a_0]]}{| \Gamma |_{\gamma'}^u \vdash | A |_{a_1[a_0]}^{a_0}} \quad [ \frac{a_0}{x} ]}{| \Gamma |_{\gamma'}^u, | \Delta |_{\delta'}^v \vdash | B |_w^{b'}} \quad (\text{cut})$$

where  $\gamma'$  and  $\delta', b'$  are obtained from  $\gamma$  and  $\delta, b$  via the substitutions  $[a_1[a_0]/y]$  and  $[a_0/x]$ , respectively.

*Tensor*

$$\frac{\frac{| \Gamma |_\gamma^u \vdash | A |_y^a \quad | \Delta |_\delta^v \vdash | B |_w^b}{| \Gamma |_\gamma^u, | \Delta |_\delta^v \vdash | A |_y^a \otimes | B |_w^b} \quad (\otimes R)}{| \Gamma |_\gamma^u, | \Delta |_\delta^v \vdash | A \otimes B |_{y,w}^{a,b}} \quad (\text{DII}) \quad \frac{| \Gamma |_\gamma^u, | A |_a^x, | B |_b^y \vdash | C |_w^c}{| \Gamma |_\gamma^u, | A |_a^x \otimes | B |_b^y \vdash | C |_w^c} \quad (\otimes L)}{| \Gamma |_\gamma^u, | A \otimes B |_{a,b}^{x,y} \vdash | C |_w^c} \quad (\text{DII})$$

*Linear implication - left introduction*

$$\frac{\frac{| \Gamma |_\gamma^u \vdash | A |_y^a}{| \Gamma |_\gamma^u [fa(b[ga])] \vdash | A |_{fa(b[ga])}^a} \quad [ \frac{fa(b[ga])}{y} ] \quad \frac{| \Delta |_{\delta[v]}^w, | B |_{b[v]}^v \vdash | C |_z^{c[v]}}{| \Delta |_{\delta[ga]}^w, | B |_{b[ga]}^{ga} \vdash | C |_z^{c[ga]}} \quad [ \frac{ga}{v} ]}{| \Delta |_{\delta[ga]}^w, | B |_{b[ga]}^{ga} \vdash | C |_z^{c[ga]}} \quad (\multimap L)}{| \Gamma |_\gamma^u [fa(b[ga])], | \Delta |_{\delta[ga]}^w, | A |_{fa(b[ga])}^a \multimap | B |_{b[ga]}^{ga} \vdash | C |_z^{c[ga]}} \quad (\text{DIII})$$

Universal quantifier

$$\frac{\frac{\Gamma|_{\gamma[z]}^u \vdash |A(z)|_y^{a[z]}}{\Gamma|_{\gamma[z]}^u \vdash |A(z)|_y^{(\lambda z.a[z])z}}}{\Gamma|_{\gamma[z]}^u \vdash |\forall z A(z)|_{y,z}^{a[z]}} \text{ (DII)} \quad \frac{\frac{\Gamma|_{\gamma[x]}^u, |A(t)|_{a[x]}^x \vdash |B|_w^{b[x]}}{\Gamma|_{\gamma[f t]}^u, |A(t)|_{a[f t]}^{f t} \vdash |B|_w^{b[f t]}} \left[ \frac{f t}{x} \right]}{\Gamma|_{\gamma[f t]}^u, |\forall z A(z)|_{a[f t], t}^f \vdash |B|_w^{b[f t]}} \text{ (DII)}$$

Existential quantifier

$$\frac{\Gamma|_{\gamma}^u \vdash |A(t)|_y^a}{\Gamma|_{\gamma}^u \vdash |\exists z A(z)|_y^{a,t}} \text{ (DII)} \quad \frac{\Gamma|_{\gamma[z]}^u, |A(z)|_{a[z]}^x \vdash |B|_y^{b[z]}}{\Gamma|_{\gamma[z]}^u, |\exists z A(z)|_{a[z]}^{x,z} \vdash |B|_y^{b[z]}} \text{ (DII)}$$

With - right introduction

$$\frac{\frac{\Gamma|_{\gamma_0}^u \vdash |A|_y^a}{\Gamma|_{T(\gamma_0, \gamma_1)}^u \vdash |A|_y^a \diamond_T |B|_w^b} \text{ (Ax. 7/ LII(iv))} \quad \frac{\Gamma|_{\gamma_1}^u \vdash |B|_w^b}{\Gamma|_{F(\gamma_0, \gamma_1)}^u \vdash |A|_y^a \diamond_F |B|_w^b}}{\frac{\Gamma|_{z(\gamma_0, \gamma_1)}^u \vdash |A|_y^a \diamond_z |B|_w^b}{\Gamma|_{z(\gamma_0, \gamma_1)}^u \vdash |A \& B|_{y,w,z}^{a,b}} \text{ (DII)}} \text{ (LII(i))}$$

With - left introduction and Plus - right introduction

$$\frac{\Gamma|_{\gamma}^u, |A|_a^x \vdash |B|_w^b}{\Gamma|_{\gamma}^u, |A|_a^x \diamond_T |C|_c^v \vdash |B|_w^b} \text{ (LII(iv))} \quad \frac{\Gamma|_{\gamma}^u \vdash |A|_y^a}{\Gamma|_{\gamma}^u \vdash |A|_y^a \diamond_T |B|_w^b} \text{ (LII(iv))}$$

$$\frac{\Gamma|_{\gamma}^u, |A \& C|_{a,c,T}^{x,v} \vdash |B|_w^b}{\Gamma|_{\gamma}^u \vdash |A \oplus B|_{y,w}^{a,b,T}} \text{ (DII)} \quad \frac{\Gamma|_{\gamma}^u \vdash |A|_y^a}{\Gamma|_{\gamma}^u \vdash |A \oplus B|_{y,w}^{a,b,T}} \text{ (DII)}$$

The other &-L and  $\oplus$ -R are similar.

Plus - left introduction

$$\frac{\Gamma|_{\gamma_0}^u, |A|_a^x \vdash |C|_w^{c_1}}{\Gamma|_{T(\gamma_0, \gamma_1)}^u, |A|_a^x \diamond_T |B|_b^v \vdash |C|_w^{T(c_1, c_2)}} \quad \frac{\Gamma|_{\gamma_1}^u, |B|_b^v \vdash |C|_w^{c_2}}{\Gamma|_{F(\gamma_0, \gamma_1)}^u, |A|_a^x \diamond_F |B|_b^v \vdash |C|_w^{F(c_1, c_2)}} \text{ (Ax. 7/ LII(iv))}$$

$$\frac{\Gamma|_{z(\gamma_0, \gamma_1)}^u, |A|_a^x \diamond_z |B|_b^v \vdash |C|_w^{z(c_1, c_2)}}{\Gamma|_{z(\gamma_0, \gamma_1)}^u, |A \oplus B|_{a,b}^{x,v,z} \vdash |C|_w^{z(c_1, c_2)}} \text{ (DII)} \text{ (LII(i))}$$

The other cases are treated similarly.  $\square$

## 2.1 Characterisation

As described in the introduction, one of the main advantages of working in the context of intuitionistic linear logic is that we no longer need (non-standard) branching quantifiers. The asymmetry introduced in ILL<sup>ω</sup> turns the symmetric games of classical linear logic into games where Eloise always plays first, so formulas  $A$  are interpreted as  $\exists x \forall y |A|_y^x$ .

**Proposition 2.** *The following principles characterise the basic interpretation presented above*



$$\begin{aligned}
AC_I & : \forall x \exists y A_V(y) \multimap \exists f \forall x A_V(f.x) \\
MP_I & : (\forall x A_{qf} \multimap B_{qf}) \multimap \exists x (A_{qf} \multimap B_{qf}) \\
IP_I & : (A_V \multimap \exists y B_V) \multimap \exists y (A_V \multimap B_V) \\
EP & : \forall x, v (A_{qf} \otimes B_{qf}) \multimap (\forall x A_{qf} \otimes \forall v B_{qf})
\end{aligned}$$

where  $A_{qf}$ ,  $B_{qf}$  and  $A_V$ ,  $B_V$  are quantifier-free formulas and purely universal formulas of  $ILL_b^\omega$  respectively. Formally,

$$ILL_b^\omega + AC_I + MP_I + IP_I + EP \vdash A \multimap \exists x \forall y |A|_y^x.$$

**Proof.** By induction on the logical structure of  $A$ . Let us consider a few cases:

*Tensor.*

$$\begin{aligned}
A \otimes B & \stackrel{(IH)}{\multimap} \exists x \forall y |A|_y^x \otimes \exists v \forall w |B|_w^v \\
& \stackrel{(EP)}{\multimap} \exists x, v \forall y, w (|A|_y^x \otimes |B|_w^v) \\
& \equiv \exists x, v \forall y, w |A \otimes B|_{y,w}^{x,v}.
\end{aligned}$$

*With.*

$$\begin{aligned}
A \& B & \stackrel{(IH)}{\multimap} \exists x \forall y |A|_y^x \& \exists v \forall w |B|_w^v \\
& \multimap \forall z (\exists x \forall y |A|_y^x \diamond_z \exists v \forall w |B|_w^v) \\
& \multimap \forall z \exists x, v (\forall y |A|_y^x \diamond_z \forall w |B|_w^v) \\
& \multimap \forall z \exists x, v \forall y, w (|A|_y^x \diamond_z |B|_w^v) \\
& \stackrel{(AC_I)}{\multimap} \exists f, g \forall z, y, w (|A|_y^{fz} \diamond_z |B|_w^{gz}) \\
& \multimap \exists x, v \forall z, y, w (|A|_y^x \diamond_z |B|_w^v) \\
& \equiv \exists x, v \forall y, w, z |A \& B|_{y,w,z}^{x,v}.
\end{aligned}$$

*Linear implication.*

$$\begin{aligned}
A \multimap B & \stackrel{(IH)}{\multimap} \exists x \forall y |A|_y^x \multimap \exists v \forall w |B|_w^v \stackrel{(IP_I, MP_I)}{\multimap} \forall x \exists v \forall w \exists y (|A|_y^x \multimap |B|_w^v) \\
& \stackrel{(AC_I)}{\multimap} \exists f, g \forall x, w (|A|_{fxw}^x \multimap |B|_w^{gx}) \equiv \exists f, g \forall x, w |A \multimap B|_{x,w}^{f,g}.
\end{aligned}$$

*Universal quantifier.*

$$\forall z A \stackrel{(IH)}{\multimap} \forall z \exists x \forall y |A|_y^x \stackrel{(AC_I)}{\multimap} \exists f \forall y, z |A|_y^{fz} \equiv \exists f \forall y, z | \forall z A |_{y,z}^f.$$

The other cases are treated similarly. In fact, for the remaining cases (once the induction hypothesis is assumed) the equivalence can be proved in  $ILL_b^\omega$  alone.  $\square$

*Remark 1.* Note that if we are embedding  $ll^\omega$  via the standard embedding  $(\cdot)^*$  then the connective  $A \otimes B$  is not needed, and hence the extra principle EP is not needed either.

### 3 Some Interpretations of $\text{ILL}^\omega$

In this section we consider a few choices of how the basic interpretation given in Definition [1](#) can be extended to full intuitionistic linear logic, i.e. we give some alternative interpretations of the modality  $!A$ . All choices considered will have the form:

$$!A|_y^x := !\forall y' \sqsubset y |A|_{y'}^x \quad (1)$$

for some notion of bounded quantified formula  $\forall y' \sqsubset y A$ . In fact, very little structure is required in order to obtain a sound interpretation.

**Proposition 3.** *Given a formula  $A[y]$ , assume the formula  $\forall y \sqsubset a A$  is such that for some terms [1](#)  $\eta(\cdot)$ ,  $(\cdot) \otimes (\cdot)$  and  $(\cdot) \circ (\cdot)$  the following are provable in  $\text{ILL}_b^\omega$*

$$(A1) \quad !\forall y \sqsubset \eta(z) A[y] \multimap A[z]$$

$$(A2) \quad !\forall y \sqsubset y_1 \otimes y_2 A[y] \multimap !\forall y \sqsubset y_1 A[y] \otimes !\forall y \sqsubset y_2 A[y]$$

$$(A3) \quad !\forall y \sqsubset f \circ z A[y] \multimap !\forall x \sqsubset z !\forall y \sqsubset f x A[y].$$

The interpretation of  $!A$  as above leads to a sound functional interpretation of  $\text{ILL}^\omega$ .

**Proof.** By Theorem [1](#) we just have to analyse the rules of contraction, weakening,  $!$ -right introduction and  $!$ -left introduction.

*Contraction*

$$\frac{\frac{\frac{|G|_\gamma^u, !A|_{a_0}^{x_0}, !A|_{a_1}^{x_1} \vdash |B|_w^b}{|G|_\gamma^u, !A|_{a_0}^x, !A|_{a_1}^x \vdash |B|_w^b} \left[ \frac{x}{x_0}, \frac{x}{x_1} \right]}{|G|_\gamma^u, !\forall y' \sqsubset a_0 |A|_{y'}^x, !\forall y' \sqsubset a_1 |A|_{y'}^x \vdash |B|_w^b} \text{ (I)}}{|G|_\gamma^u, !\forall y' \sqsubset a_0 |A|_{y'}^x \otimes !\forall y' \sqsubset a_1 |A|_{y'}^x \vdash |B|_w^b} \text{ (}\otimes\text{L)}} \text{ (A2)}}{|G|_\gamma^u, !\forall y' \sqsubset a_0 \otimes a_1 |A|_{y'}^x \vdash |B|_w^b} \text{ (I)}}{|G|_\gamma^u, !A|_{a_0 \otimes a_1}^x \vdash |B|_w^b} \text{ (I)}$$

*Weakening*

$$\frac{\frac{|G|_\gamma^u \vdash |B|_w^b}{|G|_\gamma^u, !\forall y' \sqsubset a |A|_{y'}^x \vdash |B|_w^b} \text{ (wkn)}}{|G|_\gamma^u, !A|_a^x \vdash |B|_w^b} \text{ (I)}$$

Note that every type is inhabited by a closed term.

<sup>1</sup> Note that these terms are allowed to be specific to the formula  $A$ , in particular, the free variables of  $\eta(\cdot)$ ,  $(\cdot) \otimes (\cdot)$  and  $(\cdot) \circ (\cdot)$  are assumed to be contained in the free-variables of  $\forall y A[y]$  (i.e. all free-variables of  $A$  except  $y$ ).

*Bang - right introduction*

$$\frac{\frac{\frac{!|\Gamma|_{\gamma[y']}^u \vdash |A|_y^a}{!|\forall w' \sqsubset \gamma[y']| \Gamma|_{w'}^u \vdash |A|_{y'}^a}{!|\forall y' \sqsubset y \quad !|\forall w' \sqsubset (\lambda y'. \gamma[y']) y' | \Gamma|_{w'}^u \vdash !|\forall y' \sqsubset y | A|_{y'}^a}{!|\forall w' \sqsubset (\lambda y'. \gamma[y']) \circ y | \Gamma|_{w'}^u \vdash !|\forall y' \sqsubset y | A|_{y'}^a}{!|\Gamma|_{(\lambda y'. \gamma[y']) \circ y}^u \vdash !|\Lambda|_y^a}}{\quad} \text{(A3)}$$

*Bang - left introduction*

$$\frac{\frac{|\Gamma|_{\gamma}^u, |A|_a^x \vdash |B|_w^b}{|\Gamma|_{\gamma}^u, !|\forall y \sqsubset \eta(a) | A|_y^x \vdash |B|_w^b}}{|\Gamma|_{\gamma}^u, !|\Lambda|_{\eta(a)}^x \vdash |B|_w^b}}{\quad} \text{(A1)}$$

That concludes the proof.  $\square$

*Remark 2.* Assume that the types of  $y^\rho$  and  $\mathbf{a}^{T\rho}$  in  $\forall y \sqsubset \mathbf{a} | A|_y^x$  are as shown, for a fixed  $A$ . Then, our three families of terms have types

$$\begin{aligned} \eta &: \rho \rightarrow T\rho \\ \otimes &: T\rho \times T\rho \rightarrow T\rho \\ \circ &: (\tau \rightarrow T\rho) \times T\tau \rightarrow T\rho. \end{aligned}$$

In category theory, one could think of  $(T, \eta, \circ)$  as forming a Kleisli triple ( $\sim$  monad), with  $\otimes$  being a commutative monoid on  $T\rho$ . This in turn extends to a comonad on formulas as

$$T(A[y]) := !(\forall y \sqsubset \mathbf{a} A)[\mathbf{a}].$$

See e.g. the work of Valeria de Paiva [12] and Martin Hyland ([5], section 3.1) on categorical logic for more information about the connection between functional interpretations and comonads.

**Proposition 4.** *The following are three sound interpretations of  $!A$ :*

- (a)  $!|A|_y^x := !|\forall y | A|_y^x$
- (b)  $!|A|_a^x := !|\forall y \in \mathbf{a} | A|_y^x$
- (c)  $!|A|_y^x := !|A|_y^x.$

**Proof.** (a) This interpretation of  $!A$  corresponds to the choice  $\forall y \sqsubset \mathbf{t} A[y] := \forall y A[y]$ . It is easy to check that conditions (A1), (A2) and (A3) become

$$\begin{aligned} !|\forall y A[y] \multimap A[z] \\ !|\forall y A[y] \multimap !|\forall y A[y] \otimes !|\forall y A[y] \\ !|\forall y A[y] \multimap !|\forall x !|\forall y A[y] \end{aligned}$$

respectively, which are trivially derivable in  $\text{ILL}_b^\omega$ .

(b) Consider that the language of  $\text{ILL}_b^\omega$  has a new finite type  $\sigma^*$  for each finite type  $\sigma$ . An element of type  $\sigma^*$  is a finite set of elements of type  $\sigma$ . The extended language has a relation symbol  $\in$  infixing between a term of type  $\sigma$  and a term of type  $\sigma^*$  with axioms to ensure that  $!(x \in y)$  if and only if  $x$  is an element in the set  $y$ . Consider also the existence of three more constants of types  $\sigma \rightarrow \sigma^*$ ,  $\sigma^* \rightarrow \sigma^* \rightarrow \sigma^*$  and  $\sigma^* \rightarrow (\sigma \rightarrow \rho^*) \rightarrow \rho^*$  that should be seen as terms such that  $\eta(t)$  is the singleton set with  $t^\sigma$  as the only element (in particular  $!(t \in \eta(t))$ ),  $t \otimes q$  is the union of two finite sets  $t$  and  $q$ , and  $f \circ q$  is the set that results from the union of all sets  $fx$  with  $x \in q$ . The interpretation  $!|A|_a^x := !\forall y \in a |A|_y^x$  corresponds to the choice  $\forall y \sqsubset t A[y] := \forall y \in t A[y]$ , which is an abbreviation for  $\forall y(! (y \in t) \rightarrow A[y])$ . In this context, the conditions (A1), (A2) and (A3) become

$$\begin{aligned} & !\forall y \in \eta(z) A[y] \rightarrow A[z] \\ & !\forall y \in y_1 \otimes y_2 A[y] \rightarrow !\forall y \in y_1 A[y] \otimes !\forall y \in y_2 A[y] \\ & !\forall y \in f \circ z A[y] \rightarrow !\forall x \in z !\forall y \in f x A[y], \end{aligned}$$

which are provable in the extension of  $\text{ILL}_b^\omega$  outlined above.

(c) This interpretation of  $!A$  corresponds to the choice  $\forall y \sqsubset t A[y] := A[t/y]$ . Given a formula  $A[y]$  we define  $\eta(\cdot)$ , as being the identity,  $\circ$  is defined as  $f \circ x := fx$  and  $y_1 \otimes y_2$  is

$$y_1 \otimes y_2 := \begin{cases} y_1 & \text{if } !A[y_1] \rightarrow 0 \\ y_2 & \text{if } !A[y_1]. \end{cases}$$

We are assuming that  $\text{ILL}_b^\omega$  has also an extra axiom (asserting the decidability of  $A$ )  $\vdash !A \oplus (!A \rightarrow 0)$ . Conditions (A1), (A2) and (A3) become

$$\begin{aligned} & !A[\eta(z)] \rightarrow A[z] \\ & !A[y_1 \otimes y_2] \rightarrow !A[y_1] \otimes !A[y_2] \\ & !A[f \circ z] \rightarrow !!A[fz] \end{aligned}$$

respectively. (A1) and (A3) are trivially derivable. In the derivation of (A2) use

$$\begin{aligned} & \vdash !A \oplus (!A \rightarrow 0) \\ & !A[y_1], !A[y_1 \otimes y_2] \vdash !A[y_1] \otimes !A[y_2], \text{ and} \\ & !A[y_1] \rightarrow 0, !A[y_1 \otimes y_2] \vdash 0. \end{aligned}$$

□

## 4 Relation to Standard Interpretations of $\text{IL}^\omega$

We argued in the introduction (see Proposition [11](#)) that for the purpose of analysing  $\text{IL}^\omega$  via linear logic it suffices to work with the system  $\text{ILL}_r^\omega$ . As it turns out, in  $\text{ILL}_r^\omega$ , we can simplify our definition of functional interpretation as follows:

**Proposition 5.** *When interpreting the subsystem  $\text{ILL}_r^\omega$ , the interpretation of  $A$  &  $B$  presented in Definition [1](#) can be simplified so that the parametrised interpretation*

$$\begin{aligned}
|A \multimap B|_{x,w}^{f,g} &:= |A|_{f,xw}^x \multimap |B|_w^{gx} \\
|A \otimes B|_{y,w}^{x,v} &:= |A|_y^x \otimes |B|_w^v \\
|A \& B|_{y,w}^{x,v} &:= |A|_y^x \& |B|_w^v \\
|A \oplus B|_{y,w}^{x,v,z} &:= |A|_y^x \diamond_z |B|_w^v \\
|\exists z A(z)|_{y,z}^{x,z} &:= |A(z)|_y^x \\
|\forall z A(z)|_{y,z}^f &:= |A(z)|_y^{fz} \\
|A|_y^x &:= !\forall y' \sqsubset y |A|_{y'}^x
\end{aligned}$$

is sound for  $\text{ILL}_r^\omega$ , assuming (A1), (A2), and (A3) are satisfied.

**Proof.** We just have to analyse the rules for  $\&$  having in mind that, in the case of the system under interpretation, the  $\&$ -right introduction is restricted of the form  $! \Gamma$ . The simplified interpretation of  $A \& B$  is shown sound as:

$$\frac{\frac{! \Gamma|_{\gamma_0}^a \vdash |A|_x^a \quad (\text{P5})}{!\forall y' \sqsubset \gamma_0 | \Gamma|_{y'}^a \vdash |A|_x^a} \quad (\text{A2}) \quad \frac{! \Gamma|_{\gamma_1}^b \vdash |B|_y^b \quad (\text{P5})}{!\forall y' \sqsubset \gamma_1 | \Gamma|_{y'}^b \vdash |B|_y^b} \quad (\text{A2})}{!\forall y' \sqsubset \gamma_0 \otimes \gamma_1 | \Gamma|_{y'}^a \vdash |A|_x^a \quad !\forall y' \sqsubset \gamma_0 \otimes \gamma_1 | \Gamma|_{y'}^b \vdash |B|_y^b} \quad (\&R)}{\frac{!\forall y' \sqsubset \gamma_0 \otimes \gamma_1 | \Gamma|_{y'}^a \vdash |A|_x^a \& |B|_y^b \quad (\text{P5})}{! \Gamma|_{\gamma_0 \otimes \gamma_1}^a \vdash |A \& B|_{x,y}^{a,b}}}$$

And for the left introduction:

$$\frac{\frac{|\Gamma|_{\gamma}^a, |A|_a^x \vdash |C|_w^c \quad (\&L)}{|\Gamma|_{\gamma}^a, |A|_a^x \& |B|_b^v \vdash |C|_w^c} \quad (\text{P5})}{|\Gamma|_{\gamma}^a, |A \& B|_{a,b}^{x,v} \vdash |C|_w^c}$$

The other  $\&$ -left introduction is similar.  $\square$

Since in the remaining part of this section we work with translations of intuitionistic logic into linear logic, by  $|A|_y^x$  we refer to the (simplified) parametrised interpretation described in Proposition 5. Next we prove that the three different ways of interpreting  $!A$  (cf. Proposition 4) give rise to interpretations of  $\text{ILL}_r^\omega$  that correspond (via the translations of intuitionistic logic into intuitionistic linear logic) to Kreisel's modified realizability, the Diller-Nahm interpretation, and Gödel's dialectica interpretation, as:

$ A _a^x$	Interpretation of $\text{IL}^\omega$
$!\forall y  A _y^x$	Kreisel's modified realizability
$!\forall y \in \mathbf{a}  A _y^x$	Diller-Nahm interpretation
$! A _a^x$	Gödel's dialectica interpretation.

But first a consideration concerning translation  $(\cdot)^*$  of  $\text{IL}^\omega$  into  $\text{ILL}_r^\omega$ , which we will use in the treatment of the Diller-Nahm and the dialectica interpretations (for modified realizability we use the translation  $(\cdot)^\circ$ ).

**Proposition 6.** *Consider the following simplification of Girard's translations  $(\cdot)^*$  (cf. Proposition [7](#))*

$$A_{\text{at}}^+ \quad := A_{\text{at}}, \quad \text{if } A_{\text{at}} \neq \perp$$

$$\perp^+ \quad := 0$$

$$(A \wedge B)^+ \quad := A^+ \& B^+$$

$$(A \vee B)^+ \quad := A^+ \oplus B^+$$

$$(A \rightarrow B)^+ \quad := !A^+ \multimap B^+$$

$$(\forall xA)^+ \quad := \forall xA^+$$

$$(\exists xA)^+ \quad := \exists xA^+.$$

If  $A$  is provable in  $\text{IL}^\omega$  then  $A^+$  is provable in  $\text{ILL}_r^\omega + P_\oplus + P_\exists$ , where

$$P_\oplus \quad : \quad !(A \oplus B) \multimap !A \oplus !B$$

$$P_\exists \quad : \quad !\exists xA \multimap \exists x!A.$$

**Proof.** First we show that given the principles  $P_\oplus$  and  $P_\exists$ , we have  $!A^* \multimap !A^+$ . The proof is done by induction on the complexity of the formula  $A$ . Conjunction, implication and universal quantification follow easily by induction hypothesis using that  $\text{ILL}_r^\omega$  proves:

$$!(A \& B) \multimap !A \otimes !B$$

$$!(!A \multimap B) \multimap !(!A \multimap !B)$$

$$!\forall xA \quad \multimap \quad !\forall x!A$$

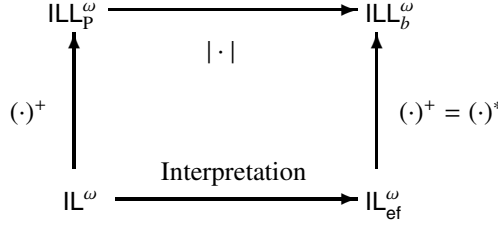
respectively. Disjunction and existential quantification are studied below:

$$!(A \vee B)^* \equiv !(!A^* \oplus !B^*) \multimap !A^* \oplus !B^*$$

$$\stackrel{(\text{IH})}{\multimap} !A^+ \oplus !B^+ \stackrel{(P_\oplus)}{\multimap} !(A^+ \oplus B^+) \equiv !(A \vee B)^+$$

and  $!(\exists xA)^* \equiv !\exists x!A^* \multimap \exists x!A^* \stackrel{(\text{IH})}{\multimap} \exists x!A^+ \stackrel{(P_\exists)}{\multimap} !\exists xA^+ \equiv !(\exists xA)^+$ . Applying Proposition [8](#), we know that from  $\text{IL}^\omega \vdash A$  we have  $\text{ILL}_r^\omega \vdash A^*$ . So,  $\text{ILL}_r^\omega \vdash !A^*$  and hence  $\text{ILL}_r^\omega + P_\oplus + P_\exists \vdash !A^*$ . Using the equivalence proved before we have  $\text{ILL}_r^\omega + P_\oplus + P_\exists \vdash !A^+$ . In particular, we conclude  $\text{ILL}_r^\omega + P_\oplus + P_\exists \vdash A^+$ .  $\square$

The reason for assuming the principles  $P_\oplus$  and  $P_\exists$  is that they are validated by the interpretations we shall consider. As such, we can make use of these to simplify the embeddings of intuitionistic logic into (this extension of) linear logic, since the interpretation of linear logic will interpret these principles taking us back to standard linear logic, as suggested by the following diagram:



In the diagram,  $\text{ILL}_p^\omega$  abbreviates  $\text{ILL}_r^\omega + \text{P}_\oplus + \text{P}_\exists$ . Note that all our interpretations transform proofs in  $\text{IL}^\omega$  into existential-free proofs, i.e. proofs in  $\text{IL}_{\text{ef}}^\omega$ , where the two translations  $(\cdot)^*$  and  $(\cdot)^+$  coincide.

#### 4.1 Modified Realizability

Kreisel's modified realizability associates to each formula  $A$  of intuitionistic logic a new formula  $x \text{ mr } A$  (see [13] for the formal definition). We are going to prove that this form of realizability once translated to the linear logic context via the  $(\cdot)^\circ$  translation corresponds (according to the theorem bellow) to the interpretation of  $\text{ILL}_r^\omega$  with  $!A|^x := !\forall y|A|^y$ . First an auxiliary result:

**Lemma 2.**  $|A^\circ|^x \multimap !|A^\circ|^x$ .

**Proof.** Note that, because of the way we interpret  $!A$ , it can be checked by induction on  $A$  that the interpretation of  $A^\circ$  has an empty tuple of challenge variables, i.e. we obtain a formula of the form  $|A^\circ|^x$ . To verify the lemma, it is enough to prove that  $|A^\circ|^x \multimap !A'$ , for some formula  $A'$ , since assuming this we have  $!|A^\circ|^x \multimap !!A' \multimap !A' \multimap |A^\circ|^x$ . The proof is done by induction on the complexity of the formula  $A$ . We just sketch the cases of conjunction and disjunction, being the other cases immediate.

$$|(A \wedge B)^\circ|^x,y \equiv |A^\circ \otimes B^\circ|^x,y \equiv |A^\circ|^x \otimes |B^\circ|^y \stackrel{(\text{IH})}{\multimap} !A' \otimes !B' \multimap !(A' \& B').$$

$$|(A \vee B)^\circ|^x,y,z \equiv |A^\circ \oplus B^\circ|^x,y,z \equiv |A^\circ|^x \diamond_z |B^\circ|^y \stackrel{(\text{IH})}{\multimap} !A' \diamond_z !B' \stackrel{(\text{IH} \vee)}{\multimap} !(A' \diamond_z !B'). \quad \square$$

**Theorem 2.**  $|A^\circ|^x \multimap (x \text{ mr } A)^\circ$ .

**Proof.** The proof is done by induction on the complexity of the formula  $A$ . If  $A$  is an atomic formula, the result is trivial. Consider the case of conjunction:

$$|(A \wedge B)^\circ|^x,y \equiv |A^\circ \otimes B^\circ|^x,y \equiv |A^\circ|^x \otimes |B^\circ|^y$$

$$\stackrel{(\text{IH})}{\multimap} (x \text{ mr } A)^\circ \otimes (y \text{ mr } B)^\circ \equiv (x \text{ mr } A \wedge y \text{ mr } B)^\circ \equiv (x, y \text{ mr } A \wedge B)^\circ.$$

The universal and existential quantifications also follow immediately from the way we define the translation and the interpretations, applying the induction hypothesis. Implication is treated as

$$|(A \rightarrow B)^\circ|^g \equiv !|(A^\circ \multimap B^\circ)|^g \equiv !\forall x|A^\circ \multimap B^\circ|^g_x \equiv !\forall x(|A^\circ|^x \multimap |B^\circ|^g_x)$$

$$\stackrel{(\text{IH})}{\multimap} !\forall x((x \text{ mr } A)^\circ \multimap (gx \text{ mr } B)^\circ) \multimap !\forall x!((x \text{ mr } A)^\circ \multimap (gx \text{ mr } B)^\circ)$$

$$\equiv (\forall x(x \text{ mr } A \rightarrow gx \text{ mr } B))^\circ \equiv (g \text{ mr } (A \rightarrow B))^\circ.$$

whereas disjunction uses the auxiliary result above:

$$\begin{aligned}
|(A \vee B)^\circ|^{x,y,z} &\stackrel{(L2)}{\circ\text{-}\circ} |!(A \vee B)^\circ|^{x,y,z} \equiv !|A^\circ \oplus B^\circ|^{x,y,z} \equiv !( |A^\circ|^{x,y,z} \diamond_z |B^\circ|^{y,z} ) \\
&\stackrel{(IH)}{\circ\text{-}\circ} !( (!(z = T) \multimap (x \text{ mr } A)^\circ) \& ( !(z = F) \multimap (y \text{ mr } B)^\circ) ) \\
&\circ\text{-}\circ !( (!(z = T) \multimap (x \text{ mr } A)^\circ) \otimes ( !(z = F) \multimap (y \text{ mr } B)^\circ) ) \\
&\equiv ((z = T \rightarrow x \text{ mr } A) \wedge (z = F \rightarrow y \text{ mr } B))^\circ \\
&\equiv (x, y, z \text{ mr } A \vee B)^\circ.
\end{aligned}$$

That concludes the proof.  $\square$

## 4.2 Gödel's Dialectica Interpretation

Recall that Gödel's dialectica interpretation associates to each formula  $A$  a quantifier-free formula  $A_D(\mathbf{x}; \mathbf{y})$  inductively, such that  $A$  is interpreted as  $\exists \mathbf{x} \forall \mathbf{y} A_D(\mathbf{x}; \mathbf{y})$  (see [1], section 2.3). The next result shows the correspondence between the dialectica interpretation and the  $\text{ILL}_r^\omega$  interpretation with  $!|A|_y^x := !|A|_y^x$ , via the simplified embedding  $(\cdot)^+$  (cf. Proposition 6).

**Theorem 3.**  $|A^+|_y^x \circ\text{-}\circ (A_D(\mathbf{x}; \mathbf{y}))^+$ .

**Proof.** The proof is easily done by induction on the complexity of the formula  $A$ . Again the atomic formulas are checked trivially and the other formulas follow immediately by induction hypothesis using the definitions of the  $(\cdot)^+$ -translation and the interpretations. We illustrate with two cases: Conjunction

$$\begin{aligned}
|(A \wedge B)^+|_{y,w}^{x,v} &\equiv |A^+ \& B^+|_{y,w}^{x,v} \equiv |A^+|_y^x \& |B^+|_w^v \stackrel{(IH)}{\circ\text{-}\circ} (A_D(\mathbf{x}; \mathbf{y}))^+ \& (B_D(\mathbf{v}; \mathbf{w}))^+ \\
&\equiv (A_D(\mathbf{x}; \mathbf{y}) \wedge B_D(\mathbf{v}; \mathbf{w}))^+ \equiv ((A \wedge B)_D(\mathbf{x}, \mathbf{v}; \mathbf{y}, \mathbf{w}))^+.
\end{aligned}$$

and disjunction:

$$\begin{aligned}
|(A \vee B)^+|_{y,w}^{x,v,z} &\equiv |A^+ \oplus B^+|_{y,w}^{x,v,z} \equiv |A^+|_y^x \diamond_z |B^+|_w^v \\
&\equiv ( !(z = T) \multimap |A^+|_y^x ) \& ( !(z = F) \multimap |B^+|_w^v ) \\
&\stackrel{(IH)}{\circ\text{-}\circ} ( !(z = T) \multimap (A_D(\mathbf{x}; \mathbf{y}))^+ ) \& ( !(z = F) \multimap (B_D(\mathbf{v}; \mathbf{w}))^+ ) \\
&\equiv (z = T \rightarrow A_D(\mathbf{x}; \mathbf{y}))^+ \& (z = F \rightarrow B_D(\mathbf{v}; \mathbf{w}))^+ \\
&\equiv ((z = T \rightarrow A_D(\mathbf{x}; \mathbf{y})) \wedge (z = F \rightarrow B_D(\mathbf{v}; \mathbf{w})))^+ \\
&\equiv ((A \vee B)_D(\mathbf{x}, \mathbf{v}, z; \mathbf{y}, \mathbf{w}))^+.
\end{aligned}$$

The other cases are treated similarly.  $\square$

Note that although  $(\cdot)^+$  translates formulas from  $\text{IL}^\omega$  into  $\text{ILL}_r^\omega + \text{P}_\oplus + \text{P}_\exists$ , since these two principles are interpretable the verifying system is still  $\text{ILL}_b^\omega$ . Let us argue that  $\text{P}_\oplus$  and  $\text{P}_\exists$  are interpretable, by showing that the interpretation of premise implies that of the conclusion (hence the identity and projection functions can be taken as realisers for the implication). It can be proved that



$$\forall x \sqsubset a(A(x) \& B) \multimap (\forall x \sqsubset a A(x) \& B) \text{ and}$$

$$\forall x \sqsubset a(B \multimap A(x)) \multimap (B \multimap \forall x \sqsubset a A(x))$$

when the variable  $x$  does not occur free in  $B$ . Also,  $!(A \diamond_b B) \multimap !A \diamond_b !B$ . So,

$$\begin{aligned} |!(A \oplus B)|_{a,c}^{x,v,b} &\equiv !\forall y \sqsubset a \forall w \sqsubset c (|A|_y^x \diamond_b |B|_w^v) \\ &\multimap !(\forall y \sqsubset a |A|_y^x \diamond_b \forall w \sqsubset c |B|_w^v) \\ &\multimap !\forall y \sqsubset a |A|_y^x \diamond_b !\forall w \sqsubset c |B|_w^v \equiv !|A \oplus B|_{a,c}^{x,v,b}. \end{aligned}$$

Similarly,  $!|\exists z A|_a^{x,z} \equiv !\forall y \sqsubset a |\exists z A|_y^{x,z} \equiv !\forall y \sqsubset a |A|_y^x \equiv !|A|_a^x \equiv !|\exists z A|_a^{x,z}$ .

### 4.3 Diller-Nahm Interpretation

The Diller-Nahm interpretation differs from Gödel's dialectica interpretation since it allows finite sets to witness the negative content of an implication. Formally, the Diller-Nahm interpretation is defined inductively as

$$\begin{aligned} (A_{\text{at}})_{dn}(\cdot) &:\equiv A_{\text{at}} \\ (A \wedge B)_{dn}(\mathbf{x}, \mathbf{v}; \mathbf{y}, \mathbf{w}) &:\equiv A_{dn}(\mathbf{x}; \mathbf{y}) \wedge B_{dn}(\mathbf{v}; \mathbf{w}) \\ (A \vee B)_{dn}(\mathbf{x}, \mathbf{v}, \mathbf{z}; \mathbf{y}, \mathbf{w}) &:\equiv (z = \text{T} \rightarrow A_{dn}(\mathbf{x}; \mathbf{y})) \wedge (z = \text{F} \rightarrow B_{dn}(\mathbf{v}; \mathbf{w})) \\ (A \rightarrow B)_{dn}(\mathbf{f}, \mathbf{g}; \mathbf{x}, \mathbf{w}) &:\equiv \forall y \in \mathbf{f} \mathbf{x} \mathbf{w} A_{dn}(\mathbf{x}; \mathbf{y}) \rightarrow B_{dn}(\mathbf{g} \mathbf{x}; \mathbf{w}) \\ (\forall z A)_{dn}(\mathbf{f}; \mathbf{y}, \mathbf{z}) &:\equiv A_{dn}(\mathbf{f} \mathbf{z}; \mathbf{y}) \\ (\exists z A)_{dn}(\mathbf{x}, \mathbf{z}; \mathbf{y}) &:\equiv A_{dn}(\mathbf{x}; \mathbf{y}). \end{aligned}$$

Next we show that the Diller-Nahm interpretation of  $\text{IL}^\omega$  corresponds to the interpretation of  $\text{ILL}^\omega$  with  $!|A|_a^x := !\forall y \in a |A|_y^x$ .

**Theorem 4.**  $|A^+|_y^x \circ\multimap (A_{dn}(\mathbf{x}; \mathbf{y}))^+$ .

**Proof.** The proof, by induction on the structure of  $A$ , is almost entirely similar to the one concerning Gödel's interpretation. The only case which needs attention is that of implication, which we analyse below.

$$\begin{aligned} |(A \rightarrow B)^+|_{\mathbf{x}, \mathbf{w}}^{\mathbf{f}, \mathbf{g}} &\equiv !|A^+ \multimap B^+|_{\mathbf{x}, \mathbf{w}}^{\mathbf{f}, \mathbf{g}} \equiv !|A^+|_{\mathbf{f} \mathbf{x} \mathbf{w}}^x \multimap |B^+|_{\mathbf{w}}^{\mathbf{g} \mathbf{x}} \\ &\equiv !\forall y \in \mathbf{f} \mathbf{x} \mathbf{w} |A^+|_y^x \multimap |B^+|_{\mathbf{w}}^{\mathbf{g} \mathbf{x}} \\ &\stackrel{(\text{IH})}{\circ\multimap} !\forall y \in \mathbf{f} \mathbf{x} \mathbf{w} (A_{dn}(\mathbf{x}; \mathbf{y}))^+ \multimap (B_{dn}(\mathbf{g} \mathbf{x}; \mathbf{w}))^+ \\ &\equiv !(\forall y \in \mathbf{f} \mathbf{x} \mathbf{w} A_{dn}(\mathbf{x}; \mathbf{y}))^+ \multimap (B_{dn}(\mathbf{g} \mathbf{x}; \mathbf{w}))^+ \\ &\equiv (\forall y \in \mathbf{f} \mathbf{x} \mathbf{w} A_{dn}(\mathbf{x}; \mathbf{y}) \rightarrow B_{dn}(\mathbf{g} \mathbf{x}; \mathbf{w}))^+ \\ &\equiv ((A \rightarrow B)_{dn}(\mathbf{f}, \mathbf{g}; \mathbf{x}, \mathbf{w}))^+. \end{aligned}$$

Note that the  $(\cdot)^+$  translation of  $\forall y \in a A$  is  $\forall y \in a A^+$ , as we can see below:

$$\begin{aligned} (\forall y \in a A)^+ &\equiv (\forall y (y \in a \rightarrow A))^+ \\ &\equiv \forall y (! (y \in a)^+ \multimap A^+) \equiv \forall y (! (y \in a) \multimap A^+) \equiv \forall y \in a A^+. \end{aligned}$$

That concludes the proof.  $\square$

**Acknowledgements.** The first author would like to thank FCT (grant SFRH/BPD/34527/2006) and CMAF, whereas the second author gratefully acknowledges support of the Royal Society (grant 516002.K501/RH/kk).

## References

1. Avigad, J., Feferman, S.: Gödel's functional ("Dialectica") interpretation. In: Buss, S.R. (ed.) Handbook of proof theory. Studies in Logic and the Foundations of Mathematics, vol. 137, pp. 337–405. North Holland, Amsterdam (1998)
2. Diller, J., Nahm, W.: Eine Variant zur Dialectica-interpretation der Heyting Arithmetik endlicher Typen. Arch. Math. Logik Grundlagenforsch 16, 49–66 (1974)
3. Girard, J.-Y.: Linear logic. Theoretical Computer Science 50(1), 1–102 (1987)
4. Gödel, K.: Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. Dialectica 12, 280–287 (1958)
5. Hyland, J.M.E.: Proof theory in the abstract. Annals of Pure and Applied Logic 114, 43–78 (2002)
6. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. In: Heyting, A. (ed.) Constructivity in Mathematics, pp. 101–128. North Holland, Amsterdam (1959)
7. Oliva, P.: Computational interpretations of classical linear logic. In: Leivant, D., de Queiroz, R. (eds.) WoLLIC 2007. LNCS, vol. 4576, pp. 285–296. Springer, Heidelberg (2007)
8. Oliva, P.: Modified realizability interpretation of classical linear logic. In: Proc. of the Twenty Second Annual IEEE Symposium on Logic in Computer Science LICS 2007. IEEE Press, Los Alamitos (2007)
9. Oliva, P.: An analysis of Gödel's dialectica interpretation via linear logic. Dialectica 62(2), 269–290 (2008)
10. Oliva, P.: Functional interpretations of linear and intuitionistic logic. Information and Computation (to appear, 2009)
11. de Paiva, V.C.V.: The Dialectica categories. In: Gray, J.W., Scedrov, A. (eds.) Proc. of Categories in Computer Science and Logic, Boulder, CO, 1987. Contemporary Mathematics, vol. 92, pp. 47–62. American Mathematical Society, Providence (1989)
12. de Paiva, V.C.V.: A Dialectica-like model of linear logic. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) Category Theory and Computer Science. LNCS, vol. 389, pp. 341–356. Springer, Heidelberg (1989)
13. Troelstra, A.S.: Metamathematical Investigation of Intuitionistic Arithmetic and Analysis. Lecture Notes in Mathematics, vol. 344. Springer, Berlin (1973)

# Fixed-Point Definability and Polynomial Time

Martin Grohe

Humboldt-Universität zu Berlin, Germany

**Abstract.** My talk will be a survey of recent results about the quest for a logic capturing polynomial time.

In a fundamental study of database query languages, Chandra and Harel [4] first raised the question of whether there exists a logic that captures polynomial time. Actually, Chandra and Harel phrased the question in a somewhat disguised form; the version that we use today goes back to Gurevich [15]. Briefly, but slightly imprecisely,<sup>1</sup> a logic  $L$  captures a complexity class  $K$  if exactly those properties of finite structures that are decidable in  $K$  are definable in  $L$ . The existence of a logic capturing PTIME is still wide open, and it is viewed as one of the main open problems in finite model theory and database theory. One reason the question is interesting is that we know from Fagin's Theorem [9] that existential second-order logic captures NP, and we also know that there are logics capturing most natural complexity classes above NP. Gurevich conjectured that there is no logic capturing PTIME. If this conjecture was true, this would not only imply that  $\text{PTIME} \neq \text{NP}$ , but it would also show that NP and the complexity classes above NP have a fundamentally different structure than the class PTIME and presumably most natural complexity classes below PTIME. (This aspect is highlighted by a result due to Dawar [6], also see [13].)

On the positive side, Immerman [18] and Vardi [23] proved that least fixed-point logic FP captures polynomial time on the class of all ordered finite structures. Here we say that a logic  $L$  captures a complexity class  $K$  on a class  $\mathcal{C}$  of finite structures if exactly those properties of structures in  $\mathcal{C}$  decidable in  $K$  are definable in  $L$ . It is easy to prove that FP does not capture PTIME on the class of all finite structures. Immerman [19] proposed the extension  $\text{FP} + \text{C}$  of fixed-point logic by *counting operators* as a candidate for a logic capturing PTIME. It is not easy to prove, but true nevertheless, that  $\text{FP} + \text{C}$  does not capture PTIME. This was shown by Cai, Fürer, and Immerman in 1992 [3].

## Fixed-Point Definability on Graphs with Excluded Minors

Even though the logic  $\text{FP} + \text{C}$  does not capture PTIME on the class of all finite structures, it does capture PTIME on many natural classes of structures. Immerman and Lander [20] proved that  $\text{FP} + \text{C}$  captures PTIME on the class of all trees. In 1998, I proved that  $\text{FP} + \text{C}$  captures PTIME on the class of all planar

---

<sup>1</sup> For a precise definition of a logic capturing PTIME, I refer the reader to Grädel's excellent survey [10] on descriptive complexity theory.

graphs [11] and around the same time, Julian Mariño and I proved that  $\text{FP} + \text{C}$  captures PTIME on all classes of structures of bounded tree width [14]. In [12], I proved the same result for the class of all  $K_5$ -free graphs, that is the class of all graphs that have no complete graph on five vertices as a minor. A *minor* of graph  $G$  is a graph  $H$  that can be obtained from a subgraph of  $G$  by contracting edges. By (the easy direction of) Kuratowski's Theorem, the class of all  $K_5$ -free graphs contains all planar graphs. We say that a class  $\mathcal{C}$  of graphs *excludes a minor* if there is a graph  $H$  that is not a minor of any graph in  $\mathcal{C}$ . Very recently, I proved the following theorem, which generalises all these previous results, because all classes of graphs appearing in these results exclude minors.

**Theorem<sup>2</sup>**  $\text{FP} + \text{C}$  captures PTIME on all classes of graphs that exclude a minor.

The main part of my talk will be devoted to this theorem.

### Stronger Logics

While  $\text{FP} + \text{C}$  captures PTIME on many interesting classes of structures, it has been known for almost twenty years that it does not capture PTIME. So what about stronger logics? Currently, the two main candidates for logics capturing PTIME are *choiceless polynomial time with counting*  $\text{CP} + \text{C}$  and *fixed-point logic with a rank operator*  $\text{FP} + \text{R}$ . The logic  $\text{CP} + \text{C}$  was introduced ten years ago by Blass, Gurevich and Shelah [1] (also see [2,8]). The formal definition of the logic is carried out in the framework of *abstract state machines*. Intuitively  $\text{CP} + \text{C}$  may be viewed as a version of  $\text{FP} + \text{C}$  where quantification and fixed-point operators not only range over elements of a structure, but instead over all objects that can be described by  $O(\log n)$  bits, where  $n$  is the size of the structure. This intuition can be formalised in an expansion of a structure by all hereditarily finite sets which use the elements of the structure as atoms. The logic  $\text{FP} + \text{R}$ , introduced recently in [7], is an extension of  $\text{FP}$  by an operator that determines the rank of definable matrices in a structure. This may be viewed as a higher dimensional version of a counting operator. (Counting appears as a special case of diagonal  $\{0, 1\}$ -matrices.)

Both  $\text{CP} + \text{C}$  and  $\text{FP} + \text{R}$  are known to be strictly more expressive than  $\text{FP} + \text{C}$ . Indeed, both logics can express the property used by Cai, Fürer, and Immerman to separate  $\text{FP} + \text{C}$  from PTIME. For both logics it is open whether they capture polynomial time, and it is also open whether one of them semantically contains the other.

### Is There a Logic Capturing PTIME ?

Let me close this note by a few thoughts on the question of whether there exists a logic capturing PTIME. As I said in the first paragraph, the question is still wide

<sup>2</sup> As this result has not been published yet, and not even a complete readable manuscript exists, the skeptic reader may treat this as a conjecture rather than a theorem.

open. By this, I do not only mean that we are far from having a proof settling the question in either direction, but actually that I do not see any evidence pointing towards one or the other answer (despite Gurevich's conjecture). If anything, I mildly lean towards believing that there is a logic for PTIME. It is known that there is a connection between the question of whether there exists a logic capturing PTIME and a variant of the graph isomorphism problem: If there is a polynomial time graph canonisation algorithm, then there is a logic capturing PTIME. But this does not really help, because the question for polynomial time graph isomorphism and canonisation algorithms is open just the same, and even if there is no polynomial time canonisation algorithm, this does not mean that there is no logic for PTIME. To gain a better understanding of the relation between the two problems, it would be interesting to see a small complexity class like uniform  $AC^0$ , which provably does not admit graph canonisation, can be captured by a logic.

## References

1. Blass, A., Gurevich, Y., Shelah, S.: Choiceless polynomial time. *Annals of Pure and Applied Logic* 100, 141–187 (1999)
2. Blass, A., Gurevich, Y., Shelah, S.: On polynomial time computation over unordered structures. *Journal of Symbolic Logic* 67, 1093–1125 (2002)
3. Cai, J., Fürer, M., Immerman, N.: An optimal lower bound on the number of variables for graph identification. *Combinatorica* 12, 389–410 (1992)
4. Chandra, A., Harel, D.: Structure and complexity of relational queries. *Journal of Computer and System Sciences* 25, 99–128 (1982)
5. Chen, Y., Flum, J.: A logic for PTIME and a parameterized halting problem. In: *Proceedings of the 24th IEEE Symposium on Logic in Computer Science* (2009)
6. Dawar, A.: Generalized quantifiers and logical reducibilities. *Journal of Logic and Computation* 5, 213–226 (1995)
7. Dawar, A., Grohe, M., Holm, B., Laubner, B.: Logics with rank operators. In: *Proceedings of the 24th IEEE Symposium on Logic in Computer Science* (2009)
8. Dawar, A., Richerby, D., Rossman, B.: Choiceless polynomial time, counting and the Cai-Fürer-Immerman graphs (Extended abstract). *Electronic Notes on Theoretical Computer Science* 143, 13–26 (2006)
9. Fagin, R.: Generalized first-order spectra and polynomial-time recognizable sets. In: Karp, R.M. (ed.) *Complexity of Computation*, SIAM-AMS Proceedings, vol. 7, pp. 43–73 (1974)
10. Grädel, E.: Finite Model Theory and Descriptive Complexity. In: Kolaitis, P.G., Libkin, L., Marx, M., Spencer, J., Vardi, M.Y., Venema, Y., Weinstein, S. (eds.) *Finite Model Theory and Its Applications*, ch. 3, pp. 125–230. Springer, Heidelberg (2007)
11. Grohe, M.: Fixed-point logics on planar graphs. In: *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, pp. 6–15 (1998)
12. Grohe, M.: Definable tree decompositions. In: *Proceedings of the 23rd IEEE Symposium on Logic in Computer Science*, pp. 406–417 (2008)
13. Grohe, M., Ebbinghaus, H.-D.: Zur Struktur dessen, was wirklich berechenbar ist. *Philosophia Naturalis* 36, 91–116 (1999)

14. Grohe, M., Mariño, J.: Definability and descriptive complexity on databases of bounded tree-width. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 70–82. Springer, Heidelberg (1998)
15. Gurevich, Y.: Logic and the challenge of computer science. In: Börger, E. (ed.) Current trends in theoretical computer science, pp. 1–57. Computer Science Press, Rockville (1988)
16. Hella, L.: Definability hierarchies of generalized quantifiers. *Annals of Pure and Applied Logic* 43, 235–271 (1989)
17. Hella, L., Kolaitis, P.G., Luosto, K.: Almost everywhere equivalence of logics in finite model theory. *Bulletin of Symbolic Logic* 2, 422–443 (1996)
18. Immerman, N.: Upper and lower bounds for first-order expressibility. *Journal of Computer and System Sciences* 25, 76–98 (1982)
19. Immerman, N.: Expressibility as a complexity measure: results and directions. In: Proceedings of the 2nd IEEE Symposium on Structure in Complexity Theory, pp. 194–202 (1987)
20. Immerman, N., Lander, E.: Describing graphs: A first-order approach to graph canonization. In: Selman, A. (ed.) Complexity theory retrospective, pp. 59–81. Springer, Heidelberg (1990)
21. Nash, A., Remmel, J.B., Vianu, V.: PTIME queries revisited. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 274–288. Springer, Heidelberg (2004)
22. Otto, M.: Bisimulation-invariant PTIME and higher-dimensional  $\mu$ -calculus. *Theoretical Computer Science* 224, 237–265 (1999)
23. Vardi, M.Y.: The complexity of relational query languages. In: Proceedings of the 14th ACM Symposium on Theory of Computing, pp. 137–146 (1982)

# Kleene's Amazing Second Recursion Theorem\*

## (Extended Abstract)

Yiannis N. Moschovakis

Department of Mathematics  
 University of California, Los Angeles, CA, USA  
 and Department of Mathematics, University of Athens, Greece  
 ynm@math.ucla.edu

This little gem is stated unbilled and proved (completely) in the last two lines of §2 of the short note [Kleene \(1938\)](#). In modern notation, with all the hypotheses stated explicitly and in a strong form, it reads as follows:

**Theorem 1 (SRT).** *Fix a set  $\mathbb{V} \subseteq \mathbb{N}$ , and suppose that for each natural number  $n \in \mathbb{N} = \{0, 1, 2, \dots\}$ ,  $\varphi^n : \mathbb{N}^{n+1} \rightarrow \mathbb{V}$  is a recursive partial function of  $(n + 1)$  arguments with values in  $\mathbb{V}$  so that **the standard assumptions** (1) and (2) hold with*

$$\{e\}(\vec{x}) = \varphi_e^n(\vec{x}) = \varphi^n(e, \vec{x}) \quad (\vec{x} = (x_1, \dots, x_n) \in \mathbb{N}^n).$$

(1) *Every  $n$ -ary recursive partial function with values in  $\mathbb{V}$  is  $\varphi_e^n$  for some  $e$ .*

(2) *For all  $m, n$ , there is a recursive (total) function  $S = S_n^m : \mathbb{N}^{m+1} \rightarrow \mathbb{N}$  such that*

$$\{S(e, \vec{y})\}(\vec{x}) = \{e\}(\vec{y}, \vec{x}) \quad (e \in \mathbb{N}, \vec{y} \in \mathbb{N}^m, \vec{x} \in \mathbb{N}^n).$$

*Then, for every recursive, partial function  $f(e, \vec{y}, \vec{x})$  of  $(1 + m + n)$  arguments with values in  $\mathbb{V}$ , there is a total recursive function  $\tilde{z}(\vec{y})$  of  $m$  arguments such that*

$$\{\tilde{z}(\vec{y})\}(\vec{x}) = f(\tilde{z}(\vec{y}), \vec{y}, \vec{x}) \quad (\vec{y} \in \mathbb{N}^m, \vec{x} \in \mathbb{N}^n). \tag{1}$$

*Proof.* Fix  $e$  so that  $\{e\}(m, \vec{y}, \vec{x}) = f(S(m, m, \vec{y}), \vec{y}, \vec{x})$  and let  $\tilde{z}(\vec{y}) = S(e, e, \vec{y})$ .<sup>†</sup>

Kleene states the theorem with  $\mathbb{V} = \mathbb{N}$ , relative to specific  $\varphi^n, S_n^m$ , supplied by his *Enumeration Theorem*,  $m = 0$  (no parameters  $\vec{y}$ ) and  $n \geq 1$ , i.e., not allowing nullary partial functions. And most of the time, this is all we need; but there are a few important applications where choosing “the right”  $\varphi^n, S_n^m$ , restricting the values to a proper  $\mathbb{V} \subsetneq \mathbb{N}$  or allowing  $m > 0$  or  $n = 0$  simplifies the proof considerably. With  $\mathbb{V} = \{0\}$  (singleton 0) and  $m = n = 0$ , for example, the characteristic equation

$$\{\tilde{z}\}() = f(\tilde{z}) \tag{2}$$

is a rather “pure” form of self-reference, where the number  $\tilde{z}$  produced by the proof (as a code of a nullary semirecursive relation) has the *property*  $f(\tilde{z})$ , at least when  $f(\tilde{z}) \downarrow$ .

---

\* Part of an expository article in preparation, written to commemorate the passage of 100 years since the birth of Stephen Cole Kleene.

Kleene uses the theorem in the very next page to prove that there is a largest initial segment of the countable ordinals which can be given “constructive notations”, in the first application of what we now call *effective grounded* (transfinite) *recursion*, one of the most useful methods of proof justified by SRT; but there are many others, touching most parts of logic and even classical analysis.

My aim in this lecture is to list, discuss, explain and in a couple of simple cases prove some of the most significant applications of the Second Recursion Theorem, in a kind of “retrospective exhibition” of the work that it has done since 1938. It is quite impressive, actually, the power of such a simple fact with a one-line proof; but part of its wide applicability stems precisely from this simplicity, which make it easy to formulate and establish it in many contexts outside ordinary recursion theory on  $\mathbb{N}$ . Some of the more important applications come up in *Effective Descriptive Set Theory*, where the relevant version of SRT is obtained by replacing  $\mathbb{N}$  by the Baire space  $\mathcal{N} = (\mathbb{N} \rightarrow \mathbb{N})$  and applying recursion theory on  $\mathcal{N}$ —also developed by Kleene.

Speaking rather loosely, the identity (II) expresses a *self-referential* property of  $\tilde{z}(\vec{y})$  and SRT is often applied to justify powerful, self-referential definitions. I will discuss some of these first, and then I will turn to applications of effective grounded recursion.

The lecture will focus on some of the following consequences of SRT (I)

**A. Self reproducing Turing machines.** It is quite simple to show using SRT that there is a Turing machine which prints its code when it is started on the blank tape. It takes just a little more work—and a careful choice of  $\varphi^n, S_n^m$ —to specify a Turing machine (naturally and literally) by a string of symbols in its own alphabet and then show

**Theorem 2.** *On every alphabet  $\Sigma$  with  $N \geq 3$  symbols, there is a Turing machine  $M$  which started on the blank tape outputs itself.*

**B. Myhill's characterization of r.e. complete sets.** Recall that a recursively enumerable (r.e.) set  $A \subseteq \mathbb{N}$  is *complete* if for each r.e. set  $B$  there is a recursive (total) function  $f$  such that  $x \in B \iff f(x) \in A$ .<sup>2</sup> An r.e. set  $A$  is *creative* if there is a unary recursive partial function  $u(e)$  such that

$$W_e \cap A = \emptyset \implies u(e) \downarrow \ \& \ u(e) \notin (A \cup W_e). \quad (3)$$

The notion goes back to Post (1944) who showed (in effect) that every r.e.-complete set is creative and implicitly asked for the converse.

<sup>1</sup> The choice of these examples was dictated by what I know and what I like, but also by the natural limitations of space in an extended abstract and time in a lecture. A more complete list would surely include examples from *Recursion in higher types* and *Realizability theory*. (For the latter, see Moschovakis (2010) which is in some ways a companion article to this.)

<sup>2</sup> And then one can find a one-to-one  $f$  with the same property, cf. Theorem VII in Rogers (1967).



**Theorem 3** (Myhill (1955)). *Every creative set is r.e.-complete.*

This clever argument of Myhill's has many applications, but it is also foundationally significant: it identifies *creativity*, which is an intrinsic property of a set  $A$  but depends on the coding of recursive partial functions with *completeness*, which depends on the entire class of r.e. sets but is independent of coding. I believe it is the first important application of SRT in print by someone other than Kleene.

**C. The Myhill-Shepherdson Theorem.** One (modern) interpretation of this classical result is that algorithms which call their (computable, partial) function arguments *by name* can be simulated by non-deterministic algorithms which call their function arguments *by value*. It is a rather simple but interesting consequence of SRT.

Let  $\mathcal{P}_r^1$  be the set of all unary recursive partial functions. A partial operation

$$\Phi : \mathbb{N}^n \times \mathcal{P}_r^1 \rightarrow \mathbb{N} \quad (4)$$

is *effective* if its partial function *associate*

$$f(\vec{x}, e) = \Phi(\vec{x}, \varphi_e^1) \quad (5)$$

is recursive. In programming terms, an effective operation calls its function argument *by name*, i.e., it needs a code of any  $p \in \mathcal{P}_r^1$  to compute the value  $\Phi(\vec{x}, p)$ .

There are cases, however, when we need to compute  $\Phi(\vec{x}, p)$  without access to a code of  $p$ , only to its values. In programming terms again, this comes up when  $p$  is computed by some other program which is not known, but which can be asked to produce any values  $p(\vec{y})$  that are required during the (otherwise effective) computation of  $\Phi(\vec{x}, p)$ . We can make this precise using a (deterministic or non-deterministic) *Turing machine  $M$  with an oracle* which can request values of the function argument  $p$  on a special *function input tape*: when  $M$  needs  $p(y)$ , it prints  $y$  on the function input tape and waits until it is replaced by  $p(y)$  before it can go on—which, in fact, may cause the computation to stall if  $p(y) \uparrow$ . In these circumstances we say that  $M$  *computes  $\Phi$  by value*.

Notice that the recursive associate  $f$  of an effective operation  $\Phi$  as in (5) satisfies the following *invariance condition*:

$$\varphi_{e_1} = \varphi_{e_2} \implies f(\vec{x}, e_1) = f(\vec{x}, e_2). \quad (6)$$

This is used crucially in the proof of the next theorem, which involves two clever applications of SRT:

**Theorem 4** (Myhill and Shepherdson (1955)). *A partial operation  $\Phi$  as in (4) is effective if and only if it is computable by a non-deterministic Turing machine<sup>3</sup>*

<sup>3</sup> The use of non-deterministic machines here is essential, because the operation

$$\Phi(p) = \begin{cases} 1, & \text{if } p(0) \downarrow \text{ or } p(1) \downarrow, \\ \perp, & \text{otherwise} \end{cases}$$

is effective but not computable by a deterministic Turing machine.

**D. The Kreisel-Lacombe-Shoenfield-Ceitin Theorem.** Let  $\mathcal{F}_r^1$  be the set of all unary total recursive functions. By analogy with operations on  $\mathcal{P}_r^1$ , a partial operation

$$\Phi : \mathbb{N}^n \times \mathcal{F}_r^1 \rightarrow \mathbb{N}$$

is *effective* if there is a recursive partial function  $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  such that

$$\varphi_e \in \mathcal{F}_r^1 \implies \Phi(\vec{x}, \varphi_e) = f(\vec{x}, e). \quad (7)$$

Notice that such a *recursive associate*  $f$  of  $\Phi$  satisfies the *invariance condition*

$$\varphi_e = \varphi_m \in \mathcal{F}_r^1 \implies f(\vec{x}, e) = f(\vec{x}, m), \quad (8)$$

but is not uniquely determined.

The next theorem is a version of the Myhill-Shepherdson Theorem appropriate for these operations. Its proof is not quite so easy, and it involves applying SRT in the middle of a relatively complex construction:

**Theorem 5** (Kreisel, Lacombe, and Shoenfield (1957), Ceitin (1962)). *Suppose  $\Phi : \mathbb{N}^n \times \mathcal{F}_r^1 \rightarrow \mathbb{N}$  is a total effective operation.*

(1)  $\Phi$  is *effectively continuous*: i.e., there is a recursive partial function  $g(e, \vec{x})$ , such that when  $\varphi_e$  is total, then  $g(e, \vec{x}) \downarrow$  for all  $\vec{x}$ , and for all  $p \in \mathcal{F}_r^1$ ,

$$(\forall t < g(e, \vec{x})) [p(t) = \varphi_e(t)] \implies \Phi(\vec{x}, p) = \Phi(\vec{x}, \varphi_e).$$

(2)  $\Phi$  is computed by a deterministic Turing machine.

The restriction in the theorem to total operations on  $\mathcal{F}_r^1$  is necessary, because of a lovely counterexample in Friedberg (1958).

Ceitin (1962) proved independently a general version of (1) in this theorem: *every recursive operator on one constructive metric space to another is effectively continuous*. His result is the central fact in the school of *constructive analysis* which was flowering in Russia at that time, and it has played an important role in the development of constructive mathematics ever since.

**E. Incompleteness and undecidability using SRT.** We formulate in this section two basic theorems which relate SRT to incompleteness and undecidability results: a version of the so-called Fixed Point Lemma, and a beautiful result of Myhill's, which implies most simple incompleteness and undecidability facts about sufficiently strong theories and insures a very wide applicability for the Fixed Point Lemma.

Working in the language of Peano Arithmetic (PA) with symbols  $0, 1, +, \cdot$ , define first (recursively) for each  $x \in \mathbb{N}$  a closed term  $\Delta x$  which denotes  $x$ , and for every formula  $\chi$ , let

$$\# \chi = \text{the code (Gödel number) of } \chi, \quad \ulcorner \chi \urcorner \equiv \Delta \# \chi = \text{the name of } \chi.$$

We assume that the Gödel numbering of formulas is sufficiently effective so that (for example)  $\#\varphi(\Delta x_1, \dots, \Delta x_n)$  can be computed from  $\#\varphi(v_1, \dots, v_n)$  and  $x_1, \dots, x_n$ . A *theory* (in the language of PA) is any set of sentences  $T$  and

$$\text{Th}(T) = \{\#\theta \mid \theta \text{ is a sentence and } T \vdash \theta\}$$

is the set of (Gödel numbers of) the theorems of  $T$ . A theory  $T$  is *sound* if every  $\theta \in T$  is true in the standard model  $(\mathbb{N}, 0, 1, +, \cdot)$ ; it is *axiomatizable* if its *proof relation*

$$\text{Proof}_T(e, y) \iff e \text{ is the code of a sentence } \sigma \\ \text{and } y \text{ is the code of a proof of } \sigma \text{ in } T$$

is recursive, which implies that  $\text{Th}(T)$  is recursively enumerable; and it is *sufficiently expressive* if every recursive relation  $R(\vec{x})$  is *numeralwise expressible* in  $T$ , i.e., for some formula  $\varphi_R(v_1, \dots, v_n)$ ,

$$R(x_1, \dots, x_n) \implies T \vdash \varphi_R(\Delta x_1, \dots, \Delta x_n), \\ \neg R(x_1, \dots, x_n) \implies T \vdash \neg \varphi_R(\Delta x_1, \dots, \Delta x_n).$$

**Theorem 6 (Fixed Point Lemma).** *If  $T$  is axiomatizable in the language of PA and  $\text{Th}(T)$  is r.e.-complete, then for every formula  $\theta(v)$  with at most  $v$  free, there is a sentence  $\sigma$  such that*

$$T \vdash \sigma \iff T \vdash \theta(\ulcorner \sigma \urcorner). \tag{9}$$

*Proof.* Let  $\psi^0, \psi^1, \dots$  be recursive partial functions satisfying the standard assumptions, let  $\square$

$$u \in A \iff (\exists n)[\text{Seq}(u) \ \& \ \text{lh}(u) = n + 1 \ \& \ \psi^n((u)_0, (u)_1, \dots, (u)_n) \downarrow],$$

so that  $A$  is r.e., and so there is a total recursive function  $\mathfrak{r}$  such that

$$u \in A \iff \mathfrak{r}(u) \in \text{Th}(T).$$

It follows that for every  $n$ -ary semirecursive relation  $R(\vec{x})$ , there is a number  $e$  such that

$$R(\vec{x}) \iff \psi^n(e, \vec{x}) \downarrow \iff \langle e, \vec{x} \rangle \in A \iff \mathfrak{r}(\langle e, \vec{x} \rangle) \in \text{Th}(T). \tag{10}$$

We will use SRT with  $\mathbb{V} = \{\{0\}\}$  and

$$\varphi^n(e, \vec{x}) = 0 \iff \mathfrak{r}(\langle e, \vec{x} \rangle) \in \text{Th}(T),$$

which (because of  $\square$ ), easily satisfy the standard assumptions.

---

<sup>4</sup> For any tuple  $\vec{x} = (x_0, \dots, x_{n-1}) \in \mathbb{N}^n$ ,  $\langle \vec{x} \rangle$  codes  $\vec{x}$  so that for suitable recursive relations and functions,

$\langle \vec{x} \rangle = f_n(x_0, \dots, x_{n-1})$ ,  $\text{Seq}(w) \iff w$  is a sequence code,  $\text{lh}(\langle \vec{x} \rangle) = n$ ,  $(\langle \vec{x} \rangle)_i = x_i$ ,

and the code  $\langle \ \rangle$  of the empty sequence is 1.

Given  $\theta(v)$ , SRT (with  $m = n = 0$ ) gives us a number  $\tilde{z}$  such that

$$\{\tilde{z}\}(\ ) = 0 \iff \mathfrak{r}(\langle \tilde{z} \rangle) \text{ is not the code of a sentence or } T \vdash \theta(\Delta\mathfrak{r}(\langle \tilde{z} \rangle)). \quad (11)$$

Now  $\mathfrak{r}(\langle \tilde{z} \rangle)$  is the code of a sentence, because if it were not, then the right-hand-side of (11) would be true, which makes the left-hand-side true and insures that  $\mathfrak{r}(\langle \tilde{z} \rangle)$  codes a sentence, in fact a theorem of  $T$ ; and if  $\mathfrak{r}(\langle \tilde{z} \rangle) = \#\sigma$ , then

$$\begin{aligned} T \vdash \sigma &\iff \mathfrak{r}(\langle \tilde{z} \rangle) \in \text{Th}(T) \iff \{\tilde{z}\}(\ ) = 0 \\ &\iff T \vdash \theta(\Delta\mathfrak{r}(\langle \tilde{z} \rangle)) \iff T \vdash \theta(\ulcorner \sigma \urcorner). \quad \dashv \end{aligned}$$

The conclusion of the Fixed Point Lemma is usually stated in the stronger form

$$T \vdash \sigma \leftrightarrow \theta(\ulcorner \sigma \urcorner),$$

but (9) is sufficient to yield the applications. For the First Incompleteness Theorem, for example, we assume in addition that  $T$  is sufficiently expressive, we choose  $\sigma$  such that

$$T \vdash \sigma \iff T \vdash \neg(\exists u)\text{Proof}_T(\ulcorner \sigma \urcorner, u) \quad (12)$$

where  $\text{Proof}_T(v, u)$  numeralwise expresses in  $T$  its proof relation, and we check that if  $T$  is consistent, then  $T \not\vdash \sigma$ , and if  $T$  is also sound, then  $T \not\vdash \neg\sigma$ . The only difference from the usual argument is that (12) does not quite say that  $\sigma$  “expresses its own unprovability”—only that it is provable exactly when its unprovability is also provable. For the Rosser form of Gödel’s Theorem, we need to assume that  $T$  is a bit stronger (as we will explain below) and *consistent*, though not necessarily sound, and the classical argument again works with the more complex Rosser sentence and this same, small different understanding of what the Rosser sentence says.

There is a problem, however, with the key hypothesis in Theorem 6 that  $\text{Th}(T)$  is *r.e.-complete*. This is trivial for sufficiently expressive and sound theories, including, of course, PA, but not so simple to verify for theories which are consistent but not sound. In fact it holds for every axiomatizable theory  $T$  which extends the system Q from Robinson (1950)—which is the standard hypothesis for incompleteness and undecidability results about consistent theories which need not be sound.<sup>5</sup>

**Theorem 7** (Myhill (1955)). *If  $T$  is any consistent, axiomatizable extension of Q, then  $\text{Th}(T)$  is creative, and hence r.e.-complete.*

---

<sup>5</sup> For a description of Q and its properties, see (for example) Boolos, Burgess, and Jeffrey (1974) or even Kleene (1952), §41. Notice also that Theorem 7 does not lose much of its foundational interest or its important applications if we replace Q by PA in its statement—and the properties of Q that are used in the proof are quite obvious for PA.

The proof uses SRT with  $\mathbb{V} = \{0, 1\}$ , and (like all arguments of Myhill), it is very clever.

So consistent, axiomatizable extensions of  $\mathbf{Q}$  are undecidable and hence incomplete; moreover, the Fixed Point Lemma Theorem 6 applies to them, and so we can construct specific, interesting sentences that they cannot decide, a la Rosser.

A minor (notational) adjustment of the proofs establishes Theorems 6 and 7 for any consistent, axiomatizable theory  $T$ , in any language, provided only that  $\mathbf{Q}$  can be interpreted in  $T$  6 including, for example, ZFC; and then a third fundamental result of Myhill (1955) implies that the sets of theorems of any two such theories are recursively isomorphic. 7

**F. Solovay's theorem in provability logic.** The (propositional) modal formulas are built up as usual using variables  $p_0, p_1, \dots$ ; a constant  $\perp$  denoting falsity; the binary implication operator  $\rightarrow$  (which we use with  $\perp$  to define all the classical propositional connectives); and a unary operator  $\Box$ , which is usually interpreted by "it is necessary that". Solovay (1976) studies interpretations of modal formulas by sentences of PA in which  $\Box$  is interpreted by "it is provable in PA that" and establishes some of the basic results of the logic of provability. His central argument appeals to SRT at a crucial point.

An interpretation of modal formulas is any assignment  $\pi$  of sentences of PA to the propositional variables, which is then extended to all formulas by the structural recursion

$$\pi(\perp) \equiv 0 = 1, \quad \pi(\varphi \rightarrow \psi) \equiv (\pi(\varphi) \rightarrow \pi(\psi)), \quad \pi(\Box\varphi) \equiv (\exists u)\text{Proof}_{\text{PA}}(\ulcorner \pi(\varphi) \urcorner, u).$$

A modal formula  $\varphi$  is PA-valid if  $\text{PA} \vdash \pi(\varphi)$  for every interpretation  $\pi$ .

The axiom schemes of the system GL are:

- (A0) All tautologies;
- (A1)  $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$  (transitivity of provability);
- (A2)  $\Box\varphi \rightarrow \Box\Box\varphi$  (provable sentences are provably provable); and
- (A3)  $(\Box(\Box\varphi \rightarrow \varphi)) \rightarrow \Box\varphi$  (Löb's Theorem).

The inference rules of GL are:

- (R1)  $\varphi \rightarrow \psi, \varphi \implies \psi$  (Modus Ponens); and
- (R2)  $\varphi \implies \Box\varphi$  (Necessitation).

**Theorem 8** (Solovay (1976)). *A modal formula  $\varphi$  is PA-valid if and only if it is a theorem of GL.*

Solovay shows also that the class of PA-valid modal formulas is decidable, and he obtains a similar decidable characterization of the modal formulas  $\varphi$  such that

<sup>6</sup> A (weak) interpretation of  $T_1$  in  $T_2$  is any recursive map  $\chi \mapsto \chi^*$  of the sentences of  $T_1$  to those of  $T_2$  such that  $T_1 \vdash \chi \implies T_2 \vdash \chi^*$  and  $T_2 \vdash (\neg\chi)^* \leftrightarrow \neg(\chi^*)$ .

<sup>7</sup> Pour-El and Kripke (1967) have interesting, stronger results of this type, whose proofs also use the Second Recursion Theorem.

every interpretation  $\pi(\varphi)$  is true (in the standard model), in terms of a related axiom system  $GL'$ .

The proof of Theorem 8 is long, complex, ingenious and depends essentially (and subtly) on the full strength of PA. It is nothing like the one-line derivations of Theorems 2 and 6 from standard facts about the relevant objects by a natural application of SRT or even the longer, clever proofs of Theorems 3, 4, 5 and 7 in which SRT still yields the punch lines. Still, I cannot see how one could possibly construct (or even think up) the key, “self-referential” closed term  $l$  of Solovay's Lemma 4.1 directly, without appealing to the Second Recursion Theorem 8 and so, in that sense, SRT is an essential ingredient of his argument.

Next we turn to effective, grounded recursion, and perhaps the best way to explain it is to describe how it was introduced in Kleene (1938).

**G. Constructive and recursive ordinals.** Ordinal numbers can be viewed as the order types of well ordered sets, but also as *extended number systems*, which go beyond  $\mathbb{N}$  and can be used to count (and regulate) transfinite iteration. Church and Kleene developed in the 1930s an extensive theory of such systems, aiming primarily at a *constructive theory of ordinals*—which, however, was only partly realized since many of their basic results could only be proved using classical logic. Kleene (1938) formulated the Second Recursion Theorem to solve one of the basic problems in this area.

A *notation system for ordinals* or  *$\tau$ -system* (in Kleene (1938)) is a set  $S \subseteq \mathbb{N}$ , together with a function  $x \mapsto |x|_S$  which assigns to each  $x$  in  $S$  a countable ordinal so that the following conditions hold:

(ON1) There is a recursive partial function  $K(x)$  whose domain of convergence includes  $S$  and such that, for  $x \in S$ ,

$$\begin{aligned} |x|_S = 0 &\iff K(x) = 0, \\ |x|_S \text{ is a successor ordinal} &\iff K(x) = 1, \\ |x|_S \text{ is a limit ordinal} &\iff K(x) = 2. \end{aligned}$$

(ON2) There is a recursive partial function  $P(x)$ , such that if  $x \in S$  and  $|x|_S$  is a successor ordinal, then  $P(x) \downarrow$ ,  $P(x) \in S$  and  $|x|_S = |P(x)|_S + 1$ .

(ON3) There is a recursive partial function  $Q(x, t)$ , such that if  $x \in S$  and  $|x|_S$  is a limit ordinal, then for all  $t$ ,  $Q(x, t) \downarrow$ ,  $|Q(x, t)|_S < |Q(x, t + 1)|_S$  and  $|x|_S = \lim_t |Q(x, t)|_S$ .

In short, an  $\tau$ -system assigns *S-names* (number codes) to some ordinals, so that we can effectively recognize whether a code  $x$  names 0, a successor ordinal or a limit ordinal, and we can compute an *S-name* for the predecessor of each *S-named* successor ordinal and (*S-names* for) a strictly increasing sequence converging to each *S-named* limit ordinal.

A countable ordinal is *constructive* if it gets a name in some  $\tau$ -system.

<sup>8</sup> Which Solovay invokes to define a function  $h : \mathbb{N} \rightarrow \{0, \dots, n\}$  by the magical words “Our definition of  $h$  will be in terms of a Gödel number  $e$  for  $h$ . The apparent circularity is handled, using the recursion theorem, in the usual way.”

The empty set is an  $\tau$ -system, as is  $\mathbb{N}$ , which names the finite ordinals, and every  $\tau$ -system (obviously) assigns names to a countable, initial segment of the ordinal numbers. It is not immediately clear, however, whether the set of constructive ordinals is countable or what properties it may have: the main result in Kleene (1938) clarifies the picture considerably by constructing a single  $\tau$ -system which names all of them.

This “maximal”  $\tau$ -system is defined by a straightforward transfinite recursion which yields the following

**Lemma.** *There is an  $\tau$ -system  $(S_1, | \cdot |_1)$  such that:*

- (i)  $1 \in S_1$  and  $|1|_1 = 0$ .
- (ii) If  $x \in S_1$ , then  $2^x \in S_1$  and  $|2^x|_1 = |x|_1 + 1$ .
- (iii) If  $\varphi_e^1$  is total and for all  $t$ ,  $\varphi_e(t) \in S_1$  and  $|\varphi_e(t)|_1 < |\varphi_e(t+1)|_1$ , then  $3 \cdot 5^e \in S_1$  and  $|3 \cdot 5^e|_1 = \lim_t |\varphi_e(t)|$ .

**Theorem 9** (Kleene (1938)). *For every  $\tau$ -system  $(S, | \cdot |_S)$ , there is a unary recursive function  $\psi$  such that*

$$x \in S \implies (\psi(x) \in S_1 \ \& \ |x|_S = |\psi(x)|_1).$$

*In particular, the system  $(S_1, | \cdot |_1)$  names all constructive ordinals.*

*Proof.* Let  $K, P, Q$  be the recursive partial functions that come with  $(S, | \cdot |_S)$ , choose a number  $e_0$  such that

$$\{S(e_0, z, x)\}(t) = \{e_0\}(z, x, t) = \{z\}(Q(x, t)),$$

fix by SRT (with  $\mathbb{V} = \mathbb{N}, m = 0, n = 1$ ) a number  $\tilde{z}$  such that

$$\varphi_{\tilde{z}}(x) = \begin{cases} 1, & \text{if } K(x) = 0, \\ 2^{\varphi_{\tilde{z}}(P(x))}, & \text{if } K(x) = 1, \\ 3 \cdot 5^{S(e_0, \tilde{z}, x)}, & \text{otherwise,} \end{cases}$$

and set  $\psi(x) = \varphi_{\tilde{z}}(x)$ . The required properties of  $\psi(x)$  are proved by a simple (possibly transfinite) induction on  $|x|_S$ . –

In effect, the map from  $S$  to  $S_1$  is defined by the obvious transfinite recursion on  $|x|_S$ , which is made effective by appealing to SRT—hence the name for the method.

The choice of numbers of the form  $3 \cdot 5^e$  to name limit ordinals was made for reasons that do not concern us here, but it poses an interesting question: which ordinals get names in  $(S'_1, | \cdot |'_1)$ , defined by replacing  $3 \cdot 5^e$  by (say)  $7^e$  in the definition of  $S_1$ ? They are the same constructive ordinals, of course, and the proof is by defining by effective grounded recursion (exactly as in the proof of Theorem 9) a pair of recursive functions  $\psi, \psi'$  such that

$$x \in S_1 \implies (\psi'(x) \in S'_1 \ \& \ |x|_1 = |\psi'(x)|'_1),$$

$$x \in S'_1 \implies (\psi(x) \in S_1 \ \& \ |x|'_1 = |\psi(x)|_1).$$

(And I do not know how else one could prove this “obvious” fact.)

The constructive ordinals are “constructive analogs” of the classical countable ordinals, and so the constructive analog of the first uncountable ordinal  $\Omega_1$  is

$$\omega_1^{\text{CK}} = \sup\{|x|_1 \mid x \in S_1\} = \text{the least non-constructive ordinal,}$$

the superscript standing for *Church-Kleene*. This is one of the most basic “universal constants” which shows up in logic—in many parts of it and under many guises. We list here one of its earliest characterizations.

A countable ordinal  $\xi$  is *recursive* if it is the order type of a recursive well ordering on some subset of  $\mathbb{N}$ .

**Theorem 10** (Markwald (1954), Spector (1955)). *A countable ordinal  $\xi$  is constructive if and only if it is recursive.*

Both directions of the theorem are proved by (fairly routine) effective, grounded recursions.

**H. The hyperarithmetical hierarchy.** Kleene (1955c) associates with each  $a \in S_1$  a set  $H_a \subseteq \mathbb{N}$  so that<sup>9</sup>

- (H1)  $H_1 = \mathbb{N}$ .
- (H2)  $H_{2^b} = H'_b$ .
- (H3) If  $a = 3 \cdot 5^e$ , then  $t \in H_a \iff (t)_1 \in H_{\varphi_e((t)_0)}$ .

A relation  $P \subseteq \mathbb{N}^n$  is *hyperarithmetical* if it is recursive in some  $H_a$ .<sup>10</sup>

This natural extension of the arithmetical hierarchy was also defined independently (and in different ways) by Davis (1950) and Mostowski (1951) who knew most of its basic properties, but not the central Theorem III below. To formulate it, we need to refer to the *arithmetical* and *analytical* relations on  $\mathbb{N}$  which were introduced in Kleene (1943, 1955a); without repeating the definitions, we just record here the fact that they fall into two *hierarchies*

$$\Delta_1^0 \subsetneq \dots \subsetneq \Sigma_\ell^0 \cup \Pi_\ell^0 \subsetneq \Delta_{\ell+1}^0 \subsetneq \dots \subsetneq \Delta_1^1 \subsetneq \dots \subsetneq \Sigma_k^1 \cup \Pi_k^1 \subsetneq \Delta_{k+1}^1 \subsetneq \dots$$

where  $\Delta_1^0$  comprises the recursive relations. Above all the arithmetical relations and at the bottom of the analytical hierarchy lie the  $\Delta_1^1$  relations which satisfy a double equivalence

$$P(\vec{x}) \iff (\exists\beta)Q(\vec{x}, \beta) \iff (\forall\beta)R(\vec{x}, \beta)$$

where  $\beta$  ranges over the Baire space  $\mathcal{N} = (\mathbb{N} \rightarrow \mathbb{N})$  and  $Q, R$  are arithmetical (or just  $\Delta_2^0$ ) relations on  $\mathbb{N}^n \times \mathcal{N}$ , suitably defined.

**Theorem 11** (Kleene (1955c)). *A relation  $P \subseteq \mathbb{N}^n$  is  $\Delta_1^1$  if and only if it is hyperarithmetical.*

<sup>9</sup> For each  $A \subseteq \mathbb{N}$ ,  $A'$  is the *jump* of  $A$ .

<sup>10</sup> Actually, Kleene defines  $H_a$  only when  $a \in O$ , a variant of  $S_1$  which has more structure and is “more constructive”. I will disregard this fine point here, as many basic facts about  $O$  can only be proved classically and the attempt to use intuitionistic logic whenever it is possible clouds and complicates the arguments.



Notice that the theorem characterizes only the  $\Delta_1^1$  relations on  $\mathbb{N}$ , so we will revisit the question later.

This is the most significant, foundational result in the sequence of articles Kleene (1935, 1943, 1944, 1955a,b,c, 1959) in which Kleene developed the theory of arithmetical, hyperarithmetical and analytical relations on  $\mathbb{N}$ , surely one of the most impressive bodies of work in the theory of definability.<sup>11</sup> Starting with the 1944 article, Kleene uses effective, grounded recursion in practically every argument: it is the key, indispensable technical tool for this theory.

From the extensive work of others in this area, we cite only one, early but spectacular result:

**Theorem 12** (The Uniqueness Theorem, Spector (1955)). *There is a recursive function  $u(a, b)$ , such that if  $|a|_1 \leq_o |b|_1$ , then  $u(a, b)$  is the code of a Turing machine  $M$  which decides  $H_a$  using  $H_b$  as oracle.*

*In particular, if  $|a|_1 = |b|_1$ , then  $H_a$  and  $H_b$  have the same degree of unsolvability.*

The definition of  $u(a, b)$  is naturally given by effective grounded recursion.

**Classical and effective descriptive set theory.** Kleene was primarily interested in relations on  $\mathbb{N}$ , and he was more-or-less dragged into introducing quantification over  $\mathcal{N}$  and the analytical hierarchy in order to find *explicit forms* for the hyperarithmetical relations. Once they were introduced, however, the analytical relations on Baire space naturally posed new problems: is there, for example, a *construction principle* for the  $\Delta_1^1$  relations which satisfy

$$P(\mathbf{x}) \iff (\exists\beta)Q(\mathbf{x}, \beta) \iff (\forall\beta)R(\mathbf{x}, \beta) \quad (13)$$

where  $\mathbf{x} = \vec{x}, \vec{\alpha}$  varies over  $\mathbb{N}^n \times \mathcal{N}^m$  and  $Q, R$  are arithmetical—a useful and interesting analog of Theorem III?

In fact, these were very old problems, initially posed (and sometimes answered, in different form, to be sure) by Borel, Lebesgue, Lusin, Suslin and many others, primarily analysts and topologists who were working in *Descriptive Set Theory* in the first half of the 20th century. The similarity between what they had been doing and Kleene's work was first noticed by Mostowski (1946) and (especially) Addison (1954, 1959), and later work by many people created a common generalization of the classical and the new results now known as *Effective Descriptive Set Theory*; Moschovakis (2009) is the standard text on the subject and it gives a more careful history and a detailed introduction to this field.

The main inheritance of effective descriptive set theory from recursion theory is the propensity to *code*—assign to objects names that determine their relevant properties and then *compute*, *decide* or *define* functions and relations on these objects by operating on the codes rather than the objects coded—except that now we use points in  $\mathcal{N}$  rather than numbers for codes. We operate on  $\mathcal{N}$ -codes

<sup>11</sup> Kleene was the first logician to receive in 1983 the *Steele Prize for a seminal contribution to research* of the American Mathematical Society, specifically for the articles Kleene (1955a,b,c).

using *recursion on  $\mathcal{N}$* , by which a partial function  $f : \mathcal{N}^n \rightarrow \mathcal{N}$  is recursive if  $f(\vec{\alpha}) = (\lambda s)f^*(\vec{\alpha}, s)$  where  $f^* : \mathcal{N}^n \times \mathbb{N} \rightarrow \mathbb{N}$  is recursive, i.e., there is a recursive relation  $R \subseteq \mathbb{N}^{n+2}$  such that

$$f^*(\vec{\alpha}, s) = w \iff (\exists t)R(\vec{\alpha}_1(t), \dots, \vec{\alpha}_n(t), s, w)$$

where  $\vec{\alpha}(t) = \langle \alpha(0), \alpha(1), \dots, \alpha(t-1) \rangle$ . It is easy to show (and important) that for these partial functions,

$$f(\vec{\alpha}) = \beta \implies \beta \text{ is recursive in } \alpha_1, \dots, \alpha_n.$$

A partial function  $g : \mathcal{N}^n \rightarrow \mathcal{N}$  is *continuous* if

$$g(\vec{\alpha}) = f(\delta_0, \vec{\alpha})$$

with some recursive  $f : \mathcal{N}^{n+1} \rightarrow \mathcal{N}$  and some  $\delta_0 \in \mathcal{N}$  12

And here is the relevant version of SRT:

**Theorem 13.** *There is a recursive partial functions  $\varphi^n : \mathcal{N}^{n+1} \rightarrow \mathcal{N}$  for each  $n \in \mathbb{N}$ , so that (1) and (2) hold with*

$$\{\varepsilon\}(\vec{\alpha}) = \varphi_\varepsilon^n(\vec{\alpha}) = \varphi^n(\varepsilon, \vec{\alpha}) \quad (\vec{\alpha} = (\alpha_1, \dots, \alpha_n) \in \mathcal{N}^n).$$

(1) *Every continuous  $g : \mathcal{N}^n \rightarrow \mathcal{N}$  is  $\varphi_\varepsilon^n$  for some  $\varepsilon$ , and every recursive  $f : \mathcal{N}^n \rightarrow \mathcal{N}$  is  $\varphi_\varepsilon^n$  for some recursive  $\varepsilon$ .*

(2) *For all  $m, n$ , there is a recursive (total) function  $S = S_n^m : \mathcal{N}^{m+1} \rightarrow \mathcal{N}$  such that*

$$\{S(\varepsilon, \vec{\beta})\}(\vec{\alpha}) = \{\varepsilon\}(\vec{\beta}, \vec{\alpha}) \quad (\varepsilon \in \mathcal{N}, \vec{\beta} \in \mathcal{N}^m, \vec{\alpha} \in \mathcal{N}^n).$$

*It follows that for every recursive, partial function  $f(\varepsilon, \vec{\beta}, \vec{\alpha})$  of  $(1 + m + n)$  arguments, there is a total recursive function  $\tilde{\gamma}(\vec{\beta})$  of  $m$  arguments such that*

$$\{\tilde{\gamma}(\vec{\beta})\}(\vec{\alpha}) = f(\tilde{\gamma}(\vec{\beta}), \vec{\beta}, \vec{\alpha}) \quad (\vec{\beta} \in \mathcal{N}^m, \vec{\alpha} \in \mathcal{N}^n). \quad (14)$$

I will confine myself here to formulating just two results from effective descriptive set theory. Both are proved by effective, grounded recursion justified by this version of SRT, and they suggest the power of the method and the breadth of its applicability.

**I. The Suslin-Kleene Theorem.** Classical descriptive set theory is the study of definable subsets of an arbitrary *Polish* (metrizable, separable, complete topological) *space*  $\mathcal{X}$ , including the real numbers  $\mathbb{R}$  and products of the form  $\mathcal{X} =$

<sup>12</sup> Notice that the domain of convergence of a recursive, partial  $f : \mathcal{N}^n \rightarrow \mathcal{N}$  is not (in general)  $\Sigma_1^0$  but  $\Pi_2^0$ ,

$$f(\vec{\alpha}) \downarrow \iff (\forall s)(\exists w)(\exists t)R(\vec{\alpha}_1(t), \dots, \vec{\alpha}_n(t), s, w).$$

It is not difficult to check that a partial  $g : \mathcal{N}^n \rightarrow \mathcal{N}$  is continuous if its domain of convergence  $D_g$  is a  $G_\delta$  ( $\Pi_2^0$ ) set and  $g$  is topologically continuous on  $D_g$ .

$\mathbb{N}^n \times \mathcal{N}^m$ . The class  $\mathbf{B} = \mathbf{B}_{\mathcal{X}}$  of the *Borel subsets* of  $\mathcal{X}$  is the smallest class which contains all the open balls and is closed under countable unions and complements, and starting with this and iterating projection on  $\mathcal{N}$  and complementation we get the *projective hierarchy*,

$$\mathbf{B} \subseteq \underline{\Delta}_1^1 \subsetneq \underline{\Sigma}_1^1 \cup \underline{\Pi}_1^1 \subsetneq \underline{\Delta}_2^1 \subsetneq \underline{\Sigma}_2^1 \cup \underline{\Pi}_2^1 \subsetneq \dots$$

The class  $\underline{\Delta}_1^1$  comprises the sets which satisfy (13) with  $Q$  and  $R$  Borel (or even  $\underline{\Delta}_3^0$ ) subsets of the product space  $\mathcal{X} \times \mathcal{N}$ , and their identification with the Borel sets is a cornerstone of the theory:

**Theorem 14** (Suslin (1917)). *For every Polish space  $\mathcal{X}$ ,  $\underline{\Delta}_1^1 = \mathbf{B}$ .*

There is an obvious resemblance in form between this result of Suslin and Kleene’s Theorem 11, which led Mostowski and Addison to talk first of “analogies” between descriptive set theory and the “hierarchy theory” of Kleene, as it was then called; but neither of these results implies the other, as Suslin’s Theorem is trivial on  $\mathbb{N}^n$  and Kleene’s Theorem says nothing about subsets of  $\mathcal{N}$ —not to mention the real numbers or other Polish spaces which are very important for the classical theory. One of the first, substantial achievements of the “marriage” of the classical and the new theory was the derivation of a basic fact which extends (and refines) both Theorems 11 and 14.

We code the  $\underline{\Sigma}_k^1$  and  $\underline{\Pi}_k^1$  subsets of each Polish space in a natural way, so that the usual operations on them (countable unions and intersections, for example) are *recursive in the codes*, also in a natural way. A  $\underline{\Delta}_1^1$ -code of a set  $A$  is the (suitably defined) pair  $\langle \alpha, \beta \rangle$  of a  $\underline{\Sigma}_1^1$  and a  $\underline{\Pi}_1^1$  code for  $A$ . Finally, we code the Borel subsets of each  $\mathcal{X}$ , so that a Borel code of a set  $A$  supplies all the information necessary to construct  $A$  from open balls by iterating the operations of countable union and complementation. And with these definitions at hand:

**Theorem 15** (The Suslin-Kleene Theorem, see Moschovakis (2009)). *For each Polish space  $\mathcal{X}$  which admits a recursive presentation, there are recursive functions  $u, v : \mathcal{N} \rightarrow \mathcal{N}$  such that if  $\alpha$  is a Borel code of a set  $A \subseteq \mathcal{X}$ , then  $u(\alpha)$  is a  $\underline{\Delta}_1^1$ -code of  $A$ , and if  $\beta$  is a  $\underline{\Delta}_1^1$ -code of  $A$ , then  $v(\beta)$  is a Borel code of  $A$ .*

*In particular, the  $\underline{\Delta}_1^1$  subsets of  $\mathcal{X}$  are exactly the Borel subsets of  $\mathcal{X}$  which have recursive codes.*<sup>13</sup>

The Suslin-Kleene Theorem implies immediately Suslin’s Theorem and (with just a little extra work) Kleene’s Theorem 11. It is shown by adapting one of

<sup>13</sup> The restriction to *recursively presented spaces* is inessential, because every Polish space can be presented *recursively in some  $\varepsilon_0 \in \mathcal{N}$* , and then the whole theory “relativises” to this  $\varepsilon_0$ . Notice also that Moschovakis (2009) defines recursive presentations only for countable and perfect Polish spaces, but the definition makes sense for arbitrary Polish spaces, and then each such  $\mathcal{X}$  is isometric with the  $\Pi_1^0$  subset  $\mathcal{X} \times \{\lambda t 0\}$  of the perfect  $\mathcal{X} \times \mathcal{N}$ ; using this representation, the Suslin-Kleene Theorem holds for all recursively presented spaces—as do many (but not all) results in Moschovakis (2009).

the classical proofs of Suslin's Theorem rather than Kleene's much more difficult argument, and, of course, effective grounded recursion.

**J. The Coding Lemma.** The last example is from the exotic world of *determinacy*, about as far from recursion theory as one could go—or so it seems at first.

**Theorem 16** (The Coding Lemma, Moschovakis (1970, 2009)). *In ZFDC+AD: if there exists a surjection  $f : \mathbb{R} \rightarrow \kappa$  of the continuum onto a cardinal  $\kappa$ , then there exists a surjection  $g : \mathbb{R} \rightarrow \mathcal{P}(\kappa)$  of the continuum onto the powerset of  $\kappa$ .*

Here ZFDC stands for ZFC with the Axiom of (full) Choice AC replaced by the weaker *Axiom of Dependent Choices* DC, and AD is the *Axiom of Determinacy*, which is inconsistent with AC. It has been shown by Martin and Steel (1988) and Woodin (1988) that (granting the appropriate large cardinal axioms), AD holds in  $L(\mathbb{R})$ , the smallest model of ZFDC which contains all ordinals and all real numbers. Long before that great (and reassuring!) result, however, AD was used systematically to uncover the structure of the analytical and projective hierarchies—so it has something to do with recursion theory after all!

It is not possible to give here a brief, meaningful explanation of all that goes into the statement of Theorem 16 which, in any case, is only a corollary of a substantially stronger and more general result. Notice, however, that in a world where it holds,  $\mathbb{R}$  is immense in size, if we measure size by surjections: it can be mapped onto  $\aleph_1$  (classically), and so onto  $\aleph_2$ , and inductively onto every  $\aleph_n$  and so onto  $\aleph_\omega$ , etc., all the way onto every  $\aleph_\xi$  for  $\xi < \aleph_1$ : and it can also be mapped onto the powerset of each of these cardinals. This *surjective size* of  $\mathbb{R}$  is actually immense in the world of AD, the Coding Lemma is one of the important tools in proving this—and it does not appear possible to prove the Coding Lemma without using SRT, which creeps in this way into the study of cardinals, perhaps the most purely set-theoretic part of set theory.

The hypothesis AD of full determinacy is covered in Section 7D, *The completely playful universe* of Moschovakis (2009), part of Chapter 7 whose title is *The Recursion Theorem*.

## References

- Addison, J.W.: On some points of the theory of recursive functions. Ph.D. Thesis, University of Wisconsin (1954)
- Addison, J.W.: Separation principles in the hierarchies of classical and effective descriptive set theory. *Fundamenta Mathematicae* 46, 123–135 (1959)
- Boolos, G.S., Burgess, J.P., Jeffrey, R.C.: *Computability and logic*. Cambridge University Press, Cambridge (1974)
- Ceitin, G.: Algorithmic operators in constructive metric spaces. *Trudy Mat. Inst. Steklov.* 67, 295–361 (1962)
- Davis, M.: Relatively recursive functions and the extended Kleene hierarchy. In: *Proceedings of the International Congress of Mathematicians*, Cambridge, Mass, p. 723 (1950)

- Friedberg, R.M.: Un contre-exemple relatif aux fonctionnelles récursives. *Comptes Rendus Acad. Sci. Paris* 247, 852–854 (1958)
- Kleene, S.C.: General recursive functions of natural numbers. *Mathematische Annalen* 112, 727–742 (1935)
- Kleene, S.C.: On notation for ordinal numbers. *Journal of Symbolic Logic* 3, 150–155 (1938)
- Kleene, S.C.: Recursive predicates and quantifiers. *Transactions of the American Mathematical Society* 53, 41–73 (1943)
- Kleene, S.C.: On the form of predicates in the theory of constructive ordinals. *American Journal of Mathematics* 66, 41–58 (1944)
- Kleene, S.C.: Introduction to metamathematics. D. Van Nostrand Co. / North Holland Co. (1952)
- Kleene, S.C.: Arithmetical predicates and function quantifiers. *Transactions of the American Mathematical Society* 79, 312–340 (1955)
- Kleene, S.C.: On the form of predicates in the theory of constructive ordinals (second paper). *American Journal of Mathematics* 77, 405–428 (1955)
- Kleene, S.C.: Hierarchies of number theoretic predicates. *Bulletin of the American Mathematical Society* 61, 193–213 (1955)
- Kleene, S.C.: Quantification of number-theoretic functions. *Compositio Mathematica* 14, 23–40 (1959)
- Kreisel, G., Lacombe, D., Shoenfield, J.R.: Partial recursive functionals and effective operations. In: Heyting, A. (ed.) *Constructivity in mathematics: proceedings of the colloquium held in Amsterdam, 1957*, pp. 195–207. North Holland Publishing Co., Amsterdam (1957)
- Markwald, W.: Zur Theorie der konstruktiven Wohlordnungen. *Mathematischen Annalen* 127, 135–149 (1954)
- Martin, D.A., Steel, J.R.: Projective determinacy. *Proceedings of the National Academy of Sciences, USA* 85, 6582–6586 (1988)
- Moschovakis, Y.N.: Determinacy and prewellorderings of the continuum. In: Bar-Hillel, Y. (ed.) *Mathematical Logic and Foundations of Set Theory*, pp. 24–62. North-Holland, Amsterdam (1970)
- Moschovakis, Y.N.: Descriptive set theory. *Mathematical Surveys and Monographs*, vol. 155. American Mathematical Society, Providence (2009)
- Moschovakis, Y.N.: Classical descriptive set theory as a refinement of effective descriptive set theory (to appear, 2010)
- Mostowski, A.: On definable sets of positive integers. *Fundamenta Mathematicae* 34, 81–112 (1946)
- Mostowski, A.: A classification of logical systems. *Studia Philosophica* 4, 237–274 (1951)
- Myhill, J.: Creative sets. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 1, 97–108 (1955)
- Myhill, J., Shepherdson, J.C.: Effective operations on partial recursive functions. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 1, 310–317 (1955)
- Post, E.L.: Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society* 50, 284–316 (1944)
- Pour-El, M.B., Kripke, S.: Deduction-preserving “recursive isomorphisms” between theories. *Fundamenta Mathematicae* 61, 141–163 (1967)
- Robinson, R.M.: An essentially undecidable axiom system. In: *Proceedings of the International Congress of Mathematics*, pp. 729–730 (1950)

- Rogers Jr., H.: Theory of recursive functions and effective computability. McGraw-Hill, New York (1967)
- Solovay, R.M.: Provability interpretations of modal logic. *Israel Journal of Mathematics* 25, 287–304 (1976)
- Spector, C.: Recursive wellorderings. *Journal of Symbolic Logic* 20, 151–163 (1955)
- Suslin, M.: Sur une definition des ensembles mesurables B sans nombres transfinis. *Comptes Rendus Acad. Sci. Paris* 164, 88–91 (1917)
- Woodin, W.H.: Supercompact cardinals, sets of reals, and weakly homogeneous trees. *Proceedings of the National Academy of Sciences, USA* 85, 6587–6591 (1988)

# Typed Applicative Structures and Normalization by Evaluation for System $F^\omega$

Andreas Abel

Department of Computer Science  
Ludwig-Maximilians-University Munich

**Abstract.** We present a normalization-by-evaluation (NbE) algorithm for System  $F^\omega$  with  $\beta\eta$ -equality, the simplest impredicative type theory with computation on the type level. Values are kept abstract and requirements on values are kept to a minimum, allowing many different implementations of the algorithm. The algorithm is verified through a general model construction using typed applicative structures, called type and object structures. Both soundness and completeness of NbE are conceived as an instance of a single fundamental theorem.

## 1 Introduction and Related Work

In theorem provers, such as Coq [INR07], Agda [Nor07], Epigram [CAM07], which are based on intensional type theory, checking the validity of proofs or typings relies on deciding equality of types. Types are recognized as equal if they have the same normal form. This is why normalization plays a key role in type theories, such as the Calculus of Constructions (CC) which underlies Coq, and Martin-Löf Type Theory which is the basis of Agda and Epigram. The hardwired type equality of Coq is restricted to computational equality ( $\beta$ ), as opposed to Agda and Epigram which have  $\beta\eta$ -equality. Our goal is to integrate  $\eta$ -laws into Coq's equality. As a prerequisite, we have to show normalization for the  $\beta\eta$ -CC.

*Normalization by evaluation* (NbE) [BS91, Dan99] is a systematic method to perform  $\beta\eta$ -normalization. In a first step, the object  $t$  of type  $T$  is evaluated. The resulting value  $d$  is then *reified* to an  $\eta$ -long  $\beta$ -normal form  $v$ . The reification process is directed by the shape of type  $T$ . NbE has proven a valid tool to structure extensional normalization, especially in the notoriously difficult case of sum types [ADHS01, BCF04, Bar08]. In previous work [ACD07], we have adapted NbE to a dependent type theory with one predicative universe and judgmental  $\beta\eta$ -equality. What is the challenge when stepping up to impredicativity? Predicative type theories allow to define the semantics of types from below via induction-recursion [Dyb00], and the reification function can be defined by induction on types. This fails in the presence of impredicativity, where one first has to lay out a lattice of semantic type candidates and then define impredicative quantification using an intersection over all candidates [GLT89]. Hence, the semantic type structure is not inductive, and reification cannot be defined by induction on types. There are at least two ways out of this dilemma: Altenkirch, Hofmann, and Streicher [AHS96] construct a total normalization function type-wise while building a model for System F. In previous work [Abe08], I have conceived reification as a deterministic

relation between value  $d$  and normal form  $v$  and their type  $T$ , and showed through a model construction that it corresponds to a total function.

In this work, we are moving one step closer to NbE for the CC: we are considering the simplest type system which features impredicativity and computation on the type level: the *higher-order* polymorphic lambda-calculus  $F^\omega$ . It adds to the problem of impredicativity the difficulty that types are no longer fixed syntactic expressions as in System F, but they need to be normalized as well. Of course, due to the simply-kinded structure of types they could be kept in long normal form using simple structural normalization. This does not scale to the CC, so we resist this temptation.

In our solution, reification of objects is directed by *type values*  $A$ . Syntactic types  $T$  are interpreted by a pair  $(A, \mathcal{A})$  of a type value  $A$  and a semantic type  $\mathcal{A}$  which is a set of objects that are reifiable at type  $A$ . Furthermore, type value  $A$  reifies to a normal form  $V$  which is  $\beta\eta$ -equal to  $T$ . These considerations lead us to the concept of a *type structure* which captures the similarities between syntactic types, type values, and semantic types. Consequently, syntactic *objects* and their values both form an *object structure* over a type structure, the syntactical type structure in case of syntactic objects and the structure of type values in case of (object) values.

Or notions of type and object structures are very general, in essence typed versions of Barendregt's syntactical applicative structures [Bar84, Def. 5.3.1]. The fundamental theorems we prove are also very general since we do not fix an interpretation of types; we only require that semantic types inhabit a *candidate space*. By choosing different candidate spaces we can harvest different results from the same fundamental theorem, e. g., soundness of NbE, completeness of NbE, or weak normalization of  $\beta$ - or  $\beta\eta$ -reduction [Abe08].

In the following developments we omit most proofs due to lack of space. They can be found in the full version of this article on the author's homepage [Abe09].

*Preliminaries.* Contexts  $\Xi, \Theta, \Gamma, \Delta, \Phi, \Psi$  are functions from variables to some codomain. We write  $\diamond$  for the totally undefined function and  $\Phi, x : a$  for the function  $\Phi'$  with domain  $\text{dom}(\Phi) \uplus \{x\}$  such that  $\Phi'(x) = a$  and  $\Phi'(y) = \Phi(y)$  for  $y \neq x$ . We say  $\Psi'$  extends  $\Psi$ , written  $\Psi' \leq \Psi$ , if  $\Psi'(x) = \Psi(x)$  for all  $x \in \text{dom}(\Psi)$ .

Families  $\mathcal{T}_\Xi$  indexed by a context  $\Xi$  are always understood to be *Kripke*, i. e.,  $\Xi' \leq \Xi$  implies  $\mathcal{T}_\Xi \subseteq \mathcal{T}_{\Xi'}$ . The notion *Kripke family* is also used for maps  $M_\Xi$ . There it implies that  $M$  does not depend on the context parameter, i. e.,  $M_\Xi(a) = M_{\Xi'}(a)$  for  $a \in \text{dom}(M_\Xi)$  and  $\Xi' \leq \Xi$ . (Note that  $\text{dom}(M_\Xi) \subseteq \text{dom}(M_{\Xi'})$  since  $M$  is Kripke.)

We write  $(a \in \mathcal{A}) \rightarrow \mathcal{B}(a)$  for the *dependent function space*  $\{f \in \mathcal{A} \rightarrow \bigcup_{a \in \mathcal{A}} \mathcal{B}(a) \mid f(a) \in \mathcal{B}(a) \text{ for all } a \in \mathcal{A}\}$ .

## 2 Syntax

In this section, we present the syntax and inference rules for System  $F^\omega$ . The system consists of three levels: On the lowest level there live the *objects*, meaning polymorphic, purely functional programs. On the middle level live the *types* of objects, and the *type constructors*, which are classified by *kinds* that themselves inhabit the highest level.

*Kinds*  $\kappa \in \text{Ki}$  are given by the grammar  $\kappa ::= \star \mid \kappa \rightarrow \kappa'$ . Kind  $\star$  classifies type constructors which are actually types, and kind  $\kappa \rightarrow \kappa'$  classifies the type constructors



---

Kinding  $\Xi \vdash T : \kappa$ . “In context  $\Xi$ , type  $T$  has kind  $\kappa$ .”

$$\frac{}{\Xi \vdash C : \Sigma(C)} \quad \frac{}{\Xi \vdash X : \Xi(X)}$$

$$\frac{\Xi, X : \kappa \vdash T : \kappa'}{\Xi \vdash \lambda X : \kappa. T : \kappa \rightarrow \kappa'} \quad \frac{\Xi \vdash T : \kappa \rightarrow \kappa' \quad \Xi \vdash U : \kappa}{\Xi \vdash T U : \kappa'}$$

Type equality  $\Xi \vdash T = T' : \kappa$ . “In context  $\Xi$ , types  $T$  and  $T'$  are  $\beta\eta$ -equal of kind  $\kappa$ .”  
 Congruence closure of the following  $\beta$ - and  $\eta$ -axioms.

$$\frac{\Xi, X : \kappa \vdash T : \kappa' \quad \Xi \vdash U : \kappa}{\Xi \vdash (\lambda X : \kappa. T) U = T[U/X] : \kappa'} \quad \frac{\Xi \vdash T : \kappa \rightarrow \kappa'}{\Xi \vdash \lambda X : \kappa. T X = T : \kappa \rightarrow \kappa'} \quad X \notin \text{dom}(\Xi)$$

Typing  $\Xi; \Gamma \vdash t : T$ . “In contexts  $\Xi, \Gamma$ , object  $t$  has type  $T$ .”

$$\frac{\Xi \vdash \Gamma}{\Xi; \Gamma \vdash x : \Gamma(x)} \quad \frac{\Xi; \Gamma, x : U \vdash t : T}{\Xi; \Gamma \vdash \lambda x : U. t : U \rightarrow T} \quad \frac{\Xi; \Gamma \vdash t : U \rightarrow T \quad \Xi; \Gamma \vdash u : U}{\Xi; \Gamma \vdash t u : T}$$

$$\frac{\Xi \vdash T : \kappa \rightarrow \star \quad \Xi, X : \kappa; \Gamma \vdash t : T X \quad X \notin \text{dom}(\Xi)}{\Xi; \Gamma \vdash \lambda X : \kappa. t : \forall^\kappa T} \quad \frac{\Xi; \Gamma \vdash t : \forall^\kappa T \quad \Xi \vdash U : \kappa}{\Xi; \Gamma \vdash t U : T U}$$

$$\frac{\Xi; \Gamma \vdash t : T \quad \Xi \vdash T = T' : \star}{\Xi; \Gamma \vdash t : T'}$$

Object equality  $\Xi; \Gamma \vdash t = t' : T$ . “In contexts  $\Xi, \Gamma$ , objects  $t$  and  $t'$  are  $\beta\eta$ -equal of type  $T$ .”  
 Congruence closure of the following  $\beta$ - and  $\eta$ -axioms.

$$\frac{\Xi; \Gamma, x : U \vdash t : T \quad \Xi; \Gamma \vdash u : U}{\Xi; \Gamma \vdash (\lambda x : U. t) u = t[u/x] : T} \quad \frac{\Xi; \Gamma \vdash t : U \rightarrow T}{\Xi; \Gamma \vdash \lambda x : U. t x = t : U \rightarrow T} \quad x \notin \text{dom}(\Gamma)$$

$$\frac{\Xi, X : \kappa; \Gamma \vdash t : T \quad \Xi \vdash U : \kappa}{\Xi; \Gamma \vdash (\lambda X : \kappa. t) U = t[U/X] : T[U/X]} \quad \frac{\Xi; \Gamma \vdash t : \forall^\kappa T}{\Xi; \Gamma \vdash \lambda X : \kappa. t X = t : \forall^\kappa T} \quad X \notin \text{dom}(\Xi)$$


---

**Fig. 1.**  $F^\omega$ : kinding, type equality, typing, object equality

which map type constructors of kind  $\kappa$  to type constructors of kind  $\kappa'$ . In the following, we will refer to all type constructors as *types*.

Assume a countably infinite set of *type variables*  $\text{TyVar}$  whose members are denoted by  $X, Y, Z$ . *Kinding contexts*  $\Xi, \Theta \in \text{KiCxt}$  are partial maps from the type variables into  $\text{Ki}$ . The set  $\text{TyCst} = \{\rightarrow, \forall^\kappa \mid \kappa \in \text{Ki}\}$  contains the *type constants*  $C$ . Their kinds are given by the signature  $\Sigma \in \text{TyCst} \rightarrow \text{Ki}$  defined by  $\Sigma(\rightarrow) = \star \rightarrow \star \rightarrow \star$  and  $\Sigma(\forall^\kappa) = (\kappa \rightarrow \star) \rightarrow \star$  for all  $\kappa \in \text{Ki}$ .

*Types* are given by the grammar  $T, U, V ::= C \mid X \mid \lambda X : \kappa. T \mid T U$ , where  $X \in \text{TyVar}$ , and form a “simply-kinded” lambda calculus. As usual, we write  $T \rightarrow U$  for  $\rightarrow T U$ . *Objects* are given by the grammar  $t, u, v ::= x \mid \lambda x : T. t \mid t u \mid \lambda X : \kappa. t \mid t U$  and form a polymorphic lambda-calculus with type abstraction and type application. Herein, object variables  $x$  are drawn from a countably infinite set  $\text{ObjVar}$  which is disjoint from  $\text{TyVar}$ . We write  $b[a/x]$  for capture-avoiding substitution of  $a$  for variable  $x$  in syntactic expression  $b$ , and  $\text{FV}$  for the function returning the set of all free type and object variables of a syntactic expression.

The judgements and inference rules of  $F^\omega$  are displayed in Figure 11. Herein, the auxiliary judgement  $\Xi \vdash \Gamma$ , read “ $\Gamma$  is a well-formed typing context in  $\Xi$ ”, is defined as  $\Xi \vdash \Gamma(x) : \star$  for all  $x \in \text{dom}(\Gamma)$ .

### 3 Abstract Normalization by Evaluation

In the following, we present normalization by evaluation (NbE) for System  $F^\omega$  for an abstract domain  $D$  of *values* and type values. This leaves the freedom to implement values in different ways, e. g.,  $\beta$ -normal forms, weak head normal forms (as in Pollack’s constructive engine [Pol94]), closures (as in Coquand’s type checker [Coq96]), tagged functions (Epigram 2 [CAM07]) or virtual machine instructions (compiled reduction in Coq [GL02]). All implementations of values that satisfy the interface given in the following can be used with our NbE algorithm, and in this article we provide a framework to prove all these implementations correct.

In this section, we will understand functions in terms of a programming language, i. e., partial and possibly non-terminating. We unify the syntax of kinds, types, and objects into a grammar of expressions  $\text{Exp}$ . Let  $\text{Var} = \text{TyVar} \cup \text{ObjVar}$ .

Expressions	$\text{Exp} \ni M, N ::= \star \mid C \mid X \mid x \mid \lambda x : M. N \mid \Lambda X : M. N \mid M N$
Values	$D \ni d, e, f, A, B, F, G$ (abstract)

Environments  $\text{Env}$  are finite maps from variables to values. Look-up of variable  $x$  in environment  $\rho$  is written  $\rho(x)$ , update of environment  $\rho$  with new value  $v$  for variable  $x$  is written  $\rho[x \mapsto v]$ , and the empty environment is written  $\diamond$ . The call  $\text{fresh}(\rho)$  returns a variable  $x$  which is not in  $\text{dom}(\rho)$ .

Application and evaluation (see Fig. 2) make values into a *syntactical applicative structure* [Bar84, 5.3.1], provided the equations below are satisfied. Such structures will appear later, in a sorted setting, as type and object structures (defs. 11 and 13). Note that establishing the laws of evaluation can be arbitrarily hard, e. g., if  $(\lfloor \_ \rfloor)$  involves an optimizing compiler.

Values are converted back to expressions through reification. However, this process can only be implemented for *term-like* value domains, in particular, we require an embedding of variables into  $D$ , and an analysis  $\text{neView}$  of values that arise as iterated application of a variable (a so-called *neutral* value) or as iterated application of a constant (a *constructed* value). Some constructed values are types or kinds, they are analyzed by  $\text{tyView}$ , which can actually be defined from  $\text{neView}$ .

Values  $d$  of type  $V$  in context  $\Delta$ , which assigns type values to variables, are reified by a call to  $\searrow^\uparrow(\Delta, d, V)$ . It is mutually defined with  $\searrow^\uparrow(\Delta, n)$  which returns the normal form  $M$  and type  $V$  of neutral value  $n$ . Later in this article, reification will be presented as two relations  $\Delta \vdash d \searrow M \Downarrow^\uparrow V$  such that  $\Delta \vdash d \searrow M \Uparrow V$  iff  $\searrow^\uparrow(\Delta, d, V) = M$  and  $\Delta \vdash d \searrow M \Downarrow V$  iff  $\searrow^\downarrow(\Delta, d) = (M, V)$ .

NbE is now obtained as reification after evaluation. For closed expressions  $M$  of type or kind  $N$  we define  $\text{nbe}(M, N) = \searrow^\uparrow(\diamond, (\lfloor M \rfloor)_\diamond, \text{tyView}(\lfloor N \rfloor)_\diamond)$ .

Applicative structure  $D$  of values.

Application	$\_ \cdot \_$	$:$	$D \rightarrow D \rightarrow D$
Evaluation	$\llbracket \_ \rrbracket \_$	$:$	$\text{Exp} \rightarrow \text{Env} \rightarrow D$
	$\llbracket x \rrbracket \rho$	$=$	$\rho(x)$
	$\llbracket \lambda x : M. N \rrbracket \rho \cdot d$	$=$	$\llbracket N \rrbracket \rho[x \mapsto d]$
	$\llbracket X \rrbracket \rho$	$=$	$\rho(X)$
	$\llbracket \Lambda X : M. N \rrbracket \rho \cdot G$	$=$	$\llbracket N \rrbracket \rho[X \mapsto G]$
	$\llbracket M N \rrbracket \rho$	$=$	$\llbracket M \rrbracket \rho \cdot \llbracket N \rrbracket \rho$

$D$  is term-like.

Embedding	<b>var</b>	$:$	$\text{Var} \rightarrow D$
View as neutral	<b>NeView</b> $\ni n$	$::=$	$C \mid X \mid x \mid e d$
	<b>neView</b>	$:$	$D \rightarrow \text{NeView}$
	<b>neView</b> $\llbracket C \rrbracket \rho$	$=$	$C$
	<b>neView</b> $(\text{var } X)$	$=$	$X$
	<b>neView</b> $(\text{var } x)$	$=$	$x$
	<b>neView</b> $(e \cdot d)$	$=$	$e d$ if <b>neView</b> $e$ is defined
View as type	<b>TyView</b> $\ni V$	$::=$	$\star \mid A \rightarrow B \mid \forall^\kappa F$
	<b>tyView</b>	$:$	$D \rightarrow \text{TyView}$
	<b>tyView</b> $\llbracket \star \rrbracket \rho$	$=$	$\star$
	<b>tyView</b> $\llbracket M \rightarrow N \rrbracket \rho$	$=$	<b>tyView</b> $\llbracket M \rrbracket \rho \rightarrow$ <b>tyView</b> $\llbracket N \rrbracket \rho$
	<b>tyView</b> $\llbracket \forall^\kappa M \rrbracket \rho$	$=$	$\forall^\kappa$ <b>tyView</b> $\llbracket M \rrbracket \rho$

Reification.

$\searrow \uparrow$	$:$	$\text{Env} \rightarrow D \rightarrow \text{TyView} \rightarrow \text{Exp}$
$\searrow \uparrow$	$\llbracket \Delta, f, A \rightarrow B \rrbracket$	$= \text{let } x = \text{fresh}(\Delta)$
		$(U, \_ ) = \searrow \downarrow \llbracket \Delta, \text{neView } A \rrbracket$
		$\text{in } \lambda x : U. \searrow \uparrow \llbracket \Delta[x \mapsto A], f \cdot \text{var } x, \text{tyView } B \rrbracket$
$\searrow \uparrow$	$\llbracket \Delta, d, \forall^\kappa F \rrbracket$	$= \text{let } X = \text{fresh}(\Delta)$
		$\text{in } \Lambda X : \kappa. \searrow \uparrow \llbracket \Delta[X \mapsto \kappa], d \cdot \text{var } X, \text{tyView}(F \cdot \text{var } X) \rrbracket$
$\searrow \uparrow$	$\llbracket \Delta, e, \star \rrbracket$	$= \text{let } (M, \_ ) = \searrow \downarrow \llbracket \Delta, \text{neView } e \rrbracket \text{ in } M$
$\searrow \downarrow$	$:$	$\text{Env} \rightarrow \text{NeView} \rightarrow \text{Exp} \times \text{TyView}$
$\searrow \downarrow$	$\llbracket \Delta, C \rrbracket$	$= (C, \Sigma(C))$
$\searrow \downarrow$	$\llbracket \Delta, X \rrbracket$	$= (X, \text{tyView}(\Delta(X)))$
$\searrow \downarrow$	$\llbracket \Delta, x \rrbracket$	$= (x, \text{tyView}(\Delta(x)))$
$\searrow \downarrow$	$\llbracket \Delta, e d \rrbracket$	$= \text{let } (M, V) = \searrow \downarrow \llbracket \Delta, e \rrbracket \text{ in case } V \text{ of}$
		$A \rightarrow B \mapsto (M (\searrow \uparrow \llbracket \Delta, d, \text{tyView } A \rrbracket), \text{tyView } B)$
		$\forall^\kappa F \mapsto (M (\searrow \uparrow \llbracket \Delta, d, \kappa \rrbracket), \text{tyView}(F \cdot d))$

Normalization by evaluation.

$$\text{nbe}(M, N) = \searrow \uparrow \llbracket \diamond, \llbracket M \rrbracket \diamond, \text{tyView}(\llbracket N \rrbracket \diamond) \rrbracket$$

**Fig. 2.** Specification of an NbE algorithm

A concrete instance of NbE is obtained by defining a recursive data type  $D$  with the constructors:

$$\begin{aligned} \text{Constr} &: \text{TyCst} \rightarrow D^* \rightarrow D \\ \text{Ne} &: \text{Var} \rightarrow D^* \rightarrow D \\ \text{Abs} &: (D \rightarrow D) \rightarrow D \end{aligned}$$

Application, evaluation, and variable embedding are given by the following equations.

$$\begin{aligned} (\text{Constr } C \ Gs) \cdot G &= \text{Constr } C \ (Gs, G) \\ (\text{Ne } x \ ds) \cdot d &= \text{Ne } x \ (ds, d) \\ (\text{Abs } f) \cdot d &= f \ d \\ (\lambda x : M. N)_\rho &= \text{Abs } f && \text{where } f \ d = \llbracket N \rrbracket_{\rho[x \mapsto d]} \\ (\lambda X : M. N)_\rho &= \text{Abs } f && \text{where } f \ G = \llbracket N \rrbracket_{\rho[X \mapsto G]} \\ (C)_\rho &= \text{Constr } C \ () \end{aligned}$$

Variables are embedded via  $\text{var } x = \text{Ne } x \ ()$ . This instance of NbE is now easily completed using the equations of the specification, and can be implemented directly in Haskell.

In this article we show that *any instance* of the NbE-specification terminates with the correct result for well-formed expressions of  $F^\omega$ , i. e., we show the following two properties:

1. Soundness: if  $\vdash M : N$ , then  $\vdash \text{nbe}(M, N) = M : N$ .
2. Completeness: if  $\vdash M : N$  and  $\vdash M' : N$ , then  $\text{nbe}(M, N) = \text{nbe}(M', N)$  (same expression up to  $\alpha$ ).

In contrast to the untyped presentation in this section, which saves us from some repetition, we will distinguish the three levels of  $F^\omega$  consequently in the remainder of the article.

## 4 Type Structures

In this section, we define type structures as an abstraction over syntactic types, type values, and semantic types. Type structures form a category which has finite products. Let  $\text{Ty}_\Xi^\kappa = \{T \mid \Xi \vdash T : \kappa\}$ .

**Definition 1 (Type structure).** An ( $F^\omega$ ) type structure is a tuple  $(\mathcal{T}, \text{Cst}, \text{App}, \llbracket \_ \rrbracket)$  where  $\mathcal{T}$  is a Kripke family  $\mathcal{T}_\Xi^\kappa$  of sets with the following Kripke families of maps:

$$\begin{aligned} \text{Cst}_\Xi &\in (C \in \text{TyCst}) \rightarrow \mathcal{T}_\Xi^{\Sigma(C)} \\ \text{App}_\Xi^{\kappa \rightarrow \kappa'} &\in \mathcal{T}_\Xi^{\kappa \rightarrow \kappa'} \rightarrow \mathcal{T}_\Xi^\kappa \rightarrow \mathcal{T}_\Xi^{\kappa'} \end{aligned}$$

Usually, we will just write  $F \cdot G$  for  $\text{App}_\Xi^{\kappa \rightarrow \kappa'}(F, G)$ . Let  $\rho \in \mathcal{T}_\Theta^\Xi$  iff  $\rho(X) \in \mathcal{T}_\Theta^\Xi(X)$  for all  $X \in \text{dom}(\Xi)$ . The interpretation function has the following properties:

$$\begin{aligned} \llbracket \_ \rrbracket &\in \text{Ty}_\Xi^\kappa \rightarrow \mathcal{T}_\Theta^\Xi \rightarrow \mathcal{T}_\Theta^\kappa && \llbracket \lambda X : \kappa. T \rrbracket_\rho \cdot G = \llbracket T \rrbracket_{\rho[X \mapsto G]} \\ \llbracket C \rrbracket_\rho &= \text{Cst}_\Theta(C) && \llbracket T U \rrbracket_\rho = \llbracket T \rrbracket_\rho \cdot \llbracket U \rrbracket_\rho \\ \llbracket X \rrbracket_\rho &= \rho(X) && \llbracket T[U/X] \rrbracket_\rho = \llbracket T \rrbracket_{\rho[X \mapsto \llbracket U \rrbracket_\rho]} \quad (*) \end{aligned}$$

If the condition  $(*)$  is fulfilled, we speak of a *combinatory* type structure, since  $(*)$  is a characterizing property of combinatory algebras. The condition  $(*)$  is only necessary since we chose to use eager substitution in the inference rules of  $F^\omega$ , it can be dropped when switching to explicit substitutions [ACD08].

We use “interpretation” and “evaluation” synonymously. Note that while the equations determine the interpretation of constants, variables, and application, there is some freedom in the interpretation of functions  $\llbracket \lambda X : \kappa. T \rrbracket_\rho$ . It could be lambda-terms (taking  $\mathcal{T} = \text{Ty}$ ), set-theoretical functions (see Def. 20), functional values in an interpreter, machine code etc.

Since  $\text{Cst}_\Xi$  is independent of  $\Xi$ , we have  $\text{Cst}_\Xi = \text{Cst}_\circ$ , we usually suppress the index  $\Xi$  in  $\text{Cst}_\Xi$ . We may even drop  $\text{Cst}$  altogether, i. e., we just write  $\rightarrow \in T_\Xi^{\kappa \rightarrow \kappa' \rightarrow \kappa}$  instead of  $\text{Cst}(\rightarrow) \in T_\Xi^{\kappa \rightarrow \kappa' \rightarrow \kappa}$ .

To avoid ambiguities when different type structures are in scope, we may write  $\rightarrow_{\mathcal{T}}$ ,  $\forall_{\mathcal{T}}^\kappa$ ,  $\cdot_{\mathcal{T}}$  and  $\mathcal{T}[\_]\_$  to emphasize that we mean the type structure operations of  $\mathcal{T}$ .

Simple examples of type structures are  $\text{Ty}$  and  $\text{Ty}$  modulo  $\beta$ ,  $\beta\eta$ , or judgmental equality. In these instances, the interpretation function is parallel substitution.

**Definition 2 (Type structure morphism).** *Given two type structures  $\mathcal{S}$  and  $\mathcal{T}$ , a type structure morphism  $M : \mathcal{S} \rightarrow \mathcal{T}$  is a Kripke family of maps  $M_\Xi^\kappa \in \mathcal{S}_\Xi^\kappa \rightarrow \mathcal{T}_\Xi^\kappa$  that commute with the operations of  $\mathcal{S}$ , i. e.,*

$$\begin{aligned} M_\Xi^\kappa(C_{\mathcal{S}}) &= C_{\mathcal{T}} \\ M_\Xi^{\kappa'}(F \cdot_{\mathcal{S}} G) &= M_\Xi^{\kappa \rightarrow \kappa'}(F) \cdot_{\mathcal{T}} M_\Xi^\kappa(G) \\ M_\Theta^\kappa(\mathcal{S}[\![T]\!]_\rho) &= \mathcal{T}[\![T]\!]_{M_\Theta^\Xi \circ \rho} \quad \text{where } (M_\Theta^\Xi \circ \rho)(X) := M_\Theta^{\Xi(X)}(\rho(X)). \end{aligned}$$

The Cartesian product  $\mathcal{S} \times \mathcal{T}$  of two type structures forms a type structure with pointwise application and tupled interpretation. The two projections  $\pi_1 : \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{S}$  and  $\pi_2 : \mathcal{S} \times \mathcal{T} \rightarrow \mathcal{T}$  are trivially type structure morphisms, and  $\times$  is a product in the category of type structures and their morphisms.

#### 4.1 Type Substructures and the Fundamental Theorem for Kinding

**Definition 3 (Type substructure).** *The Kripke family  $\mathcal{S}_\Xi^\kappa \subseteq \mathcal{T}_\Xi^\kappa$  is a type substructure of  $\mathcal{T}$  if all of  $\mathcal{T}$ 's operations are well-defined on  $\mathcal{S}$ , i. e.,  $C_{\mathcal{T}} \in \mathcal{S}_\Xi^\kappa$ ,  $\cdot_{\mathcal{T}} \in \mathcal{S}_\Xi^{\kappa \rightarrow \kappa'} \rightarrow \mathcal{S}_\Xi^\kappa \rightarrow \mathcal{S}_\Xi^{\kappa'}$ , and  $\mathcal{T}[\_]\_ \in \text{Ty}_\Xi^\kappa \rightarrow \mathcal{S}_\Theta^\Xi \rightarrow \mathcal{S}_\Theta^\kappa$ .*

In the following we simply write  $\mathcal{S} \subseteq \mathcal{T}$  to mean  $\mathcal{S}_\Xi^\kappa \subseteq \mathcal{T}_\Xi^\kappa$  for all  $\kappa, \Xi$ .

**Lemma 1 (Projection type substructure).** *If  $\mathcal{S} \subseteq \mathcal{T}_1 \times \mathcal{T}_2$  is a type substructure, so are  $\pi_1(\mathcal{S}) \subseteq \mathcal{T}_1$  and  $\pi_2(\mathcal{S}) \subseteq \mathcal{T}_2$ .*

**Definition 4 (Function space).** *We write  $K \in \widehat{\mathcal{T}}^\kappa$  if  $K$  is a Kripke family of subsets  $K_\Xi \subseteq \mathcal{T}_\Xi^\kappa$ . Given  $K \in \widehat{\mathcal{T}}^\kappa$  and  $K' \in \widehat{\mathcal{T}}^{\kappa'}$  we define the Kripke function space*

$$(K \rightarrow_{\widehat{\mathcal{T}}} K')_\Xi = \{F \in \mathcal{T}_\Xi^{\kappa \rightarrow \kappa'} \mid F \cdot G \in K'_\Xi, \text{ for all } \Xi' \leq \Xi \text{ and } G \in K_{\Xi'}\}$$

If no ambiguities arise, we write  $\rightarrow$  for  $\rightarrow_{\widehat{\mathcal{T}}}$ .

**Definition 5 (Induced type structure).** Let  $\mathcal{T}$  be a type structure and  $\mathcal{S} \subseteq \mathcal{T}$  be Kripke. If  $C_{\mathcal{T}} \in \mathcal{S}_{\Xi}^{\Sigma(C)}$  for all constants  $C$  and  $\mathcal{S}_{\Xi}^{\kappa \rightarrow \kappa'} = (\mathcal{S}^{\kappa} \rightarrow_{\hat{\mathcal{T}}} \mathcal{S}^{\kappa'})_{\Xi}$  then  $\mathcal{S}$  is called induced or an induced type substructure of  $\mathcal{T}$  (see Thm. [7](#)).

Such an  $\mathcal{S}$  is called induced since it is already determined by the choice of the denotation of the base kind  $\mathcal{S}^*$ .

**Theorem 1 (Fundamental theorem of kinding).** Let  $\mathcal{T}$  be a type structure. If  $\mathcal{S} \subseteq \mathcal{T}$  is induced, then  $\mathcal{S}$  is a type substructure of  $\mathcal{T}$ .

*Proof.* We mainly need to show that evaluation is well-defined. This is shown by induction on the kinding derivation, as usual.  $\square$

## 4.2 NbE for Types and Its Soundness

We are ready to define the reification relation for type values and show that NbE, i. e., the composition of evaluation of a syntactic type  $T$  and reification to a normal form  $V$ , is sound, i. e.,  $T$  and  $V$  are judgmentally equal. As a byproduct, we show totality of NbE on well-kinded types. The structure  $\mathcal{T}$  of type values is left abstract. However, not every  $\mathcal{T}$  permits reification of its inhabitants. It needs to include the variables which need to be distinguishable from each other and other type values. *Neutral* types, i. e., of the shape  $X \cdot \mathbf{G}$ , need to be analyzable into head  $X$  and spine  $\mathbf{G}$ . We call a suitable  $\mathcal{T}$  *term-like*; on such a  $\mathcal{T}$  we can define contextual reification [\[ACD08, Abe08\]](#).

**Definition 6 (Term-like type structure).** A type structure  $\mathcal{T}$  is term-like if there exists exists a Kripke family of maps  $\text{Var}_{\Xi} \in (X \in \text{dom}(\Xi)) \rightarrow \mathcal{T}_{\Xi}^{\Xi(X)}$  and a Kripke family of partial maps

$$\text{View}_{\Xi}^{\kappa} \in \mathcal{T}_{\Xi}^{\kappa} \rightarrow \{(C, \mathbf{G}) \in \text{TyCst} \times \mathcal{T}_{\Xi}^{\kappa} \mid \Sigma(C) = \kappa \rightarrow \kappa\} \\ + \{(X, \mathbf{G}) \in \text{TyVar} \times \mathcal{T}_{\Xi}^{\kappa} \mid \Xi(X) = \kappa \rightarrow \kappa\}$$

such that  $\text{View}(F) = (C, \mathbf{G})$  iff  $F = \text{Cst}(C) \cdot \mathbf{G}$  (“ $F$  is constructed”) and  $\text{View}(F) = (X, \mathbf{G})$  iff  $F = \text{Var}(X) \cdot \mathbf{G}$  (“ $F$  is neutral”).

Usually, we drop the index  $\Xi$  to  $\text{Var}$ . We may write  $\text{Var}_{\mathcal{T}}$  to refer to the variable embedding of type structure  $\mathcal{T}$ .

**Definition 7 (Type reification).** On a term-like type structure  $\mathcal{T}$  we define reification relations

$$\Xi \vdash F \searrow V \uparrow \kappa \quad \text{in } \Xi, F \text{ reifies to } V \text{ at kind } \kappa, \\ \Xi \vdash H \searrow U \downarrow \kappa \quad \text{in } \Xi, H \text{ reifies to } U, \text{ inferring kind } \kappa,$$

(where  $F, H \in \mathcal{T}_{\Xi}^{\kappa}$  with  $H$  neutral or constructed, and  $V, U \in \text{Ty}_{\Xi}^{\kappa}$ ) inductively by the following rules:

$$\frac{}{\Xi \vdash X \searrow X \downarrow \Xi(X)} \quad \frac{\Xi \vdash H \searrow U \downarrow \kappa \rightarrow \kappa' \quad \Xi \vdash G \searrow V \uparrow \kappa}{\Xi \vdash H \cdot G \searrow UV \downarrow \kappa'} \\ \frac{}{\Xi \vdash C \searrow C \downarrow \Sigma(C)} \quad \frac{\Xi \vdash H \searrow U \downarrow \star}{\Xi \vdash H \searrow U \uparrow \star} \quad \frac{\Xi, X : \kappa \vdash F \cdot X \searrow V \uparrow \kappa'}{\Xi \vdash F \searrow \lambda X : \kappa. V \uparrow \kappa \rightarrow \kappa'}$$

Reification is *deterministic* in the following sense: For all  $\Xi, \kappa, F$  (inputs) and neutral or constructed  $H$  (input) there is at most one  $V$  (output) such that  $\Xi \vdash F \searrow V \uparrow \kappa$  and at most one  $U$  and  $\kappa'$  (outputs) such that  $\Xi \vdash H \searrow U \downarrow \kappa'$ .

Seen as logic programs with inputs and outputs as indicated above, these relations denote partial functions, where  $\searrow \uparrow$  is defined by cases on the kind  $\kappa$  and  $\searrow \downarrow$  by cases on the neutral value  $H$ .

We continue by constructing a model of the kinding rules which proves soundness of NbE for types. Kinds  $\kappa$  are interpreted as sets  $G_{\Xi}^{\kappa}$  of pairs  $(F, T)$  *glued together* [CD97] by reification, i. e., the type value  $F$  reifies to syntactic type  $T$  up to  $\beta\eta$ -equality. Function kinds are interpreted via Tait's function space (see Def. 4), thus, the fundamental theorem of kinding yields that  $G$  is indeed a type structure.

**Definition 8 (Glueing candidate).** Fix a term-like type structure  $\mathcal{T}$ . We define the families  $\underline{G}, \overline{G} \subseteq \mathcal{T} \times \text{Ty}$  by

$$\begin{aligned} \overline{G}_{\Xi}^{\kappa} &= \{(F, T) \in \mathcal{T}_{\Xi}^{\kappa} \times \text{Ty}_{\Xi}^{\kappa} \mid \Xi \vdash F \searrow V \uparrow \kappa \text{ and } \Xi \vdash T = V : \kappa\}, \\ \underline{G}_{\Xi}^{\kappa} &= \{(H, T) \in \mathcal{T}_{\Xi}^{\kappa} \times \text{Ty}_{\Xi}^{\kappa} \mid \Xi \vdash H \searrow U \downarrow \kappa \text{ and } \Xi \vdash T = U : \kappa\}. \end{aligned}$$

A family  $S$  with  $\underline{G}^{\kappa} \subseteq S^{\kappa} \subseteq \overline{G}^{\kappa}$  is called a glueing candidate.

**Def. and Lem. 2 (Kind candidate space).**  $\underline{G}^{\kappa}, \overline{G}^{\kappa}$  form a kind candidate space, i. e., satisfy the following laws, where we write  $\underline{\kappa}$  for  $\underline{G}^{\kappa}$  and  $\overline{\kappa}$  for  $\overline{G}^{\kappa}$ .

$$\star \subseteq \overline{\kappa}, \quad \underline{\kappa} \rightarrow \overline{\kappa'} \subseteq \overline{\kappa \rightarrow \kappa'}, \quad \underline{\kappa} \rightarrow \underline{\kappa'} \subseteq \overline{\kappa} \rightarrow \underline{\kappa'}.$$

**Def. and Lem. 3 (Glueing type structure).** Given a type structure  $\mathcal{T}$ , we define  $G \subseteq \mathcal{T} \times \text{Ty}$  by  $G_{\star}^{\kappa} = \star$  and  $G_{\star}^{\kappa \rightarrow \kappa'} = G^{\kappa} \rightarrow_{\overline{\mathcal{T} \times \text{Ty}}} G^{\kappa'}$ .  $G$  is a glueing candidate, i. e.,  $\underline{G}^{\kappa} \subseteq G^{\kappa} \subseteq \overline{G}^{\kappa}$  for all  $\kappa$ .

Since  $G$  is induced, by the fundamental theorem of kinding it is a type substructure of  $\mathcal{T} \times \text{Ty}$ . Thus, for all  $T \in \text{Ty}_{\Xi}^{\kappa}$ ,  $G[[T]]_{\text{Var}_G} = (T[[T]]_{\text{Var}_T}, T) \in G_{\Xi}^{\kappa} \subseteq \overline{G}_{\Xi}^{\kappa}$ , entailing soundness.

**Theorem 4 (Soundness of NbE for types).** Let  $\mathcal{T}$  be a term-like type structure. If  $\Xi \vdash T : \kappa$  then there is a  $V \in \text{Ty}_{\Xi}^{\kappa}$  such that  $\Xi \vdash T[[T]]_{\text{Var}_T} \searrow V \uparrow \kappa$  and  $\Xi \vdash T = V : \kappa$ .

## 5 Type Groupoids

Completeness of NbE means that it models judgmental type equality, i. e., if  $\Xi \vdash T = T' : \kappa$  then  $\Xi \vdash [[T]] \searrow V \uparrow \kappa$  and  $\Xi \vdash [[T']] \searrow V \uparrow \kappa$ . Completeness will be shown by a fundamental theorem of type equality. Judgmental equality is usually modelled by partial equivalence relations (PERs), which can be seen as groupoids. Hence, we introduce the notion of a *groupoidal type structure*, or *type groupoid*. The advantage over PERs is that we can directly reuse the fundamental theorem of kinding, instantiated to a groupoidal type structure  ${}^2\mathcal{T}$  of pairs of types, instead of having to prove this theorem again for kinds modelled as PERs.

A *groupoid* is a set with inversion  $\_^{-1} : \rightarrow$  and partial but associative composition  $\_ * \_ : \times \rightarrow$  such that  $a^{-1} * a$  and  $a * a^{-1}$  are always defined, and if  $a * b$  is defined, then  $a * b * b^{-1} = a$ , and  $a^{-1} * a * b = b$ . One easily shows that  $(a^{-1})^{-1} = a$  and if  $a * b$  is defined then  $(a * b)^{-1} = b^{-1} * a^{-1}$ . Examples of groupoids include partial equivalence relations  $R$ , where  $(s, t)^{-1} = (t, s)$  and  $(r, s) * (s, t) = (r, t)$ , and any set  $S$  with the trivial groupoidal structure:  $s^{-1} = s$  and  $r * s$  is defined iff  $r = s$ , and then  $s * s = s$ .

A *subgroupoid* is a subset  $\mathcal{H} \subseteq$  that is closed under inversion and composition.

## 5.1 Type Groupoids and the Fundamental Theorem of Type Equality

**Definition 9 (Type groupoid).** A type structure is groupoidal if each  $\mathcal{T}_{\Xi}^{\kappa}$  is a groupoid, constants are preserved under inversion, and inversion and composition distribute over application, i. e.,  $C^{-1} = C$ ,  $(F \cdot G)^{-1} = F^{-1} \cdot G^{-1}$ , and  $(F \cdot G) * (F' \cdot G') = (F * F') \cdot (G * G')$ .

Given a type structure  $\mathcal{T}$  we define the square type groupoid  ${}^2\mathcal{T}$  as the product type structure  $\mathcal{T} \times \mathcal{T}$  equipped with  $(F, G)^{-1} = (G, F)$  and  $(F, G) * (G, H) = (F, H)$ . If  $K \in \widehat{\mathcal{T}}^{\kappa}$  and  $K' \in \widehat{\mathcal{T}}^{\kappa'}$  are groupoids, so is  $K \rightarrow_{\widehat{\mathcal{T}}} K' \in \widehat{\mathcal{T}}^{\kappa \rightarrow \kappa'}$ .

**Definition 10 (Induced type groupoid).** Let  $\mathcal{T}$  be a type structure and  $\mathcal{E} \subseteq {}^2\mathcal{T}$ . We say  $\mathcal{E}$  is induced if  $\mathcal{E}$  is an induced type structure and  $\mathcal{E}_{\Xi}^*$  is groupoidal for all  $\Xi$ .

Since type equality refers to kinding, we will have to refer to the fundamental theorem of kinding in the proof of the fundamental theorem of type equality.

**Lemma 2 (Fundamental theorem of kinding for type groupoids).** Let  $\mathcal{T}$  be a type structure and  $\mathcal{E} \subseteq {}^2\mathcal{T}$  be induced. Then,

1.  $\mathcal{E}$  is a type subgroupoid of  ${}^2\mathcal{T}$ , and
2. if  $\Xi \vdash T : \kappa$  and  $(\rho, \rho') \in \mathcal{E}_{\Theta}^{\Xi}$  then  $(\mathcal{T}[[T]]_{\rho}, \mathcal{T}[[T]]_{\rho'}) \in \mathcal{E}_{\Theta}^{\kappa}$ .

**Definition 11 (Model/respect type equality).**  $\mathcal{E} \subseteq {}^2\mathcal{T}$  models type equality if  $\Xi \vdash T = T' : \kappa$  and  $(\rho, \rho') \in \mathcal{E}_{\Theta}^{\Xi}$  imply  $(\mathcal{T}[[T]]_{\rho}, \mathcal{T}[[T']]_{\rho'}) \in \mathcal{E}_{\Theta}^{\kappa}$ . A type structure  $\mathcal{T}$  respects type equality if  $\Xi \vdash T = T' : \kappa$  implies  $\mathcal{T}'[[T]]_{\rho} = \mathcal{T}'[[T']]_{\rho}$  for all  $\rho \in \mathcal{T}'_{\Theta}^{\Xi}$ .

**Theorem 5 (Fundamental theorem of type equality).** Let  $\mathcal{T}$  be a combinatory type structure and  $\mathcal{E} \subseteq {}^2\mathcal{T}$  an induced type structure. Then  $\mathcal{E}$  models type equality.

## 5.2 Completeness of NbE for Types

In the following we show that the relation “reify to the same  $\eta$ -long form” gives rise to an equivalence relation on types which models type equality. This implies that NbE is complete.

**Def. and Lem. 6 (Kind candidate space for completeness).** Let  $\mathcal{T}$  be term-like.

$$\begin{aligned} \overline{\text{Per}}_{\Xi}^{\kappa} &= \{(F, F') \in {}^2\mathcal{T}_{\Xi}^{\kappa} \mid \Xi \vdash F \searrow V \uparrow \kappa \text{ and } \Xi \vdash F' \searrow V \uparrow \kappa \text{ for some } V\} \\ \underline{\text{Per}}_{\Xi}^{\kappa} &= \{(F, F') \in {}^2\mathcal{T}_{\Xi}^{\kappa} \mid \Xi \vdash F \searrow V \downarrow \kappa \text{ and } \Xi \vdash F' \searrow V \downarrow \kappa \text{ for some } V\} \end{aligned}$$

$\underline{\text{Per}}^{\kappa}$  and  $\overline{\text{Per}}^{\kappa}$  are Kripke families of subgroupoids, and form a kind candidate space.



**Def. and Lem. 7 (Type groupoid for completeness).** Let  $\mathcal{T}$  be a type structure. We define  $\mathsf{P}^\kappa \subseteq {}^2\mathcal{T}^\kappa$  by recursion on  $\kappa$ :  $\mathsf{P}^* := \underline{\mathsf{Per}}^*$  and  $\mathsf{P}^{\kappa \rightarrow \kappa'} = \mathsf{P}^\kappa \rightarrow_{\widehat{\mathcal{T}}} \mathsf{P}^{\kappa'}$ .  $\mathsf{P}$  is an induced type groupoid.

**Theorem 8 (Completeness of NbE for types).** Let  $\mathcal{T}$  be a term-like type structure. If  $\Xi \vdash T = T' : \kappa$  then  $\Xi \vdash \llbracket T \rrbracket_{\mathsf{Var}} \searrow V \uparrow \kappa$  and  $\Xi \vdash \llbracket T' \rrbracket_{\mathsf{Var}} \searrow V \uparrow \kappa$  for some  $V$ .

*Proof.* Since  $(\mathsf{Var}, \mathsf{Var}) \in \mathsf{P}_{\Xi}^{\Xi}$ , by the fundamental theorem of type equality we have  $(\llbracket T \rrbracket_{\mathsf{Var}}, \llbracket T' \rrbracket_{\mathsf{Var}}) \in \mathsf{P}_{\Xi}^{\kappa} \subseteq \overline{\mathsf{Per}}_{\Xi}^{\kappa}$  which entails the goal.  $\square$

## 6 Object Structures

In this section, we introduce object structures which model both the syntactic object structure  $\mathsf{Obj}$  indexed by syntactic types in  $\mathsf{Ty}$  and structures of values  $D$  indexed by type values from a structure  $\mathcal{T}$ . The following development, leading up the fundamental theorem of typing and the soundness of NbE for objects, parallels the preceding one on the type level. However, while we could define the glueing type structure  $G^\kappa$  by induction on kind  $\kappa$ , we cannot define a similar glueing objects structure  $\mathsf{gl}$  by induction on types, due to impredicativity. Hence, we will define  $\mathsf{gl}$  as a structure of *candidates* for semantic types.

**Definition 12 (Typing context).** Given a type structure  $\mathcal{T}$ , a  $\mathcal{T}_{\Xi}$ -context  $\Delta \in \mathcal{T}_{\Xi}^{\text{cxt}}$  is a partial map from the term variables into  $\mathcal{T}_{\Xi}^*$ . If  $\Gamma \in \mathsf{Ty}_{\Xi}^{\text{cxt}}$  and  $\rho \in \mathcal{T}_{\Theta}^{\Xi}$  then  $\llbracket \Gamma \rrbracket_{\rho} \in \mathcal{T}_{\Theta}^{\text{cxt}}$  is defined by  $\llbracket \Gamma \rrbracket_{\rho}(x) = \llbracket \Gamma(x) \rrbracket_{\rho}$ .

Let  $\mathsf{Obj}_{\Gamma}^{\Xi \vdash T} = \{t \mid \Xi; \Gamma \vdash t : T\}$ .

**Definition 13 (Object structure).** Let  $\mathcal{T}$  be a type structure. An object structure over  $\mathcal{T}$  is a family  $D^{\Xi \vdash A}$  ( $A \in \mathcal{T}_{\Xi}^*$ ) of Kripke sets indexed by  $\mathcal{T}_{\Xi}$ -contexts  $\Delta$  such that  $\Xi' \leq \Xi$  implies  $D_{\Delta}^{\Xi \vdash A} = D_{\Delta}^{\Xi' \vdash A}$ . It respects type equality, i. e.,  $\Xi \vdash T = T' : *$  implies  $D^{\Theta \vdash \llbracket T \rrbracket_{\rho}} = D^{\Theta \vdash \llbracket T' \rrbracket_{\rho}}$  for any  $\rho \in \mathcal{T}_{\Theta}^{\Xi}$ , and there are operations:

$$\begin{aligned} \mathsf{app}_{\Delta}^{\Xi \vdash A \rightarrow B} &\in D_{\Delta}^{\Xi \vdash A \rightarrow B} \rightarrow D_{\Delta}^{\Xi \vdash A} \rightarrow D_{\Delta}^{\Xi \vdash B}, \\ \mathsf{TyApp}_{\Delta}^{\Xi \vdash \forall^{\kappa} F} &\in D_{\Delta}^{\Xi \vdash \forall^{\kappa} F} \rightarrow (G \in \mathcal{T}_{\Xi}^{\kappa}) \rightarrow D_{\Delta}^{\Xi \vdash F \cdot G}. \end{aligned}$$

We write  $\cdot$  for both of these operations. For  $\Delta, \Psi \in \mathcal{T}_{\Theta}^{\text{cxt}}$ , let  $\eta \in D_{\Delta}^{\Theta \vdash \Psi}$  iff  $\eta(x) \in D_{\Delta}^{\Theta \vdash \Psi(x)}$  for all  $x \in \text{dom}(\Psi)$ . We stipulate a family of evaluation functions

$$\llbracket \_ \rrbracket_{\eta}^{\rho} \in \mathsf{Obj}_{\Gamma}^{\Xi \vdash T} \rightarrow D_{\Delta}^{\Theta \vdash \llbracket \Gamma \rrbracket_{\rho}} \rightarrow D_{\Delta}^{\Theta \vdash \llbracket T \rrbracket_{\rho}}$$

indexed by  $\rho \in \mathcal{T}_{\Theta}^{\Xi}$  which satisfy the following equations:

$$\begin{aligned} \llbracket x \rrbracket_{\eta}^{\rho} &= \eta(x) & \llbracket \lambda x : U. t \rrbracket_{\eta}^{\rho} \cdot d &= \llbracket t \rrbracket_{\eta}^{\rho[x \mapsto d]} & \text{if } d \in D_{\Delta}^{\Theta \vdash \llbracket U \rrbracket_{\rho}} \\ \llbracket r \ s \rrbracket_{\eta}^{\rho} &= \llbracket r \rrbracket_{\eta}^{\rho} \cdot \llbracket s \rrbracket_{\eta}^{\rho} & \llbracket \lambda X : \kappa. t \rrbracket_{\eta}^{\rho} \cdot G &= \llbracket t \rrbracket_{\eta}^{\rho[X \mapsto G]} & \text{if } G \in \mathcal{T}_{\Theta}^{\kappa} \\ \llbracket t \ U \rrbracket_{\eta}^{\rho} &= \llbracket t \rrbracket_{\eta}^{\rho} \cdot \llbracket U \rrbracket_{\rho} & \llbracket t[U/X] \rrbracket_{\eta}^{\sigma} &= \llbracket t \rrbracket_{\eta}^{\sigma[X \mapsto \llbracket U \rrbracket_{\sigma}]} & (*) \\ \llbracket t[u/x] \rrbracket_{\eta}^{\sigma} &= \llbracket t \rrbracket_{\eta[x \mapsto \llbracket u \rrbracket_{\eta}^{\sigma}]}^{\sigma} & & & \end{aligned}$$

Again,  $(*)$  have to hold only in *combinatory* object structures.

With parallel substitution, Obj (modulo  $\beta$ ,  $\beta\eta$ , or judgmental equality) forms an object structure over Ty (modulo the same equality).

**Definition 14 (Object substructure).** Let  $S, T$  be type structures with  $S \subseteq T$  and let  $D$  be an object structure over  $T$ . Let  $E^{\Xi \vdash A} \subseteq D^{\Xi \vdash A}$  be a Kripke family of subsets indexed by  $\Delta \in S_{\Xi}^{\text{cxt}}$  for all  $A \in S_{\Xi}^*$ . Then  $E$  is an object substructure of  $D$  over  $S$  if application and evaluation are well defined on  $E$ .

**Definition 15 (Reindexed object structure).** Let  $M : S \rightarrow T$  be a type structure morphism and  $D$  an object structure over  $T$ . The type structure  $E^{\Xi \vdash A} := D^{\Xi \vdash M(A)}$  over  $S$  with  $f \cdot_E d := f \cdot_D d$ ,  $d \cdot_E G := d \cdot_D (M(G))$ , and  $E(\_)\_ := D(\_)\_^{M \circ \rho}$  is called  $D$  reindexed by  $M$ .

Given object structures  $D_1$  over  $T_1$  and  $D_2$  over  $T_1$  we define the *product object structure*  $D_1 \times D_2$  over  $T_1 \times T_2$  in the obvious way.

## 6.1 Realizability Type Structure and the Fundamental Theorem of Typing

Fix some term-like type structure  $T$  and an object structure  $D$  over  $T$ . Let  $A \in \widehat{D}_{\Xi}^A$  if  $A_{\Delta} \subseteq D_{\Delta}^{\Xi \vdash A}$  and  $\mathcal{A}$  is Kripke.  $\widehat{D}_{\Xi}^A$  forms a complete lattice for all  $\Xi, A$ .

**Definition 16 (Function space and type abstraction on  $\widehat{D}$ ).**

$$\begin{aligned} \_ \rightarrow_{\widehat{D}} \_ &\in \widehat{D}_{\Xi}^A \rightarrow \widehat{D}_{\Xi}^B \rightarrow \widehat{D}_{\Xi}^{A \rightarrow B} \\ (\mathcal{A} \rightarrow \mathcal{B})_{\Delta} &:= \{f \in D_{\Delta}^{\Xi \vdash A \rightarrow B} \mid \text{for all } d, \Delta' \leq \Delta, d \in \mathcal{A}_{\Delta'} \text{ implies } f \cdot d \in \mathcal{B}_{\Delta'}\} \\ (\_ \multimap)_{\Delta}^{\forall^{\kappa} F} &\in (G \in T_{\Xi}^{\kappa}) \rightarrow \widehat{D}_{\Xi}^{F \cdot G} \rightarrow \widehat{D}_{\Xi}^{\forall^{\kappa} F} \\ (G \cdot \mathcal{A})_{\Delta}^{\forall^{\kappa} F} &:= \{d \in D_{\Delta}^{\Xi \vdash \forall^{\kappa} F} \mid d \cdot G \in \mathcal{A}_{\Delta}\} \end{aligned}$$

Constructors of higher kind are interpreted as operators on Kripke sets.

**Definition 17 (Kripke operators of higher kind).** We define  $\widehat{D}_{\Xi}^{E:\kappa}$  by  $\widehat{D}_{\Xi}^{A:*} := \widehat{D}_{\Xi}^A$  and  $\widehat{D}_{\Xi}^{E:\kappa \rightarrow \kappa'} := (G \in T_{\Xi}^{\kappa}) \rightarrow \widehat{D}_{\Xi}^{G:\kappa} \rightarrow \widehat{D}_{\Xi}^{F \cdot G:\kappa'}$ .

**Definition 18 (Type candidate space).** A type candidate space  $\mathcal{C}$  for  $D$  over  $T$  consists of two Kripke sets  $\underline{\mathcal{C}}^{\Xi \vdash A}, \overline{\mathcal{C}}^{\Xi \vdash A} \in \widehat{D}_{\Xi}^A$ , (written  $\underline{A}, \overline{A}$  if no ambiguities arise) for each type  $A \in T_{\Xi}^*$  such that the following conditions hold.

$$\begin{aligned} \underline{H} &\subseteq \overline{H} &\in \widehat{D}_{\Xi}^H &\quad (H \text{ neutral}) &\quad \underline{A \rightarrow B} &\subseteq \overline{A \rightarrow B} &\in \widehat{D}_{\Xi}^{A \rightarrow B} \\ \underline{\forall^{\kappa} F} &\subseteq \overline{G \cdot F \cdot G} &\in \widehat{D}_{\Xi}^{\forall^{\kappa} F} &\quad (G \in T_{\Xi}^{\kappa}) &\quad \underline{A \rightarrow \overline{B}} &\subseteq \overline{A \rightarrow B} &\in \widehat{D}_{\Xi}^{A \rightarrow B} \\ \underline{X \cdot \overline{F} \cdot \overline{X}} &\subseteq \overline{\forall^{\kappa} \overline{F}} &\in \widehat{D}_{\Xi}^{\forall^{\kappa} F} &\quad (X \notin \text{dom}(\Xi)) \end{aligned}$$

**Definition 19 (Realizable semantic types).** If  $F \in T_{\Xi}^{\kappa}$  and  $\mathcal{F} \in \widehat{D}_{\Xi}^{F:\kappa}$  then  $F \Vdash_{\mathcal{C}}^{\kappa} \mathcal{F}$  (pronounced  $F$  realizes  $\mathcal{F}$ ) is defined by induction on  $\kappa$  as follows:

$$\begin{aligned} A \Vdash_{\mathcal{C}}^* \mathcal{A} &:\iff \underline{A} \subseteq \mathcal{A} \subseteq \overline{A} \\ F \Vdash_{\mathcal{C}}^{\kappa \rightarrow \kappa'} \mathcal{F} &:\iff F \cdot G \Vdash_{\mathcal{C}}^{\kappa'} \mathcal{F}(G), \text{ for all } G \Vdash_{\mathcal{C}}^{\kappa} \end{aligned}$$

We define the Kripke families  $\mathcal{T}\widehat{D}_{\Xi}^{\kappa} = \{(F, \mathcal{F}) \in \mathcal{T}_{\Xi}^{\kappa} \times \widehat{D}_{\Xi}^{F:\kappa}\}$  and  $\mathcal{C}_{\Xi}^{\kappa} = \{(F, \mathcal{F}) \in \mathcal{T}_{\Xi}^{\kappa} \times \widehat{D}_{\Xi}^{F:\kappa} \mid F \Vdash_{\mathcal{C}}^{\kappa} \mathcal{F}\}$ . For the remainder of this section, we fix a type candidate space  $\mathcal{C}$ .

**Definition 20 (Interpretation into  $\widehat{D}$ ).** For  $T \in \text{Ty}_{\Xi}^{\kappa}$  and  $(\sigma, \rho) \in \mathcal{T}\widehat{D}_{\Theta}^{\Xi}$  we define  $\widehat{D}[[T]]_{\sigma, \rho} \in \widehat{D}_{\Theta}^{\mathcal{T}[T]_{\sigma}:\kappa}$  as follows:

$$\begin{aligned} \widehat{D}[[X]]_{\sigma, \rho} &:= \rho(X) \\ \widehat{D}[[\lambda X:\kappa. T]]_{\sigma, \rho} &:= ((G, ) \in \mathcal{T}\widehat{D}_{\Theta}^{\kappa}) \mapsto \widehat{D}[[T]]_{(\sigma, \rho)[X \mapsto (G, )]} \\ \widehat{D}[[TU]]_{\sigma, \rho} &:= \widehat{D}[[T]]_{\sigma, \rho}(\mathcal{T}[[U]]_{\sigma}, \widehat{D}[[U]]_{\sigma, \rho}) \\ \widehat{D}[[C]]_{\sigma, \rho} &:= C_{\widehat{D}} \\ \text{where } \forall_{\widehat{D}}^{\kappa} &\in \widehat{D}_{\Xi}^{\forall^{\kappa}:(\kappa \rightarrow \star) \rightarrow \star} \\ \forall_{\widehat{D}}^{\kappa}(F, \mathcal{F}) &:= \bigcap_{G \Vdash^{\kappa} G} \mathcal{F}(G, ) \end{aligned}$$

Since the kind function space is the full set-theoretic one,  $\widehat{D}$  is combinatory and respects type equality.

**Theorem 9 (Realizability).**  $\mathcal{T}\widehat{D}$  is a type structure with application  $(F, \mathcal{F}) \cdot (G, ) = (F \cdot G, \mathcal{F}(G, ))$  and evaluation  $\mathcal{T}\widehat{D}[[T]]_{\sigma, \rho} = (\mathcal{T}[[T]]_{\sigma}, \widehat{D}[[T]]_{\sigma, \rho})$ .  $\mathcal{C}$  is a type substructure of  $\mathcal{T}\widehat{D}$ .

**Theorem 10 (Fundamental theorem of typing).** Let  $D$  be an object structure over  $\mathcal{T}$  and  $\underline{\mathcal{C}}, \overline{\mathcal{C}} \in \widehat{D}$  a type candidate space. Let  $\mathcal{S} \subseteq \mathcal{C}$  be a type substructure of the associated realizability type structure  $\mathcal{C}$ . Then the family  $E_{(\Delta, \Phi)}^{\Xi \vdash (A, A)} := \mathcal{A}_{\Delta}$  is an object substructure of  $D$  reindexed by  $\pi_1 : \mathcal{S} \rightarrow \mathcal{T}$ .

## 6.2 Soundness of NbE for Objects

Term-like object structures and neutral objects are now defined analogously to term-like type structures.

**Definition 21 (Object reification).** Given a term-like type structure  $\mathcal{T}$  and a term-like object structure  $D$  over  $\mathcal{T}$ , we define the relations

$$\begin{aligned} \Xi; \Delta \vdash d \searrow v \uparrow A & \quad d \text{ reifies to } v \text{ at type } A, \\ \Xi; \Delta \vdash e \searrow u \downarrow A & \quad e \text{ reifies to } u, \text{ inferring type } A, \end{aligned}$$

(where  $d, e \in D_{\Delta}^{\Xi \vdash A}$  and  $v, u$  are syntactical objects) inductively by the following rules:

$$\frac{}{\Xi; \Delta \vdash x \searrow x \downarrow \Delta(x)} \quad \frac{\Xi; \Delta \vdash e \searrow u \downarrow A \rightarrow B \quad \Xi; \Delta \vdash d \searrow v \uparrow A}{\Xi; \Delta \vdash e d \searrow u v \downarrow B}$$

$$\frac{\Xi; \Delta \vdash e \searrow u \downarrow \forall^{\kappa} F \quad \Xi \vdash G \searrow V \uparrow \kappa}{\Xi; \Delta \vdash e G \searrow u V \downarrow F \cdot G} \quad \frac{\Xi, X:\kappa; \Delta \vdash f \cdot X \searrow v \uparrow F \cdot X}{\Xi; \Delta \vdash f \searrow \lambda X:\kappa. v \uparrow \forall^{\kappa} F}$$

$$\frac{\Xi; \Delta, x:A \vdash f \cdot x \searrow v \uparrow B \quad \Xi \vdash A \searrow U \uparrow \star}{\Xi; \Delta \vdash f \searrow \lambda x:U. v \uparrow A \rightarrow B} \quad \frac{\Xi; \Delta \vdash e \searrow u \downarrow H}{\Xi; \Delta \vdash e \searrow u \uparrow H} \text{ H neutral}$$

As for types, object reification is deterministic.

Note that we cannot say now in which  $\text{Obj}_F^{\Xi \vdash T}$  the objects  $u$  and  $v$  live. The conjecture is those  $F, T$  with  $\Xi \vdash \Delta \searrow F$  and  $\Xi \vdash A \searrow T \uparrow \star$ . However, this does not follow directly from the definition, it is a consequence of Thm. 12.

**Def. and Lem. 11 (Glueing type candidate space).** *Let  $\mathcal{S} \subseteq \mathcal{T} \times \text{Ty}$  a glueing candidate,  $\Vdash_{\text{Gl}} \mathcal{S}$ . For  $(A, T) \in \mathcal{S}_{\Xi}^*$  we define the Kripke families  $\underline{\text{gl}}^{\Xi \vdash (A, T)}, \overline{\text{gl}}^{\Xi \vdash (A, T)} \in \widehat{D \times \text{Obj}_{\Xi}^{(A, T)}}$  by*

$$\begin{aligned} \overline{\text{gl}}_{(\Delta, \Gamma)}^{\Xi \vdash (A, T)} &:= \{(d, t) \mid \Xi; \Delta \vdash d \searrow v \uparrow A \text{ and } \Xi; \Gamma \vdash t = v : T \text{ for some } v\}, \\ \underline{\text{gl}}_{(\Delta, \Gamma)}^{\Xi \vdash (A, T)} &:= \{(e, t) \mid \Xi; \Delta \vdash e \searrow u \downarrow A \text{ and } \Xi; \Gamma \vdash t = u : T \text{ for some } u\}. \end{aligned}$$

$\text{gl}$  is a type candidate space.

**Theorem 12 (Soundness of NbE for objects).** *Let  $D$  be a term-like object structure over a term-like type structure  $\mathcal{T}$ . If  $\Xi; \Gamma \vdash t : T$  then there is a long normal form  $v$  such that  $\Xi; \mathcal{T}[\![T]\!]_{\text{Var}} \vdash D(\!|t|\!)_{\text{Var}}^{\text{var}} \searrow v \uparrow \mathcal{T}[\![T]\!]_{\text{Var}}$  and  $\Xi; \Gamma \vdash t = v : T$ .*

### 6.3 Completeness of NbE for Objects

Completeness on the object level is shown analogously to completeness on the type level. Define *object groupoids* as groupoidal object structures and show that Kripke function space and impredicative quantification on  $\widehat{D}$  preserve the groupoid structure. Then prove a fundamental theorem of object equality and instantiate it to the type candidate space defined analogously to  $\text{Per}$ . Due to lack of space, we cannot spell out the details and refer to the full version of this article instead [Abe09].

## 7 Conclusion

We have developed type and object structures, which are sorted applicative structures on type and object level, in order to facilitate generic model constructions for System  $F^\omega$  — which are an alternative to categorical semantics [See87] and Bruce-Meyer-Mitchell models [BM84]. Using special instances of kind candidate spaces we have shown soundness and completeness of an abstract normalization by evaluation algorithm for types. We have gone on to show soundness and completeness of NbE for objects.

We seek to extend NbE to the Calculus of Constructions, using ideas from this work. Due to dependency, type and object levels cannot be defined in sequence, but must be defined simultaneously; this seems to be the main remaining technical difficulty.

## References

- [Abe08] Abel, A.: Weak  $\beta\eta$ -normalization and normalization by evaluation for System F. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 497–511. Springer, Heidelberg (2008)
- [Abe09] Abel, A.: Typed applicative structures and normalization by evaluation for System  $F^\omega$  (full version) (2009), <http://www.tcs.ifi.lmu.de/~abel/fomegaNbe.pdf>

- [ACD07] Abel, A., Coquand, T., Dybjer, P.: Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In: LICS 2007, pp. 3–12. IEEE CS Press, Los Alamitos (2007)
- [ACD08] Abel, A., Coquand, T., Dybjer, P.: Verifying a semantic  $\beta\eta$ -conversion test for Martin-Löf type theory. In: Audebaud, P., Paulin-Mohring, C. (eds.) MPC 2008. LNCS, vol. 5133, pp. 29–56. Springer, Heidelberg (2008)
- [ADHS01] Altenkirch, T., Dybjer, P., Hofmann, M., Scott, P.J.: Normalization by evaluation for typed lambda calculus with coproducts. In: LICS 2001, pp. 303–310. IEEE CS Press, Los Alamitos (2001)
- [AHS96] Altenkirch, T., Hofmann, M., Streicher, T.: Reduction-free normalisation for a polymorphic system. In: LICS 1996, pp. 98–106. IEEE CS Press, Los Alamitos (1996)
- [Bar84] Barendregt, H.: The Lambda Calculus: Its Syntax and Semantics. North Holland, Amsterdam (1984)
- [Bar08] Barral, F.: Decidability for non-standard conversions in lambda-calculus. PhD thesis, Ludwig-Maximilians-University, Munich (2008)
- [BCF04] Balat, V., Di Cosmo, R., Fiore, M.P.: Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In: POPL 2004, pp. 64–76. ACM, New York (2004)
- [BM84] Bruce, K.B., Meyer, A.R.: The semantics of second order polymorphic lambda calculus. In: Plotkin, G., MacQueen, D.B., Kahn, G. (eds.) Semantics of Data Types 1984. LNCS, vol. 173, pp. 131–144. Springer, Heidelberg (1984)
- [BS91] Berger, U., Schwichtenberg, H.: An inverse to the evaluation functional for typed  $\lambda$ -calculus. In: LICS 1991, pp. 203–211. IEEE CS Press, Los Alamitos (1991)
- [CAM07] Chapman, J., Altenkirch, T., McBride, C.: Epigram reloaded: a standalone type-checker for ETT. In: TFP 2005. Trends in Functional Programming, vol. 6, pp. 79–94. Intellect (2007)
- [CD97] Coquand, T., Dybjer, P.: Intuitionistic model constructions and normalization proofs. MSCS 7(1), 75–94 (1997)
- [Coq96] Coquand, T.: An algorithm for type-checking dependent types. In: MPC 1995. SCP, vol. 26, pp. 167–177. Elsevier, Amsterdam (1995)
- [Dan99] Danvy, O.: Type-directed partial evaluation. In: Hatcliff, J., Thiemann, P. (eds.) DIKU 1998. LNCS, vol. 1706, pp. 367–411. Springer, Heidelberg (1999)
- [Dyb00] Dybjer, P.: A general formulation of simultaneous inductive-recursive definitions in type theory. JSL 65(2), 525–549 (2000)
- [GL02] Grégoire, B., Leroy, X.: A compiled implementation of strong reduction. In: ICFP 2002. SIGPLAN Notices, vol. 37, pp. 235–246. ACM, New York (2002)
- [GLT89] Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in TCS, vol. 7. CUP (1989)
- [INR07] INRIA. The Coq Proof Assistant, Version 8.1. INRIA (2007), <http://coq.inria.fr/>
- [Nor07] Norell, U.: Towards a practical programming language based on dependent type theory. PhD thesis, Chalmers, Göteborg, Sweden (2007)
- [Pol94] Pollack, R.: Closure under alpha-conversion. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 313–332. Springer, Heidelberg (1994)
- [See87] Seely, R.A.G.: Categorical semantics for higher order polymorphic lambda calculus. JSL 52(4), 969–989 (1987)

# Jumping Boxes<sup>\*</sup>

## Representing Lambda-Calculus Boxes by Jumps

Beniamino Accattoli<sup>1</sup> and Stefano Guerrini<sup>2</sup>

<sup>1</sup> Dip. di Informatica e Sistemistica A. Ruberti, Sapienza Università di Roma  
Via Ariosto, 25 - 00185 Roma, Italy  
beniamino.accattoli@gmail.com

<sup>2</sup> Dipartimento di Informatica, Sapienza Università di Roma  
Via Salaria, 113 - 00198 Roma, Italy  
guerrini@di.uniroma1.it

**Abstract.** Boxes are a key tool introduced by linear logic proof nets to implement lambda-calculus beta-reduction. In usual graph reduction, on the other hand, there is no need for boxes: the part of a shared graph that may be copied or erased is reconstructed on the fly when needed. Boxes however play a key role in controlling the reductions of nets and in the correspondence between nets and terms with explicit substitutions.

We show that boxes can be represented in a simple and efficient way by adding a jump, i.e. an extra connection, for every explicit sharing position (exponential cut) in the graph, and we characterize our nets by a variant of Lamarche’s correctness criterion for essential nets. The correspondence between explicit substitutions and jumps simplifies the already known correspondence between explicit substitutions and proof net exponential cuts.

## 1 Introduction

Graph reduction [14,11] and proof nets [4,13] are two well established approaches based on graphs for the analysis and the design of implementations of  $\lambda$ -calculus  $\beta$ -reduction. The first one has mainly been driven by the practice of the implementation of functional languages, the second one by the Curry-Howard correspondence between programs and proofs extended to linear logic.

One of the main logical and computational novelties introduced by linear logic is the clear decomposition of  $\beta$ -reduction between logical and structural rules, and the use of boxes to represent the part of a term/proof that must be treated as a unique block. In proof nets, boxes enclose subnets that have a non-linear behaviour. In graph reduction instead there is no notion of boxes: the parts of a graph that can be shared or must be erased or copied are determined by need. Remarkably,  $\lambda$ -calculus proof nets [13] and graph-reduction do not rest on any notion of typing (therefore we shall consider untyped  $\lambda$ -calculus only).

Another key feature of linear logic nets is the use of correctness criteria for the characterization of the nets that are the image of a proof/program. Graph-reduction instead

---

<sup>\*</sup> Partially supported by the MIUR PRIN grant “CONCERTO” and by the Sapienza S.M.F.N. grant “Applicazione di Strumenti Logici alla Progettazione e Analisi di Sistemi Software”.

does not rest on any notion of correctness criterion, but only on the analysis of the reducts of some graph representation of a  $\lambda$ -term.

One could try to combine the two approaches, proof nets and graph-reduction with sharing, following a net style definition based on some correctness criterion, but avoiding any explicit marking of boxes. Unfortunately, such a solution works almost smoothly in the case without weakening only (the so-called  $\lambda I$ -calculus); moreover, in many cases, boxes play a key role in the fine correspondence with the calculus, or even in its definition, and then cannot be completely ignored.

In the literature boxes have alternatively been represented as sets of links [13], or introducing additional edges (or links) marking the border of every box [9], or by some additional distributed information that allows to recover them (e.g., by indexing the nodes/links of the structure [6]). We show a way of representing boxes by means of jumps. Jumps are a well-known tool for defining dependencies in proof nets, introduced by Girard in [5], and which have been recently used to analyze and control the sequentialization of multiplicative proof nets [3]. Our jump representation of boxes seems a fair compromise between the box-free approach of graph reduction and the usual proof net approach, since to characterize the boxing of a  $\lambda$ -dag (our graph representation of  $\lambda$ -terms with sharing), we add only few local constraints, one for each sharing node. Moreover, if we aim at modeling and studying explicit substitutions via proof nets, we get even more information than the information usually encoded in boxes: the quotient induced by jumps is exactly the one given by the permutation of independent substitutions, whereas in the usual approaches based on boxes, the quotient is more primitive, and the corresponding nets are more difficult to sequentialize (i.e., the readback of a  $\lambda$ -term with explicit substitution from a correct net is less direct) [27]. Summarizing, the most surprising facts concerning the use of jumps are that: (i) they naturally allow to associate a connected dag with every  $\lambda$ -term with explicit substitutions; (ii) they lead to a simple correctness criterion that is just an extension of Lamarche's correctness criterion for multiplicative essential nets [8,10]; (iii) any correct  $\lambda$ -dag has just one unique correct box assignment and a unique sequentialization, up to the permutation of independent substitutions (that is, up to changing the order of sequentialization of two disjoint shared subterms); (iv) boxes are implicit, induced by a local decoration of  $\lambda$ -dags (with jumps), instead of being explicitly spatially bound.

To give an idea of how jumps work, Fig. 1 analyzes two terms:  $x[M/x][N/y]$  and  $x[M[N/y]/x]$ , with  $y \in M$ . First,  $a$  shows the graph with neither boxes nor jumps, which is the same for both terms. Then  $b_1$  and  $b_2$  show how boxes make the difference between the two, and then  $c_1$  and  $c_2$  show how using jumps one obtains the same effect.

The results presented in the paper can be easily reformulated and stated on pure proof nets [13]. Nevertheless we have chosen a graph notation closer to the  $\lambda$ -calculus and in the style of that used in term graph rewriting [12] for three main reasons: (i) this notation does not require any linear logic notation; (ii) our approach does not depend on any encoding of  $\lambda$ -calculus into linear logic (and then does not depend on any calling or evaluation mechanism); (iii) we think that it is important to export characterizations based on correctness criteria to graphical formalisms different from linear logic proof nets.

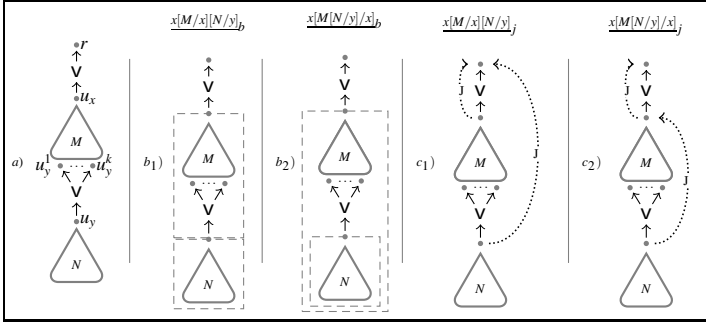


Fig. 1. Boxes and jumps

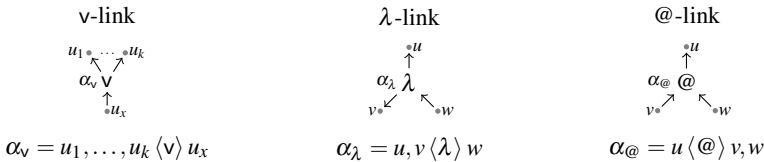
In section 2 we present the graphs, named  $\lambda$ -trees, that we shall use to represent  $\lambda$ -terms and we introduce the main notations and formalisms. We shall give a correctness criterion for  $\lambda$ -structure which allows to associate a readback to every correct  $\lambda$ -structure, and we shall define their sharing reductions, introducing the connections between sharing  $\lambda$ -structures and explicit substitutions. In section 3 we introduce jumps. We show how to use them to represent boxes, and how they allow to obtain a faithful and economic representation of explicit substitutions.

## 2 Graph Reduction Reloaded

In this section we introduce our graphical formalism. As customary in the linear logic literature we define a set of graphs, here called  $\lambda$ -structures, which act as candidates for representing  $\lambda$ -terms, and a correctness criterion that precisely characterizes the subset of  $\lambda$ -structures corresponding to  $\lambda$ -terms (without explicit substitutions, in this section). We use the case of ordinary  $\lambda$ -terms as a guide to gradually introduce our formalism, and the example of Fig. 1 to concretely explain our definitions.

### 2.1 $\lambda$ -Structures, $\lambda$ -Trees

A  $\lambda$ -structure  $G$  is a directed graph over a set of nodes  $N_G$  connected by a set of links  $L_G$ . Every link has a set of incoming/outgoing edges that connects it to a set of nodes and is of one of the following types:



- A v-link  $\alpha_v$  has an arbitrary number  $k \geq 0$  of outgoing edges, the *occurrence edges*  $(\alpha_v, u_i)$ , and one incoming edge, the *variable edge*  $(u_x, \alpha_v)$ . The number  $k$  is the cardinality of the v-link and  $u_x$  its v-node.



- A  $\lambda$ -link  $\alpha_\lambda$  has two outgoing edges, the *abstraction edge*  $(\alpha_\lambda, u)$  and the *binding edge*  $(\alpha_\lambda, v)$ , and one incoming edge, the *body edge*  $(w, \alpha_\lambda)$ .
- A  $@$ -link  $\alpha_@$  has one outgoing edge, the *composition edge*  $(\alpha_@, u)$ , and two incoming edges, the *function edge*  $(v, \alpha_@)$  and the *argument edge*  $(w, \alpha_@)$ .

The nodes and links of a  $\lambda$ -structure satisfy the following additional conditions:

*Node*: every node is connected to a link and has at most an incoming edge and at most an outgoing edge.

*Lambda*: the binding edge of a  $\lambda$ -link must be connected to a  $v$ -node.

*Output*:  $G$  has only one node with no outgoing edges, named the *root* of  $G$ .

*Input*: every node of  $G$  with no incoming edges is a  $v$ -node.

The occurrence edges of a  $v$ -link are not ordered. Given a permutation  $i_1, \dots, i_k$  of  $1, \dots, k$ , the same  $v$ -link is denoted by  $u_1, \dots, u_k \langle v \rangle u_0$  and  $u_{i_1}, \dots, u_{i_k} \langle v \rangle u_0$ .

A  $\lambda$ -structure is uniquely determined by its set of links. We shall use “ $\cup$ ” to denote the union of two sets of links and we shall omit curly brackets for singletons; thus,  $\alpha_1; \dots; \alpha_n$  is the  $\lambda$ -structure corresponding to the set  $\{\alpha_i\}_{1 \leq i \leq n}$ . We shall also write  $\alpha \in G$ , to mean that  $\alpha$  is a link of  $G$ . For instance, the  $\lambda$ -structure at  $u_{@_1}$  in Fig. 2d is denoted by:  $u_{@_1} \langle @ \rangle u_{@_2}, u_{\lambda_2}; u_{@_2} \langle @ \rangle u_{z_1}^1, u_{z_1}^2; u_{z_1}^1, u_{z_1}^2 \langle v \rangle u_{z_1}; u_{\lambda_2}, u_{y_1} \langle \lambda \rangle u_{y_2}^1; u_{y_2}^1 \langle v \rangle u_{y_2}; \langle v \rangle u_{y_1}$

**Nodes and links terminology.** A node with no incoming edges is an *input* (node), as  $u_{z_1}$ ,  $u_{z_2}$  and  $u_{y_2}$  in Fig. 2d, while a node with no outgoing edges is an *output* (node) (as  $r$ ). A  $v$ -node (defined above) is *free* if it is an input; it is *abstracted* if it is connected to a  $\lambda$ -link by a binding edge (as  $u_{x_2}$  and  $u_{y_1}$ ); otherwise, it is *substituted* (as  $u_{x_1}$ ). A  $v$ -link is free/abstracted/substituted if the corresponding  $v$ -node is free/abstracted/substituted. We shall sometimes call *substitution* a substituted  $v$ -node/link.

A  $v$ -link with cardinality  $k$  is a *weakening* when  $k = 0$  (as  $\alpha_{w_1}$  and  $\alpha_{w_2}$ ), a *dereliction* when  $k = 1$ , (as  $\alpha_{d_1}$ ,  $\alpha_{d_2}$  and  $\alpha_{d_3}$ ), or a *contraction* otherwise (as  $\alpha_c$ ). A  $v$ -node is a weakening, a dereliction, or a contraction depending on the type of the corresponding  $v$ -link. A weakening is isolated when it is free, i.e., when its  $v$ -node is an input (as  $\alpha_{w_1}$ ).

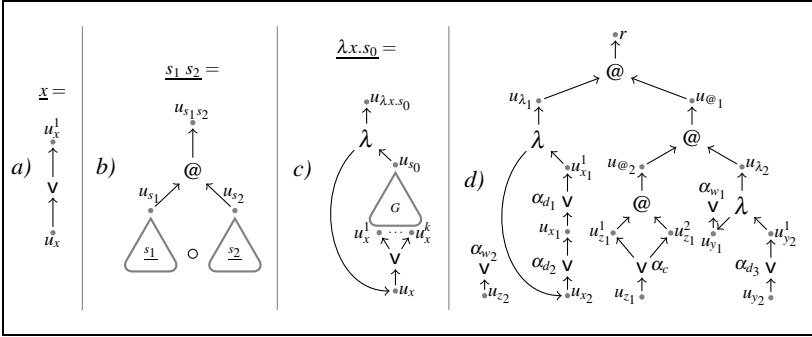
**Translation of  $\lambda$ -terms.** A  $\lambda$ -term  $t$  with the disjoint variable property (otherwise, take any  $\alpha$ -equivalent one with the property) is translated into a  $\lambda$ -structure  $\underline{t}$  with a node  $u_{\underline{s}}$  for every subterm  $s$  of  $t$ , plus a node  $u_x$  for every variable  $x$  (either free or abstracted), by (see Fig. 2a-c)

$$\underline{x} = u_x^1 \langle v \rangle u_x$$

$$s_1 s_2 = u_{s_1 s_2} \langle @ \rangle u_{s_1}, u_{s_2}; (s_1 \circ s_2) \xrightarrow{\lambda x. s_0} = \begin{cases} u_{\lambda x. s_0}, u_x \langle \lambda \rangle u_{s_0}; s_0 & x \in \text{FV}(s_0) \\ u_{\lambda x. s_0}, u_x \langle \lambda \rangle u_{s_0}; \langle v \rangle u_x; s_0 & x \notin \text{FV}(s_0) \end{cases}$$

where  $\text{FV}(t)$  is the set of free variables of  $t$ .

For every subterm  $s$ , the node  $u_{\underline{s}}$  is the root of  $\underline{s}$ . For every variable  $x$  in  $s$ , the  $\lambda$ -structure  $\underline{s}$  contains a  $v$ -link  $u_x^1, \dots, u_x^k \langle v \rangle u_x$ , where every node  $u_x^i$  corresponds to an occurrence of  $x$  in  $s$ . The inputs (free  $v$ -nodes) of  $\underline{s}$  are exactly the nodes  $u_x$  s.t.  $x$  is a free variable of  $s$ . The abstraction case in Fig. 2 assumes  $s_0 = G; u_x^1, \dots, u_x^k \langle v \rangle u_x$ . It may happen that in an application  $s_1 s_2$  some free variables in  $s_1$  occur free in  $s_2$ . In that case the two  $v$ -links  $u_1, \dots, u_k \langle v \rangle u \in \underline{s_1}$  and  $u_{k+1}, \dots, u_{k+h} \langle v \rangle u \in \underline{s_1}$  corresponding to each of these variables are replaced in  $\underline{s_1 s_2}$  by a unique link  $u_1, \dots, u_k, u_{k+1}, \dots, u_{k+h} \langle v \rangle u$ . This *gluing* of the common variables in  $\underline{s_1}$  and  $\underline{s_2}$  is denoted by  $\underline{s_1} \circ \underline{s_2}$  (not explicitly



**Fig. 2.** *a,b,c*) Translation of  $\lambda$ -terms; *d*) The guiding example (names on nodes and  $v$ -links)

represented on Fig. 2b, we just denote it with  $\circ$ ). An example: the  $v$ -link  $\alpha_c$  in Fig. 2d can be considered as coming from the gluing  $\underline{z}_1 \circ \underline{z}_1$  required by  $\underline{z}_1 \underline{z}_1$ .

Given a superset  $X$  of the free variables of  $t$ , the translation  $\underline{t}_X$  of  $t$  w.r.t. the context  $X$  is obtained by adding a weakening for every variable in the context which is not a free variable of  $t$ . namely, given  $X \supseteq \text{FV}(t)$ ,  $\underline{t}_X = \underline{t} \circ \{\langle v \rangle u_x\}_{x \in X} = \underline{t}; \{\langle v \rangle u_x\}_{x \in (X \setminus \text{FV}(t))}$ .

For instance, the example in Fig. 2 lies in the context  $\{z_1, z_2, y_2\}$ .

**Definition 1 ( $\lambda$ -tree).** A  $\lambda$ -tree  $T$  is a  $\lambda$ -structure s.t.  $T = \underline{t}_X$  for some  $\lambda$ -term  $t$  and some context  $X \supseteq \text{FV}(t)$ .

A  $\lambda$ -tree has no substituted  $v$ -link (then, the guiding example in Fig. 2d is not a  $\lambda$ -tree) and, by erasing its binding edges and  $v$ -links, we obtain the syntax tree of the translated  $\lambda$ -term.

**Paths.** The correctness criterion for  $\lambda$ -trees will be formulated in terms of node-to-node directed paths over a  $\lambda$ -structure  $G$ .

**Definition 2 (correction graph).** Given a  $\lambda$ -structure  $G$ , its correction (directed) graph  $G^*$  is obtained by reversing the orientation of the binding edges in  $G$ .

We reverse the orientation of binding edges, instead of erasing them, because this allows a simpler and proper formulation of the main properties of  $\lambda$ -structures and of  $\lambda$ -structures with jumps which we shall introduce in section 3 (for instance see Theorem 1). But we prefer not to endow  $\lambda$ -structures with this orientation, as this would introduce nodes with more than one outgoing edge.

Given two nodes  $u_1, u_2$  of  $G$ , we write  $u_1 \leq u_2$  when in  $G^*$  there is a directed path from  $u_1$  to  $u_2$ . We shall also say that a node  $u_2$  dominates a node  $u_1$ , written  $u_1 \sqsubseteq u_2$ , when in  $G^*$  there is a path from  $u_1$  to  $u_2$  and any path starting from  $u_1$  either contains or can be prolonged into a path that contains  $u_2$ . Formally,  $u_1 \sqsubseteq u_2$  iff  $u_1 \leq u_2$  and for every  $u$  s.t.  $u_1 \leq u$ , either  $u \leq u_2$  or  $u_2 \leq u$ .

We consider node-to-node paths only, but we extend notations and terminology to links. We write  $u \leq \alpha$ , when there exists a directed path from a node  $u$  to a node  $u'$  which contains an incoming edge of  $\alpha$  in the correction graph, and we say that such a

path *passes through*  $\alpha$ . A link  $\alpha$  dominates a node  $u$ , written  $u \sqsubseteq \alpha$ , when (in  $G^*$ ) every directed path from  $u$  passes through  $\alpha$  or can be prolonged to do so.

**Correctness criterion.**  $\lambda$ -trees can be characterized by a correctness criterion for  $\lambda$ -structures that is a variant of Lamarche's criterion for multiplicative essential nets [8] (a good reference for the multiplicative case is [10] also). First of all we need an acyclicity condition stating that the only cycles a  $\lambda$ -structure may contain are those introduced by the binding edges linking abstractions to their (non-weakening) variables.

(D) *DAG*: the directed graph  $G^*$  obtained from  $G$  by reversing the orientation of its binding edges has no directed cycle, namely it is a DAG.

Condition (D) does not suffice to characterize  $\lambda$ -trees among the  $\lambda$ -structures with no substituted  $\nu$ -link. We need a condition stating that the binding edges in the  $\lambda$ -structure induce a correct mapping between variables and binders in the corresponding  $\lambda$ -term. For instance (see Fig. 3c),  $u_{@} \langle @ \rangle u_{\lambda}, u_x^2; u_{\lambda}, u_x \langle \lambda \rangle u_x^1; u_x^1, u_x^2 \langle \nu \rangle u_x$  cannot be correct, since it would correspond to a  $\lambda$ -term  $(\lambda x.x)x$ , in which the  $\lambda$  binds the occurrence of a variable that is not in its body.

(L) *Lamarche*: Let  $u_x$  be a  $\nu$ -node of a  $\lambda$ -structure. If  $u_x$  is abstracted by the  $\lambda$ -link  $\alpha_{\lambda} = u_{\lambda}, u_x \langle \lambda \rangle u_b$ , then  $u_x \sqsubseteq \alpha_{\lambda}$ .

Notice that the domination in condition (L) is relative to the link, and that, in the correction graph, abstraction links have their binding edges reversed. This implies that, in the case of an abstracted weakening  $\langle \nu \rangle u_x$ , the node  $u_x$  is dominated by its binder  $\alpha_{\lambda} = u_{\lambda}, u_x \langle \lambda \rangle u_b$ , since every non-trivial node-to-node path from  $u_x$  passes through  $u_{\lambda}$ , and then through  $\alpha_{\lambda}$ . We also remark that, in a  $\lambda$ -structure satisfying condition (D),  $u \sqsubseteq u'$  iff  $u \leq u'$  and all the paths from  $u$  to the root pass through  $u'$ .

**Definition 3 (L-correctness).** A  $\lambda$ -structure is L(amarche)-correct if condition (D) holds and condition (L) holds for every abstracted  $\nu$ -node.

**Readback.** To read a term back from an L-correct  $\lambda$ -structure  $T$  with no substitutions we need to reason about subgraphs of a  $\lambda$ -structure.

**Definition 4 (Substructures and dominions).** Let  $G$  be a  $\lambda$ -structure. A  $\lambda$ -structure  $H$  s.t.  $N_H \subseteq N_G$  is a substructure of  $G$ , say  $H \triangleleft G$ , when every  $\lambda / @$  link of  $H$  is a link of  $G$  and, for every  $\nu$ -link  $u_1, \dots, u_h \langle \nu \rangle u_0$  of  $H$  there is a corresponding  $\nu$ -link  $u'_1, \dots, u'_k \langle \nu \rangle u_0$  of  $G$  s.t.  $u_1, \dots, u_h$  is a subset of  $u'_1, \dots, u'_k$ .

Given a node  $u$  of  $G$ , its dominion  $G \downarrow_u$  is a subgraph of  $G$  that contains every node and every link dominated by  $u$  and completed by a suitable set of  $\nu$ -links. Namely,  $G \downarrow_u$  is the least graph s.t. every  $u' \sqsubseteq u$  is a node of  $G \downarrow_u$  and, for every  $\alpha \in G$ : (i) if  $\alpha$  is a  $\lambda$  or  $@$ -link,  $\alpha \in G \downarrow_u$  iff  $u' \sqsubseteq u$ , for every node  $u'$  of  $\alpha$ ; (ii) if  $\alpha = u_1, \dots, u_k \langle \nu \rangle u_0 \in G$ , with  $h \geq k \geq 0$  and  $u_i \sqsubseteq u$  for  $i \leq k$ , then the  $\nu$ -link  $\alpha' = u_1, \dots, u_k \langle \nu \rangle u_0 \in G \downarrow_u$  iff either (a)  $k > 0$  and  $u_i \not\sqsubseteq u$  for  $i > k$  or (b)  $h = 0$  and  $u_0 \sqsubseteq u$  but  $u_0 \neq u$ .

The aim of the previous definition is to ensure that, in a correct  $\lambda$ -structure, the dominion of a node  $u$  is a  $\lambda$ -structure with root  $r$  that contains every node and link dominated

by  $u$ . We cannot simply say that  $G \downarrow_u$  is the set of links dominated by  $u$  since, in order to get a  $\lambda$ -structure, we have to add a  $\nu$ -node/link in the following cases: when  $u$  dominates only some of the occurrences of a variable (in Fig. 2d the dominion of  $u_{z_1}^1$  is  $u_{z_1}^1 \langle \nu \rangle u_{z_1} \rangle$ ); when  $u$  dominates a  $\lambda$ -link that binds a weakening (indeed  $G \downarrow_{u_{z_2}}$  contains  $\alpha_{w_1} = \langle \nu \rangle u_{y_1} \rangle$ ); when  $u$  dominates a bound variable but it does not dominate its binder ( $G \downarrow_{u_{x_1}}$  is  $u_{x_1} \langle \nu \rangle u_{x_2}$ , even if  $u_{x_2} \not\sqsubseteq u_{x_1}$ ).

Let  $G$  be an L-correct  $\lambda$ -structure without substitutions, and  $u$  one of its nodes, non minimal w.r.t.  $\leq$ . It is easily seen that  $G \downarrow_u$  is a  $\lambda$ -structure, and a substructure of  $G$ , i.e.,  $G \downarrow_u \triangleleft G$ . Moreover  $G \downarrow_u$  is L-correct. This suggests the definition of a recursive map that associates a  $\lambda$ -term  $\overline{G}$  to any L-correct  $\lambda$ -structure  $G$  that does not contain substitutions.

**Definition 5 (Readback).** *Given an L-correct  $\lambda$ -structure  $G$  with root  $r$  that does not contain substitutions, let us associate a distinct variable with every  $\nu$ -node of  $G$  and let us denote by  $u_x$  the  $\nu$ -node associated with the variable  $x$ . The corresponding readback of  $G$  is the  $\lambda$ -term recursively defined by*

$$r \langle \nu \rangle u_x = x \quad r, u_x \langle \lambda \rangle u_0; \overline{G} = \lambda x. \overline{G \downarrow_{u_0}} \quad r \langle @ \rangle u_1, u_2; \overline{G} = \overline{G \downarrow_{u_1}} \overline{G \downarrow_{u_2}}$$

where  $u_x$  denotes a  $\nu$ -node with which the variable  $x$  has been associated.

By definition, the readback depends on the names assigned to the free variables, and equates L-correct  $\lambda$ -structures differing only for isolated weakenings. As an example, the readback of the sub-graph rooted at  $u_{@_1}$  in Fig. 2d is  $(z_1 \ z_1) \lambda y_1. y_2$ .

**Proposition 1.** *Let  $G$  be a  $\lambda$ -structure in which no  $\nu$ -node is a substitution. Connectedness: If  $G$  is L-correct then  $G = G \downarrow_r; W$ , where  $W$  is the set of isolated weakenings of  $G$ . Sequentialization:  $G$  is a  $\lambda$ -tree iff it is L-correct.*

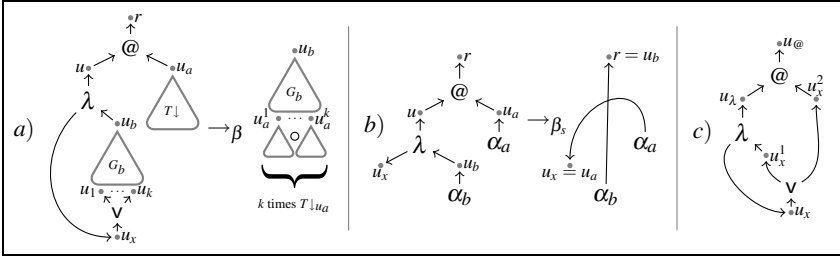
**Contexts.** We shall write  $G = G'[H_1, \dots, H_k]$  when a  $\lambda$ -structure  $G$  factors as  $G'[u_1, \dots, u_k] \circ H_1 \circ \dots \circ H_k$  where: (i)  $G'[u_1, \dots, u_k]$  is a  $\lambda$ -structure (that may not satisfy the *input condition* of  $\lambda$ -structures), called *context*; (ii) each  $H_i$  is a  $\lambda$ -structure; (iii)  $u_1, \dots, u_k$  are inputs of  $G'$ , called the *holes* of the context; (iv) each  $u_i$  is also the root of the corresponding  $\lambda$ -structure  $H_i$ . Here the notion of gluing is stronger than the one defined above. It may happen that some of the  $\nu$ -nodes appearing free in one of the  $H_i$  are abstracted or substituted in  $G$ . For this reason from now on gluing is not restricted to pairs of free  $\nu$ -links, but we also allow a free  $\nu$ -link of  $H_i$  to be glued to a abstracted/substituted  $\nu$ -link in  $G'$  (assuming that they denote the same variable). For instance, let  $G$  be the left subtree of the root  $@$ -link in Fig. 2d. It can be factorized as  $G[H]$ , where  $G[u_{x_1}^1] = u_{\lambda_1}, u_{x_2} \langle \lambda \rangle u_{x_1}^1; \langle \nu \rangle u_{x_2} \rangle$  and  $H = u_{x_1}^1 \langle \nu \rangle u_{x_1}; u_{x_1} \langle \nu \rangle u_{x_2}$ . The reader is invited to check that  $G$  can be also factorized with respect to  $H' = u_{x_1} \langle \nu \rangle u_{x_2}$  and  $H'' = u_{x_1}^1 \langle \nu \rangle u_{x_1}$ .

**$\beta$ -reduction.** Given a  $\lambda$ -tree  $T$ , a  $\beta$ -redex at the node  $r$  is a  $\lambda$ -tree (see Fig. 3a)

$$R = T \downarrow_r = r \langle @ \rangle u, u_a; u, u_x \langle \lambda \rangle u_b; T \downarrow_{u_b} \circ T \downarrow_{u_a}$$

If  $T \downarrow_{u_b} = G_b[u_1, \dots, u_k]; u_1, \dots, u_k \langle \nu \rangle u_x$ . The  $\beta$ -contraction of  $R$  in  $T$  is

$$T[T \downarrow_r] \rightarrow T[G_b[T \downarrow_{u_a}^1, \dots, T \downarrow_{u_a}^k]]$$



**Fig. 3.** a-b)  $\lambda$ -tree ( $\beta$ ) and sharing ( $\beta_s$ )  $\beta$ -rule; c) an example of incorrect  $\lambda$ -structure

where  $T \downarrow_{u_a}^i$  denotes an isomorphic copy of  $T \downarrow_{u_a}$  of root  $u_a^i$  in which the free  $v$ -nodes are the same of  $T \downarrow_{u_a}$ , while all the other nodes are distinct. This rule is illustrated in Fig. 3. As usual,  $\beta$ -reduction is the transitive and reflexive closure of  $\beta$ -contraction.  $\beta$ -reduction of  $\lambda$ -trees preserves L-correctness and simulates the corresponding reduction of  $\lambda$ -terms. We overload the notation  $\rightarrow$ : it denotes  $\beta$ -reduction on both  $\lambda$ -terms and  $\lambda$ -trees

**Proposition 2.** For any  $\lambda$ -term  $t$ . If  $\underline{t}_X \rightarrow^* S$ , then  $t \rightarrow^* s$  for some  $\lambda$ -term  $s$  s.t.  $\underline{s}_X = S$ . Moreover, if  $t \rightarrow^* s$ , then  $\underline{t}_X \rightarrow^* \underline{s}_X$ .

### 2.2 Sharing Reductions and Explicit Substitutions

One of the main advantages in moving from term to graph rewriting is that one can easily share multiple copies of the same term. Duplication can be performed locally, as in the so-called optimal implementations approach [1], or globally, by duplicating a whole subterm [14, 11]. In this paper we shall not analyze optimal implementations, but concentrate on implementing global duplications.

**Sharing  $\beta$ -rule.** The sharing version of the  $\beta$ -rule does not execute the instantiation of the substituting variables of the redex by copying the subgraph corresponding to the argument of the redex, but denotes it by a substitution, and leaves that task up to the rules which manipulate substitutions (the C,W,D-rules in Fig. 4).

A redex for the sharing  $\beta$ -rule, or  $\beta_s$ -rule (see Fig. 3b), is just a pair of  $\lambda$ / $@$ -links  $r \langle @ \rangle u, u_a; u, u_x \langle \lambda \rangle u_b$ ; its contraction erases the  $\lambda$ / $@$ -links, merges the root node  $r$  of the redex with the root  $u_b$  of the body of the  $\lambda$ -link, and directly connects the  $v$ -link abstracted by the  $\lambda$ -link to the root of the argument (i.e., the root  $u_a$  of the argument is merged with the  $v$ -node  $u_x$  of the  $v$ -link).

**Explicit substitutions.** The reduct of  $(\lambda x.t)s$  by the sharing  $\beta$ -rule (see Fig. 4d) is no longer a  $\lambda$ -tree (because of its substituted  $v$ -node), but can be described in the term calculus introducing a new operator, the so-called explicit substitution. An explicit substitution  $t[s/x]$  (which can also be written  $\text{let } x = s \text{ in } t$ ) denotes that a term  $s$  should be replaced for the occurrences of  $x$  in  $t$  and acts as a binder (see [7] for an account of the main results on explicit substitutions). In the  $\lambda$ -calculus with explicit substitutions, the  $\beta$ -rule becomes

$$(\lambda x.t)s \rightarrow_{\beta_{es}} t[s/x]$$

We extend the translation of  $\lambda$ -terms to the case with explicit substitution by adding (see Fig. 4d)

$$t[s/x] = \underline{t[s]}_x = \underline{t[u_x]}_x \circ \underline{s}$$

where  $\underline{t[u_x]}_x = \underline{t} \circ \langle v \rangle u_x$  and  $u_x$  is the root of  $\underline{s}$ . In the  $\lambda$ -structure  $\underline{t[s/x]}$ , a substitution  $t[s/x]$  maps then to a substituted  $v$ -node, whose outgoing edge is connected to the  $v$ -link corresponding to  $x$  in  $\underline{t}$  (a weakening if  $x \notin \text{FV}(t)$ ), while its incoming edge is connected to the root link of  $\underline{s}$ . The example in Fig 2d shows the translation of  $(\lambda x_1.(x_2[x_1/x_2]))((z_1 z_1)(\lambda y_1.y_2))$  in the context  $\{z_1, z_2, y_2\}$ . The above definitions imply that  $(\lambda x.t)_{s_X} \rightarrow_{\beta} \underline{t[s/x]}_X$ .

**Definition 6 ( $\lambda$ -dag).** A  $\lambda$ -dag (directed acyclic graph)  $D$  is a  $\lambda$ -structure s.t.  $D = \underline{t}_X$ , for some  $\lambda$ -term with explicit substitutions  $t$  and some context  $X \supseteq \text{FV}(t)$ .

Unfortunately, the correctness criterion in Def. 3 allows to characterize  $\lambda$ -dags only in the case without weakening—the main reason being that we loose the property that every node is connected to the root.

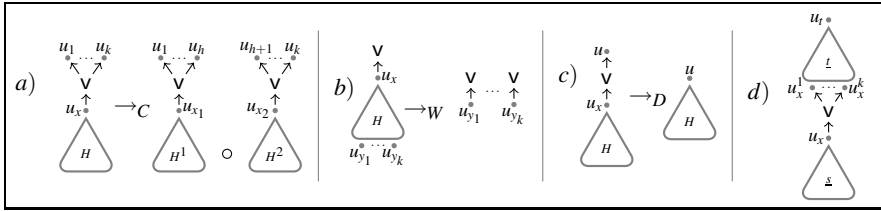
**Boxes.** In 2 boxes contain the (translation of the) term to be substituted by an explicit substitution, so that whenever duplication or erasing is required there is no ambiguity of which part of the graph should be involved. Let us start with a naive notion of box.

**Definition 7 ( $\lambda$ -box).** Given a  $\lambda$ -structure  $G$  a  $\lambda$ -box with principal port  $u$  in  $G$  is a  $\lambda$ -structure  $H \triangleleft G$  with root  $u$ .

Generally, in a  $\lambda$ -structure  $G$ , there are many distinct  $\lambda$ -structures  $H \triangleleft G$  with the same root  $u$  (consider Fig. 1a). Therefore, if we do not associate an explicit box with every substituted  $v$ -node, when we want to apply a contraction or an erasing at a given substituted  $v$ -node  $u$  we would have to non-deterministically choose a substructure rooted at  $u$  as the box to duplicate or erase. In order to avoid this non-determinism, the standard solution is decorating the structure with the required boxes; which also implies the introduction of additional rules, for instance to move a box inside another box. Another solution might be to introduce a standard choice among the set of possible boxes for a given node; for instance, we could define the box of a node  $u$  as the smallest (or the largest) correct  $\lambda$ -structure with root  $u$ , the so-called kingdom (empire) of  $u$ . But, because of weakenings, it would be very difficult to compute the kingdom (empire) of a node on the fly. Anyway, as the main purpose of this section is to explain the reduction rules of  $\lambda$ -terms with sharing, for the moment we omit the box assignment problem. In the next section, we shall see that jumps allow an easy and efficient solution to this problem: by adding a suitable number of jumps to a correct  $\lambda$ -structure we shall obtain that every node is the root of exactly one correct substructure, its box.

**Duplication, erasing, dereliction.** The two rules that operate on  $\lambda$ -boxes are duplication, i.e., the C(ontraction)-rule, and erasing, i.e., the W(eakening)-rule. Their behaviour is non-linear—they correspond to the exponential rules of linear logic—and this is why in 2 they required !-boxes. The third rule that operates on substituted nodes is the D(ereliction)-rule. This last rule allows the erasing of useless substituted  $v$ -nodes and, since it is linear, does not directly involve boxes.

The C,W,D-rules are described in Fig. 4. Given a substitution  $u_x$  of a  $\lambda$ -structure  $S$ , we can always apply one of these three rules. In fact, let  $H$  be a  $\lambda$ -box with root  $u_x$ .



**Fig. 4.** a,b,c) Duplication, erasing, and dereliction rules; d) the  $\beta_s$ -reduct of  $(\lambda x.t)s$

- If the  $v$ -link  $\alpha_x$  of  $u_x$  has  $k > 1$  occurrences edges, then we can apply the C-rule, where the two isomorphic copies  $H^1$  and  $H^2$  of  $H$  (with roots  $u_{x1}$  and  $u_{x2}$ , respectively) are glued to context of the redex.
- If  $\alpha_x$  has  $k = 0$  occurrence edges, then  $\alpha_x$  is a weakening and we can apply the W-rule. In the reduct, the  $\lambda$ -box  $H$  is replaced by a set of weakening links  $W_{FV(H)}$ , one for each free  $v$ -node  $u_y$  of  $H$ , added to ensure that the lhs and the rhs of the rule have the same free  $v$ -nodes (again, more precisely, they are glued to the context).
- If  $\alpha_x$  has  $k = 1$  occurrence edges, we can apply the D-rule, which executes the (linear) substitution by removing  $\alpha_x$ . The D-rule is local, as it does not depend on the  $\lambda$ -box associated with  $u_x$ .

### 3 Jumps

A *jump* or *j-link* is a link with an incoming edge, the *source edge*, that connects the link to its source (node)  $u_s$ , and an outgoing edge, the *anchor edge*, that connects the link to its anchor (node)  $u_a$ . A jump from the source  $u_s$  to the anchor  $u_a$  is written  $u_a \langle j \rangle u_s$  and represented as

$$u_a \leftarrow \dots j \dots u_s$$

**j-translation of explicit substitutions.** We refine the translation of  $\lambda$ -terms with explicit substitutions by adding a jump from the root of  $\underline{s}$  to the root of  $\underline{t}$ , for every explicit substitution  $t[s/x]$ . Thus, the definition of  $\underline{t}_j$  is the same as  $\underline{t}$ , but for (see Fig. 5f)

$$\underline{t[s/x]}_j = \underline{t}_j[\underline{s}_j]_x; u_{\underline{t}_j} \langle j \rangle u_{\underline{s}_j}$$

We shall see that this economic decoration of  $\lambda$ -dags with jumps suffices to define an unambiguous notion of  $\lambda$ -box in terms of domination. The notion of  $\lambda$ -structure is therefore extended in order to encompass jumps.

**Definition 8 ( $\lambda_j$ -structure,  $\lambda_j$ -dag).** A  $\lambda_j$ -structure  $G$  is a  $\lambda$ -structure  $G_\lambda$  plus a set of  $j$ -links  $G_j$  s.t.: (Source) the source of every jump is a substitution and every substitution of  $G_\lambda$  is the source of one and only one  $j$ -link; (Anchor) every anchor is not an abstracted or free  $v$ -node of  $G_\lambda$

A  $\lambda_j$ -dag  $D$  is a  $\lambda_j$ -structure s.t.  $D = \underline{t}_j X$ , for some  $\lambda$ -term with explicit substitutions  $t$  and some context  $X \supseteq FV(t)$ .

Let us stress that in a  $\lambda_J$ -structure nodes may have more than one incoming edge, but in such a case exactly one edge is not the anchor edge of a jump. In particular a node may be the anchor of many jumps. All and only the substituted  $v$ -nodes have more than one outgoing edge. More precisely, they have two outgoing edges, one which is the source edge of a jump, and one which is not.

The definitions of path and partial orders for  $\lambda$ -structures, and consequently, that of dominion on Definition 4 naturally extend to  $\lambda_J$ -structures. In particular, any jump from the source  $u_s$  to the anchor  $u_a$  adds a new path from  $u_s$  to  $u_a$ .

### 3.1 Correctness Criterion

Jumps are a sort of back-connections similar to the binding edges of  $\lambda$ -links. It seems thus rather natural to ask that they comply with a binding condition similar to the L-condition required for  $\lambda$ -links.

(LJ) *Lamarche for Jumps*: Let  $\alpha_j = u_a \langle j \rangle u_s$  be a  $j$ -link of a  $\lambda_J$ -structure. Then  $u_s \sqsubseteq u_a$ .

**Definition 9 (LJ-correctness,  $\lambda_J$ -dag).** A  $\lambda_J$ -structure with root  $r$  is LJ-correct when condition (D) holds, condition (L) holds for every bound  $v$ -node, and condition (LJ) holds for every substitution.

Let us remark that these conditions hold by taking into account jumps. For instance, by adding a jump  $r \langle j \rangle u_{x_1}$  in Fig. 2d we obtain an incorrect  $\lambda_J$ -graph, because  $u_{x_2}$  is no longer dominated by its abstraction. In particular no term would translate to such an incorrect  $\lambda_J$ -structure, thus the criterion *has to* consider paths using jumps. We also stress that, since in a  $\lambda_J$ -structure we have a jump for every substitution, and since a non-isolated weakening is a substitution, condition (D) and the existence of only one output force in a  $\lambda_J$ -dag that every node  $u$  has a path to the root of  $G$  (except the  $v$ -nodes of the context variables), and that domination can be expressed in terms of paths to the root (as before).

On an L-correct  $\lambda_J$ -structure the LJ-condition implies a nesting of jumps similar to the nesting of boxes in proof nets. In fact, given two distinct jumps  $u_a \langle j \rangle u_s$  and  $u'_a \langle j \rangle u'_s$  for which condition (LJ) holds, it is not the case that  $u_s \leq u'_s \leq u_a \leq u'_a \leq r$ , unless  $u'_s = u'_a$ . Therefore, condition (LJ) could be replaced by the following jump nesting condition.

(JN) *Jump Nesting*: Let  $\alpha_j = u_a \langle j \rangle u_s$  and  $\alpha'_j = u'_a \langle j \rangle u'_s$ . If  $u_s \leq u'_s \leq r$ , then any path  $\phi$  from  $u_s$  to  $r$  that contains  $u'_s$  contains also  $u_a$  and  $u'_a$ , and either  $\phi = u_s, \dots, u_a, \dots, u'_s, \dots, u'_a, \dots, r$  or  $\phi = u_s, \dots, u'_s, \dots, u'_a, \dots, u_a, \dots, r$ .

In the particular case  $u_s = u'_s$  (and then  $u_a = u'_a$  also), condition (NJ) implies that every path from  $u_s$  to  $u_r$  also contains  $u_a$ , i.e., condition (LJ).

### 3.2 $\lambda_J$ -Boxes

**Definition 10 (source-closed,  $j$ -substructure).** Let  $G$  be a  $\lambda_J$ -structure,  $H \subseteq G$  a subset of its links.  $H$  is source-closed when, if the anchor  $u_a$  of some jump  $\alpha = u_a \langle j \rangle u_s$



of  $G$  is a node of  $H$ , then  $\alpha \in H$  (and, as a consequence, the source of  $\alpha$ , is a node of  $H$ ).  $H$  is a  $\mathcal{J}$ -substructure of  $G$ , say  $H \triangleleft_{\mathcal{J}} G$ , if  $H$  is a  $\lambda_{\mathcal{J}}$ -structure,  $H_{\lambda} \triangleleft G_{\lambda}$ , and  $H$  is source-closed.

According to this definition,  $G \downarrow_u \triangleleft_{\mathcal{J}} G$  for every  $u$  that is not an abstracted or free  $\nu$ -node of  $G$  (i.e. a node that is not minimal w.r.t.  $\leq$ ). The definition of box can now be refined to include jumps, by taking into account that a box must be a  $\mathcal{J}$ -substructure of the  $\lambda_{\mathcal{J}}$ -structure in which it is contained.

**Definition 11 ( $\lambda_{\mathcal{J}}$ -box).** Given a  $\lambda_{\mathcal{J}}$ -structure  $G$ , a  $\lambda_{\mathcal{J}}$ -box with principal port  $u$  in  $G$  is a  $\lambda_{\mathcal{J}}$ -structure  $H \triangleleft_{\mathcal{J}} G$  with root  $u$ .

A  $\lambda_{\mathcal{J}}$ -box  $H$  in  $G$  is source-closed. Furthermore, being a  $\lambda_{\mathcal{J}}$ -structure, all its jumps must have a substitution as source. Thus, we get the following property: if  $\alpha = u_a \langle \mathcal{J} \rangle u_s \in G$ , then  $u_s \in N_H$  implies  $u_a \in N_H$  iff  $u_s$  is not an input (a free  $\nu$ -node) of  $H$ . We can prove that in a  $\lambda_{\mathcal{J}}$ -dag  $G$  a  $\lambda_{\mathcal{J}}$ -box is a  $\lambda_{\mathcal{J}}$ -dag too. Moreover, for every node  $u$  to which we may (and we need to) assign a box, there is one and only one  $\lambda_{\mathcal{J}}$ -box  $H$  with principal port  $u$ , which is indeed the only LJ-correct  $H \triangleleft_{\mathcal{J}} G$  with root  $u$ , and which coincide with  $G \downarrow_u$  also.

**Theorem 1 (box uniqueness).** Let  $G$  be an LJ-correct  $\lambda_{\mathcal{J}}$ -structure. For every node  $u$  of  $G$ , except the abstracted and free  $\nu$ -nodes, we have that  $G \downarrow_u$  is an LJ-correct  $\lambda_{\mathcal{J}}$ -structure s.t.  $G \downarrow_u \triangleleft_{\mathcal{J}} G$ , and  $G \downarrow_u$  is the unique  $\lambda_{\mathcal{J}}$ -structure  $H$  with root  $u$  s.t.  $H \triangleleft_{\mathcal{J}} G$ .

### 3.3 Readback of $\lambda_{\mathcal{J}}$ -Dags

We want to associate a term  $t$  to every LJ-correct  $\lambda_{\mathcal{J}}$ -structure  $G$ , extending the recursive readback  $\bar{\tau}$  of section 2.2 to a map  $\bar{\tau}$ . If the root  $r$  of  $G$  is not the anchor of any jump, we can apply the recursive rules of section 2.2. If  $r$  is the anchor of just one jump  $r \langle \mathcal{J} \rangle u_x$  then we can readback the term  $t[s/x]$  s.t.  $s = \overline{G \downarrow_{u_x}}$  and  $t = \overline{G'[u_x]}$ , for a context  $G'[\cdot]$  s.t.  $G = G'[G \downarrow_{u_x}]$ . But when  $r$  is the anchor of many jumps we have to pay attention to the order in which we remove jumps, because there can be dependencies among them. Let us consider for instance  $G = \overline{t[s_1/x_1][s_2/x_2]}$ , with  $x_2 \in \text{FV}(s_1)$ . The root is the anchor of the two jumps from the substitution nodes  $u_{x_1}$  and  $u_{x_2}$ , but we cannot remove the jump from  $u_{x_1}$  before the one from  $u_{x_2}$ , as we would obtain the term  $t[s_2/x_2][s_1/x_1]$  whose  $\mathcal{J}$ -translation is not  $G$  (because of  $x_2 \in \text{FV}(s_1)$ ). In terms of paths, the situation described above corresponds to the case  $u_{x_2} \leq u_{x_1}$ . Only when  $u_{x_1} \not\leq u_{x_2}$  and  $u_{x_2} \not\leq u_{x_1}$  the order of removals is irrelevant, and the two substitutions can safely be permuted. Let us notice however that by the correctness of  $G$  there cannot be mutual dependencies.

Thus, for every node  $u$  in  $G$  we assume given a partial order  $\preceq_u$  on  $\mathcal{J}(u)$ , the set of substitutions anchored to  $u$ , s.t.  $\preceq_u$  is total and  $u' \preceq u''$  if  $u' \leq u''$ . Our readback map depends on such partial orders.

Let us assign a distinct variable to every  $\nu$ -node of  $G$ , including its substitutions, and let us denote by  $u_x$  the  $\nu$ -node corresponding to the variable  $x$ . If  $r$  is the root of  $G$  and  $\mathcal{J}(r)$  is not empty, there is a context  $G'[\cdot]$  s.t.  $G'[u_x]$  is a  $\lambda_{\mathcal{J}}$ -dag and  $G = G'[H]; r \langle \mathcal{J} \rangle u_x$  where  $H = G \downarrow_{u_x}$  and  $u_x = \min_{\preceq_r}(\mathcal{J}(r))$ . The readback of  $G$  w.r.t.  $\preceq$  and the given variable assignment is  $\overline{G} = \overline{G'[u_x]}[\overline{H}/x]$ .

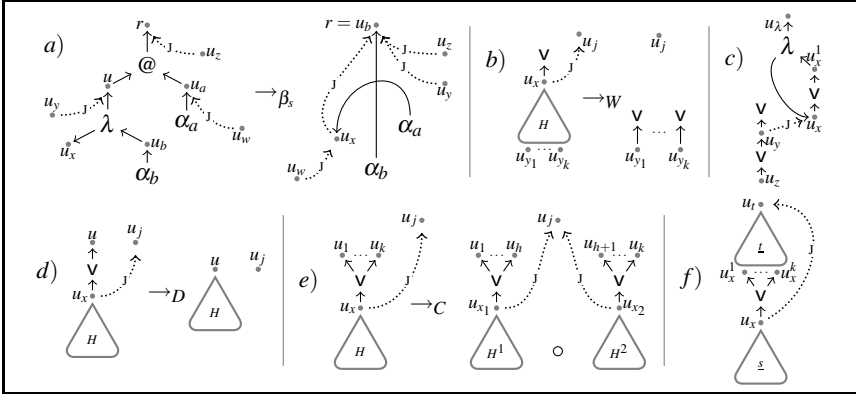


Fig. 5. a,b,d,e) Rules for  $\lambda_J$ -dags; c) A non  $\lambda_J$ -structure; f) The  $J$ -translation of  $t[s/x]$

When  $J(r)$  is empty, the readback of the LJ-correct  $\lambda_J$ -structure is obtained by replacing  $\nabla$  for  $\bar{\nabla}$  in the rules given for the readback of a  $\lambda$ -tree (Def. 5).

**Proposition 3 (Sequentialization).** *Let  $G$  be a  $\lambda_J$ -structure. Connectedness: if  $G$  is LJ-correct then  $G = G \downarrow_r ; W$ , where  $W$  is the set of isolated weakenings of  $G$ . Sequentialization:  $G$  is a  $\lambda_J$ -dag iff it is LJ-correct.*

Modulo isolated weakenings and free variable names, the readback is univocally determined given the linear orders  $\preceq_u$  and, since two linearizations differ for the permutation of independent substitutions only, we get  $t_{1 \downarrow X} = t_{2 \downarrow X}$  if and only if  $t_1 \simeq t_2$ , where  $\simeq$  is the least congruence induced by  $t[s_1/x_1][s_2/x_2] \sim t[s_2/x_2][s_1/x_1]$  when  $x_1 \notin \text{FV}(s_2)$  and  $x_2 \notin \text{FV}(s_1)$ .

We can now explain the anchor condition in the definition of  $\lambda_J$ -structures. It asks that no anchor is an abstracted or free  $v$ -node: these are the only nodes of a  $\lambda_J$ -dag  $D$  which have no corresponding subterm in  $\bar{D}^J$ . Thus, they cannot be used as sequentialization points for a substitution (see Fig. 5c) for an example).

### 3.4 Reductions

The reduction rules for  $\lambda_J$ -dags are the same as those described in section 2.2, with the addition of jumps, see Fig. 5, where  $H$  is the unique  $\lambda_J$ -box with root  $u_x$  induced by the jumps of  $G$ . The only rule that creates a new jump in the resulting graph is the sharing  $\beta$ -rule, which introduces a  $J$ -link  $u_b \langle J \rangle u_x$  from the  $v$ -node  $u_x$ , that after the reduction becomes a substitution, to the root  $u_b$  (which is indeed merged with  $r$ ). In the  $\beta_s$ -rule, the jumps on the redex move in accord to the way nodes are merged in the reduct: every jump in the redex whose anchor is the root  $r$ , or the abstraction node  $u$  or the body node  $u_b$  are anchored in the reduct to the new root  $u_b$ , while the jump anchored to the argument node  $u_a$  are moved to the substitution  $u_x$  (by the definition of  $\lambda_J$ -structure no jump can be anchored to  $u_x$  in the redex).

The other rules move, duplicate, or erase jumps depending on the action performed on the corresponding source. In the dereliction and erasing rules, the jumps from the

substitution  $u_x$  in the redex disappear. In the duplication rules, every jump in the  $\lambda_J$ -box  $H$  leads to a new jump between the copied nodes, while the jump from  $u_x$  leads to two new jumps with the same anchor. Note that by source-closure every jump anchored to  $u_x$  is in  $H$ . In contrast to linear logic there is no specific rule for moving a box into another. In the D-rule this happens implicitly : it removes the jump, and thus changes the domination order.

Given two  $\lambda$ -terms  $r$  and  $s$ , let us denote by  $r\{s/x\}$  the  $\lambda$ -term obtained by replacing  $s$  for  $x$  (i.e., the standard substitution of  $\lambda$ -calculus). Given a  $\lambda$ -term with explicit substitutions  $t$ , we shall denote by  $t^\bullet$  the  $\lambda$ -term obtained by replacing every  $r\{s/x\}$  in the term with the  $\lambda$ -term  $r^\bullet\{s^\bullet/x\}$ .

First of all, we can state that the reduction rules for  $\lambda_J$ -dags defined so far implement a sound reduction of  $\lambda$ -terms with explicit substitutions: (i) any reduct  $G$  of a  $\lambda_J$ -dag  $D$  is a  $\lambda_J$ -dag; (ii) if  $t$  is a  $\lambda$ -term with explicit substitutions s.t.  $\underline{t}_{jX} = D$ , then there is a  $\lambda$ -term with explicit substitutions  $s$  s.t.  $\underline{s}_{jX} = G$ , and the  $\lambda$ -term (without explicit substitutions)  $t^\bullet$  reduces to the  $\lambda$ -term (without explicit substitutions)  $s^\bullet$ .

**Proposition 4.** *Let  $D$  be a  $\lambda_J$ -dag and  $t$  a  $\lambda$ -term with explicit substitutions s.t.  $\underline{t}_{jX} = D$  for some  $X$ . If  $D \rightarrow G$ , then  $G$  is a  $\lambda_J$ -dag, and  $t^\bullet \rightarrow_\beta^* s^\bullet$ , for some  $s$  s.t.  $\underline{s}_{jX} = G$ .*

Then, we can state that the readback of a  $\lambda_J$ -dag  $D$  is internalized by the C,W,D-rules, the  $\sigma$ -rules, of  $\lambda_J$ -dags. In other words, we have that: (i) in a  $\lambda_J$ -dag, the effective execution of an implicit substitution can be implemented by the  $\sigma$ -reductions; (ii) the  $\sigma$ -reductions of a  $\lambda_J$ -dag are terminating and confluent and, as a consequence, every  $\lambda_J$ -dag has a unique  $\sigma$ -normal form; (iii) if  $t$  is a  $\lambda$ -term with explicit substitutions s.t.  $\underline{t}_{jX} = D$ , the unique normal form of the  $\lambda_J$ -dag  $D$  is the  $\lambda$ -tree  $T$  corresponding to the  $\lambda$ -term (without explicit substitutions) represented by  $t$ , that is,  $T = \underline{t}^\bullet_X$ .

**Proposition 5.** *Let  $\rightarrow_\sigma = \rightarrow_D \cup \rightarrow_C \cup \rightarrow_W$ . For every  $\lambda$ -term with explicit substitutions  $t$ , we have that*

1.  $t\{s/x\}_{jX} \rightarrow_\sigma^* \underline{t\{s/x\}}_{jX}$ , for every  $s$ ;
2.  $\rightarrow_\sigma$  is terminating and confluent on the  $\lambda_J$ -dag  $D = \underline{t}_{jX}$  and, as a consequence,  $D$  has a unique  $\sigma$ -normal form;
3. the unique  $\sigma$ -normal form of  $D$  is the  $\lambda$ -tree  $\underline{t}^\bullet_X$ .

The first item of the previous proposition states that two  $\sigma$ -equivalent  $\lambda_J$ -dags have the same readback  $t$  and are just distinct shared representation of the  $\lambda$ -dag  $T$  associated to  $t$ . Every C-rule decreases the amount of sharing in the  $\lambda_J$ -dag  $D$ , while the D-rule and the W-rule decrease the size of the  $\lambda_J$ -dag. Therefore, the  $\sigma$ -reduction is terminating. Confluence can be proved directly by inspection of the rules or by observing that the normal form of  $D$  w.r.t.  $\sigma$ -reduction does not contain substitutions and then it is a  $\lambda$ -tree; in particular, since readback is preserved, the normal form of  $D$  must be its readback.

Finally, we can conclude that the  $\beta$ -reductions of  $\lambda$ -terms can be simulated by  $\rightarrow_j = \rightarrow_{\beta_s} \cup \rightarrow_\sigma$ , the sharing reductions with jumps of  $\lambda_J$ -dags, and that every sharing reduction with jumps of a  $\lambda$ -tree corresponds indeed to some  $\beta$ -reduction of the  $\lambda$ -term represented by the  $\lambda$ -tree.

**Theorem 2.** *Let  $t$  be a  $\lambda$ -term. If  $t \rightarrow_\beta^* s$ , then  $\underline{t}_{jX} \rightarrow_j^* \underline{s}_{jX}$ . If  $\underline{t}_{jX} \rightarrow_j^* D$ , then there is  $t \rightarrow_\beta^* s$  s.t.  $D \rightarrow_\sigma^* \underline{s}_{jX}$ .*

## 4 Conclusions and Further Work

In the paper we have seen a way to represent boxes in  $\lambda$ -dags—the graphs corresponding to the proof nets of  $\lambda$ -calculus—by means of jumps. Jumps give a local and indirect implementation of boxes, which is very economic—we only need a jump for every substitution (every exponential cut in the corresponding proof net)—and does not require to deal with boxes auxiliary ports.

When we interpret jumps as  $\lambda$ -terms with explicit substitutions, jumps have also a very clear interpretation: any jump gives the exact position of a substitution in the term, leading to a unique sequentialization of  $\lambda$ -dags (when we assume that sequences of independent substitutions correspond to a sort of multiple parallel substitution). However, since the main purpose of the paper was the analysis of  $\lambda_J$ -dags from the point of view of nets—correctness, graph reduction, sequentialization—we have not analyzed, or compared with the calculi in the literature, the calculus with explicit substitutions induced by the  $\lambda_J$ -dag reduction. In fact, let  $|t|_x$  be the number of free occurrences of  $x$  in  $t$ ; the rules correspond to

$$\begin{array}{lll}
 (\beta_{es}) & (\lambda x.t)[\bar{v}/\bar{y}] u \rightarrow t[u/x][\bar{v}/\bar{y}] & \\
 (\text{Gc}) & t[u/x] \rightarrow t & |t|_x = 0 \\
 (\text{V}) & t[u/x] \rightarrow t\{u/x\} & |t|_x = 1 \\
 (\text{Dup}) & t[u/x] \rightarrow t\{y|x\}[u/x][u/y] & |t|_x \geq 2 \text{ \& } y \text{ fresh}
 \end{array}$$

where  $t[\bar{v}/\bar{y}]$  stands for  $t[v_1/y_1] \dots [v_n/y_n]$  for some  $n \geq 0$ , and  $t\{y|x\}$  means that  $k$  occurrences of  $x$  in  $t$ , chosen in any way and with  $0 < k < |t|_x$ , are replaced by a fresh variable  $y$ . In particular, the second rule corresponds to  $\rightarrow_W$ , the third one to  $\rightarrow_D$  and the last one to  $\rightarrow_C$ . The detailed analysis of the main properties of such a calculus are the subject of our ongoing research.

## References

1. Asperti, A., Guerrini, S.: The Optimal Implementation of Functional Programming Languages. Cambridge Tracts in Theoretical Computer Science, vol. 45. Cambridge University Press, Cambridge (1998)
2. Cosmo, R.D., Kesner, D., Polonovski, E.: Proof nets and explicit substitutions. *Mathematical Structures in Computer Science* 13(3), 409–450 (2003)
3. Di Giamberardino, P., Faggian, C.: Jump from parallel to sequential proofs: Multiplicatives. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 319–333. Springer, Heidelberg (2006)
4. Girard, J.-Y.: Linear logic. *Theoretical Computer Science* 50(1), 1–102 (1987)
5. Girard, J.-Y.: Quantifiers in linear logic II. In: *Prépublications de l'Équipe de Logique* 19. Université Paris VII, Paris (1991)
6. Guerrini, S., Martini, S., Masini, A.: Coherence for sharing proof nets. *Theoretical Computer Science* 294(3), 379–409 (2003)
7. Kesner, D.: The theory of calculi with explicit substitutions revisited. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 238–252. Springer, Heidelberg (2007)
8. Lamarche, F.: Proof nets for intuitionistic linear logic I: Essential nets. Preliminary report (April 1994)
9. Mackie, I.: Linear logic with boxes. In: *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pp. 309–320 (1998)

10. Murawski, A.S., Ong, C.-H.L.: Dominator trees and fast verification of proof nets. In: LICS 2000: Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science, pp. 181–191. IEEE Computer Society, Los Alamitos (2000)
11. Peyton Jones, S.: The Implementation of Functional Programming Languages. International Series in Computer Science. Prentice-Hall, Englewood Cliffs (1987)
12. Plump, D.: Term graph rewriting. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools, ch. 1, vol. 2, pp. 3–61. World Scientific, Singapore (1999)
13. Regnier, L.: Lambda-calcul et réseaux. Thèse de doctorat, Université Paris, 7 (1992)
14. Wadsworth, C.P.: Semantics and pragmatics of the lambda-calculus. Phd Thesis, Oxford, ch. 4 (1971)

# Tree-Width for First Order Formulae

Isolde Adler<sup>1</sup> and Mark Weyer<sup>2</sup>

<sup>1</sup> Department of Informatics, University of Bergen, Norway

<sup>2</sup> Humboldt Universität zu Berlin, Berlin, Germany

**Abstract.** We introduce tree-width for first order formulae  $\varphi$ ,  $\text{fotw}(\varphi)$ . We show that computing  $\text{fotw}$  is fixed-parameter tractable with parameter  $\text{fotw}$ . Moreover, we show that on classes of formulae of bounded  $\text{fotw}$ , model checking is fixed parameter tractable, with parameter the length of the formula. This is done by translating a formula  $\varphi$  with  $\text{fotw}(\varphi) < k$  into a formula of the  $k$ -variable fragment  $\mathcal{L}^k$  of first order logic. For fixed  $k$ , the question whether a given first order formula is equivalent to an  $\mathcal{L}^k$  formula is undecidable. In contrast, the classes of first order formulae with bounded  $\text{fotw}$  are fragments of first order logic for which the equivalence is decidable.

Our notion of tree-width generalises tree-width of conjunctive queries to arbitrary formulae of first order logic by taking into account the quantifier interaction in a formula. Moreover, it is more powerful than the notion of elimination-width of quantified constraint formulae, defined by Chen and Dalmau (CSL 2005): For quantified constraint formulae, both bounded elimination-width and bounded  $\text{fotw}$  allow for model checking in polynomial time. We prove that  $\text{fotw}$  of a quantified constraint formula  $\varphi$  is bounded by the elimination-width of  $\varphi$ , and we exhibit a class of quantified constraint formulae with bounded  $\text{fotw}$ , that has unbounded elimination-width. A similar comparison holds for strict tree-width of non-recursive stratified datalog as defined by Flum, Frick, and Grohe (JACM 49, 2002).

Finally, we show that  $\text{fotw}$  has a characterization in terms of a robber and cops game without monotonicity cost.

## 1 Introduction

Model checking is an important problem in complexity theory. It asks for a given formula  $\varphi$  of some class  $\mathcal{C}$  of formulae and a structure  $\mathcal{A}$ , whether  $\mathcal{A}$  satisfies  $\varphi$ .

$\text{MC}(\mathcal{C})$
<b>Input:</b> A structure $\mathcal{A}$ and a formula $\varphi \in \mathcal{C}$ .
<b>Question:</b> $\mathcal{A} \models \varphi$ ?

Let  $\mathcal{L}$  denote first order logic. It is well-known, that  $\text{MC}(\mathcal{L})$  is PSPACE-complete. Motivated by this, much research has been done on finding fragments of  $\mathcal{L}$  having a tractable model checking problem. For instance, for fixed  $k$ , the

problem  $\text{MC}(\mathcal{L}^k)$  can be solved in polynomial time, where  $\mathcal{L}^k$  denotes the fragment of first order formulae with at most  $k$  variables (see e.g. [15]).

The class of *conjunctive queries*, CQ, is an important fragment of first order logic. Many queries that occur in practice are conjunctive queries, and model checking of conjunctive queries on relational databases (i.e. relational structures) is an important and well-studied problem in database theory [25,7,16,9,17,20]. It is equivalent to conjunctive query containment, to the constraint satisfaction problem studied in artificial intelligence and to the homomorphism problem for structures [6,12]. A *conjunctive query* is a first order formula starting with a quantifier prefix using only existential quantifiers, followed by a conjunction of relational atoms. While  $\text{MC}(\text{CQ})$  is NP-hard in general, several researchers proved independently that conjunctive queries of bounded *tree-width* can be evaluated in polynomial time [7,16]. One way to prove this is the following. Suppose  $\varphi$  is a conjunctive query having tree-width  $k - 1$ . Then we can compute a tree decomposition of width  $k - 1$  in linear time using Bodlaender’s algorithm [5]. From the decomposition we can actually read off the syntax of an equivalent formula  $\varphi' \in \mathcal{L}^k$ . Finally, we use the fact that  $\text{MC}(\mathcal{L}^k)$  is solvable in polynomial time.

In this paper, we introduce a notion of *tree-width for first order formulae*  $\varphi$ ,  $\text{fotw}(\varphi)$ . Our notion generalises the notion of tree-width of conjunctive queries, and we show that the class  $\mathcal{C}_k$  of all first order formulae  $\varphi$  with  $\text{fotw}(\varphi) \leq k$  satisfies the following properties.

1.  $\mathcal{C}_k$  has a polynomial time membership test (Corollary 2).
2.  $\mathcal{C}_k$  has the same expressive power as  $\mathcal{L}^k$ , the fragment of first order formulae with at most  $k$  variables (Theorem 2).
3. There is an algorithm that computes for given  $\varphi \in \mathcal{C}_k$  an equivalent formula  $\varphi' \equiv \varphi$  with  $\varphi' \in \mathcal{L}^k$  (Theorem 2).
4.  $\text{MC}(\mathcal{C}_k)$  is fixed parameter tractable with parameter the length of  $\varphi$ , i.e. for input  $\varphi \in \mathcal{C}_k$  and  $\mathcal{A}$ , the running time is  $p(\|\mathcal{A}\|)f(|\varphi|)$  for a polynomial  $p$  and a computable function  $f$  (Corollary 3).

Obviously, properties 1 and 3 imply property 4. While  $\text{MC}(\mathcal{L}^k)$  is solvable in polynomial time, we do not obtain a polynomial algorithm for  $\text{MC}(\mathcal{C}_k)$ . Nevertheless, in typical applications one can expect the length of the formula to be small compared to the size of the structure (database). For a fixed formula the running time is polynomial, and moreover, the problem is *fixed-parameter tractable* (in FPT), meaning that changing  $\varphi$  does not alter the exponent of the polynomial (see [10,15]).

Note that for fixed  $k > 0$  it is undecidable, whether a first order formula  $\varphi$  is equivalent to an  $\mathcal{L}^k$  formula. Hence it is not surprising that our notion of  $k$ -bounded first order tree-width does not capture semantic equivalence to  $\mathcal{L}^k$  (we will give more details in Section 5). The idea is to compute a tree decomposition of a formula  $\varphi \in \mathcal{L}$  layerwise, respecting the quantifier interaction.

*Quantified constraint formulae* generalise conjunctive queries by allowing arbitrary quantifiers in the quantifier prefix. In [8], Chen and Dalmau introduce

*elimination orderings* for quantified constraint formulae. These elimination orderings must respect the quantifier prefix. In this way, Chen and Dalmau obtain a notion of *elimination-width*<sup>1</sup>, which allows for model checking of quantified constraint formulae of bounded elimination-width in polynomial time, using a consistency algorithm. Hereby, they answer a question posed in [19] positively, whether bounded tree-width methods work for formulae more general than conjunctive queries. Introducing a notion of tree-width for arbitrary first order formulae, we even go further. We show that for quantified constraint formulae  $\varphi$ , elimination-width of  $\varphi$  is at least as large as  $\text{fotw}(\varphi)$ , and we exhibit a class of quantified constraint formulae with bounded first order tree-width and unbounded elimination-width. We show that quantified constraint formulae of bounded  $\text{fotw}$  allow for model checking in polynomial time. Hence  $\text{fotw}$  is more powerful than elimination-width.

In [13], Flum, Frick and Grohe introduce *strict tree-width for non-recursive stratified datalog (NRS) programs*. They show that model checking for NRS programs of bounded strict tree-width can be done in polynomial time. Since first order formulae can be easily translated into NRS programs, their notion of strict tree-width can be transferred to first order formulae. We show that if an NRS program obtained from a formula  $\varphi$  has strict tree-width at most  $k$ , then  $\text{fotw}(\varphi) \leq k$ , and there are classes of formulae with bounded first order tree-width, whose corresponding NRS programs have unbounded strict tree-width<sup>2</sup>. Hence our notion of first order tree-width yields larger subclasses of  $\mathcal{L}$ , that still allow for tractable model checking.

Actually, we obtain first order tree-width as a special case of a more abstract notion which we term stratified tree-width. We expect that stratified tree-width will find further, quite different, applications.

The rest of this paper is organised as follows. Section 2 fixes some terminology. Section 3 introduces the notion of stratified tree-width and the special case of first order tree-width. In Section 4 we show how to compute stratified tree decompositions and, in particular, how to compute first order tree-width. In Section 5 we prove that model checking for formulae of bounded first order tree-width is fixed-parameter tractable. In Section 6 we relate our notion to existing notions and give a game characterisation of stratified tree-width. We conclude with some open problems in Section 7.

## 2 Well-Known Definitions

A *vocabulary*  $\sigma = \{R_1, \dots, R_n, c_1, \dots, c_m\}$  is a finite set of *relation symbols*  $R_i$ ,  $1 \leq i \leq n$ , and *constant symbols*  $c_j$ ,  $1 \leq j \leq m$ . Every  $R_i$  has an associated *arity*,

<sup>1</sup> Actually, the notion is called *tree-width* for quantified constraint formulae in [8]. But since the notion is defined via elimination orderings, we prefer the term *elimination-width*.

<sup>2</sup> In [13], the authors also introduce a notion of *tree-width for first order formulae*. But their notion disregards the quantifier interaction, and they only use it for conjunctive queries with negation.



an integer  $\text{ar}(R_i) > 0$ . A  $\sigma$ -structure is a tuple  $\mathcal{A} = (A, R_1^{\mathcal{A}}, \dots, R_n^{\mathcal{A}}, c_1^{\mathcal{A}}, \dots, c_m^{\mathcal{A}})$  where  $A$  is a finite set, the *universe* of  $\mathcal{A}$ ,  $R_i^{\mathcal{A}} \subseteq A^{\text{ar}(R_i)}$  for  $1 \leq i \leq n$ , and  $c_j \in A$  for  $1 \leq j \leq m$ .

Given a  $\sigma$ -structure  $\mathcal{A}$  we distinguish between the cardinality  $|A|$  of the universe  $A$  of  $\mathcal{A}$  and the *size*  $\|\mathcal{A}\|$  of  $\mathcal{A}$ , given by  $\|\mathcal{A}\| = |\sigma| + |A| + \sum_{i=1}^n |R_i^{\mathcal{A}}| \cdot \text{ar}(R_i)$ .

We use  $\mathcal{L}$  to denote relational first order logic with constants, and for simplicity, we refer to  $\mathcal{L}$  as *first order logic*. We assume that the reader is familiar with the basic notions of first order logic (see for instance [11]). For a formula  $\varphi$  we let  $\text{free}(\varphi)$  denote the set of free variables of  $\varphi$ . Formula  $\varphi$  is a *sentence*, if  $\text{free}(\varphi) = \emptyset$ . We sometimes write  $\varphi(x_1, \dots, x_n)$  to indicate that  $\text{free}(\varphi) \subseteq \{x_1, \dots, x_n\}$ .

For a structure  $\mathcal{A}$  and a formula  $\varphi(x_1, \dots, x_n)$  we let

$$\varphi(\mathcal{A}) := \{(a_1, \dots, a_n) \mid \mathcal{A} \models \varphi(a_1, \dots, a_n)\}.$$

For sentences we have  $\varphi(\mathcal{A}) = \text{TRUE}$ , if  $\mathcal{A}$  satisfies  $\varphi$ , and **FALSE** otherwise. If the vocabularies of  $\varphi$  and  $\mathcal{A}$  are different, we let  $\varphi(\mathcal{A}) = \emptyset$ .

The *Query Evaluation Problem* for a class  $\mathcal{C}$  of formulae is the following problem:

EVAL( $\mathcal{C}$ )
<b>Input:</b> A structure $\mathcal{A}$ and a formula $\varphi \in \mathcal{C}$ .
<b>Problem:</b> Compute $\varphi(\mathcal{A})$ .

Note that if  $\varphi$  is a sentence, then  $\text{EVAL}(\mathcal{C})$  and  $\text{MC}(\mathcal{C})$  coincide. We say that a formula  $\varphi \in \mathcal{L}$  is *straight*, if no variable in  $\varphi$  is quantified over twice, if no free variable is also a quantified variable, and if each quantified variable actually occurs in some atom. All formulae are straight, unless stated otherwise. Moreover, we assume that all formulae are in negation normal form, i.e. the negation symbols only appear in front of atoms.

A *tree decomposition* of a graph  $G = (V, E)$  is a pair  $(T, B)$ , consisting of a rooted tree  $T$  and a family  $B = (B_t)_{t \in T}$  of subsets of  $V$ , the *pieces* of  $T$ , satisfying:

- (TD1). For each  $v \in V$  there exists  $t \in T$ , such that  $v \in B_t$ . We say the node  $t$  *covers*  $v$ .
- (TD2). For each edge  $e \in E$  there exists  $t \in T$ , such that  $e \subseteq B_t$ . We say the node  $t$  *covers*  $e$ .
- (TD3). For each  $v \in V$  the set  $\{t \in T \mid v \in B_t\}$  is connected in  $T$ .

The *width* of  $(T, B)$  is defined as  $w(T, B) := \max \{|B_t| \mid t \in T\} - 1$ .

The *tree-width* of  $G$  is defined as

$$\text{tw}(G) := \min \{w(T, B) \mid (T, B) \text{ is a tree decomposition of } G\}.$$

### 3 First Order Tree-Width

We start with defining stratified tree-width. Then, first order tree-width is defined as a special case. Although it is our only application of stratified tree-width,

stating results in greater generality allows us to focus on their essence. It is also quite possible, that further applications will arise in the future.

Any rooted tree  $T$  induces a natural partial order  $<_T$  on its nodes, where the smallest element is the root. For a tree decomposition  $(T, B)$  of a graph  $G$  and a vertex  $v \in V(G)$ , let  $t_v \in T$  denote the  $<_T$ -minimal tree node that covers  $v$ . By (TD3), the node  $t_v$  is well-defined. Now, let  $d: V(G) \rightarrow \mathbb{N}$  be a function. We say that a tree decomposition  $(T, B)$  of  $G$  is  $d$ -stratified, if all  $u, v \in V(G)$  with  $t_u <_T t_v$  satisfy  $d(u) \leq d(v)$ . The *tree-width* of  $(G, d)$  is defined as

$$\text{tw}(G, d) := \min \{w(T, B) \mid (T, B) \text{ is a } d\text{-stratified tree decomposition of } G\}.$$

For a formula  $\varphi$ , the *formula graph* is the undirected graph  $G_\varphi$ , with vertices  $\text{var}(\varphi)$ , and edges  $\{x, y\}$  whenever  $x$  and  $y$  are free variables, or when  $x$  and  $y$  occur together in some atom of  $\varphi$ . (If  $\varphi$  is not straight, then we obtain the formula graph of  $\varphi$  by first making it straight.) Note that the formula graph depends on the syntax of the formula. Logically equivalent formulae may have different formula graphs.

We now introduce a partial order  $\preceq_\varphi$  on the variables of a formula  $\varphi$ , from which we then obtain the *essential alternation depth*,  $d_\varphi(x)$ , of a variable  $x \in \text{var}(\varphi)$ . Given a tree decomposition of  $G_\varphi$  of width  $k+1$  that respects  $d_\varphi$ , we then transform the formula  $\varphi$  bottom up along the decomposition into an equivalent  $\mathcal{L}^k$ -formula. In this transformation, we want to ‘reuse’ as many variables as possible, so, intuitively, the ‘worst case’ is that  $\varphi$  is in prenex normal form. Hence we want to ‘undo’ prenex normal form, pushing quantifiers as far as possible away from the root in the syntax tree. Of course, we have to make sure that we obtain an equivalent formula. Intuitively,  $\preceq_\varphi$  gives us a partial order of quantifications that we have to respect while undoing prenex normal form.

For a bound variable  $x$ , let  $Q_x \in \{\exists, \forall\}$  be the type of quantifier used to quantify  $x$ . For bound variables  $x, y$  of  $\varphi$ , we write  $x \leq_\varphi y$  to denote that  $x = y$  or  $y$  is quantified in the scope of  $x$ . For a set  $X$  of variables, we use  $\varphi_{[X]}$  to denote the minimal subformula of  $\varphi$  which contains all atoms that use variables from  $X$ . Let  $\preceq_\varphi$  be the minimal (with respect to  $\subseteq$ ) binary relation on  $\text{var}(\varphi)$ , such that the following hold, where we say that two variables  $x$  and  $y$  are entangled, if there are  $X \subseteq (x \preceq_\varphi)$  (meaning  $x \preceq_\varphi x'$  for all  $x' \in X$ ) and  $Y \subseteq (y \preceq_\varphi)$ , such that  $x$  occurs in  $\varphi_{[Y]}$  and  $y$  occurs in  $\varphi_{[X]}$ .

1.  $\preceq_\varphi$  is reflexive.
2.  $\preceq_\varphi$  is transitive.
3. If  $x \leq_\varphi y$ ,  $Q_x \neq Q_y$  and there is a sequence  $x = z_0, \dots, z_n = y$  of bound variables such that for all  $0 \leq i < n$  we have that  $z_i, z_{i+1}$  are entangled and that  $x \preceq_\varphi z_i$  or  $y \preceq_\varphi z_i$ , then  $x \preceq_\varphi y$  (*Alternation*).

It is clear, that these closure conditions are monotone on  $\preceq_\varphi$ . (Some explanations of the term monotonicity in this context might be in order. The above conditions on  $\preceq_\varphi$  can be seen as closure of  $\preceq_\varphi$  under certain operations on binary relations. For example transitivity is closure under the operation  $R \mapsto R^2$ , that is  $\preceq_\varphi$  must satisfy  $(\preceq_\varphi)^2 \subseteq (\preceq_\varphi)$ . The statement is, that the operations implicit in the three

conditions are monotone with respect to  $\subseteq$ .) Hence they constitute an inductive definition.

- Remark 1.* 1. The relation  $\preceq_\varphi$  is a subrelation of  $\leq_\varphi$ :  $(\preceq_\varphi) \subseteq (\leq_\varphi)$ ,  
 2. the relation  $\preceq_\varphi$  is a partial order, and  
 3.  $x \preceq_\varphi y$  holds whenever  $x$  and  $y$  are entangled,  $Q_x \neq Q_y$ , and  $x \leq_\varphi y$ .

*Proof.* 1 follows since  $\leq_\varphi$  satisfies all closure conditions.

2:  $x \preceq_\varphi y$  is reflexive and transitive by definition, and it inherits anti-symmetry from  $(\leq_\varphi)$  by 1.

3: This follows by letting  $n = 1$  in Alternation.  $\square$

We have defined entanglement somewhat complicated, to make monotonicity clear. Setting  $\varphi_x := \varphi_{\llbracket x \preceq \rrbracket}$ , we can now simplify the definition, by noting that  $x$  and  $y$  are entangled, if and only if  $x$  occurs in  $\varphi_y$  and  $y$  occurs in  $\varphi_x$ . Observe, that  $\varphi_x$  is a subformula of the scope of  $x$ .

*Example 1.* Let  $\varphi := \exists x \forall y \exists z (Pxy \wedge \forall u (Ryu \vee Pzu))$ . Then  $x \preceq_\varphi y$  and  $z \preceq_\varphi u$  by Remark [1](#).3,  $y \preceq_\varphi z$  by Alternation (witnessed by the sequence  $y, u, z$ ), and  $x \preceq_\varphi z$ ,  $y \preceq_\varphi u$ , and  $x \preceq_\varphi u$  by Transitivity.

Last, let  $d_\varphi(x)$  denote the *essential alternation depth* of  $x$  in  $\varphi$ , that is the maximum over all  $\preceq_\varphi$ -paths  $P$  ending in  $x$  of the number of quantifier changes in  $P$ , adding  $+1$  in case the first variable on  $P$  is existentially quantified and  $+2$  if it is universally quantified. If  $x$  is a free variable, we set  $d_\varphi(x) = 0$ . Observe, that we have  $d_\varphi \leq t$  for  $\varphi \in \Sigma_t$ . For instance, formula  $\varphi$  of Example [1](#) satisfies  $d_\varphi(x) = 1$ ,  $d_\varphi(y) = 2$ ,  $d_\varphi(z) = 3$ , and  $d_\varphi(u) = 4$ .

For a formula  $\varphi$  we let  $\text{fotw}(\varphi) := \text{tw}(G_\varphi, d_\varphi)$ . Accordingly, we say that  $(T, B)$  is a *tree decomposition for  $\varphi$* , if  $(T, B)$  is a  $d_\varphi$ -stratified tree decomposition for  $G_\varphi$ . We will make frequent use of the following well-known fact about tree decompositions:

**Fact.** *Let  $(T, B)$  be a tree decomposition of some graph  $G$ , and let  $C \subseteq V(G)$ . If for all  $v, w \in C$ , some piece of  $(T, B)$  covers both  $v$  and  $w$ , then there is some piece  $B_t$  covering  $C$  entirely, i.e.  $C \subseteq B_t$ .*

*In particular, each clique of  $G$  is covered by some piece.*

Note that for a formula  $\varphi$ , the variables  $\text{free}(\varphi)$ , as well as the variables of any atom or literal in  $\varphi$  induce cliques in  $G_\varphi$ . In particular, since  $d_\varphi(x) = 0$  for any free variable  $x$ , it is no restriction to require that the free variables be covered in the root of a tree decomposition.

Prenex normal form aims to make the scopes of quantifiers as large as possible. Working in the opposite direction, we obtain what we term xenerp normal form.

**Definition 1.** *A subformula  $\chi$  of a formula  $\varphi$  is in xenerp normal form with respect to  $\varphi$ , if for all variables  $x$  quantified in  $\chi$  the following holds:  $\varphi_x$  is immediately preceded by a quantifier sequence which contains  $Q_x x$ .*

*A formula  $\varphi$  is in xenerp normal form, if it is in xenerp normal form with respect to itself.*

*Example 2.* Let  $\varphi := \forall x \exists y \forall z (Pzy \vee (Rx \wedge Ry))$ , and let  $\psi := \exists y \forall z (Pzy \vee (\forall x Rx \wedge Ry))$ . Then  $\varphi$  and  $\psi$  are equivalent and  $\psi$  is in xenerp normal form, whereas  $\varphi$  is not in xenerp normal form. Moreover, we have  $d_\varphi(x) = 2$ ,  $d_\varphi(y) = 1$ ,  $d_\varphi(z) = 2$ , and  $d_\psi = d_\varphi$ .

The following Lemma presents equivalent transformations of formulae, such that neither the formula graph, nor the essential alternation depth, nor the corresponding tree decompositions change.

**Lemma 1.** *Let  $\varphi$  and  $\psi$  be formulae satisfying either 1, 2 or 3.*

1. *There are formulae  $\chi_1, \dots, \chi_n$ , a positive Boolean combination  $\theta$  of  $n$  arguments, and a variable  $x$  which does not occur in  $\chi_2, \dots, \chi_n$ , such that  $\psi$  is obtained from  $\varphi$  by replacing a subformula  $\theta(Q_x x \chi_1, \chi_2, \dots, \chi_n)$  by  $Q_x x \theta(\chi_1, \chi_2, \dots, \chi_n)$ .*
2. *There are a formula  $\chi$ , and variables  $x, y$  with  $Q_x = Q_y$  such that  $\psi$  is obtained from  $\varphi$  by replacing a subformula  $Q_x x Q_y y \chi$  by  $Q_y y Q_x x \chi$ .*
3. *There are a formula  $\chi$ , and variables  $x, y$  with  $\varphi_x$  a proper subformula of  $\varphi_y$ , such that  $\psi$  is obtained from  $\varphi$  by replacing a subformula  $Q_x x Q_y y \chi$  by  $Q_y y Q_x x \chi$ , where  $Q_y y \chi$  is xenerp with respect to  $\varphi$ .*

Then  $\varphi \equiv \psi$ ,  $d_\varphi = d_\psi$ , and  $G_\varphi = G_\psi$ . Consequently, tree decompositions for  $\varphi$  coincide with tree decompositions for  $\psi$  and in particular  $\text{fotw}(\varphi) = \text{fotw}(\psi)$ .

Implicitly, we assume in these cases that  $Q_x$  and  $Q_y$  are the same with respect to the formula  $\varphi$  and with respect to the formula  $\psi$ .

The (rather technical) proof is deferred to the full version. Observe, that the first class of replacements in the previous lemma are exactly what is used in turning a formula into prenex normal form. All classes are used for xenerp normal form.

**Corollary 1.** *Let  $\varphi$  be a formula and let  $\psi$  be a prenex normal form (obtained in the usual way) of  $\varphi$ . Then  $\text{fotw}(\varphi) = \text{fotw}(\psi)$ .*

**Lemma 2.** *From a formula  $\varphi$  we can compute in polynomial time a formula  $\psi$  in xenerp normal form, such that  $\varphi \equiv \psi$ ,  $G_\varphi = G_\psi$ ,  $d_\varphi = d_\psi$ , and consequently tree decompositions for  $\varphi$  coincide with those for  $\psi$ .*

The proof can be found in the full version.

A (Boolean) conjunctive query is a sentence  $\varphi = \exists x_1 \dots \exists x_n \psi$ , where  $\psi$  is a conjunction of relational atoms such that  $\text{var}(\psi) = \{x_1, \dots, x_n\}$ . The tree-width of a conjunctive query  $\varphi$ ,  $\text{tw}(\varphi)$ , is defined as the tree-width of  $G_\varphi$  (see [21]). For a conjunctive query  $\varphi$  we have  $d_\varphi = 1$ . Hence the notion of  $\text{fotw}$  generalises the notion of tree width of conjunctive queries.

*Remark 2.* Any conjunctive query  $\varphi$  satisfies  $\text{fotw}(\varphi) = \text{tw}(\varphi)$ . □

For the relation of  $\text{fotw}$  to tree-width of non-recursive stratified datalog and to elimination-width we need some more definitions, and the reader is referred to Section 6. In general, the difference between  $\text{fotw}(\varphi)$  and  $\text{tw}(G_\varphi)$  can be unbounded:

**Proposition 1.** *For every  $n > 0$  there is a formula  $\varphi_n$  with  $\text{fotw}(\varphi_n) = n$  and  $\text{tw}(G_{\varphi_n}) = 1$ .*

*Proof.* Let

$$\varphi_n = \exists x_1 \dots \exists x_n \forall y \bigwedge_{1 \leq i \leq n} E x_i y.$$

Then  $G_{\varphi_n}$  is the  $n$ -star with center  $y$ , and we have  $d_{\varphi_n}(x_i) = 1$  for all  $1 \leq i \leq n$ , and  $d_{\varphi_n}(y) = 2$ . It is easy to see that any  $d_{\varphi_n}$ -stratified tree decomposition  $(T, B)$  of  $G_{\varphi_n}$  has a piece  $\{x_1, x_2, \dots, x_n, y\}$ , namely  $B_{t_y}$ . On the other hand, such a tree decomposition needs no other pieces. Hence  $\text{fotw}(\varphi_n) = n$ . Since  $G_{\varphi_n}$  is a tree we have  $\text{tw}(G_{\varphi_n}) = 1$ .  $\square$

## 4 Computing Stratified Tree Decompositions

In this section, we show that computing stratified tree decompositions of optimal width is fixed-parameter tractable, where the parameter is the stratified tree-width. In fact, the running time is linear for bounded stratified tree-width. In the next section, we will use the algorithm here developed as a first step for formula evaluation.

**Definition 2.** *Let  $G = (V, E)$  be a graph and  $d : V \rightarrow \mathbb{N}$ . We say that  $(G, d)$  is normalized, if for some  $n \in \mathbb{N}$  we have  $V = \{1, \dots, n\}$  and  $d(v) \leq n$  for all  $v \in V$ .*

**Theorem 1.** *There is an algorithm that, given a normalized  $(G, d)$ , computes a  $d$ -stratified tree decomposition of  $G$  of minimum width in time  $O(|V(G)| \cdot 2^{\text{poly}(\text{tw}(G, d))})$ .*

The proof of the theorem will be included in the full version. As each  $(G, d)$  can be normalized in time  $O(n \log n)$ , a similar statement holds for arbitrary  $(G, d)$ , albeit not with linear running time. Theorem [1](#) implies that we can efficiently decide whether a formula  $\varphi$  satisfies  $\text{fotw}(\varphi) \leq k$ :

**Corollary 2.** *There is an algorithm that, given a formula  $\varphi \in \mathcal{L}$ , computes a tree decomposition of  $\varphi$  of minimum width in time  $\text{poly}(\|\varphi\|) + |\text{var}(\varphi)| \cdot 2^{\text{poly}(\text{fotw}(\varphi))}$ .*

*Proof.* Given  $\varphi$ , it is first turned into a straight formula. Then,  $G_\varphi$  and  $d_\varphi$  are computed and normalized in polynomial time. The last step is a call to the algorithm from Theorem [1](#). The only part that might need further clarification is, that computing  $\preceq_\varphi$  (as needed to compute  $d_\varphi$ ) can be done in polynomial time. This follows from the three closure operators each being computable in polynomial time and the size of final relation  $\preceq_\varphi$  being polynomially bounded by  $\|\varphi\|$ .  $\square$

## 5 Query Evaluation on Bounded First Order Tree-Width

This section contains the second main result: Evaluating formulae of bounded tree-width is fixed parameter tractable with parameter the length of the formula. Moreover, we show that evaluating quantified constraint formulae of bounded first order tree-width can be done in polynomial time. This is a stronger than Chen and Dalmau’s result [8] for quantified constraint formulae of bounded elimination-width: As we will see in Section 6, bounded elimination-width (i.e. bounded *tree-width* in [8]) implies bounded first order tree-width, but there are classes of quantified constraint formulae with unbounded elimination-width, that have bounded bounded first order tree-width.

For an integer  $k \geq 0$  let  $\mathcal{L}^k$  be the fragment of formulae  $\varphi$  of  $\mathcal{L}$ , such that all subformulae of  $\varphi$  have at most  $k$  free variables. Note that  $\mathcal{L}^k$  is equivalent to the fragment of  $\mathcal{L}$  consisting of all (not necessarily straight) formulae with at most  $k$  variables.

The question whether a given formula is equivalent to an  $\mathcal{L}^k$  formula for some fixed  $k$  is undecidable. (This can be seen by a reduction from satisfiability: First, satisfiability reduces to satisfiability by an infinite structure using relativization. Then, let  $\varphi$  be some fixed formula, such that  $\varphi$  does not have finite models, and such that  $\varphi$  is not equivalent to any  $\mathcal{L}^k$ -formula. Now, a given formula  $\psi$  is unsatisfiable by infinite structures, if and only if  $\varphi \wedge \psi'$  is equivalent to an  $\mathcal{L}^k$  formula, where  $\psi'$  is obtained from  $\psi$  by renaming all symbols to be disjoint from those of  $\varphi$ .) As a consequence, there is no computable width parameter such that width- $k$  captures all formulae logically equivalent to a formula of  $\mathcal{L}^k$ . For any width parameter, only formulae which are ‘syntactically close’ to a formula of  $\mathcal{L}^k$  are captured, for varying values of ‘syntactically close’.

*Example 3.* Let  $n > 2$ .  $\varphi_n = \psi_n \vee \chi$ , where  $\text{fotw}(\chi) = 2$  and  $\text{fotw}(\psi_n) = n$ , but  $\psi_n$  is unsatisfiable. Then  $\varphi_n \equiv \chi \in \mathcal{L}^2$ , but  $\text{fotw}(\varphi_n) = n$ .

First order sentences of tree-width at most  $k - 1$  have the same expressive power as  $\mathcal{L}^k$ . More generally we have the following.

**Theorem 2.** *Let  $k \geq 0$ .*

1. *For any formula  $\varphi$  with  $\text{fotw}(\varphi) \leq k - 1$  there is a formula  $\psi \in \mathcal{L}^k$  with  $\varphi \equiv \psi$  which is computable from  $\varphi$ .*
2. *Any formula  $\psi \in \mathcal{L}^k$  satisfies  $\text{fotw}(\psi) \leq k - 1$ , and we can compute a tree decomposition witnessing this in time linear in the size of  $\psi$ .*

The rather technical proof is deferred to the full version. It is well-known that first order query evaluation for  $\mathcal{L}^k$  is in polynomial time [24].

**Corollary 3.** *Evaluating queries of bounded first order tree-width is fixed parameter tractable with parameter the length of the formula.*

*More precisely, given a finite structure  $\mathcal{A}$  and a formula  $\varphi \in \mathcal{L}$ , there is an algorithm that computes  $\varphi(\mathcal{A})$  in time  $\|\mathcal{A}\|^{\text{fotw}(\varphi)+O(1)} f(|\varphi|)$  for some computable function  $f$ .*

Here the function  $f$  is basically the running time we need for translating the formula  $\varphi$  with  $\text{fotw}(\varphi) \leq k - 1$  into an  $\mathcal{L}^k$  formula. It is  $q$ -times exponential, where  $q$  is the alternation depth of  $\varphi$ . The exponentiations arise from converting some subformulae into disjunctive or conjunctive normal form. In particular, if  $\varphi$  does not use any disjunctions to start with (for example  $\varphi$  is a quantified constraint formula), then all respective subformulae already are in disjunctive as well as conjunctive normal form, and the running time is much lower.

**Corollary 4.** *Evaluating quantified constraint formulae of bounded first order tree-width can be done in polynomial time.*

## 6 Relation to Similar Notions

A *quantified constraint formula* [8] is a sentence

$$\varphi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \psi,$$

where  $Q_i \in \{\forall, \exists\}$  for  $i = 1 \dots n$  and  $\psi$  is a conjunction of relational atoms. In [8], Chen and Dalmau introduce the notion of *tree-width* of a quantified constraint formula and they show that model checking for quantified constraint formulae of bounded tree-width can be done in polynomial time using the  $k$ -consistency algorithm. Since their notion of tree-width is defined via an elimination ordering rather than via a decomposition, we call it *elimination-width* instead of *tree-width*. We generalise the definition of elimination-width to formulae of  $\mathcal{L}$ , and we show that  $\text{fotw}$  is less or equal to elimination-width. Thus  $\text{fotw}$  yields a larger class of tractable formulae than elimination width.

Let  $G = (V, E)$  be a graph and  $d : V \rightarrow \mathbb{N}$ . An *elimination ordering* for  $(G, d)$  is a linear ordering  $(x_1, \dots, x_n)$  of  $V$  which *respects*  $d$ , i.e.  $i < j$  implies  $d(x_i) \leq d(x_j)$ . With an elimination ordering we associate a sequence of graphs as follows:

- $G_n := G$
- $G_{i-1} := G_i - x_i + \{\{u, v\} \mid \{u, x_i\}, \{x_i, v\} \in E(G_i)\}$ , for  $1 < i \leq n$

The *width* of an elimination ordering is  $\max_{i \in [n]} \{\deg(x_i) \text{ in } G_i\}$ . The *elimination-width* of  $(G, d)$ ,  $\text{ew}(G, d)$ , is the minimum width of an elimination ordering of  $(G, d)$ . For a formula  $\varphi$  we define  $\text{ew}(\varphi) := \text{ew}(G_\varphi, d_\varphi)$ . (If  $\varphi$  is not straight, then we first transform  $\varphi$  into a straight formula  $\varphi'$  and we let  $\text{ew}(\varphi) := \text{ew}(\varphi')$ .) It is well-known that the tree-width of a graph  $G$  equals the elimination-width of  $G$ , and this fact can be generalised to our setting.

**Theorem 3.** *Let  $G$  be a graph and  $d : V(G) \rightarrow \mathbb{N}$ . Then  $\text{tw}(G, d) = \text{ew}(G, d)$ . In particular, any formula  $\varphi \in \mathcal{L}$  satisfies  $\text{fotw}(\varphi) = \text{ew}(\varphi)$ .*

The proofs of Theorem [3] and of the following lemma can be found in the full version.

Chen and Dalmau's notion of elimination-width [8] can be equivalently reformulated in our setting as follows. Let  $\varphi$  be a quantified constraint formula with

formula graph  $G_\varphi$ . Let  $d'_\varphi$  be the mapping that assigns to a variable  $v \in \text{var}(\varphi)$  the alternation depth of  $v$  (i.e. the number of quantifier changes occurring before  $v$  in the quantifier prefix of  $\varphi$ ), adding +1. Then Chen and Dalmau's notion of elimination-width is  $\text{ew}(G_\varphi, d'_\varphi)$ .

**Lemma 3.** *Let  $\varphi$  be a quantified constraint formula. Then  $\text{fotw}(\varphi) = \text{ew}(\varphi) \leq \text{ew}(G_\varphi, d'_\varphi)$ .*

The following example exhibits a class of quantified constraint formulae having first order tree-width 1, where Chen and Dalmau's elimination-width is unbounded.

*Example 4.* For an integer  $n > 0$  let

$$\varphi_n := \exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_n \forall y_n \exists z \left( \bigwedge_{i=1}^n R x_i z \wedge \bigwedge_{i=1}^n P y_i \right).$$

Then for all  $i \leq n$  we have  $d_{\varphi_n}(x_i) = 1 = d_{\varphi_n}(z)$  and  $d_{\varphi_n}(y_i) = 2$ , and hence  $\text{fotw}(\varphi_n) = 1$ . Moreover,  $\text{ew}(G_{\varphi_n}, d'_{\varphi_n}) = n$ .

In [13], Flum, Frick and Grohe define tree-width of non-recursive stratified datalog (NRSD) programs and they show that the evaluation problem for NRSD programs can be solved in polynomial time on programs of bounded strict tree-width [13, Corollary 5.26]. NRSD programs have the same expressive power as  $\mathcal{L}$  and every NRSD program corresponds to a first order formula and vice versa. This allows us to compare their notion with  $\text{fotw}$ . We denote the *strict tree-width* of a datalog program  $\Pi$  by  $\text{stw}(\Pi)$  and we refer the reader to [13] for the definition of strict tree-width of NRSD programs.

We show that if the NRSD program has strict tree-width at most  $k$ , then the corresponding first order formula has  $\text{fotw}$  at most  $k$ , and we exhibit a class of formulae with bounded  $\text{fotw}$ , whose corresponding NRSD programs have unbounded strict tree-width. Let  $\varphi_\Pi$  denote the first-order formula associated with an NRSD program  $\Pi$ .

**Lemma 4.** *For any NRSD program  $\Pi$  we have  $\text{fotw}(\varphi_\Pi) \leq \text{stw}(\Pi)$ .*

Again, the proof must be deferred to the full version. As with quantified constraint formulae, the difference can be unbounded in the opposite direction.

*Remark 3.* There is a class  $\mathcal{C}$  of NRSD programs with unbounded strict tree-width, such that  $\text{fotw}(\varphi_\Pi) = 0$  for all  $\Pi \in \mathcal{C}$ .

The programs from  $\mathcal{C}$  are NRSD versions of the formulae from Example 4.

We now introduce the robber and cops game as defined in [23]. Let  $G$  be a graph and let  $k \geq 0$  be an integer. The robber and cops game on  $G$  (with game parameter  $k$ ) is played by two players, the *cop player* and the *robber player*, on the graph  $G$ . The cop player controls  $k$  cops and the robber player controls the robber. Both the cops and the robber move on the vertices of  $G$ . Some of



the cops move to at most  $k$  vertices and the robber stands on a vertex  $r$  not occupied by the cops. In each move, some of the cops fly in helicopters to at most  $k$  new vertices. During the flight, the robber sees which position the cops are approaching and before they land she quickly tries to escape by running arbitrarily fast along paths of  $G$  to a vertex  $r'$ , not being allowed to run through a standing cop. Hence, if  $X \subseteq V(G)$  is the cops' first position, the robber stands on  $r \in V(G) \setminus X$ , and after the flight, the cops occupy the set  $Y \subseteq V(G)$ , then the robber can run to any vertex  $r'$  within the connected component of  $G \setminus (X \cap Y)$  containing  $r$ . The cops win if they land a cop via helicopter on the vertex occupied by the robber. The robber wins if she can always elude capture. *Winning strategies* are defined in the usual way. The *cop-width* of  $G$ ,  $\text{cw}(G)$ , is the minimum number of cops having a winning strategy on  $G$ .

A winning strategy for the cops is *monotone*, if for any sequence  $X_1, X_2, \dots$  of cop positions and all possible responses of the robber, the connected components  $R_i$  of  $G \setminus X_i$  containing the robber form a decreasing (with respect to  $\subseteq$ ) sequence. The  $R_i$  are called the *robber spaces*. The *monotone cop-width* of  $G$ ,  $\text{mon-cw}(G)$ , is the minimum number of cops having a monotone winning strategy on  $G$ .

**Theorem 4 (Seymour, Thomas [23]).** *Any graph  $G$  satisfies  $\text{tw}(G) + 1 = \text{cw}(G) = \text{mon-cw}(G)$ .*

Now let  $G$  be a graph and let  $d$  be a function  $d: V(G) \rightarrow \mathbb{N}$ . The  *$d$ -stratified* robber and cops game on  $G$  is played as the robber and cops game on  $G$ , but in every move the cops have to satisfy the following additional condition. Intuitively, they can only clear vertices  $x$  with  $d(x) = i$  after they have cleared all vertices  $y$  with  $d(y) < i$ . More precisely: For every move  $(X, R)$ , where  $X \subseteq V(G)$  is the cop position and  $R$  is the robber space, the cops have to make sure that  $\max\{d_\varphi(x) \mid x \in X\} \leq \min\{d_\varphi(r) \mid r \in R\}$ . Then  $\text{cw}(G, d)$  and  $\text{mon-cw}(G, d)$  are defined analogously, and for a formula  $\varphi$  we let  $\text{cw}(\varphi) := \text{cw}(G_\varphi, d_\varphi)$  and  $\text{mon-cw}(\varphi) := \text{mon-cw}(G_\varphi, d_\varphi)$ .

Although proving the following theorem is not very hard, it seems interesting to know that  $\text{cw}(G, d)$  and  $\text{mon-cw}(G, d)$  coincide. In many generalisations of the robber and cops game to other settings, the analogous statements become false [12, 22], and it might be helpful to explore the borderline. The proof can be found in the full version.

**Theorem 5.** *Let  $G = (V, E)$  be a graph and let  $d: V \rightarrow \mathbb{N}$ . Then  $\text{tw}(G, d) + 1 = \text{mon-cw}(G, d) = \text{cw}(G, d)$ .*

*In particular, any first order formula  $\varphi$  satisfies  $\text{fotw}(\varphi) + 1 = \text{mon-cw}(\varphi) = \text{cw}(\varphi)$ .*

## 7 Conclusion

We introduced a notion of tree-width for first order formulae  $\varphi$ ,  $\text{fotw}(\varphi)$ , generalising tree-width of conjunctive queries and elimination-width of quantified constraint formulae [8]. Our notion can also be seen as an adjustment of the

notion of tree-width of first order formulae as defined in [13] (which only works for conjunctive queries with negation).

We proved that computing  $\text{fotw}$  is fixed-parameter tractable with parameter  $\text{fotw}$  (Theorem 1). Moreover, we showed that evaluating formulae of  $k$ -bounded first order tree-width is fixed-parameter tractable, with parameter the length of the formula (Theorem 2). This is done by first computing a tree decomposition of width at most  $k$  for the formula, and then translating the formula equivalently into a formula of the  $k$ -variable fragment  $\mathcal{L}^k$  of first order logic. It is well-known that evaluating  $\mathcal{L}^k$  formulae can be done in polynomial time. When translating the formula  $\varphi$  into an equivalent  $\mathcal{L}^k$  formula, we get a non-elementary explosion in the running time.

*Conjecture 1.* When translating a formula  $\varphi$  satisfying  $\text{fotw}(\varphi) \leq k$  into an equivalent  $\mathcal{L}^k$  formula, a non-elementary explosion cannot be avoided.

Moreover, the precise complexity of evaluating queries of bounded first order tree-width is still unknown.

We show that first order tree-width can be characterised by other notions such as elimination-width (Theorem 3), and the minimum number of cops necessary to catch the robber in the stratified robber and cops game, as well as the minimum number of cops necessary in the monotone version of the game (Theorem 5). Hence our notion is very natural and robust.

Moreover, we showed that  $\text{fotw}$  is more powerful than the notion of elimination-width of quantified constraint formulae as defined in [8]: For quantified constraint formulae, both bounded elimination-width and bounded  $\text{fotw}$  allow for model checking in polynomial time. We proved that if  $\varphi$  is a quantified constraint formula, then  $\text{fotw}(\varphi)$  is bounded by the elimination-width of  $\varphi$ , and there are classes of quantified constraint formulae with bounded  $\text{fotw}$  and unbounded elimination-width.

In [13], Flum, Frick and Grohe define strict tree-width of non-recursive stratified datalog (NRSD) programs and they show that the evaluation problem for NRSD programs can be solved in polynomial time on programs of bounded strict tree-width. NRSD programs have the same expressive power as first order logic, in the sense that NRSD programs correspond to first order formulae and vice versa. We showed that first-order tree-width of (formula versions of) NRSD programs is bounded by the strict tree-width of the programs and that there are classes of first order formulae with bounded  $\text{fotw}$ , whose corresponding NRSD programs have unbounded strict tree-width.

For conjunctive query evaluation, methods more powerful than bounded tree-width are known. Conjunctive queries of bounded hypertree-width [17], bounded fractional hypertree-width [20] and bounded (hyper)closure tree-width [4] yield even larger tractable classes of instances. For example, conjunctive queries of bounded hypertree-width correspond to the  $k$ -guarded fragment of first order logic [18], and similar correspondences can be found for the other invariants. Why not generalise these notions to first order formulae? By generalising these notions to first order formulae  $\varphi$  in the obvious way, a decomposition of bounded

width would not give us an instruction how to translate  $\varphi$  into the corresponding guarded fragment of first order logic (transforming subformulae of  $\varphi$  into conjunctive normal form as in the proof of Theorem 2.1 does not necessarily yield guarded subformulae).

Nevertheless, generalising these notions to quantified constraint formulae should indeed yield classes with an efficient query evaluation, that are strictly larger than classes of quantified constraint formulae of bounded first order tree-width. It would be interesting to find the largest fragment of first order formulae for which such a generalization is possible.

## References

1. Adler, I.: Marshals, monotone marshals, and hypertree-width. *Journal of Graph Theory* 47, 275–296 (2004)
2. Adler, I.: Directed tree-width examples. *J. Comb. Theory, Ser. B* 97(5), 718–725 (2007)
3. Adler, I.: Tree-related widths of graphs and hypergraphs. *SIAM J. Discrete Math.* 22(1), 102–123 (2008)
4. Adler, I.: Tree-width and functional dependencies in databases. In: Lenzerini, M., Lembo, D. (eds.) *PODS*, pp. 311–320. ACM, New York (2008)
5. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* 25(6), 1305–1317 (1996)
6. Chandra, A.K., Merlin, P.M.: Optimal implementation of conjunctive queries in relational data bases. In: *STOC*, pp. 77–90. ACM, New York (1977)
7. Chekuri, C., Rajaraman, A.: Conjunctive query containment revisited. *Theor. Comput. Sci.* 239(2), 211–229 (2000)
8. Chen, H., Dalmau, V.: From pebble games to tractability: An ambidextrous consistency algorithm for quantified constraint satisfaction. In: Ong, L. (ed.) *CSL 2005*. LNCS, vol. 3634, pp. 232–247. Springer, Heidelberg (2005)
9. Dalmau, V., Kolaitis, P.G., Vardi, M.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Van Hentenryck, P. (ed.) *CP 2002*. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
10. Downey, R.G., Fellows, M.R.: *Parameterized Complexity*. Springer, Heidelberg (1999)
11. Ebbinghaus, H.-D., Flum, J.: *Finite Model Theory*. Springer, Heidelberg (1990)
12. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.* 28(1), 57–104 (1998)
13. Flum, J., Frick, M., Grohe, M.: Query evaluation via tree-decompositions. *J. ACM* 49(6), 716–752 (2002)
14. Flum, J., Grohe, M.: *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, Secaucus (2006)
15. Flum, J., Grohe, M.: *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer, Secaucus (2006)
16. Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: *AAAI*, pp. 4–9 (1990)
17. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences* 64, 579–627 (2002)

18. Gottlob, G., Leone, N., Scarcello, F.: Robbers, marshals, and guards: Game theoretic and logical characterizations of hypertree width. *Journal of Computer and System Sciences* 66, 775–808 (2003)
19. Gottlob, G., Greco, G., Scarcello, F.: The complexity of quantified constraint satisfaction problems under structural restrictions. In: Kaelbling, L.P., Saffiotti, A. (eds.) *IJCAI*, pp. 150–155. Professional Book Center (2005)
20. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: *Proceedings of the of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms* (2006) (to appear)
21. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.* 61(2), 302–332 (2000)
22. Kreutzer, S., Ordyniak, S.: Digraph decompositions and monotonicity in digraph searching. In: *CoRR*, abs/0802.2228 (2008)
23. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B* 58, 22–33 (1993)
24. Vardi, M.Y.: On the complexity of bounded-variable queries (extended abstract), pp. 266–276 (1995)
25. Yannakakis, M.: Algorithms for acyclic database schemes. In: *VLDB*, pp. 82–94. IEEE Computer Society, Los Alamitos (1981)

# Algorithmic Analysis of Array-Accessing Programs<sup>\*</sup>

Rajeev Alur, Pavol Černý, and Scott Weinstein

University of Pennsylvania  
{alur, cernyp, weinstein}@cis.upenn.edu

**Abstract.** For programs whose data variables range over boolean or finite domains, program verification is decidable, and this forms the basis of recent tools for software model checking. In this paper, we consider algorithmic verification of programs that use boolean variables, and in addition, access a single read-only array whose length is potentially unbounded, and whose elements range over a potentially unbounded data domain. We show that the reachability problem, while undecidable in general, is (1) PSPACE-complete for programs in which the array-accessing `for`-loops are not nested, (2) decidable for a restricted class of programs with doubly-nested loops. The second result establishes connections to automata and logics defining languages over data words.

## 1 Introduction

Verification questions concerning programs are undecidable in general. However, for finite-state programs — programs whose data variables range over finite types such as boolean, the number of bits needed to encode a program state is a priori bounded, and verification questions such as reachability are decidable. This result, coupled with progress on symbolic techniques for searching the state-space of finite-state programs, and abstraction techniques for extracting boolean over-approximations of general programs, forms the basis of recent tools for software model checking [3,16].

We focus on algorithmic verification of programs that access a single array. The length of the input array is potentially unbounded. The elements of the array range over  $\Sigma \times D$ , where  $\Sigma$  is a finite set, and  $D$  is a data domain that is potentially unbounded and totally ordered. The array is thus modeled as a *data word*, that is, a sequence of pairs in  $\Sigma \times D$ . For example an array that contains employees' names, and for each name a tag indicating whether the employee is a programmer, a manager, or a director, can be modeled by setting  $D$  to be the set of strings, and  $\Sigma$  to be a set with three elements. The program can have Boolean variables, index variables ranging over array positions, and data variables ranging over  $D$ . Programs can access  $\Sigma$  directly, but can only perform equality and order tests on elements of  $D$ . The expressions in the program can

---

<sup>\*</sup> This research was partially supported by NSF Cybertrust award CNS 0524059.

use constants in  $D$ , and equality tests and ordering over index and data variables. The programs are built using assignments, conditionals, and `for`-loops over the array. Even with these restrictions, one can perform interesting computational tasks including searching for a specific value, finding the minimum data value, checking that all values in the array are within specific bounds, or checking for duplicate data values. For example, Java midlets designed to enhance features of mobile devices include simple programs accessing the address books, and our methods can lead to an automatic verification tool that certifies their correctness before being downloaded. For programs that fall outside the restrictions mentioned above, it is possible to use abstract interpretation techniques such as predicate abstraction [13] to abstract some of the features of the program, and analyze the property of interest on the abstract program. As the abstract programs are nondeterministic, we will consider nondeterministic programs.

Our first result is that the reachability problem for programs in which there are no *nested* loops is decidable. The construction is by mapping such a program to a finite-state abstract transition system such that every finite path in the abstract system is feasible in the original program for an appropriately chosen array. We show that the reachability problem for programs with non-nested loops is PSPACE-complete, which is the same complexity as that for finite-state programs with only boolean variables.

Our second result shows decidability of reachability for programs with doubly-nested loops with some restrictions on the allowed expressions. The resulting complexity is non-elementary, and the interest is mainly due to the theoretical connections with the recently well-studied notions of automata and logics over *data words* [6,5,17]. Among different kinds of automata over data words that have been studied, *data automata* [6] emerged as a good candidate definition for the notion of regularity for languages on data words. A data automaton first rewrites the  $\Sigma$ -component to another finite alphabet  $\Gamma$  using a nondeterministic finite-state transducer, and then checks, for every data value  $d$ , whether the projection obtained by deleting all the positions in which the data value is not equal to  $d$ , belongs to a regular language over  $\Gamma$ . In order to show decidability of the reachability problem for programs with doubly nested loops, we extend this definition as follows: An *extended data automaton* first rewrites the data word as in case of data automata. For every data value  $d$ , the corresponding projection contains more information than in case of data automata. It is obtained by replacing each position with data value different from  $d$  by the special symbol 0. The projection is required to be in a regular language over  $\Gamma \cup \{0\}$ . We prove that the reachability problem for extended data automata can be reduced to emptiness of multi-counter automata (or equivalently, to Petri nets reachability), and is thus decidable. We then show that a program containing doubly-nested loops can be simulated, under some restrictions, by an extended data automaton. Relaxing these restrictions leads to undecidability of the reachability problem for programs with doubly-nested loops.

Analyzing reachability problem for programs brings a new dimension to investigations on logics and automata on data words. We establish some new connections,

in terms of expressiveness and decidability boundaries, between programs, logics, and automata over data words. Bojańczyk et al. [6] consider logics on data words that use two binary predicates on positions of the word: (1) an equivalence relation  $\approx$ , such that  $i \approx j$  if the data values at positions  $i$  and  $j$  are equal, and (2) an order  $\prec$  which gives access to order on data values, in addition to standard successor ( $+1$ ) and order  $<$  predicates over the positions. They show that while the first order logic with two variables,  $\text{FO}^2(\approx, <, +1)$ , is decidable, introducing order on data values causes undecidability, that is,  $\text{FO}^2(\approx, \prec, <, +1)$  is undecidable. In this context, our result on programs with non-nested loops is perhaps surprising, as we show that the undecidability does not carry over to these programs, even though they access order on the data domain and have an arbitrary number of index and data variables.

Details of proofs are available in the companion report [1].

## 2 Programs

In this section, we define the syntax and semantics of programs that we will consider. We start by defining arrays. Let  $D$  be an infinite set of data values. We will consider domains  $D$  equipped with equality  $(D, =)$ , or with both equality and linear order  $(D, =, <)$ . Let  $\Sigma$  be a finite set of symbols. An array is a data word  $w \in (\Sigma \times D)^*$ . The program can access the elements of the array via indices into the array.

**Syntax.** The programs have one array variable  $A$ . Variables  $b, b1, b2, \dots$  are boolean. Variables  $p, p1, p2, \dots$  range over  $\mathbb{N}$ , and are called index variables. Variables  $i, j, i1, i2, \dots$  range over  $\mathbb{N}$  and are called loop variables. Variables  $v, v1, v2, \dots$  range over  $D$  and are called data variables. Constants  $c, c1, c2, \dots$  are in  $D$ , and constants  $s, s1, s2, \dots$  are in  $\Sigma$ . We make a distinction between loop and index variables because loop variables cannot be modified outside of the loop header. Index expressions  $IE$  are defined by the following grammar  $IE ::= p \mid i$ . Data expressions  $DE$  are of the form  $DE ::= v \mid c \mid A[IE].d$ , where  $A[IE].d$  accesses the data part of the array.  $\Sigma$ -expressions  $SE$  are of the form  $SE ::= s \mid A[IE].s$ , where  $A[IE].s$  accesses the  $\Sigma$  part of the array. Boolean expressions are defined by the following grammar:  $B ::= \text{true} \mid \text{false} \mid b \mid B \mid B \mid \text{not } B \mid IE = IE \mid IE < IE \mid DE = DE \mid DE < DE \mid SE = SE$ . The programs are defined by the grammar:

$$\begin{aligned} P ::= & \text{skip} \mid \{ P \} \mid b := B \mid p := IE \mid v := DE \\ & \mid \text{if } B \text{ then } P \text{ else } P \mid \text{if } * \text{ then } P \text{ else } P \\ & \mid \text{for } i := 1 \text{ to } \text{length}(A) \text{ do } P \mid P; P \end{aligned}$$

The commands include a nondeterministic conditional. We consider nondeterministic programs in this paper, in order to enable modeling of abstracted programs. Software model checking approaches [13, 3, 16] often rely on predicate abstraction. For example, if the original program contains an assignment of the form  $b := E$ , where  $E$  is a complicated expression that falls out of scope of the intended analysis, the assignment is abstracted into a nondeterministic assignment

to `b`. This is modeled as `if * then b:=true else b:=false` in the language presented here.

**Semantics.** A *global state* of the program is a valuation of its boolean, loop, index and data variables, as well as of the array variable. We denote global states by  $g, g_1$ , and the set of global states by  $G$ . For a boolean, index, loop or data variable  $v$ , we denote the value of  $v$  by  $g[v]$ . The value of the array variable  $A$  is a word  $w \in (\Sigma \times D)^*$ . It is denoted by  $g[A]$ . The length of the array at global state  $g$  is denoted by  $l(g[A])$  and evaluates to the length of  $w$ . Note that the length and the contents of the array do not change over the course of the computation.

Semantics of boolean, index, data and  $\Sigma$  expressions is a partial function:  $\llbracket B \rrbracket : G \rightarrow \mathbb{B}$ ,  $\llbracket IE \rrbracket : G \rightarrow \mathbb{N}$ ,  $\llbracket DE \rrbracket : G \rightarrow D$  and  $\llbracket SE \rrbracket : G \rightarrow \Sigma$ . It is not defined only in cases when there is an array access out of bounds. For example, in a state  $g$  where  $g[A]$  is a word of length 10 and  $g[p]$  is 20, the semantics of the expression  $A[p].d$  is undefined. The semantics of commands is defined as a relation on  $G$ ,  $\llbracket P \rrbracket \subseteq G \times G$ . The definition is presented in detail in [1].

Given a program, a global state is *initial* if either i) the array variable contains a nonempty word, all boolean variables are set to *false*, all index and loop variables are set to 1, and all data variables are set to the same value as the first element of the array; or ii) the array variable contains an empty word, all boolean variables are set to *false*, all index and loop variables are set to 1, and all data variables are set to constant  $c_D \in D$ . The intention is that the only unspecified part of the initial state, the part that models input of the program, is the array. A boolean state is a valuation of all the boolean variables of a program. For a given global state  $g$ , we denote the corresponding boolean state by  $bool(g)$ . For any boolean variable  $b$  of the program, we have that  $bool(g)[b] = g[b]$ . We denote boolean states by  $m, m_1$  and the set of boolean states by  $M$ .

**Restricted fragments.** We classify programs using the nesting depth of loops. We denote programs with only non-nested loops by ND1, programs with nesting depth at most 2 by ND2, etc. Restricted-ND2 programs are programs with nesting depth at most 2, that do not use index or data variables, and do not refer to order on data or indices. Furthermore, a key restriction, such that if it is lifted, the reachability problem becomes undecidable, is a restriction on the syntax of the code inside the inner loop. Let P1 be the code inside an inner loop, and let  $i$  be the loop variable of the outer loop and let  $j$  be the loop variable for the inner loop. P1 must be of the following form: `if A[i].d=A[j].d then P2 else P3`. Furthermore, P3 cannot refer to  $A[j]$ , i.e. it does not contain occurrences of  $A[j].d$  or  $A[j].s$ .

**Examples.** We present three examples illustrating these classes of programs.

*Example 1.* We consider a simple array accessing program `Min` that scans through an array to find a minimal data value. It has one index variable, `p`, and it is an ND1 program, as it does not contain nested loops:

```
for i:= 1 to length(A) do { if A[i].d < A[p].d then p := i }
```

Note that by definition of program semantics, `p` is initialized to 1. The correctness requirement for this program is that the index `p` points to a minimal



```

b:=true;
for i:= 1 to length(A) do {
  if A[i].d < v then b:=false
    else skip;
  v := A[i].d
}

```

Fig. 1. Example 2

```

b:=false;
for i:= 1 to length(A) do {
  for j:= 1 to length(A) do
    if (A[i].d = A[j].d) then {
      if (not (i = j)) then b:=true
        else skip
    } else skip
}

```

Fig. 2. Example 3

element, that is  $\forall i: A[i].d \geq A[p].d$ . Verifying the correctness of the program can be reduced to checking reachability, as the requirement itself can be expressed as a program, by appending to the program `Min` above the following:

```

b:=true;
for i:= 1 to length(A) do { if A[i].d < A[p].d then b:=false }

```

We can now ask a reachability question: Does the control reach the end of the program in a state where `b == false` holds?

*Example 2.* Figure 1 shows an ND1 program that tests whether the array is sorted. It uses one data variable called `v` (note that by definition of the semantics, `v` is initialized to the same value as the first element of the array).

*Example 3.* The Restricted-ND2 program in Figure 2 tests whether there is a data value that appears twice in the array.

### 3 Reachability

Given a program  $P$ , a boolean state  $m$  is *reachable* if and only if there exists an initial global state  $g_I$  and a global state  $g$  such that  $(g_I, g) \in \llbracket P \rrbracket$  and  $\text{bool}(g) = m$ . The reachability problem is to determine, for a given program  $P$  and a given boolean state  $m$ , whether  $m$  is reachable. We will use a notion of a *local state*. Given a program, a local state is a valuation of all its boolean, index, loop, and data variables, as well as the values of array elements corresponding to index and loop variables. For each index and loop variable  $v$ , local states have an additional variable  $A_v$  that stores the value of the array element at position given by  $v$ . The main difference between local and global states of a program  $P$  is that local states do not contain valuation of the array, they store instead at most a fixed finite number  $k_P$  of values from the unbounded domain  $D$ , where  $k_P$  is bounded by the total number of index and loop variables occurring in  $P$ .

**Theorem 1.** *Reachability for ND1 programs is decidable. The problem is PSPACE-complete.*

The structure of the proof is as follows. We first characterize the semantics of a program in terms of a transition system  $T$  whose states are (tuples of) local states. Let us first consider the following simple program  $P$ : `for i1:=1`

to `length(A)` do P1. Here, and in the rest of the proof, we assume that the length of the array is non-zero. (In the case the length of the array is zero, the program effectively contains no loops, and reachability can be computed in time polynomial in number of variables.) The program P can be seen as a transition system whose states are local states of P and which processes an input word in  $\Sigma \times D$ , with each iteration consuming one symbol of the word. For sequential composition of commands, a product construction (augmented with some bookkeeping) is used.

Note that  $T$  is still an infinite-state system, as its states store values from  $D$ . Therefore, we construct a finite state system  $T^\alpha$  that abstracts the infinite part of the local states, that is the values of index, loop and data variables. The abstract state transition system  $T^\alpha$  keeps only order and equality information on the index, loop and data variables. Let  $IV$  be the set of index and loop variables of P. Let  $DV$  be the set of data variables of P. An abstract state is a tuple  $(m, SI, SD)$ , where  $m$  is a boolean state in  $M$ ,  $SI$  is a total order on equivalence classes on  $IV$  and  $SD$  is a total order on equivalence classes on  $DV \cup IV$ . An abstract state represents a set of local states. For example, if a program has an index variable `p1`, a loop variable `i1` and a data variable `v1`, a possible abstract state is  $(m, p1 < i1, p1 = i1 < v1)$ . This abstract state represents a set of concrete states whose boolean state is  $m$  and, the value of `p1` is less than the value of `i1`, the value of the array at position `p1` is the same as the value of the array at position `i1`, which is less than the value of `v1`.

We show that reachability of a boolean state  $m$  can be decided on the abstract system, in the sense that  $m$  is reachable in  $T$  if and only if it is reachable in  $T^\alpha$ . (A boolean state  $m$  is reachable in  $T^\alpha$  iff there exist  $SI$  and  $SD$  such that  $(m, SI, SD)$  is reachable in  $T^\alpha$ .) The main part of the proof shows that every finite path in the abstract transition system is feasible in the concrete transition system. The first idea for a proof might be to show that the abstraction defines a bisimulation between abstract and concrete transition systems. However, this is not the case. We present a simple counterexample. Let us consider a program P and let us focus on two data variables `v1` and `v2`. Let  $q_1$  be a local state such that its boolean component is  $m$ , the value of `v1` at  $q$  is 5 and the value of `v2` at  $q$  is 6. The abstract state corresponding to  $r_1$ ,  $r_1^\alpha$  is thus  $(m, SI, SD)$ , where  $SD$ , the order on data and index variables, includes  $v1 < v2$ . Furthermore, let us suppose that the program is such that the abstract state  $r_1^\alpha$  can transition (in a way that does not change the values of `v1` and `v2`) to an abstract state  $r_2^\alpha$  that requires that another data variable `v3` has a value greater than the value of `v1`, but smaller than the value of `v2`. Note now that the concrete state  $r_1$  cannot transition to any state that would correspond to the order on data variables required by  $r_2^\alpha$ , because there is no value between 5 and 6.

In a key part of the proof, we show that if an abstract state  $r_2^\alpha$  is reachable from  $r_1^\alpha$ , then there exists a state  $r_1$  (abstracted by  $r_1^\alpha$ ) and a state  $r_2$  (abstracted by  $r_2^\alpha$ ) such that  $r_2$  is reachable from  $r_1$ . The main idea for proof by induction is that we can choose  $r_1$  in such a way that the gaps between values are large enough. More precisely, if (1)  $r_1^\alpha$  requires that e.g.  $v1 > v2$  for two data variables

$v_1$  and  $v_2$  and (2)  $r_2^\alpha$  is reachable from  $r_1^\alpha$  in  $k$  steps, then it is sufficient to choose  $r_1$  such that  $v_1 - v_2 > 2^k$ .

The above argument gives rise to a PSPACE algorithm for deciding reachability of a boolean state  $m$ .

## 4 Programs, Automata and Logics on Data Words

In this section, we will examine the decidability boundary for array-accessing programs, and compare the expressive power of these programs to that of logics and automata on data words. We will show that the reachability problem for Restricted-ND2 programs is decidable, and that it is undecidable for full ND2 programs. We start by reviewing the results on automata and logics on data words, as these will be needed for the decidability proof. We will reduce the reachability problem for Restricted-ND2 programs to the nonemptiness problem of extended data automata, a new variation of data automata. The latter is a definition intended to correspond to the notion of regular automata on finite words.

### 4.1 Background

We briefly review the results on automata and logics on data words from [6]. Recall that a data word is a sequence of pairs  $\Sigma \times D$ . A *data language* is a set of data words. Let  $w$  be a data word  $(a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$ . The string  $str(w) = a_1 a_2 \dots a_n$  is called the string projection of  $w$ . Given a data language  $L$ , we write  $str(L)$  to denote the set  $\{str(w) \mid w \in L\}$ . A class is a maximal set of positions in a data word with the same data value. Let  $\mathcal{S}(w)$  be the set of all classes of the data word  $w$ . For a class  $X$  in  $\mathcal{S}(w)$  with positions  $i_1 < \dots < i_k$ , the class string  $str(w, X)$  is  $a_{i_1} \dots a_{i_k}$ .

**Data automata.** A *data automaton* (DA)  $\mathcal{A} = (G, C)$  consists of a transducer  $G$  and a class automaton  $C$ . The transducer  $G$  is a nondeterministic finite-state letter-to-letter transducer from  $\Sigma$  to  $\Gamma$  and  $C$  is a finite-state automaton on  $\Gamma$ . A data word  $w = (a_1, d_1)(a_2, d_2) \dots (a_n, d_n)$  is accepted by a data automaton  $\mathcal{A}$  if there is an accepting run of  $G$  on the string projection of  $w$ , yielding an output string  $b = b_1 \dots b_n$ , and for each class  $X$  in  $\mathcal{S}(w')$ , the class automaton  $C$  accepts  $str(w', X)$ , where  $w' = w'_1 \dots w'_n$  is defined by  $w'_i = (b_i, d_i)$ , for all  $i$  such that  $1 \leq i \leq n$ . Given a DA  $\mathcal{A}$ ,  $L(\mathcal{A})$  is the language of data words accepted by  $\mathcal{A}$ . The nonemptiness problem for data automata is decidable. The proof is by reduction to a computationally complex problem, the reachability problem in Petri nets.

*Example 4.* We present a data automaton  $\mathcal{A}$  such that  $str(L(\mathcal{A}))$ , the set of string projections, is exactly the set of all words over  $\{a, b, c\}$  that contain the same number of  $as$ ,  $bs$ , and  $cs$ . The transducer of  $\mathcal{A}$  computes the identity function, i.e. it accepts all words and its output string is the same as its input string. The class automaton ensures, for each class, that the class contains exactly one occurrence of  $a$ , one occurrence of  $b$  and one occurrence of  $c$ .

**Logics on data words.** We define logics whose models are data words. Following [6], we consider two predicates on positions in a data word whose definition also involves the data values at these positions. The predicate  $i \approx j$  is satisfied if both positions  $i$  and  $j$  have the same data value. The predicate  $i < j$  is satisfied if the data value at position  $i$  is smaller than the data value at position  $j$ . Furthermore, standard successor and order predicates on positions in a data word are used.

Let us first consider logics that use the  $\approx$  predicate and not the  $<$  predicate. We first note that for a first order logic  $\text{FO}(\approx, <, +1)$  satisfiability is undecidable, even if we restrict the number of variables to three. If we restrict the number of variables to two, the logic becomes decidable, and the proof is by reduction to the nonemptiness problem of data automata. The decidability naturally extends to existentially quantified second order monadic logic with two first order variables, denoted by  $\text{EMSO}^2(\approx, +1, \oplus 1)$ . Moreover,  $\text{EMSO}^2(\approx, +1, \oplus 1)$  is precisely equivalent in expressive power to data automata. The predicate  $\oplus 1$  denotes the class successor, and  $i \oplus 1 = j$  is satisfied if  $i$  and  $j$  are two successive positions in the same class of the data word. Furthermore, the logic  $\text{EMSO}^2(\approx, <, +\omega, \oplus 1)$  is included in  $\text{EMSO}^2(\approx, +1, \oplus 1)$ . The symbol  $+\omega$  represents all predicates of the form  $+k$ ,  $k \in \mathbb{N}$ , i.e. the logic includes all predicates  $i + 2 = j$ ,  $i + 3 = j$ , etc.

## 4.2 Extended Data Automata

**Position-preserving class string.** Note that the class automaton does not know the positions of symbols in the word  $w$ . The symbols from other classes have simply been erased. However, let us consider a program with a doubly-nested loop where  $i$  is the loop variable of the outer loop and  $j$  is the loop variable of the inner loop, and let us suppose that the program inside the inner loop is of the form: `if (A[i].d=A[j].d) then P1 else P2`. The inner loop of the program scans the array from left to right and modifies the state in two different ways (given by P1 and P2), depending on whether `(A[i].d=A[j].d)` holds or not. Simply erasing the positions from other classes seems therefore not good enough. We thus define an extension of the notion of class string and a corresponding extension of the class automaton.

Given a data word  $w \in (\Sigma \times D)^*$ , a *position-preserving class string*  $pstr(w, X)$  is a string over  $\Sigma \cup \{0\}$ . (We assume that  $0 \notin \Sigma$ .) Let  $w = w_1 w_2 \dots w_n$ , let  $i$  be a position in  $w$ , and let  $w_i$  be  $(a_i, d_i)$ . The string  $v = pstr(w, X)$  has the same length as  $w$ , and for  $v_i$  we have that  $v_i = a_i$  iff  $i \in X$ , and  $v_i = 0$  otherwise. That is, for each position  $i$  which does not belong to  $X$ , the symbol from  $\Sigma$  at the position  $i$  is replaced by 0.

An *extended data automaton* (EDA)  $\mathcal{E} = (G, C)$  consists of a transducer  $G$  and a class automaton  $C$ . The transducer  $G$  is a finite-state letter-to-letter transducer from  $\Sigma$  to  $\Gamma$  and  $C$  is a finite-state automaton over  $\Gamma \cup \{0\}$ . A data word  $w = w_1 \dots w_n$  is accepted by the EDA  $\mathcal{E}$  if there is an accepting run of  $G$  on the string projection of  $w$ , yielding an output string  $b = b_1 \dots b_n$ , and for each class  $X$  in  $\mathcal{S}(w')$ , the class automaton  $C$  accepts  $pstr(w', X)$ , where  $w' = w'_1 \dots w'_n$  is defined as follows:  $w'_i = (b_i, d_i)$ , for all  $i$  such that  $1 \leq i \leq n$ . Given an EDA  $\mathcal{E}$ ,  $L(\mathcal{E})$  is the language of data words accepted by  $\mathcal{E}$ .

*Example 5.* We consider  $L$ , a language of data words defined by the following property: A data word  $w$  is in  $L$  iff for every class  $X$  in  $\mathcal{S}(w)$ , we have that between every two successive positions in the class, there is exactly one position from another class. We show that there exists an EDA  $\mathcal{E} = (G, C)$  such that  $L(\mathcal{E}) = L$ . The transducer  $G$  computes the identity function. The class automaton  $C$  is given by the following regular expression:  $0^*(\Sigma 0)^*0^*$ . It is easy to see that  $\mathcal{E}$  accepts  $L$ .

We first note that for each DA  $\mathcal{A}$ , it is easy to find an EDA  $\mathcal{E}$  such that  $L(\mathcal{E}) = L(\mathcal{A})$ . We just modify the class automaton  $C$ , by adding the tuple  $(q, 0, q)$ , for each  $q$ , to the transition relation. This means that on reading 0 the state of the class automaton does not change.

We will also show in this section that for each EDA  $\mathcal{E}$  we can find an equivalent DA  $\mathcal{A}$ . This might not be obvious at a first glance, as class automata of DAs do not get to see the distances between positions in a class. Indeed, we show that the language from Example 5 cannot be captured by a deterministic DA. However, we show that  $\text{EMSO}^2(\approx, +1, \oplus 1)$  and EDAs are expressively equivalent, and since  $\text{EMSO}^2(\approx, +1, \oplus 1)$  and DAs are also expressively equivalent, we conclude that for every EDA there exists a DA that accepts the same language. Showing that for every EDA there exists an equivalent  $\text{EMSO}^2(\approx, +1, \oplus 1)$  formula also establishes that non-emptiness is decidable for EDAs. However, the proof of decidability of satisfiability of  $\text{EMSO}^2(\approx, +1, \oplus 1)$  formulas is rather involved. We present a direct proof for decidability of emptiness for EDAs, as it also gives an intuitive reason why emptiness is decidable for EDAs.

**Theorem 2.** *Given an EDA  $\mathcal{E}$ , it is decidable whether  $L(\mathcal{E}) = \emptyset$ .*

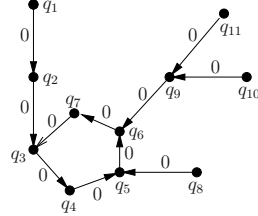
*Proof.* Let  $\mathcal{E} = (G, C)$  be an EDA, let  $G$  be defined by a tuple  $(Q_G, \Sigma, \Gamma, \delta_G, q_0^G, F_G)$ , and let  $C$  be defined by a tuple  $(Q_C, \Gamma, \delta_C, q_0^C, F_C)$ . We start by describing a more operational view of EDAs. A *run* of an EDA on a data word  $w$  is a function  $\varrho$  from positions in  $w$  to tuples of the form  $(q, o, c)$ , where  $q \in Q_G$  is a state of the transducer  $G$ ,  $o$  (a symbol from  $\Gamma$ ) is the output of the transducer, and  $c$  is a function from  $\mathcal{S}(w)$  to  $Q_C$ , the set of states of  $C$ . Furthermore, we require that  $\varrho$  is consistent with  $\delta_G$  and  $\delta_C$ , the transition functions of  $G$  and  $C$ . We define  $\varrho(0)$  to be  $(q_0^G, \gamma, \lambda X. q_0^C)$ , i.e. the transducer and all the copies of the class automaton are in initial states. Furthermore, for each position  $i$ ,  $\varrho(i)$  is equal to  $(q', o', c')$  if and only if  $w_i = (a, d)$ ,  $\varrho(i-1) = (q, o, c)$  and (i)  $(q', o') \in \delta_G(q, a)$ , (ii) for the unique  $X$  such that  $i \in X$  we have  $c'(X) \in \delta_C(c(X), o')$ , (iii) for  $X$  such that  $i \notin X$  we have  $c'(X) \in \delta_C(c(X), 0)$ .

A run is *accepting* iff  $\varrho(n) = (q, o, c)$ ,  $q$  is a final state of  $G$  and for all  $X$  in  $\mathcal{S}(w)$ , we have that  $c(X)$  is a final state of  $C$ .

Let us consider the class automaton  $C$ . Without loss of generality, we suppose that  $C$  is a complete deterministic automaton on  $\Gamma \cup \{0\}$ . The transition function  $\delta_C$  defines a directed graph  $C_0$  with states of  $C$  as vertices and 0-transitions as edges, i.e. there is an edge  $(p_1, p_2)$  in  $C_0$  if and only if  $\delta_C(p_1, 0) = p_2$ . Every vertex in  $C_0$  has exactly one outgoing edge (and might have multiple incoming edges). Therefore, each connected component of  $C_0$  has exactly one cycle. A

vertex is called cyclic if it is part of a cycle, and it is called non-cyclic otherwise. It is easy to see that each connected component is formed by the cyclic vertices and their 0-ancestors. An example of a connected component is in Figure 3. The vertex labeled  $q_6$  is cyclic, its ancestors  $q_9, q_{10}, q_{11}$  are non-cyclic.

The graph  $C_0$  consists of a number of connected components. We denote these components by  $C_0^j$ , for  $j \in [1..k]$ , where  $k$  is the number of the components. Let  $W$  be the set of all non-cyclic vertices. For each non-cyclic vertex  $v$ , let  $D(v)$  be defined as follows:  $D(v) = d$  for non-cyclic vertices connected to a cycle, where  $d$  is the length of the unique path connecting  $v$  to the closest cyclic vertex. For the graph  $C_0$ , we define  $D(C_0)$  to be  $\max_{v \in W} D(v)$ .



**Fig. 3.** A connected component of a graph  $C_0$  corresponding to an EDA  $\mathcal{E}$

Let  $i$  be a position in a data word  $w$ . The data word  $w_1 w_2 \dots w_i$  is denoted by  $prefix(w, i)$ . Let us consider a position  $i$  in a data word  $w$  and the set of classes  $\mathcal{S}(w)$ . Let  $\mathcal{S}_{act}(w, i)$  be a set of *active* classes, i.e. classes  $X$  such that there is a position in  $X$  to the left of the position  $i$ . More formally, a class  $X \in \mathcal{S}(w)$  is in  $\mathcal{S}_{act}(w, i)$  if the string  $str(prefix(w, i), X)$  is not equal to  $0^i$ .

**Lemma 1.** *Let  $\rho$  be a run of  $\mathcal{E}$  on  $w$ . Let  $i$  be a position in  $w$ . Let  $\rho(i)$  be  $(q, o, c)$ . The number  $N$  of classes  $X$ , such that  $X$  is in  $\mathcal{S}_{act}(w, i)$  and  $c(X)$  is a noncyclic vertex, is bounded by  $D(C_0)$ , i.e.  $N \leq D(C_0)$ .*

*Proof.* Let  $i$  be a position in a word  $w$ . If  $i \leq D(C_0)$ , then the number of active classes is at most  $D(C_0)$ , and we conclude immediately. Let us consider the case  $i > D(C_0)$ . Let  $\rho(i)$  be  $(q, o, c)$  and let  $s$  be the string of length  $D(C_0)$  defined by  $s = w_{i-D(C_0)+1} w_{i-D(C_0)+2} \dots w_i$ . There are two possible cases for each class  $X$  in  $\mathcal{S}(w)$ . The first case is the case when  $pstr(s, X) = 0^{D(C_0)}$ . Let  $\rho(i - D(C_0)) = (q', o', c')$ , and let  $c'(X) = v$ . We can easily prove that  $\delta_C^*(p, 0^e)$  is not in  $W$ , for all  $p$  and for all  $e \geq D(p)$ . By definition,  $D(C_0) \geq D(q')$ . Therefore, we can conclude that  $c(X) \notin W$ . The second case is the case when  $pstr(s, X) \neq 0^{D(C_0)}$ . This is true for at most  $D(C_0)$  classes, because, for all positions  $x$ , there is exactly one class  $X$ , such that the symbol at the position  $x$  of the class string  $pstr(s, X)$  is not 0. Thus we have that  $c(X) \in W$  for at most  $D(C_0)$  classes.

We reduce emptiness of EDAs to emptiness of multicounter automata. Multicounter automata are equivalent to Petri nets [11], and thus the emptiness of multicounter automata is decidable. We use the definition of multicounter automata from [6]. A *multicounter automaton* is a finite, non-deterministic automaton extended by a number  $k$  of counters. It can be described as a tuple  $(Q, \Sigma, k, \delta, q_I, F)$ . The set of states  $Q$ , the input alphabet, the initial state  $q_I \in Q$  and final states  $F \subseteq Q$  are as in a usual finite automaton. The transition relation is a subset of  $Q \times (\Sigma \cup \{\varepsilon\}) \times \{inc(i), dec(i)\} \times Q$ . The idea is that in each step,

the automaton can change its state and modify the counters, by incrementing ( $inc(i)$  increments counter number  $i$ ) or decrementing them, according to the current state and the current letter on the input (which can be  $\varepsilon$ ). Whenever it tries to decrement a counter of value zero the computation stops and rejects. The transition of a multicounter automaton does not depend on the value of the counters in any other way. In particular, it cannot test whether a counter is exactly zero. The automaton accepts when the state is final and all the counters are empty.

**Lemma 2.** *Let  $\mathcal{E}$  be an EDA. A multicounter automaton  $V$  such that  $str(L(\mathcal{E})) = L(V)$  can be computed from  $\mathcal{E}$ .*

*Proof.* We present the construction of a multicounter automaton  $V$  that simulates  $\mathcal{E}$ . The multicounter automaton  $V$  simulates the transducer  $G$  and a number of copies of  $C$ . There is one copy per class in  $\mathcal{S}(w)$ , where  $w$  is the word the automaton is reading. We say that a class automaton performs a 0-transition if the input symbol it reads is 0, and it performs a  $\Gamma$ -transition if the input symbol it reads is from  $\Gamma$ . Intuitively, at each step, the automaton  $V$ : (i) Simulates the transducer  $G$  using the finite state part (i.e. not the counters), and (ii) It guesses to which class the current position belongs, and it executes the  $\Gamma$ -transition of the automaton for that class with the symbol that is the output of the transducer at this step. For all the other simulated automata,  $V$  executes the 0-transition. (This is sufficient because each position belongs to exactly one equivalence class.) The counters of the multicounter automaton  $V$  correspond to the cyclic vertices in  $C_0$ . (In what follows, we call a state of  $C$  (non-)cyclic if it corresponds to a (non-)cyclic vertex in  $C_0$ .) The value of the counter  $h$  corresponds to the number of copies of  $C$  currently in the state  $h$ . The finite part of the automaton state tracks the number of copies in each non-cyclic state. The key idea of the proof is that the total number of copies in non-cyclic states is finite and bounded (by  $D(C_0)$ ). This fact is implied by Lemma [□](#).

Furthermore, one copy  $e$  of the class automaton is used to keep track of all the classes that are not active yet, i.e. not in  $\mathcal{S}_{act}(w, i)$  at step  $i$  - thus when a position-preserving class string contains a symbol in  $\Gamma$  for the first time, a new copy of the automaton  $C$  is started from the state at which the copy  $e$  is.

Let  $\gamma \in \Gamma$  be the current input symbol. The automaton works as follows: The first step consists of the automaton  $V$  nondeterministically guessing the equivalence class  $X$  to which the current position belongs. The copy of the class automaton for  $X$  is then set aside while the second step is performed. That is, if the copy is in state  $s$ , then  $s$  is remembered in a separate part of the finite state. In the second step, the automaton  $V$  simulates 0-transitions for all the other copies (other than the copy that performed the  $\Gamma$ -transition). For copies in non-cyclic states, this is done by a transition modifying the finite state of  $V$ . The copies that transition from a non-cyclic to a cyclic state are dealt with by modifying the finite state and increasing the corresponding counter. The copies in cyclic states are tracked in the counters. Note that if we restrict the graph to only cyclic states, each state has exactly one incoming and one outgoing 0-edge.

For all the copies in cyclic states, the 0-transition is accomplished by “relabeling” the counters. This is done by remembering in the the finite state of  $V$  for each loop for one particular state to which counter it corresponds. This is then shifted in the direction of the 0-transition.

The third step is to perform the  $\Gamma$  transition for the class  $X$ . For the copy of the automaton corresponding to this class, a  $\Gamma$ -transition is performed. That is, if it is in state  $q$ , and  $\delta(q, \gamma) = q'$ , then there are four possibilities.: (i) if  $q, q'$  are cyclic states, the counter corresponding to  $q$  is decreased and the counter corresponding to  $q'$  is increased; (ii) if  $q, q'$  are non-cyclic state, a transition that changes the state of  $V$  is made; (iii) if  $q$  is a cyclic state and  $q'$  is a non-cyclic state, the counter corresponding to  $q$  is decreased, and the finite state of  $V$  is changed to reflect that the number of copies in  $q'$  has increased; (iv) if  $q$  is a noncyclic state and  $q'$  is a cyclic state, the transition is simulated similarly.

This concludes the proof of Theorem [2](#).

### 4.3 Restricted Doubly-Nested Loops

We will reduce the reachability problem of Restricted-ND2 programs to the emptiness problem of EDAs. The main idea of the proof is that the transducer  $G$  guesses an accepting run of the outer loop, while the class automaton  $C$  checks that the inner loop can be executed in a way that is consistent with the guess of the transducer.

**Theorem 3.** *Reachability for Restricted-ND2 programs is decidable.*

The proof of Theorem [3](#) gives a decision procedure, but one whose running time is non-elementary. The reason is that while the problem of reachability in multicounter automata is decidable, no elementary upper bound is known.

However, the following proposition shows that the problem is at least as hard as the reachability in multicounter automata, which makes it unlikely that a more efficient algorithm exists. The best lower bound for the latter problem is EXPSPACE [\[19\]](#).

**Proposition 4.** *The reachability problem for multicounter automata can be reduced to the reachability problem for Restricted-ND2 programs.*

### 4.4 Undecidable Extensions

We show that if we lift the restrictions we imposed on Restricted-ND2 programs, the reachability problem becomes undecidable.

**Theorem 5.** *The reachability problem for ND2 programs is undecidable.*

The proof is by reduction from the reachability problem of two-counter automata [\[20\]](#). We note that the proof also implies that the reachability problem is undecidable even for ND2 programs that do not use order on the data domain and do not use index or data variables.



We investigate the case of Restricted-ND2 programs with access to order on the data domain and with data index variables. We show that if we add order on the data domain and at least one data variable, the reachability problem becomes undecidable. The proof is by reduction from the Post's Correspondence Problem, and is similar to the proof of Proposition 21 of [6].

**Proposition 6.** *Reachability for Restricted-ND2 programs that use order on  $D$  and at least one data variable is undecidable.*

A natural question, which is now open, is whether it is possible to add only one of these features (order on data domain or data (index) variables) to Restricted-ND2 programs without losing decidability of the reachability problem.

#### 4.5 Expressiveness

In this section, we compare expressiveness of logics and automata on data words and array-accessing programs. We make our comparisons in terms of languages of data words these formalisms can define. Due to a lack of space, we present only the results in this subsection.

**Language of a program.** In order to define the language of a program, we extend the notion of a program by adding a final state. That is, in this section we will assume that every program  $P$  has a final state  $m_f$ , where  $m$  is a boolean state of  $P$ . The language  $L_m(P)$  is the set of data words  $w$ , such that there exist an initial state  $g_I$  and a state  $g$ , such that  $g_I[A] = w$ ,  $bool(g) = m$ , and  $(g_I, g) \in \llbracket P \rrbracket$ . We say that a program  $P$  *accepts* the language  $L_m(P)$ , where  $m$  is the final state.

The following proposition shows that EDAs and  $EMSO^2(\approx, +1, \oplus 1)$  are equally expressive. This means that somewhat surprisingly, DAs and EDAs are expressively equivalent.

**Proposition 7.** *EDAs and  $EMSO^2(\approx, +1, \oplus 1)$  are equally expressive.*

The following proposition sheds light on the difference between DAs and EDAs. We saw that DAs and EDAs are expressively equivalent. However, one difference between EDAs and DAs is that deterministic EDAs are more expressive than deterministic DAs. It is the nondeterminism that then levels the difference.

**Proposition 8.** *Deterministic EDAs are more expressive than deterministic DAs.*

We show that nondeterminism adds to the expressive power of EDAs, as there exists a language accepted by a nondeterministic EDA, but no deterministic EDA can accept it. This implies the following proposition.

**Proposition 9.** *Deterministic EDAs are strictly less expressive than EDAs.*

We will now compare the expressive power of array-accessing programs to logics and automata on data words. Specifically, we will use the logic  $EMSO^2(\approx, +1, \oplus 1)$  for comparison. Recall that this logic is expressively equivalent to data automata. We first show that Restricted-ND2 programs are not as expressive as  $EMSO^2(\approx, +1, \oplus 1)$ . We also compare the expressive power of ND1 programs and  $EMSO^2(\approx, +1, \oplus 1)$ .

**Proposition 10.** *Restricted-ND2 programs are strictly less expressive than  $EMSO^2(\approx, +1, \oplus 1)$ .*

**Proposition 11.** *There exists an  $EMSO^2(\approx, +1, \oplus 1)$  property that is not expressible by an ND1 program.*

Note that ND1 programs allow order on the data domain, and thus can check a property specifying that the elements in the input data word are in increasing order. It is easy to see that this property is not specifiable in  $EMSO^2(\approx, +1, \oplus 1)$ . However, if we syntactically restrict ND1 programs not to use order on  $D$ , they can be captured by  $EMSO^2(\approx, +1, \oplus 1)$  formulas. The reason is that ND1 programs that do not refer to the order on  $D$  can be simulated by register automata introduced in [17]. For every register automaton, there is an equivalent data automaton [5]. Another natural question is whether there is an order-invariant property that can be captured by ND1 programs (that have access to order), but is not expressible in  $EMSO^2(\approx, +1, \oplus 1)$ . We leave this question for future work.

## 5 Related Work

Our results establish connections between verification of programs accessing arrays and logics and automata on data words. Kaminski and Francez [17] initiated the study of finite-memory automata on infinite alphabets. They introduced register automata, that is automata that in addition to finite state have a fixed number of registers that can store data values. The results of Kaminski and Francez were recently extended in [21,6,5,4]. Data automata introduced in this line of research were shown to be more expressive than register automata. Furthermore, the logic  $EMSO^2(\approx, +1, \oplus 1)$  was introduced, and [6] shows that  $EMSO^2(\approx, +1, \oplus 1)$  and data automata are equally expressive. The reduction from  $EMSO^2(\approx, +1, \oplus 1)$  to data automata and the fact that emptiness is decidable for data automata imply that satisfiability is decidable for  $EMSO^2(\approx, +1, \oplus 1)$ . We show that Restricted-ND2 programs can be encoded in  $EMSO^2(\approx, +1, \oplus 1)$ . However, adding a third variable to the logic or allowing access to order on data variable makes satisfiability undecidable for the resulting logic, even for the first order fragment. We show, perhaps somewhat surprisingly, that the undecidability does not translate into undecidability of reachability for ND1 programs that access order on the data domain and have an arbitrary number of index and data variables. The results on automata and logics on data words model were applied in the context of XML reasoning [21] and extended temporal logics [9]. The connection to verification of programs with unbounded data structures is the first to the best of our knowledge.

Deutsch et al. [10] consider a model of database-driven systems similar in some aspects to our model of programs. The key difference is that they consider a dense order. They specifically note that the model-checking problem they consider is open for the case of a discrete order. It would be interesting to see if our result on programs on structures with discrete order can be extended to the setting of database-driven systems. Fragments of first order logic on arrays have been shown decidable in [8,15,2,7]. These fragments do not restrict the number of variables

(as was the case with  $\text{EMSO}^2(\approx, +1, \oplus 1)$ ), but restrict the number of quantifier alternations. These papers focus on theory of arrays, rather than on analysis of array-accessing programs. Decidability of reachability for polymorphic systems with arrays (PSAs) was studied e.g. in [18]. PSAs use well-typed  $\lambda$ -terms and do not allow iteration over arrays.

Static analysis of programs that access arrays is an active research area, with recent results including [12,14,2]. The approach consists in finding inductive invariants for loops using abstraction methods, such as abstract domains that can represent universally quantified facts [14] and a predicate abstraction approach to shape analysis [2]. In contrast, our results yield decision procedures for array-accessing programs. However, the methods based on abstraction are applicable to a richer class of programs. Note that the abstract domains used for examples and applications also track equality and order on array elements.

## References

1. Alur, R., Černý, P., Weinstein, S.: Algorithmic analysis of array-accessing programs. Technical Report MS-CIS-08-35. University of Pennsylvania (2008)
2. Balaban, I., Pnueli, A., Zuck, L.D.: Shape analysis by predicate abstraction. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 164–180. Springer, Heidelberg (2005)
3. Ball, T., Rajamani, S.: The SLAM project: debugging system software via static analysis. In: POPL, pp. 1–3 (2002)
4. Björklund, H., Bojańczyk, M.: Shuffle expressions and words with nested data. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 750–761. Springer, Heidelberg (2007)
5. Björklund, H., Schwentick, T.: On notions of regularity for data languages. In: Csuhaaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 88–99. Springer, Heidelberg (2007)
6. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, pp. 7–16 (2006)
7. Bouajjani, A., Habermehl, P., Jurski, Y., Sighireanu, M.: Rewriting systems with data. In: Csuhaaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 1–22. Springer, Heidelberg (2007)
8. Bradley, A.R., Manna, Z., Sipma, H.B.: What’s decidable about arrays? In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 427–442. Springer, Heidelberg (2005)
9. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. In: LICS, pp. 17–26 (2006)
10. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: ICDT, pp. 252–267 (2009)
11. Gischer, J.: Shuffle languages, Petri nets, and context-sensitive grammars. *Commun. ACM* 24(9), 597–605 (1981)
12. Gopan, D., Reps, T., Sagiv, M.: A framework for numeric analysis of array operations. In: POPL, pp. 338–350 (2008)
13. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: CAV, pp. 72–83 (1997)

14. Gulwani, S., McCloskey, B., Tiwari, A.: Lifting abstract interpreters to quantified logical domains. In: POPL, pp. 235–246 (2008)
15. Habermehl, P., Iosif, R., Vojnar, T.: What else is decidable about integer arrays? In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 474–489. Springer, Heidelberg (2008)
16. Henzinger, T.A., Jhala, R., Majumdar, R., Necula, G.C., Sutre, G., Weimer, W.: Temporal-safety proofs for systems code. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 526–538. Springer, Heidelberg (2002)
17. Kaminski, M., Francez, N.: Finite-memory automata. *Theoretical Computer Science* 134(2), 329–363 (1994)
18. Lazić, R.: Decidability of reachability for polymorphic systems with arrays: A complete classification. *ENTCS* 138(3), 3–19 (2005)
19. Lipton, R.: The reachability problem requires exponential space. Technical Report Dept. of Computer Science, Research report 62. Yale University (1976)
20. Minski, M.: Recursive unsolvability of Post’s problem of ‘tag’ and other topics in theory of Turing machines. *Annals of Mathematics* 74, 437–455 (1962)
21. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Logic* 5(3), 403–435 (2004)

# Decidable Relationships between Consistency Notions for Constraint Satisfaction Problems

Albert Atserias<sup>1</sup> and Mark Weyer<sup>2</sup>

<sup>1</sup> Universitat Politècnica de Catalunya, Barcelona, Spain

<sup>2</sup> Humboldt Universität zu Berlin, Berlin, Germany

**Abstract.** We define an abstract pebble game that provides game interpretations for essentially all known consistency algorithms for constraint satisfaction problems including arc-consistency,  $(j, k)$ -consistency,  $k$ -consistency,  $k$ -minimality, and refinements of arc-consistency such as peek arc-consistency and singleton arc-consistency. Our main result is that for any two instances of the abstract pebble game where the first satisfies the additional condition of being stacked, there exists an algorithm to decide whether consistency with respect to the first implies consistency with respect to the second. In particular, there is a decidable criterion to tell whether singleton arc-consistency with respect to a given constraint language implies  $k$ -consistency with respect to the same constraint language, for any fixed  $k$ . We also offer a new decidable criterion to tell whether arc-consistency implies satisfiability which pulls from methods in Ramsey theory and looks more amenable to generalization.

## 1 Introduction

Comparing finite structures with respect to some preorder or equivalence relation is a classic theme in logic and algorithms. Notable examples include isomorphisms, embeddings, and homomorphisms, as well as preservation of formulas in various logics, and (bi-)simulation relations of various types.

Let  $\leq$  and  $\leq'$  be preorders on finite structures, where  $\leq$  is a *refinement* of  $\leq'$  which is however *harder* than  $\leq'$ . More precisely,  $\leq$  is a refinement of  $\leq'$  in that the implication

$$\mathbf{A} \leq \mathbf{B} \implies \mathbf{A} \leq' \mathbf{B} \tag{1}$$

holds true, but the reverse implication is not true in general. Also,  $\leq$  is harder than  $\leq'$  in the sense that determining whether  $\mathbf{A}$  is smaller than  $\mathbf{B}$  is computationally harder for  $\leq$  than for  $\leq'$ . A question of interest in this situation is to characterize the structures  $\mathbf{A}$  (resp.  $\mathbf{B}$ ) for which the implication in (1) is actually an equivalence. Let us summarize a few known instances where this equivalence holds.

### 1.1 Some Examples

For a collection of first-order formulas  $L$ , we write  $\mathbf{A} \leq^L \mathbf{B}$  if every sentence from  $L$  that is true in  $\mathbf{A}$  is also true in  $\mathbf{B}$ . Clearly,  $\leq^L$  defines a preorder, and if

$L$  is closed under negation, it defines an equivalence relation  $\equiv^L$ . When  $L$  is FO, the collection of all first-order formulas, it is well known that  $\equiv^L$  agrees with isomorphism on finite structures. On the other hand, the  $k$ -variable fragment of first-order logic  $\text{FO}^k$  gives a coarsening of isomorphism which, for fixed  $k$ , can be decided in polynomial time. Thus, we are asking for the finite structures  $\mathbf{A}$  for which the equivalence

$$\mathbf{A} \equiv^{\text{FO}^k} \mathbf{B} \iff \mathbf{A} \cong \mathbf{B} \quad (2)$$

holds for every finite  $\mathbf{B}$ .

If  $\mathbf{A}$  is a colored path (a word), then (2) holds for  $k \geq 3$  and every  $\mathbf{B}$  [16]. More generally, for every colored tree  $\mathbf{A}$ , equation (2) holds with  $k \geq d + 1$  and every  $\mathbf{B}$ , where  $d$  is a bound on the degree of the tree. Other fascinating examples arise for graphs embedded in surfaces: if  $\mathbf{A}$  is a 3-connected planar graph, (2) holds for every large constant  $k$  and every  $\mathbf{B}$ , and indeed  $k \geq 15$  suffices [13,21].

When  $L$  is  $\exists\text{FO}^+$ , the existential-positive fragment of FO, the preorder  $\mathbf{A} \leq^L \mathbf{B}$  coincides with the existence of a homomorphism from  $\mathbf{A}$  into  $\mathbf{B}$ . Also its  $k$ -variable fragment  $\exists\text{FO}^{k,+}$  gives a coarsening that can be decided in polynomial time. Thus, in this case we are asking for the finite structures  $\mathbf{A}$  for which the equivalence

$$\mathbf{A} \leq^{\exists\text{FO}^{k,+}} \mathbf{B} \iff \mathbf{A} \rightarrow \mathbf{B} \quad (3)$$

holds true for every  $\mathbf{B}$ , where  $\mathbf{A} \rightarrow \mathbf{B}$  denotes the existence of a homomorphism.

If  $\mathbf{A}$  is a colored tree, it is easy to show that (3) holds for  $k \geq 2$  and every  $\mathbf{B}$ . More generally, if the treewidth of  $\mathbf{A}$  is less than  $k$ , even if up to homomorphic equivalence, then (3) holds for every  $\mathbf{B}$  [8]. Interestingly, this result is tight: if (3) holds for every  $\mathbf{B}$ , then the treewidth of  $\mathbf{A}$  is less than  $k$ , up to homomorphic equivalence [2].

For equation (3), restrictions imposed on  $\mathbf{B}$  have a different meaning than restrictions imposed on  $\mathbf{A}$ , and obtaining tight characterizations becomes much harder. Still, some cases are known. For example, if  $\mathbf{B}$  is a bipartite graph, then (3) holds for  $k \geq 3$  and every  $\mathbf{A}$ . And on restriction to graphs, this is one of the few instances where we get a characterization: if  $\mathbf{B}$  is a graph (with at least one edge) for which (3) holds for every  $\mathbf{A}$ , then  $k \geq 3$  and  $\mathbf{B}$  is bipartite [20]. For general relational structures, the state of affairs is much more complicated, as discussed next.

## 1.2 On Characterization Results and CSPs

Unfortunately, full characterizations as those discussed in the previous section may be hopeless even for natural instances of  $\leq$  and  $\leq'$ . Consider for example the problem that, given  $\mathbf{A}$ , asks whether the equivalence in (2) holds for every  $\mathbf{B}$ . For  $k = 2$ , an algorithm follows from the fact that the finite satisfiability problem for  $\text{FO}^2$  is decidable. But for  $k = 3$ , equivalence with respect to  $\text{FO}^k$  is able to encode Diophantine problems [14] and we quickly face undecidability.

For coarser preorders, such as homomorphism, there is still some hope. For example, the results mentioned above show that the equivalence in (3) holds

for every  $\mathbf{B}$  if and only if the treewidth of  $\mathbf{A}$  is less than  $k$  up to homomorphic equivalence, which is a decidable criterion. On the other hand, the dual question of characterizing the finite structures  $\mathbf{B}$  for which (3) holds for every  $\mathbf{A}$  is one of the main questions in the seminal work by Feder and Vardi [10] on constraint satisfaction problems. It corresponds to the question of characterizing all constraint languages for which the so-called *k-consistency algorithm* (defined in Section 3) solves any instance. In symbols:

$$k\text{-CON}(\mathbf{B}) \stackrel{?}{=} \text{CSP}(\mathbf{B}) \quad (4)$$

where  $k\text{-CON}(\mathbf{B})$  denotes the collection of all instances that are  $k$ -consistent with respect to  $\mathbf{B}$ , and  $\text{CSP}(\mathbf{B})$  is the collection of all  $\mathbf{A}$  such that  $\mathbf{A} \rightarrow \mathbf{B}$ .

### 1.3 New Results

Motivated by question (4), we offer a unifying approach to the *consistency algorithms* that were considered in the literature. These include arc-consistency,  $k$ ,  $\ell$ -consistency,  $k$ -consistency,  $k$ -minimality, and refined versions of arc-consistency such as peek arc-consistency and singleton arc-consistency. For pairs of these algorithms, which we denote as preorders  $\leq$  and  $\leq'$ , we want to be able to decide for which finite structures  $\mathbf{B}$  the equivalence

$$\mathbf{A} \leq \mathbf{B} \iff \mathbf{A} \leq' \mathbf{B} \quad (5)$$

holds true for every finite  $\mathbf{A}$ . Along the lines of Kolaitis and Vardi [18], we phrase each of these algorithms as an instance of a general pebble game. This abstract setting allows us to prove that the equivalence in (5) is decidable for pairs of algorithms including arc-consistency, peek arc-consistency, singleton arc-consistency, and some others. The simple argument pivots around three components: the fact that such games enjoy treewidth duality, the identification of a subclass of games –called *stacked*– that have definitions in monadic second-order logic, and the decidability of MSO on structures of bounded treewidth. It is worth pointing out, as an interesting feature, that the MSO definitions of stacked games span different levels of the so-called “closure of monadic NP” introduced by Ajtai, Fagin, and Stockmeyer [1]. In particular, they seem to go beyond monadic NP.

One further consequence of these results is that, for a given finite structure  $\mathbf{B}$ , the equality  $\text{SAC}(\mathbf{B}) = k\text{-CON}(\mathbf{B})$  is decidable, where  $\text{SAC}(\mathbf{B})$  denotes the collection of all instances that are singleton arc-consistent with respect to  $\mathbf{B}$ . To our knowledge, this sort of result was unknown before. Another remarkable consequence is that a solution to problem (4) automatically gives a solution to problem  $\text{SAC}(\mathbf{B}) = \text{CSP}(\mathbf{B})$ , and similarly for other pairs of algorithms.

Finally, we close the paper by offering a new decidable criterion for the problem  $\text{AC}(\mathbf{B}) = \text{CSP}(\mathbf{B})$ , where  $\text{AC}(\mathbf{B})$  denotes the instances that are arc-consistent with respect to  $\mathbf{B}$ . Our new proof pulls from ideas in Ramsey theory and looks more amenable to generalization when compared to the previous direct proof by Feder and Vardi [10]. Indeed, our method was introduced by Kolaitis and Vardi [17] for solving a completely different problem related to the asymptotic probability of strict NP properties, which indicates its wider generality.

## 2 Preliminaries

We use standard notation and terminology in finite model theory; see [9]. All our vocabularies are finite and relational, perhaps with additional constant symbols. Homomorphisms preserve tuples and constants, strong homomorphisms preserve also non-tuples, embeddings are injective homomorphisms, and strong embeddings are injective strong homomorphisms. We write  $h : \mathbf{A} \rightarrow \mathbf{B}$  to denote that  $h$  is a homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$ . If  $h$  does not matter, we write  $\mathbf{A} \rightarrow \mathbf{B}$  to denote its existence. We use the convention that if  $\mathbf{A}$  and  $\mathbf{B}$  do not share the same vocabulary, automatically  $\mathbf{A} \not\rightarrow \mathbf{B}$ . The same conventions apply to embeddings  $\xrightarrow{e}$ , strong embeddings  $\xrightarrow{s}$ , and isomorphisms  $\cong$ .

The structure  $\mathbf{A}$  is a substructure of  $\mathbf{B}$  if  $A \subseteq B$  and the identity mapping is an embedding. It is an induced substructure if the embedding is strong. In this case,  $\mathbf{A}$  is the substructure of  $\mathbf{B}$  induced by  $A$ , and we denoted it by  $\mathbf{B} \upharpoonright A$ . The union of  $\mathbf{A}$  and  $\mathbf{B}$  is the structure with universe  $A \cup B$  where the relation  $R$  is interpreted by  $R^A \cup R^B$ . The disjoint union of  $\mathbf{A}$  and  $\mathbf{B}$  is the union of two copies of  $\mathbf{A}$  and  $\mathbf{B}$  with disjoint universes. If  $C = A \cap B$  and  $\mathbf{A}$  and  $\mathbf{B}$  agree on  $C$  in the sense that  $\mathbf{A} \upharpoonright C = \mathbf{B} \upharpoonright C$ , the union of  $\mathbf{A}$  and  $\mathbf{B}$  is called the glued union through  $C$ , where  $\mathbf{C} = \mathbf{A} \upharpoonright C = \mathbf{B} \upharpoonright C$ .

For the definitions of treewidth and tree-decompositions of graphs and relational structures we refer the reader to, say, [11]. We write  $\text{TW}(k)$  for the class of all finite structures of treewidth at most  $k$ .

For the definitions of first and second-order logic, MSO, least and greatest fixed-point logic, and Datalog see [9]. Co-Datalog stands for the negations of Datalog formulae. If  $k$  is an integer,  $k$ -ary Datalog has all recursive predicates of arity at most  $k$ . An SNP formula is a formula of the form  $\exists \overline{X} \forall \overline{x} \varphi$ , where  $\overline{X}$  is a sequence of relation variables,  $\overline{x}$  is a sequence of first-order variables, and  $\varphi$  is a quantifier-free formula. A  $k$ -ary SNP formula has all relation variables of arity at most  $k$ . The closure of monadic SNP stands for the collection of formulas of the form  $\exists \overline{X}_1 \forall \overline{x}_1 \cdots \exists \overline{X}_m \forall \overline{x}_m \varphi$ , where all relation variables are unary and  $\varphi$  is quantifier-free. The closure of monadic NP was introduced in [1]. It follows from standard techniques that every formula of  $k$ -ary co-Datalog is equivalent to a  $k$ -ary SNP formula, and that every formula of monadic universal greatest fixed-point logic is equivalent to a formula in the closure of monadic SNP.

A *consistency notion* is just any reflexive transitive relation between structures, which is isomorphism invariant. Thus,  $\rightarrow$ ,  $\xrightarrow{e}$ ,  $\xrightarrow{s}$ , and  $\cong$  are consistency notions. If  $L$  is a logic and  $\leq^L$  denotes preservation of  $L$ -formulae, then  $\leq^L$  is also a consistency notion. Let  $\leq$  and  $\leq'$  be two consistency notions. We say that  $\leq$  is a *refinement* of  $\leq'$ , or  $\leq'$  a *coarsening* of  $\leq$ , if  $\mathbf{A} \leq \mathbf{B}$  implies  $\mathbf{A} \leq' \mathbf{B}$ .

## 3 Generalized Pebble Game

In this section we define the abstract pebble game for which we prove our results. The methods would work for versions of the game that are even more general, but as this is at the expense of intuition, we have made the definition only as general as necessary to include the important consistency notions in the literature.



Before we start, let us introduce some necessary notation and terminology. Let  $\{c_1, c_2, \dots\}$  be a countable set of constant symbols. For a natural number  $k$ , a *k-numbered structure* is a structure  $\mathbf{D}$  for a vocabulary that contains  $\{c_1, \dots, c_k\}$  such that  $D = \{c_1^{\mathbf{D}}, \dots, c_k^{\mathbf{D}}\}$ . Observe, that this does not imply  $|D| = k$ . For a structure  $\mathbf{D}$  and  $d_1, \dots, d_k \in D$ , let  $(\mathbf{D}, d_1, \dots, d_k)$  denote the *k-numbered structure*, which is obtained from  $\mathbf{D} \upharpoonright \{d_1, \dots, d_k\}$  by interpreting  $c_i$  by  $d_i$  for all  $1 \leq i \leq k$ . All *k-numbered structures* can be represented this way.

### 3.1 Definition of the Game

The game comes parameterized by two sets  $G$  and  $S$ . The *growing set*  $G$  is a collection of pairs  $(k, \mathbf{D})$ , where  $k$  is a natural number and  $\mathbf{D}$  is an  $\ell$ -numbered structure for some  $\ell \geq k$ . The *shrinking set*  $S$  is a collection of pairs  $(k, K)$ , such that  $k$  is a natural number and  $K \subseteq \{1, \dots, k\}$ . Further we require, that if  $(k, \mathbf{D}) \in G$  and  $\mathbf{D} \cong \mathbf{D}'$ , then also  $(k, \mathbf{D}') \in G$ .

The game  $\mathcal{G}^{G,S}$  is played by two players, Spoiler and Duplicator, on a board formed by two structures  $\mathbf{A}$  and  $\mathbf{B}$ . The positions of a play of the game are sequences

$$((a_1, b_1), \dots, (a_k, b_k)), \tag{6}$$

where  $a_i \in A$  and  $b_i \in B$ . The initial position is the empty sequence. From a position  $p$  as in (6), Spoiler has a set of options:

1. *Growing round*: Spoiler may announce a growing round in which he picks some  $\ell \geq k$ , some  $a_{k+1}, \dots, a_\ell$  from  $A$ , and an  $\ell$ -numbered substructure  $\mathbf{S}$  of  $(\mathbf{A}, a_1, \dots, a_\ell)$ , provided that the pair  $(k, \mathbf{S})$  belongs to  $G$ . Then it is Duplicator's turn, who is required to pick some  $b_{k+1}, \dots, b_\ell$  from  $B$  such that  $\mathbf{S} \rightarrow (\mathbf{B}, b_1, \dots, b_\ell)$ . If she succeeds, the next position is  $((a_1, b_1), \dots, (a_\ell, b_\ell))$ ; if she does not, the game is over.
2. *Shrinking round*: For every  $(k, K)$  in  $S$ , Spoiler has the option to move to  $((a_i, b_i) : i \in K)$ , the subsequence of  $p$  induced by  $K$ .

Duplicator wins a play if she can play infinitely. We write  $\mathbf{A} \leq^{G,S} \mathbf{B}$  if Duplicator has a winning strategy to win every play of  $\mathcal{G}^{G,S}$  on the board formed by  $\mathbf{A}$  and  $\mathbf{B}$ . If  $b$  is an integer, we say that the game  $\mathcal{G}^{G,S}$  is *grow-bounded* by  $b$  if every pair  $(k, \mathbf{D})$  in  $G$  has  $k \leq b$ . We say that it is *fully-bounded* by  $b$  if for every such pair,  $\mathbf{D}$  is  $\ell$ -numbered for some  $\ell \leq b$ .

The first thing we need to observe is that our pebble games define relations that are always coarser than homomorphisms:

**Lemma 1.** *Let  $G$  and  $S$  define a pebble game. Then  $\rightarrow$  is a refinement of  $\leq^{G,S}$ , and  $\leq^{G,S}$  is reflexive.*

*Proof.* If  $h : \mathbf{A} \rightarrow \mathbf{B}$ , then Duplicator has a winning strategy by answering all growing rounds with  $h$ . In other words, in position  $((a_1, h(a_1)), \dots, (a_k, h(a_k)))$ , if Spoiler picked  $a_{k+1}, \dots, a_\ell$  and a substructure  $\mathbf{S}$  of  $(\mathbf{A}, a_1, \dots, a_\ell)$ , Duplicator replies with  $h(a_{k+1}), \dots, h(a_\ell)$ . Then  $\mathbf{S} \rightarrow (\mathbf{A}, a_1, \dots, a_\ell) \rightarrow (\mathbf{B}, h(a_1), \dots, h(a_\ell))$  so this is a valid move. In this way, Duplicator can play infinitely to win. The

second claim is immediate from considering the identity homomorphism from  $\mathbf{A}$  to  $\mathbf{A}$ .  $\square$

On the other hand, not all  $\leq^{G,S}$  are transitive, hence not all of them induce proper consistency notions. All of our examples are transitive, though.

### 3.2 Examples

The  $k$ -consistency algorithm was studied by Freuder [12]. It can be defined as follows. Let  $\mathbf{A}$  and  $\mathbf{B}$  be two structures. Let  $H$  be the collection of all partial homomorphisms  $h$  from  $\mathbf{A}$  to  $\mathbf{B}$  such that  $|\text{Dom}(h)| \leq k$ . For every  $h$  in  $H$  with  $|\text{Dom}(h)| \leq k$  and every  $a$  in  $A$ , if there does not exist any  $b$  in  $B$  such that  $g := h \cup \{(a, b)\}$  is a partial homomorphism from  $\mathbf{A}$  to  $\mathbf{B}$ , for which every  $f \subseteq g$  with  $|\text{Dom}(f)| \leq k$  belongs to  $H$ , remove  $h$  from  $H$ , and repeat. Whenever  $H$  does not change anymore, stop. If  $H$  stabilizes to a non-empty set, we say that  $\mathbf{A}$  is  $k$ -consistent with respect to  $\mathbf{B}$ . Otherwise we say that it is  $k$ -inconsistent.

The form we presented of the  $k$ -consistency algorithm is somewhat closer to the game interpretation given by Kolaitis and Vardi [18]. Our framework, of course, also captures it. We formulate a slightly more general version of it, called  $k, \ell$ -consistency, that appears in [10] and goes back to [12]:

*Example 1.* Let  $k$  and  $\ell$  be natural numbers with  $0 < k < \ell$ . Define

$$G = \{(i, \mathbf{D}) : i \leq k \text{ and } \mathbf{D} \text{ is } j\text{-numbered with } i \leq j \leq \ell\}$$

$$S = \{(i, K) : K \subseteq \{1, \dots, i\}, \text{ and } |K| \leq k\},$$

Then  $\leq^{G,S}$  is called  $k, \ell$ -consistency and we denote it by  $\leq^{k,\ell}$ . This game is grow-bounded by  $k$  and fully-bounded by  $\ell$ . The special case  $\leq^{k,k+1}$  is called  $k$ -consistency.

A variant of the  $k$ -consistency algorithm was introduced by Bulatov, who called it  $k$ -minimality. Although the differences are minor, specially when the vocabulary is finite, we show how to phrase it in our framework as the particular case of 1-minimality is a very well-known algorithm called arc-consistency.

Before we phrase the  $k$ -minimality algorithm in game-theoretic terms, let us present the algorithmic view of arc-consistency. Let  $\mathbf{A}$  and  $\mathbf{B}$  be two structures. The algorithm maintains a set  $S_a \subseteq B$  for every  $a \in A$ , initially set to  $B$ . For every  $a \in A$ , every  $b \in S_a$ , every relation symbol  $R$ , and every  $(a_1, \dots, a_r) \in R^{\mathbf{A}}$ , if there does not exist any  $(b_1, \dots, b_r) \in R^{\mathbf{B}}$  such that, for every  $j \in \{1, \dots, r\}$ , it holds that  $b_j \in S_{a_j}$ , and  $b_j = b$  whenever  $a_j = a$ , remove  $b$  from  $S_a$ , and repeat. Whenever the  $S_a$ 's do not change anymore, stop. If the  $S_a$ 's stabilize to non-empty sets, we say that  $\mathbf{A}$  is arc-consistent with respect to  $\mathbf{B}$ . Otherwise we say that it is arc-inconsistent. The algorithmic version of  $k$ -minimality is a straightforward generalization of this algorithm that maintains relations of arity at most  $k$ .

In order to define  $k$ -minimality in game terms, we need the following additional concept. Let  $i \leq j$ . An  $i, j$ -tuple structure is a  $j$ -numbered structure  $\mathbf{D}$  such that

$R^{\mathbf{D}} \neq \emptyset$  holds for exactly one  $R$  in the vocabulary, such that  $R^{\mathbf{D}}$  has exactly one tuple  $(d_1, \dots, d_r)$ , and such that  $\{d_1, \dots, d_r\} = \{c_i^{\mathbf{D}}, \dots, c_j^{\mathbf{D}}\}$ .

*Example 2.* Let  $k$  be a natural number with  $k > 0$ . Define

$$\begin{aligned} G &= \{(i, \mathbf{D}) : i \leq k \text{ and } \mathbf{D} \text{ is } j\text{-numbered with } i \leq j \leq k\} \cup \\ &\quad \{(k, \mathbf{D}) : \mathbf{D} \text{ is a } 1, j\text{-tuple structure with } j \geq k\} \\ S &= \{(i, K) : K \subseteq \{1, \dots, i\} \text{ and } |K| \leq k\}, \end{aligned}$$

Then  $\leq^{G,S}$  is called  $k$ -minimality and we denote it by  $\leq^{k\text{-MIN}}$ . This game is grow-bounded by  $k$ , and fully-bounded by the maximum of  $k$  and the maximum arity of the relation symbols in the vocabulary. The particular case  $\leq^{1\text{-MIN}}$  is called arc-consistency and we denote it by  $\leq^{\text{AC}}$ .

We discuss two more examples that will re-appear later in the paper. These are refinements of arc-consistency that have been studied in the literature, sometimes interchangeably. The first refinement is called peek arc-consistency in [6]. In algorithmic form, this stands for the procedure that, for every  $a \in A$ , checks if there exists some  $b \in B$  for which the arc-consistency algorithm started with  $S_a = \{b\}$ , and  $S_{a'} = B$  for  $a' \neq a$ , stabilizes with non-empty sets. As a game, this is phrased as follows:

*Example 3.* Define

$$\begin{aligned} G &= \{(i, \mathbf{D}) : \mathbf{D} \text{ is } j\text{-numbered with } i \leq j \leq 2\} \cup \\ &\quad \{(2, \mathbf{D}) : \mathbf{D} \text{ is a } 2, j\text{-tuple structure with } j \geq 2\} \\ S &= \{(i, \{1, j\}) : 2 \leq j \leq i\}, \end{aligned}$$

Then  $\leq^{G,S}$  is called peek arc-consistency and we denote it by  $\leq^{\text{PAC}}$ . This game is grow-bounded by 2 and fully-bounded by the maximum arity of the relation symbols in the vocabulary plus one.

The last example is singleton arc-consistency [5]. Algorithmically, we maintain sets  $T_a \subseteq B$ , initially set to  $B$ , and for every  $a \in A$  and every  $b \in T_a$  check whether arc-consistency started with  $S_a = \{b\}$ , and  $S_{a'} = T_{a'}$  for  $a' \neq a$ , stabilizes with non-empty sets. If it does not, we remove  $b$  from  $T_a$ , and repeat. Game-theoretically, here is how this is defined:

*Example 4.* Define

$$\begin{aligned} G &= \{(i, \mathbf{D}) : \mathbf{D} \text{ is } j\text{-numbered with } i \leq j \leq 2\} \cup \\ &\quad \{(2, \mathbf{D}) : \mathbf{D} \text{ is a } 2, j\text{-tuple structure with } j \geq 2\} \\ S &= \{(i, \{j\}) : 1 \leq j \leq i\} \cup \{(i, \{1, j\}) : 2 \leq j \leq i\}, \end{aligned}$$

Then  $\leq^{G,S}$  is called singleton-arc-consistency and we denote it by  $\leq^{\text{SAC}}$ . Again, this game is grow-bounded by 2 and fully-bounded by the maximum arity of the relation symbols in the vocabulary plus one.

### 3.3 Definability

We turn now to definability. We say that  $\leq$  induces a consistency notion that is *definable* in some logic if, for every finite structure  $\mathbf{B}$ , there exists a formula  $\varphi$  in the logic, such that for every finite structure  $\mathbf{A}$  we have  $\mathbf{A} \leq \mathbf{B}$  iff  $\mathbf{A} \models \varphi$ . We say that the definition is *effective* if furthermore such a  $\varphi$  can be computed from  $\mathbf{B}$ . The following is well-known:

**Lemma 2.**  $\rightarrow$  induces a consistency notion that is definable in monadic SNP. Furthermore, the definition is effective.

For the general pebble game, it does not seem possible to stay within monadic SNP, not even monadic second-order logic. However, standard methods give the following:

**Lemma 3.** Let  $G$  and  $S$  define a pebble game that is grow-bounded by  $k$  and fully-bounded. Then,  $\leq^{G,S}$  induces a consistency notion that is definable in  $k$ -ary co-Datalog and hence in  $k$ -ary SNP. Furthermore, if  $G$  and  $S$  are decidable, the definition is effective.

Note, in particular, that  $\leq^{\text{AC}}$  induces a consistency notion that is definable in monadic co-Datalog and hence in monadic SNP. According to this lemma,  $\leq^{\text{PAC}}$  and  $\leq^{\text{SAC}}$  induce consistency notions that are definable in binary co-Datalog and binary SNP, as they are both grow-bounded by 2. We will show more as both notions are definable in a monadic fragment of second-order logic. This will follow from a general condition on pebble games that we define next.

Let  $G$  and  $S$  define a pebble game. The game is called *stacked* if for every  $(k, K)$  in  $S$  there exist  $0 \leq i, j \leq k$  such that  $K \setminus \{j\} = \{1, \dots, i\}$ . Arc-consistency,  $1, \ell$ -consistency, peek-arc-consistency, and singleton-arc-consistency are all stacked. Note also that these examples are grow-bounded by 2 but, in general, stacked pebble games need not be grow-bounded by any fixed  $k$ . Thus, the following result, which is the main result of this section, gets interesting when compared to Lemma 3.

**Lemma 4.** Let  $G$  and  $S$  define a fully-bounded stacked pebble game. Then,  $\leq^{G,S}$  induces a consistency notion that is definable in monadic universal greatest fixed-point logic and hence in the closure of monadic SNP. Furthermore, if  $G$  and  $S$  are decidable, the definition is effective.

*Proof (rough sketch).* The construction makes heavy use of nested and simultaneous fixed points. Intuitively, the nesting levels correspond to the different  $i$  for shrinking sets of the form  $\{1, \dots, i, j\}$ . In this setting, only the  $j$  are able to change entries in positions, which leads to monadic fixed points being sufficient.  $\square$

### 3.4 Treewidth Duality

Let  $\mathcal{C}$  be a class of structures. The binary relation  $\leq$  has  $\mathcal{C}$ -duality, if for every pair of structures  $\mathbf{A}$  and  $\mathbf{B}$  such that  $\mathbf{A} \not\leq \mathbf{B}$ , there exists a  $\mathbf{C} \in \mathcal{C}$  such that

$\mathbf{C} \leq \mathbf{A}$  and  $\mathbf{C} \not\leq \mathbf{B}$ . We say that  $\leq$  has *treewidth- $k$ -duality*, if it has  $\mathcal{C}$ -duality for some  $\mathcal{C} \subseteq \text{TW}(k)$ .

Observe that this differs in formulation from the duality used in [10]. There, it is established that for a certain  $\leq$ , namely  $k$ -consistency,  $\mathbf{A} \not\leq \mathbf{B}$  implies the existence of  $\mathbf{C}$  in  $\text{TW}(k)$ , such that  $\mathbf{C} \rightarrow \mathbf{A}$  and  $\mathbf{C} \not\rightarrow \mathbf{B}$ . As for this  $\leq$  it turns out that  $\mathbf{C} \leq \mathbf{D}$  and  $\mathbf{C} \rightarrow \mathbf{D}$  are equivalent for every  $\mathbf{C}$  in  $\text{TW}(k)$  and every structure  $\mathbf{D}$ , the different formulations amount to the same. However, this equivalence does not carry over to other  $\leq$ . In particular, the following result does not follow directly from the fact that every bounded pebble game induces a consistency notion that is definable in co-Datalog. It requires its own proof.

**Lemma 5.** *Let  $G$  and  $S$  define a pebble game fully-bounded by  $k$ . Then  $\leq^{G,S}$  has treewidth- $(k - 1)$ -duality.*

*Proof (rough sketch).* For the structure  $\mathbf{C}$ , we use an unravelling of  $\mathbf{A}$ . The actual notion of unravelling is game specific, but it coincides with the usual one for the  $k$ -consistency game. In order to obtain a finite  $\mathbf{C}$ , we truncate the unravelling tree at a depth which is large enough to contain all relevant moves in a game between  $\mathbf{A}$  and  $\mathbf{B}$ . □

## 4 Application: Decidable Relative Consistency Results

The *relative consistency problem* for  $\leq$  and  $\leq'$  is the problem of, given some finite structure  $\mathbf{B}$ , deciding whether the implication

$$\mathbf{A} \leq \mathbf{B} \implies \mathbf{A} \leq' \mathbf{B} \tag{7}$$

holds for every finite  $\mathbf{A}$ . A simple application of the decidability of the satisfiability problem for MSO on structures bounded treewidth (see [11]) gives:

**Theorem 1.** *Let  $\leq$  and  $\leq'$  induce consistency notions such that:*

1. *both notions are effectively definable in MSO.*
2.  *$\leq'$  has treewidth duality.*
3.  *$\leq'$  is a refinement of  $\leq$ .*

*Then, the relative consistency problem for  $\leq$  and  $\leq'$  is decidable.*

*Proof.* Let  $\mathbf{B}$  be given and let  $k$  be such that  $\leq'$  has  $\text{TW}(k)$  duality. Let  $\varphi$  be an MSO definition of the consistency notion induced by  $\leq$  and  $\mathbf{B}$ , and let  $\varphi'$  be the one for  $\leq'$  and  $\mathbf{B}$ . We will show that the implication (7) fails if and only if there exists some  $\mathbf{C}$  in  $\text{TW}(k)$  such that  $\mathbf{C} \leq \mathbf{B}$  and  $\mathbf{C} \not\leq' \mathbf{B}$ . This last condition is equivalent to the satisfiability of  $\varphi \wedge \neg\varphi'$  in  $\text{TW}(k)$ , which is decidable.

The ‘if’ part of the claim is immediate. For the ‘only if’ part, let  $\mathbf{A}$  be such that  $\mathbf{A} \leq \mathbf{B}$  and  $\mathbf{A} \not\leq' \mathbf{B}$ . Using duality, let  $\mathbf{C}$  in  $\text{TW}(k)$  be such that  $\mathbf{C} \leq' \mathbf{A}$  and  $\mathbf{C} \not\leq' \mathbf{B}$ . From  $\mathbf{C} \leq' \mathbf{A}$  we obtain  $\mathbf{C} \leq \mathbf{A}$  because  $\leq'$  is a refinement of  $\leq$ , and then  $\mathbf{C} \leq \mathbf{B}$  using transitivity. □

Even though  $k$ -consistency induces a consistency notion that is probably not definable in MSO when  $k > 1$ , we can still prove the following result:

**Theorem 2.** *Let  $k \geq \text{ar}(\sigma)$ , and let  $\leq$  induce a consistency notion such that:*

1. *the notion is effectively definable in MSO.*
2.  *$\leq^k$  is a refinement of  $\leq$ .*

*Then, the relative consistency problem for  $\leq$  and  $\leq^k$  is decidable.*

*Proof.* The proof follows the same lines as in Theorem 1, using  $\text{TW}(k)$  duality of  $\leq^k$ . It remains to replace MSO definability of  $\leq^k$ . For this purpose, note that the previous proof only needed the formula  $\varphi'$  to describe whether  $\mathbf{A} \leq^k \mathbf{B}$  for  $\mathbf{A}$  in  $\text{TW}(k)$ , not for all finite structures. Recall that  $\leq^k$  and  $\rightarrow$  coincide on  $\text{TW}(k)$  (see 3), and that  $\rightarrow$  is definable in MSO. This is all we need.  $\square$

It is also possible to generalize this result to any consistency notion defined by a fully-bounded game replacing  $\leq^k$ . This requires some additional techniques, including a notion of *game treewidth*, which will appear in the full version of the paper.

Combining lemmata 4 and 5 and theorems 1 and 2, we obtain the following:

**Corollary 1.** *The following relative consistency problems are decidable:*

1. *Arc-consistency and peek arc-consistency.*
2. *Peek arc-consistency and singleton arc-consistency.*
3. *Singleton arc-consistency and  $k$ -consistency for  $k \geq \text{ar}(\sigma)$ .*
4.  *$1, \ell$ -consistency and  $1, \ell'$ -consistency for  $\ell \leq \ell'$ .*
5.  *$1, \ell$ -consistency and  $k$ -consistency for  $k \geq \ell - 1$ .*
6. *Transitive combinations of the above.*

As a side note we give a further result, which in particular implies that the injective variant of the relative consistency problem for  $\leq^k$  and  $\rightarrow$  is decidable.

**Theorem 3.** *Let  $\leq$  induce a consistency notion such that:*

1.  *$\leq$  is decidable.*
2.  *$\xrightarrow{e}$  is a refinement of  $\leq$ .*

*Then, the relative consistency problem for  $\leq$  and  $\xrightarrow{e}$  is decidable.*

*Proof.* Although a more elementary presentation would be possible, we proceed along the lines of the previous proofs. First,  $\xrightarrow{e}$  has size-plus-one duality: If  $\mathbf{A} \not\xrightarrow{e} \mathbf{B}$ , then there is some  $\mathbf{C}$  such that  $\mathbf{C} \xrightarrow{e} \mathbf{A}$ ,  $\mathbf{C} \not\xrightarrow{e} \mathbf{B}$ , and  $|C| \leq |B| + 1$ : If  $|A| \leq |B| + 1$ , we let  $\mathbf{C} := \mathbf{A}$ . Otherwise, let  $C$  be any subset of  $A$  of size  $|B| + 1$  and let  $\mathbf{C} = \mathbf{A} \upharpoonright C$ . Then  $\text{id} : \mathbf{C} \xrightarrow{e} \mathbf{A}$  and  $\mathbf{C} \not\xrightarrow{e} \mathbf{B}$  follows from injectivity.

Hence, we only need to decide whether  $\mathbf{A} \leq \mathbf{B}$  implies  $\mathbf{A} \xrightarrow{e} \mathbf{B}$  for all  $\mathbf{A}$  with  $|A| \leq |B| + 1$ . As  $\leq$  is decidable, this can be solved by brute force.  $\square$

## 5 Decidable Criterion for Arc-Consistency

In this section we concentrate on arc-consistency. We want to be able to decide, for a given  $\mathbf{B}$ , whether  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  implies  $\mathbf{A} \rightarrow \mathbf{B}$  for every finite  $\mathbf{A}$ . Put differently, we want to detect if there exists some *counterexample*: a finite  $\mathbf{A}$  such that  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  and yet  $\mathbf{A} \not\rightarrow \mathbf{B}$ . Since our goal is to build an arc-consistent instance, we start by developing the closure properties of this class of structures.

For the rest of this section, we fix a finite structure  $\mathbf{B}$  with vocabulary  $\sigma$ . For every  $b$  in  $B$ , let  $P_b$  be a new unary relation symbol and let  $\tau$  be  $\sigma \cup \{P_b : b \in B\}$ . For every  $\mathbf{A}$  such that  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$ , let  $\mathcal{W}(\mathbf{A})$  be the collection of all  $\tau$ -expansions of  $\mathbf{A}$  whose interpretations for  $\{P_b : b \in B\}$  satisfy the following conditions:

- i. for every  $a \in A$  there exists some  $b \in B$  such that  $a \in P_b^{\mathbf{A}}$ ,
- ii. for every  $a \in A$ , every  $b \in B$  such that  $a \in P_b^{\mathbf{A}}$ , every  $R \in \sigma$ , and every  $(a_1, \dots, a_r) \in R^{\mathbf{A}}$ , there exists  $(b_1, \dots, b_r) \in R^{\mathbf{B}}$  such that, for every  $j \in \{1, \dots, r\}$ , it holds that  $a_j \in P_{b_j}^{\mathbf{A}}$ , and  $b_j = b$  whenever  $a_j = a$ .

The condition  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  is equivalent to the statement that  $\mathcal{W}(\mathbf{A})$  is non-empty. Alternatively, we could have taken this as our definition of  $\leq^{\text{AC}}$ . Let  $\mathcal{W}$  denote the union of all  $\mathcal{W}(\mathbf{A})$  as  $\mathbf{A}$  ranges over all finite structures such that  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$ .

**Lemma 6.**  *$\mathcal{W}$  is closed under induced substructures and glued unions.*

*Proof.* Closure under induced substructures is immediate since the conditions i. and ii. defining  $\mathcal{W}(\mathbf{A})$  are universal on  $\mathbf{A}$ . We concentrate on glued unions. Let  $\mathbf{A}_1$  and  $\mathbf{A}_2$  be two structures in  $\mathcal{W}$  that agree on the common part: that is, for  $A_0 = A_1 \cap A_2$ , we have  $\mathbf{A}_1 \upharpoonright A_0 = \mathbf{A}_2 \upharpoonright A_0$ . Let  $\mathbf{A}_3$  be the glued union of  $\mathbf{A}_1$  and  $\mathbf{A}_2$ , and let  $\mathbf{A}$  be its  $\sigma$ -reduct. We claim that the sets  $\{P_b^{\mathbf{A}_3} : b \in B\}$  satisfy the conditions i. and ii. that define  $\mathcal{W}(\mathbf{A})$ . This will show that  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  and at the same time put  $\mathbf{A}_3$  in  $\mathcal{W}(\mathbf{A})$  and  $\mathcal{W}$ .

For condition i., fix an element  $a \in A_3$ . If  $a \in A_k$  for  $k \in \{1, 2\}$ , then there exists some  $b \in B$  such that  $a \in P_b^{\mathbf{A}_k}$  and the same  $b$  serves for  $\mathbf{A}_3$ . The fact that  $\mathbf{A}_1$  and  $\mathbf{A}_2$  agree on  $A_0$  guarantees that this is well-defined. For condition ii., fix  $a \in A_3$ ,  $b \in B$ ,  $R \in \sigma$ , and  $(a_1, \dots, a_r) \in R^{\mathbf{A}}$  as in its statement. Since  $(a_1, \dots, a_r)$  belongs to  $R^{\mathbf{A}}$  and  $\mathbf{A}$  is also the glued union of the  $\sigma$ -reducts of  $\mathbf{A}_1$  and  $\mathbf{A}_2$ , it must necessarily be the case that either  $\{a_1, \dots, a_r\} \subseteq A_1$  or  $\{a_1, \dots, a_r\} \subseteq A_2$ . In case  $\{a_1, \dots, a_r\} \subseteq A_k$  for  $k \in \{1, 2\}$ , let  $b_1, \dots, b_r \in B$  be given by condition ii. on  $\mathbf{A}_k$ . Again the fact that  $\mathbf{A}_1$  and  $\mathbf{A}_2$  agree on  $A_0$  guarantees that this choice is well-defined and valid for  $\mathbf{A}_3$ .  $\square$

It follows from the lemma that  $\mathcal{W}$  is an amalgamation class and, by Fraïssé’s construction (see [15]), there exists a countably infinite structure  $\mathbf{S}^+$  satisfying the following three properties:

1. every finite induced substructure of  $\mathbf{S}^+$  is isomorphic to a structure in  $\mathcal{W}$ ,
2. every structure in  $\mathcal{W}$  is isomorphic to a finite induced substructure of  $\mathbf{S}^+$ ,
3. for every two finite subsets  $S_1$  and  $S_2$  of  $S^+$ , if  $\mathbf{S}^+ \upharpoonright S_1$  and  $\mathbf{S}^+ \upharpoonright S_2$  are isomorphic, then there exists an automorphism of  $\mathbf{S}^+$  that maps  $S_1$  to  $S_2$ .

From now on, we write  $\mathbf{S}$  for the  $\sigma$ -reduct of  $\mathbf{S}^+$ . Except for the fact that it is infinite,  $\mathbf{S}$  is the candidate counterexample we are looking for. To establish this, the first and second properties of  $\mathbf{S}^+$  will suffice; the third property will be discussed later on. We start showing that  $\mathbf{S}$  is arc-consistent:

**Lemma 7.**  $\mathbf{S} \leq^{\text{AC}} \mathbf{B}$

*Proof.* If we show that the sets  $\{P_b^{\mathbf{S}^+} : b \in B\}$  satisfy the conditions i. and ii. that define  $\mathcal{W}(\mathbf{S})$ , it will follow that the duplicator has a winning strategy witnessing that  $\mathbf{S} \leq^{\text{AC}} \mathbf{B}$ . For condition i., fix an element  $a \in S^+$ . Let  $\mathbf{A}$  be the finite substructure  $\mathbf{S}^+ \upharpoonright \{a\}$ . By the first property of  $\mathbf{S}^+$ , the structure  $\mathbf{A}$  belongs to  $\mathcal{W}$ . Let then  $b$  be the witness to condition i. for  $\mathbf{A}$ . The same  $b$  works for  $\mathbf{S}^+$ . For condition ii., fix an element  $a \in S^+$ ,  $b \in B$ ,  $R \in \sigma$ , and  $(a_1, \dots, a_r) \in R^{\mathbf{S}}$  as in its statement. Let  $\mathbf{A}$  be the finite substructure  $\mathbf{S}^+ \upharpoonright \{a_1, \dots, a_r\}$ . By the first property of  $\mathbf{S}^+$ , the structure  $\mathbf{A}$  belongs to  $\mathcal{W}$ . Let then  $b_1, \dots, b_r \in B$  be the witnesses to condition ii. for  $\mathbf{A}$ . The same witnesses work for  $\mathbf{S}^+$ .  $\square$

Next we show that the existence of a homomorphism  $\mathbf{S} \rightarrow \mathbf{B}$  determines if arc-consistency solves  $\text{CSP}(\mathbf{B})$ .

**Lemma 8.** *The following are equivalent:*

1.  $\mathbf{S} \rightarrow \mathbf{B}$ .
2.  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  implies  $\mathbf{A} \rightarrow \mathbf{B}$  for every finite  $\mathbf{A}$ .

*Proof.* Assume  $\mathbf{S} \rightarrow \mathbf{B}$  and let  $\mathbf{A}$  be a finite structure such that  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$ . This means that  $\mathcal{W}(\mathbf{A})$  is not empty; let  $\mathbf{A}^+$  be a member of  $\mathcal{W}(\mathbf{A})$  and therefore of  $\mathcal{W}$ . By the second property of  $\mathbf{S}^+$ , the structure  $\mathbf{A}^+$  embeds into  $\mathbf{S}^+$ , and hence  $\mathbf{A}$  also embeds into  $\mathbf{S}$ . Since  $\mathbf{S} \rightarrow \mathbf{B}$ , also  $\mathbf{A} \rightarrow \mathbf{B}$ .

The converse is proved by a standard compactness argument. As we will not really need this implication in what follows, we omit the standard proof. At any rate, it will be a consequence of the results below (of course, without falling in a circularity; see the proof of Theorem 5).  $\square$

Our next goal is to finitize  $\mathbf{S}^+$ . Since we cannot satisfy the three properties of  $\mathbf{S}^+$  in a finite structure, we relax them significantly. This will give us a very naive first candidate to a finitized  $\mathbf{S}^+$  which we will strengthen later on. Let  $r$  be the maximum arity of the relations in  $\sigma$ . Let  $\mathbf{N}^+$  be a finite  $\tau$ -structure satisfying the following two properties:

1. every induced substructure of  $\mathbf{N}^+$  is isomorphic to some structure in  $\mathcal{W}$ ,
2. every structure in  $\mathcal{W}$  of cardinality at most  $r$  is isomorphic to some induced substructure of  $\mathbf{N}^+$ .

Note that the disjoint union of all structures in  $\mathcal{W}$  of cardinality at most  $r$  does the job. We will take this canonical example as our  $\mathbf{N}^+$ . Note that  $\mathbf{N}^+$  belongs to  $\mathcal{W}$  as  $\mathcal{W}$  is closed under glued unions and hence under disjoint unions. From now on, let  $\mathbf{N}$  be the  $\sigma$ -reduct of  $\mathbf{N}^+$ .



By itself,  $\mathbf{N}$  is way too naive. If  $\mathbf{N}$  is unsatisfiable, meaning that  $\mathbf{N} \not\rightarrow \mathbf{B}$ , we are certainly done as we have a counterexample. But if it is satisfiable, there is not much we can say. We will ask then for a stronger condition on  $\mathbf{N}$  that comes inspired by the third property of  $\mathbf{S}^+$ . We will say that  $\mathbf{N}$  is *strongly  $\mathbf{B}$ -satisfiable* if there exists a homomorphism  $f : \mathbf{N} \rightarrow \mathbf{B}$  such that  $f(a_1) = f(a_2)$  for every pair of points  $a_1$  and  $a_2$  in  $\mathbf{N}$  for which  $\mathbf{N}^+ \upharpoonright \{a_1\}$  and  $\mathbf{N}^+ \upharpoonright \{a_2\}$  are isomorphic. The following lemma links our naive candidate  $\mathbf{N}$  with our ideal candidate  $\mathbf{S}$ , in one direction:

**Lemma 9.** *If  $\mathbf{N}$  is strongly  $\mathbf{B}$ -satisfiable, then  $\mathbf{S} \rightarrow \mathbf{B}$ .*

*Proof.* Let  $f : \mathbf{N} \rightarrow \mathbf{B}$  be a homomorphism witnessing that  $\mathbf{N}$  is strongly satisfiable. Let  $c_1, c_2, c_3, \dots$  be a fixed enumeration of the universe of  $\mathbf{S}$ . We define a sequence of mappings  $h_0, h_1, h_2, h_3, \dots$  where  $h_i$  has domain  $\{c_1, \dots, c_i\}$ , inductively. Let  $h_0$  be the empty mapping. Let  $i > 0$  and suppose that  $h_{i-1}$  is already defined. Let  $a_i$  be any element of  $\mathbf{N}^+$  for which  $\mathbf{S}^+ \upharpoonright \{c_i\}$  and  $\mathbf{N}^+ \upharpoonright \{a_i\}$  are isomorphic. Such an  $a_i$  must exist by the first property of  $\mathbf{S}^+$  and the definition of  $\mathbf{N}^+$ . Let  $h_i$  be the extension of  $h_{i-1}$  that sets  $h_i(c_i) = f(a_i)$ . From the fact that  $\mathbf{N}$  is strongly satisfied by  $f$ , this does not depend on the choice of  $a_i$ .

We claim that the map  $h = \bigcup_i h_i$  is a homomorphism from  $\mathbf{S}$  to  $\mathbf{B}$ . Fix a tuple  $(c_{i_1}, \dots, c_{i_r})$  in some relation  $R^{\mathbf{S}}$ . Let  $d_1, \dots, d_r$  be such that  $\mathbf{S}^+ \upharpoonright \{c_{i_1}, \dots, c_{i_r}\}$  and  $\mathbf{N}^+ \upharpoonright \{d_1, \dots, d_r\}$  are isomorphic with  $c_{i_j}$  mapped to  $d_j$ . Such  $d_1, \dots, d_r$  exist by the first property of  $\mathbf{S}^+$  and the definition of  $\mathbf{N}^+$ . Since  $f$  is a homomorphism, we have  $(f(d_1), \dots, f(d_r)) \in R^{\mathbf{B}}$ . On the other hand,  $\mathbf{S}^+ \upharpoonright \{c_{i_j}\}$  is isomorphic to both  $\mathbf{N}^+ \upharpoonright \{d_j\}$  and  $\mathbf{N}^+ \upharpoonright \{a_{i_j}\}$ . It follows that  $f(d_j) = f(a_{i_j}) = h(c_{i_j})$ , and therefore also  $(h(c_{i_1}), \dots, h(c_{i_r})) \in R^{\mathbf{B}}$ . Thus  $h$  is a homomorphism.  $\square$

Our next goal is to reverse the implication in Lemma 9. For this we need to introduce some terminology from Ramsey theory.

Let  $\mathbf{C}$  and  $\mathbf{D}$  be structures and let  $p \geq 1$  and  $c \geq 1$  be integers. We write  $\mathbf{D} \rightarrow (\mathbf{C})_c^p$  if for every mapping  $f : \binom{D}{p} \rightarrow \{1, \dots, c\}$  there exists a strong embedding  $e : \mathbf{C} \xrightarrow{s} \mathbf{D}$  such that for every two sets  $A, B \subseteq C$  with  $|A| = |B| = p$  and  $\mathbf{C} \upharpoonright A \cong \mathbf{C} \upharpoonright B$ , it holds that  $f(e(A)) = f(e(B))$ . Here, the notation  $\binom{M}{p}$  stands for the collection of all subsets of  $M$  of size  $p$ . A classic result in Ramsey theory states that for every  $p$  and  $c$  and every finite structure  $\mathbf{C}$ , there exists a finite structure  $\mathbf{D}$  such that  $\mathbf{D} \rightarrow (\mathbf{C})_c^p$ . See [19] for a beautiful exposition and a discussion on the long history of this result.

On the one hand, we require the Ramsey result for the much simpler case of  $p = 1$ . On the other, we require it relative to a particular class of finite structures. If  $\mathcal{K}$  is a class of finite structures and  $p \geq 1$ , we say that  $\mathcal{K}$  is a  *$p$ -Ramsey class* if for every  $c \geq 1$  and every  $\mathbf{C}$  in  $\mathcal{K}$ , there exists a  $\mathbf{D}$  in  $\mathcal{K}$  such that  $\mathbf{D} \rightarrow (\mathbf{C})_c^p$ . We say that  $\mathcal{K}$  is a *pigeonhole class* if it is a 1-Ramsey class. Relativized Ramsey theorems are also known and have an equally long history. The version stated below seems not to appear in the literature but can be proved by standard methods in the area. We note that the restriction to  $p = 1$  is essential in all known approaches.

**Theorem 4.** *Let  $\mathcal{K}$  be a class of finite structures that is closed under induced substructures and glued unions. Then  $\mathcal{K}$  is a pigeonhole class.*

We see how this solves our problem by reversing the implication in Lemma 9. First note that, by Lemma 6 and Theorem 4, the class  $\mathcal{W}$  is a pigeonhole class. Let  $\mathbf{M}^+$  be the structure  $\mathbf{D}$  given by the Theorem 4 with  $\mathbf{C} = \mathbf{N}^+$  and  $c = |B|$ . Let  $\mathbf{M}$  be the  $\sigma$ -reduct of  $\mathbf{M}^+$ . These two structures will be used in the following:

**Lemma 10.** *If  $\mathbf{S} \rightarrow \mathbf{B}$ , then  $\mathbf{N}$  is strongly  $\mathbf{B}$ -satisfiable.*

*Proof.* Let  $h : \mathbf{S} \rightarrow \mathbf{B}$ . By the second property of  $\mathbf{S}^+$ , there exists  $f : \mathbf{M} \xrightarrow{e} \mathbf{S}$ . Composing we get  $h \circ f : \mathbf{M} \rightarrow \mathbf{B}$ . As  $\mathbf{M}^+ \rightarrow (\mathbf{N}^+)^1_{|B|}$ , there exists  $e : \mathbf{N}^+ \xrightarrow{s} \mathbf{M}^+$  such that if  $\mathbf{N}^+ \upharpoonright \{a_1\} \cong \mathbf{N}^+ \upharpoonright \{a_2\}$ , then  $h(f(e(a_1))) = h(f(e(a_2)))$ . Thus,  $h \circ f \circ e$  is a homomorphism witnessing that  $\mathbf{N}$  is strongly satisfiable.  $\square$

Finally, we obtain the characterization:

**Theorem 5.** *The following conditions are equivalent:*

1.  $\mathbf{S} \rightarrow \mathbf{B}$ ,
2.  $\mathbf{M} \rightarrow \mathbf{B}$ ,
3.  $\mathbf{N}$  is strongly  $\mathbf{B}$ -satisfiable,
4.  $\mathbf{A} \leq^{\text{AC}} \mathbf{B}$  implies  $\mathbf{A} \rightarrow \mathbf{B}$  for every finite  $\mathbf{A}$ .

*Proof.* Implication 1. to 2. follows from the second property of  $\mathbf{S}^+$  and the fact that  $\mathbf{M}^+$  belongs to  $\mathcal{W}$ . Implication 2. to 3. is in the proof of Lemma 10. Implication 3. to 1. is Lemma 9. This shows that 1., 2., and 3. are equivalent. The equivalence between 1. and 4. is Lemma 8. But since we proved only that 1. implies 4. in that Lemma, let us note how 4. implies 2.:  $\mathbf{M}^+$  belongs to  $\mathcal{W}$  and hence  $\mathbf{M} \leq^{\text{AC}} \mathbf{B}$ , which means that if 4. holds, then 2. holds as well.  $\square$

Note that 3. is a perfectly decidable condition. Condition 2. is also decidable as  $\mathbf{M}^+$  and  $\mathbf{M}$  are explicitly defined from  $\mathbf{N}^+$  (this is implicit in the full proof).

## 6 Concluding Remarks

Important progress on the analysis of the  $k$ -consistency algorithm was achieved recently through the algebraic approach to CSPs. Complete decidable classifications are now known for digraphs without sources or sinks [4] and for special triads [3]. Even for general structures a solution was announced recently. As soon as this breakthrough is confirmed, our results give also decidability for SAC and other stacked games by transitivity. A natural next step would be understanding this decidability proof through some explicit algebraic condition, or perhaps by showing that  $k$ -consistency is no more powerful than SAC for solving CSPs. On a related note, we are not aware of algebraic conditions that allow comparing the relative strength of two different algorithms as in Corollary 11. Again, this could be because different consistency algorithms collapse after all, or because some refinement of the algebraic approach awaits for discovery.

Finally, the decidable criterion we gave for AC has an appealing combinatorial flavour that calls for generalization. An explicit question we were unable to answer and that stopped our progress is this: Is the class of all instances that are  $k$ -consistent with respect to a fixed  $\mathbf{B}$  the collection of reducts of some amalgamation class? Results in the style of [7] indicate that this might be possible.

*Acknowledgment.* We thank Elitza Maneva for useful insights and discussions. The first author was partially supported by CICYT TIN2007-68005-C04-03.

## References

1. Ajtai, M., Fagin, R., Stockmeyer, L.J.: The closure of monadic NP. In: 30th Symp. on the Theory of Computing (1998)
2. Atserias, A., Bulatov, A.A., Dalmau, V.: On the power of  $k$ -consistency. In: 34th Intl. Colloq. on Automata, Languages and Programming, pp. 279–290 (2007)
3. Barto, L., Kozik, M., Maroti, M., Niven, T.: CSP dichotomy for special triads. Proc. Amer. Math. Soc. (to appear, 2009)
4. Barto, L., Kozik, M., Niven, T.: The CSP dichotomy holds for digraphs with no sources and no sinks. SIAM Journal on Computing 38(5), 1782–1802 (2009)
5. Bessiere, C., Debruyne, R.: Theoretical analysis of singleton arc consistency and its extensions. Artificial Intelligence 172(1), 29–41 (2008)
6. Bodirsky, M., Chen, H.: Peek arc consistency. In: CoRR, abs/0809.0788 (2008)
7. Covington, J.: Homogenizable relational structures. Illinois Journal of Mathematics 34(4), 731–743 (1990)
8. Dalmau, V., Kolaitis, P.G., Vardi, M.Y.: Constraint satisfaction, bounded treewidth, and finite-variable logics. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 310–326. Springer, Heidelberg (2002)
9. Ebbinghaus, H., Flum, J.: Finite Model Theory. Springer, Heidelberg (1995)
10. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction. SIAM Journal on Computing 28(1), 57–104 (1998)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
12. Freuder, E.C.: A sufficient condition for backtrack-free search. Journal of the ACM 29(1), 24–32 (1982)
13. Grohe, M.: Fixed-point logics on planar graphs. In: 13th IEEE Symposium on Logic in Computer Science, pp. 6–15 (1998)
14. Grohe, M.: Large Finite Structures with Few  $L^k$ -Types. Information and Computation 179(2), 250–278 (2002)
15. Hodges, W.: A shorter model theory. Cambridge University Press, Cambridge (1997)
16. Immerman, N., Kozen, D.: Definability with bounded number of bound variables. Information and Computation 83, 121–139 (1989)
17. Kolaitis, P.G., Vardi, M.Y.: The decision problem for the probabilities of higher-order properties. In: 19th Symp. Theory of Comp., pp. 425–435 (1987)
18. Kolaitis, P.G., Vardi, M.Y.: A game-theoretic approach to constraint satisfaction. In: 17th Nat. Conf. on Artificial Intelligence, pp. 175–181 (2000)
19. Nešetřil, J.: Ramsey classes and homogeneous structures. Combinatorics, Probability & Computing 14(1-2), 171–189 (2005)
20. Nešetřil, J., Zhu, Z.: On bounded treewidth duality of graphs. Journal of Graph Theory 23, 151–162 (1996)
21. Verbitsky, O.: Planar graphs: Logical complexity and parallel isomorphism tests. In: 24th Symp. on Theoretical Aspects of Computer Science, pp. 682–693 (2007)

# Cardinality Quantifiers in MLO over Trees<sup>\*</sup>

Vince Bárány<sup>1</sup>, Lukasz Kaiser<sup>2</sup>, and Alexander Rabinovich<sup>3</sup>

<sup>1</sup> Oxford University, Computing Laboratory

vbarany@comlab.ox.ac.uk

<sup>2</sup> RWTH Aachen, Mathematische Grundlagen der Informatik

kaiser@logic.rwth-aachen.de

<sup>3</sup> Tel Aviv University, School of Computer Science

rabinoo@post.tau.ac.il

**Abstract.** We study an extension of monadic second-order logic of order with the uncountability quantifier “there exist uncountably many sets”. We prove that, over the class of finitely branching trees, this extension is equally expressive to plain monadic second-order logic of order.

Additionally we find that the continuum hypothesis holds for classes of sets definable in monadic second-order logic over finitely branching trees, which is notable for not all of these classes are analytic.

Our approach is based on Shelah’s composition method and uses basic results from descriptive set theory. The elimination result is constructive, yielding a decision procedure for the extended logic. Furthermore, by the well-known correspondence between monadic second-order logic and tree automata, our findings translate to analogous results on the extension of first-order logic by cardinality quantifiers over injectively presentable Rabin-automatic structures, generalizing the work of Kuske and Lohrey.

## 1 Introduction

Monadic second-order logic of order, MLO, extends first-order logic by allowing quantification over *subsets* of the domain. The binary relation symbol  $<$  and unary predicate symbols  $P_i$  are its only non-logical relation symbols. MLO plays a very important role in mathematical logic and computer science. The fundamental connection between MLO and automata was discovered independently by Büchi, Elgot and Trakhtenbrot when the logic was proved to be decidable over the class of finite linear orders and over  $(\omega, <)$ . Moving away from linear orders, Rabin proved that monadic second-order theory of the full binary tree, S2S for short, is decidable [14]. This theorem, obtained using the notion of tree automata, is one of the most celebrated results in theoretical computer science, sometimes even called “the mother of all decidability results”.

*First-order cardinality quantifiers*, also known under the name of Magidor-Malitz quantifiers, count the number of elements with a given property. These

---

\* This research was facilitated by the ESF project AutoMathA. The first author was supported in part by ANR-06-MDCA-05 (2007-2009) DocFlow, and by the EPSRC grant EP/E010865/1.

quantifiers have been widely investigated in mathematical logic with respect to both decidability and the possibility of elimination. The book [1] presents results on decidability and other properties of first-order logic extended with such cardinality quantifiers over various natural classes of structures.

*Second-order cardinality quantifiers* in MLO, which we study in this paper, have been mostly considered in the context of automata and automatic structures. The first, basic result [2,3] shows that the quantifier “there exist infinitely many words” can be eliminated on automatic structures. By the standard correspondence between automata and MLO mentioned above, this is equivalent to eliminating the quantifier “there exist infinitely many sets” from *weak* MLO over  $(\omega, <)$ . The case of full MLO over  $(\omega, <)$  corresponds to injectively presented  $\omega$ -automatic structures and was solved by Kuske and Lohrey in [7,8]. Let us remark that, while cardinality quantifiers are hardly ever used directly in specifications, the structural properties of  $\omega$ -regular languages identified in these results gave important insights into automatic structures and their properties.

Motivated by previous work on  $(\omega, <)$  that used word automata, we investigate cardinality quantifiers over finitely branching trees, in particular over the binary tree with arbitrary labelings, which corresponds to tree automata with additional parameters. The parameterless question was previously studied by Nawiński, who in [12] proved that a regular language of infinite trees is uncountable if and only if it contains a non-regular tree.

This paper deals with the expressive power of the extension of MLO by cardinality quantifiers “there exist infinitely many subsets  $X$  such that” ( $\exists^{\aleph_0}$ ), “there exist uncountably many subsets  $X$  such that” ( $\exists^{\aleph_1}$ ) and “there exist at least continuum many subsets  $X$  such that” ( $\exists^{2^{\aleph_0}}$ ). We study the extension of MLO by these quantifiers,  $\text{MLO}(\exists^{\aleph_0}, \exists^{\aleph_1}, \exists^{2^{\aleph_0}})$ , over *simple trees*. These are finitely-branching trees every branch of which is either finite or of order type  $\omega$ . Our main results are summarized in the next two theorems.

**Theorem 1 (Elimination of the uncountability quantifier).** *For every  $\text{MLO}(\exists^{\aleph_0}, \exists^{\aleph_1}, \exists^{2^{\aleph_0}})$  formula  $\varphi(\bar{Y})$  there exists an MLO formula  $\psi(\bar{Y})$ , computable from  $\varphi$ , that is equivalent to  $\varphi(\bar{Y})$  over the class of simple trees.*

In addition to the above, the reduction will show that over simple trees the quantifiers  $\exists^{\aleph_1} X$  and  $\exists^{2^{\aleph_0}} X$  are equivalent, i.e. that the continuum hypothesis holds for MLO-definable families of sets. Though not surprising, this is not obvious for it is known that in MLO one can define non-analytic classes of sets [13] and that CH is independent of ZFC already for co-analytic sets [11].

**Theorem 2.** *For every MLO formula  $\varphi(X, \bar{Y})$ ,  $\exists^{\aleph_1} X \varphi(X, \bar{Y})$  is equivalent to  $\exists^{2^{\aleph_0}} X \varphi(X, \bar{Y})$  over simple trees.*

These results naturally extend to cardinality quantifiers  $\exists^{\aleph_0} \bar{X}$ ,  $\exists^{\aleph_1} \bar{X}$  and  $\exists^{2^{\aleph_0}} \bar{X}$  counting (finite) tuples of sets. This follows from the basic fact that for any cardinal  $\kappa \geq \aleph_0$  it holds  $\exists^\kappa (U, \bar{V}) \varphi \equiv \exists^\kappa U (\exists \bar{V} \varphi) \vee \exists^\kappa \bar{V} (\exists U \varphi)$ .

Note that  $\exists^\kappa X \varphi$  means “there exist *at least*  $\kappa$  sets  $X$  satisfying  $\varphi$ ”.

Our results bear relevance to the theory of automatic structures. Call a structure  $\mathfrak{A}$  *generalized tree-automatic* [4], or specifically  $\mathfrak{T}$ -automatic, if there is a *subset interpretation* of  $\mathfrak{A}$  in a labeled simple tree  $\mathfrak{T}$ . As introduced in [4], subset interpretations differ from monadic second-order interpretations in that the free variables of their constituent formulas are set variables. A structure  $\mathfrak{A}$  is thus  $\mathfrak{T}$ -automatic if it has a concrete representation with subsets of  $\mathfrak{T}$  as elements and atomic relations given by MLO formulas, equivalently, by Rabin tree-automata, hence the name. The first-order theory of  $\mathfrak{A}$  is thus interpreted in the MLO theory of  $\mathfrak{T}$ . Such a representation is called *injective* if equality is left uninterpreted [6]. Theorem 1 entails the following.

**Corollary 3.** *Cardinality quantifiers can be effectively eliminated from first-order logic on injectively presented generalized tree-automatic structures.*

This supersedes the previously mentioned results from [2,3] and [7,8] and generalizes the theorem of Niwiński from [12], which follows from a parameterless instance of our theorem. Certain structural insight gained from some of our intermediate lemmas might be of independent interest. More specifically we show that counting sets of nodes satisfying an MLO-formula on a simple tree can be effectively reduced to a combination of counting branches satisfying a certain MLO-formula, and counting chains with certain MLO-definable properties on individual branches. While the latter essentially amounts to dealing with the special case treated in [7,8], relying on basic results from descriptive set theory we show that counting of branches can also be implemented in MLO.

## 1.1 Organization

We begin by noting in Section 2 some observations considered folklore regarding the second-order infinity quantifier  $\exists^{\aleph_0} X$ . In Section 3 we fix terminology and notation on trees and recollect some essentials of Shelah’s composition method for MLO. The rest of the paper is devoted to the proof of Theorems 1 and 2.

In Section 4 we start by reducing the question of the existence of uncountably many sets  $X$  satisfying a given MLO formula  $\varphi(X, \bar{Y})$  with parameters  $\bar{Y}$  over a simple tree to a disjunction of three conditions: A, B and C. Condition A deals with MLO-properties of antichains; Condition C deals with a simpler version of the uncountability quantifier, namely with the quantifier “there exist uncountably many branches”. Ultimately, condition B is concerned with the cardinality of chains with a specific MLO property on individual branches, but it is postulated first in a far broader form for deductive advantages.

In Section 5, we show that Condition B can be significantly weakened assuming that conditions A and C are not satisfied. Relying on the elimination results on  $(\omega, <)$  from [7,8], we formalize this weakened form of Condition B in MLO and prove, that it guarantees the existence of continuum many sets satisfying  $\varphi$ .

In Section 6 we consider Condition C in the special case of the complete binary tree. The key theorem that we prove there, which might be of independent interest, is that MLO-definable sets of branches of the binary tree are Borel. This

opens the way to formalizing Condition C in plain MLO, first over the binary tree and finally, in Section 7, over arbitrary simple trees.

The proofs of our main theorems are summarized in Section 8, Section 9 states further results.

## 2 Infinity Quantifier

With regard to the second-order infinity quantifier  $\exists^{\aleph_0} X$  the following observations are worth making. While it clearly cannot be eliminated over all structures, it is easily expressible in monadic second-order logic (MSO) with the auxiliary predicate  $\text{Inf}(Z)$  asserting that the set  $Z$  is infinite, or equivalently, with the help of the first-order infinity quantifier  $\exists^{\aleph_0} x$ .

**Proposition 4.** *For every MSO( $\exists^{\aleph_0}$ ) formula  $\varphi(\overline{Y})$  there exists an MSO(Inf) formula  $\psi(\overline{Y})$  equivalent to  $\varphi(\overline{Y})$  over all structures.*

*Proof.* Observe that the following are equivalent:

- (1) There are only finitely many  $X$  which satisfy  $\varphi(X, \overline{Y})$ .
- (2) There is a finite set  $Z$  such that any two different sets  $X_1, X_2$  which both satisfy  $\varphi(X_i, \overline{Y})$  differ on  $Z$ , i.e.

$$\exists Z \left( \neg \text{Inf}(Z) \wedge \forall X_1 X_2 \left( (\varphi(X_1, \overline{Y}) \wedge \varphi(X_2, \overline{Y}) \wedge X_1 \neq X_2) \rightarrow \exists z \in Z (z \in X_1 \leftrightarrow z \notin X_2) \right) \right).$$

Item (2) implies (1) as a collection of sets pairwise differing only on a finite set  $Z$  has cardinality at most  $2^{|Z|}$ . Conversely, if  $X_1, \dots, X_k$  are all the sets that satisfy  $\varphi(X_i, \overline{Y})$ , then choose for every pair of distinct sets  $X_i, X_j$  an element  $z_{i,j}$  in the symmetric difference of  $X_i$  and  $X_j$  and define  $Z$  as the set of these chosen elements.  $\square$

Over simple trees  $\text{Inf}(Z)$  can of course be expressed in MLO. Indeed, with König's Lemma in mind,  $Z$  is infinite iff there is an infinite chain, equivalently, an unbounded set of nodes each having an element of  $Z$  below it:

$$\psi_{\text{Inf}}(Z) = \exists C \forall v \in C \exists w \in C, z \in Z : v < w \wedge v \leq z$$

**Corollary 5.** *MLO( $\exists^{\aleph_0}$ ) collapses effectively to MLO over the class of simple trees.*

Observe that the converse of Proposition 4 holds as well. In fact, the predicate  $\text{Inf}(Z)$  can be defined over all structures by the formula  $\exists^{\kappa} Y (Y \subseteq Z)$  for any  $\aleph_0 \leq \kappa \leq 2^{\aleph_0}$ . Therefore, by Proposition 4, any of the quantifiers  $\exists^{\kappa} Y$  with  $\aleph_0 < \kappa \leq 2^{\aleph_0}$  can be used to define  $\exists^{\aleph_0} X$  over arbitrary structures.

### 3 Preliminaries

For a given set  $A$  we denote by  $A^*$  the set of all finite sequences of elements of  $A$ , by  $A^\omega$  the set of all infinite sequences of elements of  $A$  (i.e. functions  $\omega \rightarrow A$ ), and  $A^{\leq\omega} = A^* \cup A^\omega$ . For any sequence  $s = s_0s_1s_2\dots \in A^{\leq\omega}$  we denote by  $|s|$  the length of  $s$  (either a natural number or  $\omega$ ) and by  $s|_n = s_0\dots s_{n-1}$  the finite sequence composed of the first  $n$  elements of  $s$ , with  $s|_0 = \varepsilon$ , the empty sequence. We write  $s[n]$  for the  $(n + 1)$ st element of  $s$  (we count from 0), so  $s[n] = s_n$  for  $n \in \mathbb{N}$ . Given a finite sequence  $s$  and a sequence  $t \in A^{\leq\omega}$  we denote by  $s \cdot t$  (or just  $st$ ) the concatenation of  $s$  and  $t$ . Moreover, we write  $s \preceq t$  if  $s$  is a prefix of  $t$ , i.e. if there exists a sequence  $r$  such that  $t = sr$ . A subset  $B$  of  $A^{\leq\omega}$  is said to be prefix-closed if for every  $t \in B$  and  $s \preceq t$  it holds that  $s \in B$ .

#### 3.1 Trees

For a number  $l \in \mathbb{N}, l > 0$ , an  $l$ -tree is a structure  $\mathfrak{T} = (T, <, P_1, \dots, P_l)$ , where the  $P_i$ 's are unary predicates and  $<$  is the irreflexive and transitive binary *ancestor* relation with a least element called the *root of  $\mathfrak{T}$*  and such that for every  $v \in T$  the set  $\{u \in T \mid u < v\}$  of ancestors of  $v$  is linearly ordered by  $<$ . Elements of a tree are referred to as *nodes*, maximal linearly ordered sets of nodes are called *branches*, ancestor-closed and linearly ordered sets of nodes are called *paths*, whereas *chains* are arbitrary linearly ordered subsets. An *antichain* is a set of pairwise incomparable nodes. Given a node  $v$ , the subtree of  $\mathfrak{T}$  rooted in  $v$  is obtained by restricting the structure to the domain  $T_v = \{u \in T \mid u \geq v\}$  and is denoted  $\mathfrak{T}_v$ .

Given a finite set  $A$ , we denote by  $\mathfrak{T}(A)$  the full tree over  $A$ , which is a structure with the universe  $A^*$ ,  $<$  interpreted as the prefix ordering and unary predicates  $P_a = A^*a$  for each  $a \in A$ . For finite  $A$  with  $|A| = n$ , this structure is axiomatizable in MLO and its MLO theory is the same as  $SnS$ , the monadic second-order theory of  $n$  successors (modulo trivial MLO-interpretations in  $\mathfrak{T}(n)$ ).

We identify a path  $B$  of  $\mathfrak{T}(A)$  with the sequence  $\beta = a_0a_1a_2\dots \in A^{\leq\omega}$  such that  $B = \{a_0\dots a_s \mid s \leq |\beta|\}$ . Conversely, given a sequence  $\beta \in A^{\leq\omega}$  we write  $\text{Pref}(\beta)$  for the corresponding path  $B$ .

Ordered sums of trees are defined as follows.

**Definition 6.** Let  $l > 0, \mathfrak{J} = (I, <^{\mathfrak{J}})$  be an unlabeled tree and let  $\mathfrak{T}_i = (T_i, <^i, P_1^i, \dots, P_l^i)$  be an  $l$ -tree for each  $i \in I$ . The tree sum of  $(\mathfrak{T}_i)_{i \in \mathfrak{J}}$ , denoted  $\sum_{i \in \mathfrak{J}} \mathfrak{T}_i$ , is the  $l$ -tree

$$\mathfrak{T} = \left( \bigcup_{i \in I} \{i\} \times T_i, <^{\mathfrak{T}}, \bigcup_{i \in I} \{i\} \times P_1^i, \dots, \bigcup_{i \in I} \{i\} \times P_l^i \right),$$

such that  $(i, a) <^{\mathfrak{T}} (j, b)$  for  $i, j \in I, a \in T_i, b \in T_j$  iff:

$$i <^{\mathfrak{J}} j \text{ and } a \text{ is the root of } \mathfrak{T}_i, \text{ or } i = j \text{ and } a <^i b.$$

Unless explicitly noted, we will not make a distinction between  $\mathfrak{T}_i$  and the isomorphic subtree  $\{i\} \times \mathfrak{T}_i$  of  $\mathfrak{T}$ .



A particular special case of the sum we will be using is when the index structure  $\mathcal{J}$  consists of a single branch, i.e. is a linear ordering. For every linear order  $(I, <)$  and chain  $\langle \mathfrak{T}_i \mid i \in I \rangle$  of trees, the sum  $\mathfrak{T} = \sum_{i \in I} \mathfrak{T}_i$  is well defined, and  $(I, <)$  forms a path (not necessarily maximal) of  $\mathfrak{T}$ .

We remark that not every tree can be decomposed as a sum along an arbitrarily chosen path. Such discrepancies can be ruled out by requiring that every two nodes possess a greatest common ancestor, i.e. an infimum. In this paper we consider only *simple trees*, which trivially fulfill this requirement.

**Definition 7.** *A simple tree is a finitely branching tree every branch of which is either finite or of order type  $\omega$ .*

### 3.2 MLO and the Composition Method

We will work with labeled trees in the relational signature  $\{<, P_1, \dots, P_l\}$  where  $<$  is a binary relation symbol denoting the ancestor relation of the tree, and the  $P_i$ 's are unary predicates representing a labeling.

Monadic second-order logic of order, MLO for short, extends first-order logic by allowing quantification over *subsets* of the domain. MLO uses first-order variables  $x, y, \dots$  interpreted as elements, and set variables  $X, Y, \dots$  interpreted as subsets of the domain. Set variables will always be capitalized to distinguish them from first-order variables. The atomic formulas are  $x < y$ ,  $x \in P_i$  and  $x \in X$ , all other formulas are built from the atomic ones by applying Boolean connectives and the universal and existential quantifiers for both kinds of variables. Concrete formulas will be given in this syntax, taking the usual liberties and short-hands, such as  $X \cup Y$ ,  $X \cap Y$ ,  $X \subseteq Y$ , guarded quantifiers and relativization of formulas to a set.

The quantifier rank of a formula  $\varphi$ , denoted  $\text{qr}(\varphi)$ , is the maximum depth of nesting of quantifiers in  $\varphi$ . For fixed  $n$  and  $l$  we denote by  $\text{Form}_{n,l}$  the set of formulas of quantifier depth  $\leq n$  and with free variables among  $X_1, \dots, X_l$ . Let  $n, l \in \mathbb{N}$  and  $\mathfrak{T}_1, \mathfrak{T}_2$  be  $l$ -trees. We say that  $\mathfrak{T}_1$  and  $\mathfrak{T}_2$  are  *$n$ -equivalent*, denoted  $\mathfrak{T}_1 \equiv^n \mathfrak{T}_2$ , if for every  $\varphi \in \text{Form}_{n,l}$ ,  $\mathfrak{T}_1 \models \varphi$  iff  $\mathfrak{T}_2 \models \varphi$ .

Clearly,  $\equiv^n$  is an equivalence relation. For any  $n \in \mathbb{N}$  and  $l > 0$ , the set  $\text{Form}_{n,l}$  is infinite. However, it contains only finitely many semantically distinct formulas, so there are only finitely many  $\equiv^n$ -classes of  $l$ -structures. In fact, we can compute representatives for these classes as follows.

**Lemma 8 (Hintikka Lemma).** *For  $n, l \in \mathbb{N}$ , we can compute a finite set  $H_{n,l} \subseteq \text{Form}_{n,l}$  such that:*

- For every  $l$ -tree  $\mathfrak{T}$  there is a unique  $\tau \in H_{n,l}$  such that  $\mathfrak{T} \models \tau$ .
- If  $\tau \in H_{n,l}$  and  $\varphi \in \text{Form}_{n,l}$ , then either  $\tau \models \varphi$  or  $\tau \models \neg\varphi$ . Furthermore, there is an algorithm that, given such  $\tau$  and  $\varphi$ , decides which of these two possibilities holds.

Elements of  $H_{n,l}$  are called  $(n, l)$ -Hintikka formulas.

Given an  $l$ -tree  $\mathfrak{T}$  we denote by  $\text{Tp}^n(\mathfrak{T})$  the unique element of  $H_{n,l}$  satisfied in  $\mathfrak{T}$  and call it the  $n$ -type of  $\mathfrak{T}$ . Thus,  $\text{Tp}^n(\mathfrak{T})$  determines (effectively) which formulas of quantifier-depth  $\leq n$  are satisfied in  $\mathfrak{T}$ .

We sometimes speak of the  $n$ -type of a tuple of subsets  $\bar{V} = V_1, \dots, V_m$  of a given  $l$ -tree  $\mathfrak{T}$ . This is synonymous with the  $n$ -type of the  $(l + m)$ -tree  $(\mathfrak{T}, \bar{V})$  obtained by expansion of  $\mathfrak{T}$  with the predicates  $P_{l+1}, \dots, P_{l+m}$  interpreted as the sets  $V_1, \dots, V_m$ . This type will be denoted by  $\text{Tp}^n(\mathfrak{T}, \bar{V})$  and often referred to as an  $n$ -type in  $m$  variables, whereby the  $n$ -type of the  $(l + m)$ -tree  $(\mathfrak{T}, \bar{V})$  is understood. Moreover, when considering substructures, e.g.  $\mathfrak{T}' \subseteq \mathfrak{T}$ , and given sets  $\bar{X} \subseteq \mathfrak{T}$ , we write  $\text{Tp}^n(\mathfrak{T}', \bar{X})$  to denote  $\text{Tp}^n(\mathfrak{T}', \bar{X} \cap \mathfrak{T}')$ .

The essence of the composition method is that certain operations on structures, such as disjoint union and certain ordered sums, can be projected to  $n$ -types. A general composition theorem for MLO from which most other follow was proved by Shelah in [15]. We only cite the composition theorem that we use [9], a more detailed presentation of the method can be found in [16,5].

**Theorem 9 (Composition Theorem for Trees).** *For every MLO-formula  $\varphi(\bar{X})$  in the signature of  $l$ -trees having  $m$  free variables and quantifier rank  $n$ , and given the enumeration  $\tau_1(\bar{X}), \dots, \tau_k(\bar{X})$  of  $H_{n,l+m}$ , there exists an MLO-formula  $\theta(Q_1, \dots, Q_k)$  such that for every tree  $\mathfrak{J} = (I, <^I)$  and family  $\{\mathfrak{I}_i \mid i \in I\}$  of  $l$ -trees and subsets  $V_1, \dots, V_m$  of  $\sum_{i \in I} \mathfrak{I}_i$ ,*

$$\sum_{i \in I} \mathfrak{I}_i \models \varphi(\bar{V}) \iff \mathfrak{J} \models \theta(Q_1, \dots, Q_k)$$

where  $Q_r = Q_r^{I;\bar{V}} = \{i \in I \mid \text{Tp}^n(\mathfrak{I}_i, \bar{V}) = \tau_r\}$  for each  $1 \leq r \leq k$ . Moreover,  $\theta$  is computable from  $\varphi$ , and does not depend on the decomposition of  $\mathfrak{T}$ .

## 4 D-Nodes versus U-Nodes and Relevant Branches

To eliminate the uncountability quantifier from  $\exists^{\aleph_1} X \varphi(X, \bar{V})$  over an  $l$ -tree  $\mathfrak{T}$ , we will consider certain colorings of segments of  $\mathfrak{T}$ . Let us first fix  $m$  sets  $\bar{V}$ ,  $n$  as the quantifier rank of  $\varphi$ , and  $k$  as the number of  $n$ -types in  $l + m + 1$  variables.

An *interval* of a tree is a connected and convex set  $I$  of nodes, i.e. such that for every  $u, w \in I$  if  $u$  and  $w$  are incomparable, then their greatest common ancestor is in  $I$ , and if  $u < w$  then for every  $u < v < w$  also  $v \in I$ . We denote by  $\mathfrak{T}|_I$  the restriction of an  $l$ -tree  $\mathfrak{T}$  to the interval  $I$ .

An interval having a minimal element is called a *tree segment*. Observe that every interval of a simple tree is a tree segment and that the summands  $\mathfrak{I}_i$  of a tree sum  $\mathfrak{T} = \sum_{i \in I} \mathfrak{I}_i$  are tree segments of  $\mathfrak{T}$ . In fact any subtree  $\mathfrak{T}_z$  of a tree  $\mathfrak{T}$  is a tree segment.

Let  $Z$  be a subset of a tree  $\mathfrak{T}$  and  $z$  be an element of  $\mathfrak{T}$ . We use the notation  $\mathfrak{T}_{z \setminus Z}$  for the restriction of  $\mathfrak{T}$  to the set  $\mathfrak{T}_z \setminus (\bigcup_{w \in Z, z < w} \mathfrak{T}_w)$ . Any tree segment  $\mathfrak{T}'$  with a minimal element  $z$  can be written in the form  $\mathfrak{T}_{z \setminus Z}$ , where  $Z$  is the set  $\{u \mid u \geq z \wedge u \notin \mathfrak{T}'\}$ .

**Definition 10.** Let  $\mathfrak{T} = (T, <, \overline{P}, X, \overline{Y})$  be an  $l + m + 1$ -tree such that  $\mathfrak{T} \models \varphi(X, \overline{Y})$  and let  $I$  be an interval of  $\mathfrak{T}$ .

(1)  $I$  is a U-interval for  $\varphi, X, \overline{Y}$  iff

$$\mathfrak{T}|_I \models \forall Z \tau(Z, \overline{Y}) \rightarrow Z = X,$$

where  $\tau(X, \overline{Y})$  is the  $n$ -type of  $\mathfrak{T}|_I$  in  $m + 1$  variables<sup>1</sup>

(2)  $I$  is a D-interval for  $\varphi, X, \overline{Y}$  iff it is not a U-interval.

(3) In the special case of  $I = \{u \mid u \geq z\}$  we say that the subtree  $\mathfrak{T}_z$  is a U-tree or D-tree, respectively, and further say that  $z$  is a U-node or D-node for  $\varphi, X, \overline{Y}$ .

(4) The set of D-nodes for  $\varphi, X, \overline{Y}$  is denoted  $D(X)$ .

(5) An infinite path  $P$  is called a D-path for  $\varphi, X, \overline{Y}$  if every  $v \in P$  is a D-node for  $\varphi, X, \overline{Y}$ , i.e. if  $P \subseteq D(X)$ .

Whenever  $\varphi, X, \overline{Y}$  are clear from the context, we will write “D-interval for  $X$ ” instead of “D-interval for  $\varphi, X, \overline{Y}$ ”, and similarly for the other notions above.

Observe that  $D(X)$  is prefix-closed since if  $u < v$  and  $\mathfrak{T}_v$  is a D-tree then, by composition,  $\mathfrak{T}_u$  is a D-tree as well. Therefore  $D(X)$  can be thought of as a tree whose infinite paths are precisely the infinite D-paths for  $X$ .

We note that each of the notions introduced in Definition 10 is formalizable in MLO. Let us start by constructing the formula  $\text{DINT}_\varphi(I, X, \overline{Y})$ , expressing that  $I$  is a D-interval for  $\varphi, X$  and  $\overline{Y}$ . By Lemma 8, the set of  $n$ -types  $H_{n, l+m+1}$  can be computed and is finite. Thus, we can write the formula

$$\psi_{\text{eqtp}}(X, X', \overline{Y}) = \bigwedge_{\tau \in H_{n, l+m+1}} \tau(X, \overline{Y}) \leftrightarrow \tau(X', \overline{Y}),$$

expressing that  $X$  and  $X'$  have the same  $n$ -type on the tree  $\mathfrak{T}$ . Let  $\psi_{\text{eqtp}}^{\text{rel}}(X, Z, \overline{Y}, I)$  be the relativization of  $\psi_{\text{eqtp}}(X, Z, \overline{Y})$  to an interval  $I$ , which expresses that  $X$  and  $Z$  have the same  $n$ -type on  $I$ .  $\text{DINT}_\varphi(I, X, \overline{Y})$  can now be written as

$$\varphi(X, \overline{Y}) \wedge \exists Z (\psi_{\text{eqtp}}^{\text{rel}}(X, Z, \overline{Y}, I) \wedge X \cap I \neq Z \cap I).$$

Using this formula we can also write the other formulas  $\text{DPATH}_\varphi(P, X, \overline{Y})$  and  $\text{DNODE}_\varphi(v, X, \overline{Y})$ , expressing, respectively, that  $P$  is a D-path and that  $v$  is a D-node for  $\varphi, X, \overline{Y}$ , and the formula  $\text{DSET}_\varphi(D, X, \overline{Y})$  which holds iff  $D = D(X)$ .

The following lemma is the first step in eliminating the  $\exists^{N_1}$  quantifier from MLO over simple trees.

**Lemma 11.** Let  $\mathfrak{T}$  be a simple  $l$ -tree and  $\varphi(X, \overline{Y})$  an MLO-formula in the signature of  $l$ -trees. Then, for every tuple of subsets  $\overline{V}$  of  $\mathfrak{T}$ ,

$$\mathfrak{T} \models \exists^{N_1} X \varphi(X, \overline{V})$$

if and only if one of the following conditions is satisfied.

<sup>1</sup> As set before,  $n$  is the quantifier rank of  $\varphi$  and  $m$  is the length of  $\overline{Y}$ .

- A. There is a set  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$  and there is an infinite antichain  $A$  of  $D$ -nodes for  $\varphi, U, \overline{V}$ .
- B. There is an infinite branch  $B$  of  $\mathfrak{T}$  which is a  $D$ -path for uncountably many  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$ .
- C. The following set of branches of  $\mathfrak{T}$

$$\{B \mid \text{there exists a set } U \text{ such that } B \text{ is a } D\text{-path for } \varphi, U, \overline{V}\}$$

is uncountable.

*Proof.* Note first that over simple trees, where König’s Lemma applies, condition A is properly subsumed by, in other words implies B and is enlisted here for deductive reasons only.

Indeed, A is arguably the most natural (easily expressible) condition sufficient for the existence of continuum many sets  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$ . To see that, let  $U$  and  $A$  be as in A and let  $v_0$  denote the root of  $\mathcal{T} = (\mathfrak{T}, U, \overline{V})$ . Then  $\mathcal{T}$  can be decomposed as  $\mathcal{T} = \mathcal{T}_{v_0 \setminus A} + \sum_{w \in A} \mathcal{T}_w$ . Applying the Composition Theorem (Th.9) to this decomposition, we get that  $\mathfrak{T} \models \varphi(U', \overline{V})$  for every  $U'$  such that  $U' \cap \mathfrak{T}_{v_0 \setminus A} = U \cap \mathfrak{T}_{v_0 \setminus A}$  and  $\text{Tp}^n(\mathfrak{T}_w, U', \overline{V}) = \text{Tp}^n(\mathfrak{T}_w, U, \overline{V})$  for all  $w \in A$ . By the choice of  $A$ ,  $U$  can be modified independently on each subtree  $\mathcal{T}_w$  without changing its type  $\text{Tp}^n(\mathcal{T}_w)$ . Hence there are continuum many different sets  $U'$  as above.

Furthermore,  $\neg A$  amounts to saying that for each  $U$  satisfying  $\varphi(U, \overline{V})$  the set  $D(U)$  induces a tree comprised of only finitely many branches. In particular, there are only finitely many infinite  $D$ -paths for each such  $U$ .

Condition B explicitly requires the existence of uncountably many sets satisfying  $\varphi(X, \overline{V})$ , so it too is sufficient for  $\exists^{\aleph_1} X \varphi(X, \overline{V})$  to hold. Hence it remains to be shown that when B fails then C is both sufficient and necessary hereto.

Assuming B does not hold in some  $\mathfrak{T}$  then A fails there as well, hence, as pointed out above, in this case there are only finitely many infinite  $D$ -paths for each  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$ . Also by the failure of B every branch is a  $D$ -path for at most countably many  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$ . It follows that every such set  $U$  shares its set of  $D$ -nodes with at most countably many other such  $U'$ . Indeed, this is clear from the above whenever  $D(U)$  contains an infinite  $D$ -path. If on the other hand  $D(U)$  is finite then  $U$  is fully determined by  $U \cap D$  and the  $n$ -types of all those  $U$ -nodes that are successors of some  $D$ -node, which only allows for a finite number of choices of  $U$  given that  $\mathfrak{T}$  is simple.

Thus we have established that whenever B fails in some  $\mathfrak{T}$  then there are uncountably many  $U$  satisfying  $\mathfrak{T} \models \varphi(U, \overline{V})$  iff there are uncountably many sets  $D(U)$  with  $\mathfrak{T} \models \varphi(U, \overline{V})$  iff condition C holds. □

Let us note again that if condition A holds then there are in fact continuum many sets  $X$  satisfying the formula  $\varphi(X, \overline{Y})$ . The description of Condition A can be directly formalized in MLO(Inf), hence, over simple trees, also in MLO as follows:

$$\begin{aligned} \psi_A(\overline{Y}) = & \exists U \exists A (\varphi(U, \overline{Y}) \wedge \text{Inf}(A) \wedge \text{antichain}(A) \wedge \\ & (\forall w \in A \text{ DNODE}_{\varphi}(w, U, \overline{Y}))), \end{aligned}$$

where  $\text{antichain}(A) = \forall x, y \in A \neg(x < y \vee y < x)$ .

## 5 Condition B

In this section, we show that a branch  $B$  is a witness for Condition B if and only if this branch satisfies a disjunction of three sub-conditions: Ba, Bb and Bc. Moreover, if both Condition A and Condition C fail, then already the sub-conditions Ba and Bc are sufficient. Finally, we express both Ba and Bc in MLO and show, that in fact both these sub-conditions guarantee the existence of continuum many sets  $X$  satisfying the formula  $\varphi(X, \overline{Y})$  in consideration.

As in the previous section, we assume that the formula  $\varphi(X, \overline{Y})$  of quantifier rank  $n$  is fixed together with a simple  $l$ -tree  $\mathfrak{T}$  and  $m$  parameters  $\overline{Y}$ , and let  $k$  be the number of  $n$ -types in  $l + m + 1$  variables. Additionally, we fix a branch  $B$  and introduce the formula  $\psi(X, \overline{Y}, P)$  stating that  $P$  is an infinite D-path for  $X$  and that  $\varphi(X, \overline{Y})$  holds:

$$\psi(X, \overline{Y}, P) = \text{DPATH}_\varphi(P, X, \overline{Y}) \wedge \text{Inf}(P) \wedge \varphi(X, \overline{Y}).$$

Note that the branch  $B$  witnesses Condition B if and only if  $\exists^{\aleph_1} U \psi(U, \overline{Y}, B)$ .

To break up Condition B, we decompose  $\mathcal{T} = (\mathfrak{T}, X, \overline{Y})$  along the branch  $B$ ,  $\mathcal{T} = \sum_{w \in B} \mathcal{T}_{w \setminus B}$ , and apply the Composition Theorem (Th.9) to this decomposition and the formula  $\psi$ . This yields a formula  $\theta$  such that

$$\mathcal{T} \models \psi(X, \overline{Y}, B) \iff (B, <) \models \theta(P_1, \dots, P_r),$$

where  $r$  is the number of  $\text{qr}(\psi)$ -types in  $l + m + 2$  variables, which we enumerate as  $\tau_1, \dots, \tau_r$ , and

$$P_i = \{w \in B \mid (\mathcal{T}_{w \setminus B}, \{w\}) \models \tau_i\}.$$

Note that we use the expansion of  $\mathcal{T}_{w \setminus B}$  by  $\{w\}$  as  $w$  is the only element of  $\mathcal{T}_{w \setminus B}$  that belongs to  $B$ .

With this reformulation it is clear that a branch  $B$  satisfies condition B if and only if either there are uncountably many different  $\overline{P}$  satisfying  $\theta$ , or some  $\overline{P}$  satisfying  $\theta$  has uncountably many  $X$  corresponding to it. Along these lines one obtains the following breakdown of condition B.

**Lemma 12.** *There are uncountably many  $X \subseteq \mathfrak{T}$  satisfying the formula  $\psi(X, \overline{Y}, B)$  in  $\mathfrak{T}$  iff one of the following sub-conditions holds.*

- (Ba) *There exists a set  $X$  such that  $\mathfrak{T}_{w \setminus B}$  is a D-interval for  $\varphi, X, \overline{Y}$  for infinitely many  $w \in B$ .*
- (Bb) *There exists a set  $X$  satisfying  $\psi$  and a  $w \in B$  so that*

$$\mathfrak{T}_{w \setminus B} \models \exists^{\aleph_1} X' \tau_i(X', \overline{Y} \cap \mathfrak{T}_{w \setminus B}, \{w\}),$$

where  $\tau_i = \text{Tp}^{\text{qr}(\psi)}(\mathfrak{T}_{w \setminus B}, X, \overline{Y}, \{w\})$ .

- (Bc) *It holds that*

$$(B, <) \models \exists^{\aleph_1} \overline{P} \left( \theta(\overline{P}) \wedge \bigwedge_{i=1}^r P_i \subseteq Q_i \wedge \forall x \left( \bigvee_{i=1}^r (x \in P_i \wedge \bigwedge_{j \neq i} x \notin P_j) \right) \right),$$

where for each  $1 \leq i \leq r$ ,  $Q_i$  is the set of nodes on the branch  $B$  in which the type  $\tau_i$  is satisfied by some set  $X$ , i.e.

$$Q_i = \{w \in B \mid \mathfrak{T}_{w \setminus B} \models \exists X \tau_i(X, \overline{Y} \cap \mathfrak{T}_{w \setminus B}, \{w\})\}$$

Observe that (Ba) already subsumes A in the sense that if condition A holds then there is a branch satisfying (Ba). Also observe that Condition (Bb) is itself just another instance of our initial problem. It is important to note, however, that the above cases classify conditions under which an *individual branch* may satisfy B. At closer inspection we find that if no branch satisfies either (Bc) or (Ba) (so that in particular A fails) and moreover condition C fails too, then B cannot hold either.

**Lemma 13.** *If over a simple tree  $\mathfrak{T}$  both Conditions A and C fail, then Condition B holds iff some branch satisfies Condition (Ba) or Condition (Bc).*

One intuitive way to see this is that if all the conditions A, (Ba), (Bc) and C fail on a tree, and thereby also on every tree segment of that tree, then for (Bb) to hold for a proper tree segment that tree segment would have to contain a proper tree segment on which (Bb) holds, and so on indefinitely. This would ultimately trace an infinite branch witnessing (Ba) contrary to the initial assumption.

Next we will construct MLO formulas  $\psi_{Ba}(B, \overline{Y})$  and  $\psi_{Bc}(B, \overline{Y})$  formalizing sub-conditions (Ba) and (Bc), respectively. By the above, we can then use the formula  $\psi_B(\overline{Y}) = \exists B(\psi_{Ba}(B, \overline{Y}) \vee \psi_{Bc}(B, \overline{Y}))$  in place of Condition B in Lemma 11.

### 5.1 Formalization of Condition Ba

Much like condition A, (Ba) is naturally expressible in MLO(Inf) and thus, over simple trees, in pure MLO as well by the formula

$$\psi_{Ba}(B, \overline{Y}) = \exists X \exists^{\aleph_0} w \text{ DINT}(T_{w \setminus B}, X, \overline{Y}),$$

where  $T_{w \setminus B}$  is just a notation for the set defined by

$$x \in T_{w \setminus B} \iff w \leq x \wedge \neg \exists b \in B (b > w \wedge b \leq x).$$

The fact that Condition (Ba) is sufficient for the existence of continuum many sets  $U$  satisfying  $\varphi(U, \overline{V})$  can be arrived at by appealing to the Composition Theorem in the same manner as for Condition A in the proof of Lemma 11, because the set  $X$  can be left intact or changed to another one with the same type on any of the infinitely many trees  $\mathfrak{T}_{w \setminus B}$  which are D-intervals for  $X$ .

### 5.2 Formalization of Condition Bc

In order to eliminate the explicit use of the uncountability quantifier from Condition (Bc) over  $(B, <) \cong (\omega, <)$ , we use Proposition 2.5 from [8] reformulated using the standard equivalence of automata and MLO on  $(\omega, <)$ , as stated in the following proposition.

**Proposition 14** (cf. [7,8]). *For every MLO formula  $\varphi(\overline{X}, \overline{Y})$  there exists an effectively constructible formula  $\psi(\overline{Y})$  such that over  $(\omega, <)$*

$$\psi(\overline{Y}) \equiv \exists^{\aleph_1} \overline{X} \varphi(\overline{X}, \overline{Y}) \equiv \exists^{2^{\aleph_0}} \overline{X} \varphi(\overline{X}, \overline{Y}).$$

Applying this result to the formula on the right hand side of Condition (Bc), with  $\overline{Q}$  as parameters, we obtain a formula  $\vartheta(\overline{Q})$  such that Condition (Bc) holds iff  $(B, <) \models \vartheta(\overline{Q})$ , with  $\overline{Q}$  as specified there.

By Proposition [4], if  $\vartheta(\overline{Q})$  holds, then there are even continuum many sets  $\overline{P}$  satisfying Condition (Bc). This in turn ensures the existence of continuum many sets  $X$  satisfying  $\varphi(X, \overline{Y})$ , because for each  $\overline{P}$  accounted for in  $\vartheta(\overline{Q})$  a corresponding  $X$  satisfying  $\psi(X, \overline{Y}, B)$  can be found and this association is necessarily injective.

To formalize Condition (Bc) in MLO over the tree  $\mathfrak{T}$ , we first define the sets  $Q_i$  on  $\mathfrak{T}$ . As the set of types is computable, we can compute each  $\tau_i$  and thus effectively construct the formula  $\alpha_i(w, B, \overline{Y})$  expressing that  $w$  is a node on the branch  $B$  such that  $\mathfrak{T}_{w \setminus B} \models \exists X \tau_i(X, \overline{Y} \cap \mathfrak{T}_{w \setminus B}, \{w\})$ , i.e.  $w \in Q_i$ . Using this formula we can express Condition (Bc) as  $\psi_{\text{Bc}}(B, \overline{Y}) =$

$$\exists \overline{Q} \left( \bigwedge_{i=1}^r (w \in Q_i \leftrightarrow \alpha_i(w, B, \overline{Y})) \wedge \vartheta^B(\overline{Q}) \right),$$

where  $\vartheta^B$  is a relativization of  $\vartheta$  to the branch  $B$ .

## 6 The Full Binary Tree and the Cantor Space

In order to formalize Condition C in MLO over simple trees, we first analyze the problem only on the full binary tree and identify and prove the following key topological property that distinguishes counting branches from counting arbitrary sets.

On the full binary tree  $\mathfrak{T}(2) = (\{0, 1\}^*, \prec, S_0, S_1)$  where  $\prec$  is the prefix-order and  $S_i = \{0, 1\}^*i$ , we show that the set of branches satisfying any given MLO formula is a Borel set in the Cantor topology and hence it has the *perfect set property*: it is uncountable iff it contains a perfect subset iff it has the cardinality of the continuum. A *perfect set* is a closed set without isolated points.

The Cantor-Bendixson Theorem states that closed subsets of a Polish space have the *perfect set property*: they are either countable or contain a perfect subset and thus have cardinality continuum. A set  $P$  is *perfect* if it is closed and if every point  $p \in P$  is a condensation point of  $P$ , i.e. if every neighborhood of  $p$  contains another point from  $P$ . We shall rely on the following fundamental result of Souslin.

**Theorem 15** (cf. e.g. in [11]). *A subset of a Polish space is Borel if and only if it is both analytic and co-analytic. Moreover, every uncountable analytic set contains a perfect subset.*

Note that whether co-analytic sets, or all sets on higher levels of the projective hierarchy, satisfy the continuum hypothesis is independent of ZFC [11].

A key observation that our formalization will exploit is that, even though there are non-Borel sets of trees definable in MLO, sets of definable paths are Borel. Recall that for a sequence  $\pi \in \{0, 1\}^*$  we denote by  $\text{Pref}(\pi)$  the path through  $\mathfrak{T}(2)$  that corresponds to this sequence, which formally is the set of prefixes of  $\pi$ .

**Theorem 16 (MLO definable sets of branches are Borel).** *Let  $U_1, \dots, U_m$  be subsets of  $\mathfrak{T}(2)$  and let  $\psi(X, \bar{Y})$  be an MLO formula over  $\mathfrak{T}(2)$ . Then the set*

$$\mathcal{X} = \{ \pi \in \{0, 1\}^\omega \mid \mathfrak{T}(2) \models \psi(\text{Pref}(\pi), \bar{U}) \}$$

*of branches of the binary tree satisfying  $\psi(X, \bar{U})$  is Borel and therefore it has the perfect set property.*

*Proof.* Note that the complement of  $\mathcal{X}$  is also definable by  $\neg\psi(X, \bar{U})$ . We will show that every definable set of branches is analytic. Therefore, by Souslin’s Theorem, it is Borel. To prove this, we will use the following variation of the Composition Theorem (cf. [9]).

**Lemma 17.** *Let  $\psi(X, Y_1, \dots, Y_m)$  be an MLO formula with quantifier rank  $n \geq 2$ , and let  $k$  be the number of  $(n + 2)$ -types in  $m + 1$  variables. Then there exists an MLO formula  $\theta(I, Z_1, \dots, Z_k)$  such that*

$$\mathfrak{T}(2) \models \psi(\text{Pref}(\pi), \bar{U}) \iff (\omega, <) \models \theta(\{n \mid \pi[n] = 1\}, \bar{Q}),$$

where for each  $1 \leq i \leq k$  we define  $Q_i = Q_i^{\pi, \bar{U}}$  as

$$Q_i = \{j \in \omega \mid \text{Tp}^{n+2}(\mathfrak{T}(2)_{\pi|_j}, \bar{U}) = \tau_i\}.$$

Let  $\theta$  be the formula obtained by applying the above lemma to  $\psi$ . Then, by the well-known correspondence of MLO and finite automata on  $\omega$ -words, there is an  $\omega$ -regular language  $\mathcal{L}_\theta \subseteq (\{0, 1\}^{k+1})^\omega \cong \{0, 1\}^\omega \times (\{0, 1\}^k)^\omega$ , such that  $\mathcal{L}_\theta$  consists of those pairs of sequences  $(\pi, \rho)$  for which  $(\omega, <) \models \theta(P, \bar{Q})$ , where  $P$  and  $\bar{Q}$  are subsets of  $\omega$  with characteristic sequences  $\pi \in \{0, 1\}^\omega$  and  $\rho \in (\{0, 1\}^k)^\omega$ . By McNaughton’s theorem [10],  $\mathcal{L}_\theta \in \Sigma_3^0$ .

Let  $\mathcal{T}$  be the extension of  $\mathfrak{T}(2)$  with each node  $w$  labeled by  $(\sigma, \bar{q})$  such that  $w$  is the  $\sigma$ -th successor of its parent (i.e.  $w \in S_\sigma$ ) and  $\bar{q} = (0, \dots, 0, 1, 0, \dots, 0)$  with the 1 in position  $i$  if  $\text{Tp}^{n+2}(\mathfrak{T}(2)_w, \bar{U}) = \tau_i$ . The set  $[\mathcal{T}]$  of labeled infinite branches of  $\mathcal{T}$  is closed in the Cantor topology.

By construction,  $\mathcal{X}$  is the projection of  $\mathcal{L}_\theta \cap [\mathcal{T}]$  to its first component, and is analytic as  $\mathcal{L}_\theta \in \Sigma_3^0$  and  $[\mathcal{T}] \in \Pi_1^0$ . □

## 7 Formalizing Condition C

The perfect set property established in Theorem 16 provides an MLO-definable characterization of Condition C of Lemma 11 over the full binary tree (with arbitrary labeling). Via interpretations, this can be extended to all simple trees to yield the following characterization.



**Proposition 18 (Eliminating uncountably-many-branches quantifier).** *For every MLO formula  $\varphi(X, \bar{Y})$  the assertion “ $\exists^{\aleph_1} B \text{ branch}(B) \wedge \varphi(B, \bar{Y})$ ” is equivalent over all simple trees to the existence of a perfect set of branches  $B$ , each satisfying  $\varphi(B, \bar{Y})$ . The latter ensures that there are in fact continuum many such branches.*

Towards an MLO formulation, note that the collection of nodes of a perfect set of branches induces a perfect tree, and vice versa. A perfect tree is one without isolated branches, equivalently, one in which for every node  $u$  there are incomparable nodes  $v, w > u$ . Perfectness is thus first-order definable.

**Corollary 19.** *Over simple trees Condition C is expressible in MLO as*

$$\psi_C(\bar{Y}) = \exists P \text{ perfect}(P) \forall B \subset P, \text{branch}(B) \exists X \text{DPATH}_\varphi(B, X, \bar{Y})$$

*Hence if Condition C holds then there are continuum many  $D$ -paths altogether for all sets  $U$  satisfying  $\varphi(U, \bar{Y})$ .*

## 8 Summary of the Proofs

As we have shown above, the conditions of Lemma 11 can be formalized in MLO over simple trees, thus we can again state the conclusion of this Lemma:  $\mathfrak{T} \models \exists^{\aleph_1} X \varphi(X, \bar{Y})$  holds if and only if

$$\mathfrak{T} \models \psi_A(\bar{Y}) \vee \exists B (\psi_{B_a}(B, \bar{Y}) \vee \psi_{B_c}(B, \bar{Y})) \vee \psi_C(\bar{Y}).$$

Using the above, we can reduce any formula of  $\text{MLO}(\exists^{\aleph_1})$  to an MLO formula equivalent over the class of simple trees by inductively eliminating the inner-most occurrence of a cardinality quantifier. Theorem 1 follows. Moreover, as we have shown in the corresponding sections, each of the conditions of Lemma 11 implies the existence of continuum many sets  $X$  satisfying  $\varphi(X, \bar{Y})$ , whence Theorem 2.

## 9 Further Results

The technique we used here can be applied to linear orders and leads to the following generalization of the theorem of Kuske and Lohrey (c.f. Proposition 14).

**Theorem 20 (Eliminating uncountability quantifier on linear orders).**

- (1) *For every  $\text{MLO}(\exists^{\aleph_1})$  formula  $\varphi(\bar{Y})$  there exists an MLO formula  $\psi(\bar{Y})$  that is equivalent to  $\varphi(\bar{Y})$  over the class of all ordinals.*
- (2) *For every  $\text{MLO}(\exists^{\aleph_1})$  formula  $\varphi(\bar{Y})$  there exists an MLO formula  $\psi(\bar{Y})$  that is equivalent to  $\varphi(\bar{Y})$  over the class of all countable linear orders. Moreover,  $\exists^{\aleph_1} X \varphi(X, \bar{Y})$  is equivalent to  $\exists^{2^{\aleph_0}} X \varphi(X, \bar{Y})$  over the class of countable linear orders.*

*Furthermore, in all these cases  $\psi$  is computable from  $\varphi$ .*

The proof will be provided in an extension of this paper. Note that the elimination result in (2) cannot be obtained simply by interpretation of countable linear orders in the full binary tree.

**Acknowledgment.** We are very grateful to Sasha Rubin for insightful discussions at an earlier stage of this work.

## References

1. Baudisch, A., Seese, D., Tuschik, H.P., Weese, M.: Decidability and Generalized Quantifiers. Akademie-Verlag, Berlin (1980)
2. Blumensath, A.: Automatic structures. Diploma thesis, RWTH Aachen (1999)
3. Blumensath, A., Grädel, E.: Automatic Structures. In: Proceedings of 15th IEEE Symposium on Logic in Computer Science LICS 2000, pp. 51–62 (2000)
4. Colcombet, T., Löding, C.: Transforming structures by set interpretations. Logical Methods in Computer Science 3(2:4), 1–36 (2007)
5. Gurevich, Y.: Monadic second-order theories. In: Barwise, J., Feferman, S. (eds.) Model-Theoretical Logics, pp. 479–506. Springer, Heidelberg (1985)
6. Hjørth, G., Khoussainov, B., Montalbán, A., Nies, A.: From automatic structures to Borel structures. In: 23rd Symp. on Logic in Computer Science, LICS (2008)
7. Kuske, D., Lohrey, M.: First-order and counting theories of  $\omega$ -automatic structures. In: Aceto, L., Ingólfssdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 322–336. Springer, Heidelberg (2006)
8. Kuske, D., Lohrey, M.: First-order and counting theories of omega-automatic structures. Journal of Symbolic Logic 73, 129–150 (2008)
9. Lifsches, S., Shelah, S.: Uniformization, choice functions and well orders in the class of trees. Journal of Symbolic Logic 61, 1206–1227 (1996)
10. McNaughton, R.: Testing and Generating Infinite Sequences by a Finite Automaton. Information and Control 9(5), 521–530 (1966)
11. Moschovakis, Y.N.: Descriptive Set Theory. Studies in Logic and the Foundations of Mathematics, vol. 100. North-Holland Publishing Company, Amsterdam (1980)
12. Nawiński, D.: On the cardinality of sets of infinite trees recognizable by finite automata. In: Tarlecki, A. (ed.) MFCS 1991. LNCS, vol. 520, pp. 367–376. Springer, Heidelberg (1991)
13. Nawiński, D., Walukiewicz, I.: A gap property of deterministic tree languages. Theoretical Computer Science 1(303), 215–231 (2003)
14. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 141, 1–35 (1969)
15. Shelah, S.: The monadic theory of order. Annals of Mathematics 102, 379–419 (1975)
16. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) Structures in Logic and Computer Science. LNCS, vol. 1261, pp. 118–143. Springer, Heidelberg (1997)

# From Coinductive Proofs to Exact Real Arithmetic

Ulrich Berger

Swansea University, Swansea SA2 8PP, Wales, UK  
u.berger@swansea.ac.uk

**Abstract.** We give a coinductive characterization of the set of continuous functions defined on a compact real interval, and extract certified programs that construct and combine exact real number algorithms with respect to the binary signed digit representation of real numbers. The data type corresponding to the coinductive definition of continuous functions consists of finitely branching non-wellfounded trees describing when the algorithm writes and reads digits. This is a pilot study in using proof-theoretic methods for certified algorithms in exact real arithmetic.

**Keywords:** Proof theory, program extraction, exact real number computation, coinduction.

## 1 Introduction

Most of the recent work on exact real number computation describes algorithms for functions on certain exact representations of the reals (for example streams of signed digits [1,2] or linear fractional transformations [3]) and proves their correctness using a certain proof method (for example coinduction [4,5,6,7]). Our work has a similar aim, and builds on the work cited above, but there are two important differences. The first is *methodological*: we do not ‘guess’ an algorithm and then verify it, instead we *extract* it from a proof, by some (once and for all) proven correct method. That this is possible in principle is well-known. Here we want to make the case that it is also feasible, and that interesting and nontrivial new algorithms can be obtained (see also [8,9] for related work on program extraction in constructive analysis and inductive definitions). The second difference is *algorithmic*: we do not represent a real function by a function on representations of reals, but by an infinite tree-like structure that contains not only information about the real function as a point map, but also and foremost information about the modulus of continuity. Since the representing tree is a pure data structure (without function component) a lazy programming language, like Haskell, will memoize computations which may improve performance in certain situations.

A crucial ingredient in the proofs (that we use for program extraction) is a coinductive definition of the notion of uniform continuity (u.c.). Although, classically, continuity and uniform continuity are equivalent for functions defined

on a compact interval (we only consider such functions), it is a suitable constructive definition of *uniform* continuity which matters for our purpose. For convenience, we consider as domain and range of our functions only the interval  $\mathbb{I} := [-1, 1] = \{x \in \mathbb{R} \mid |x| \leq 1\}$  and, for the purpose of this introduction, only unary functions. However, later we will also look at functions of several variables where one has to deal with the non-trivial problem of deciding which of the input stream the next digit is to be consumed of. This choice can have a big influence on the performance of the program.

We let  $\text{SD} := \{-1, 0, 1\}$  be the set of *signed digits*. By SDS we denote the set of all infinite streams  $a = a_0 : a_1 : a_2 : \dots$  of signed digits  $a_i \in \text{SD}$ . A signed digit stream  $a \in \text{SDS}$  represents the real number

$$\sigma(a) := \sum_{i \geq 0} a_i 2^{-(i+1)} \in \mathbb{I}$$

A function  $f : \mathbb{I} \rightarrow \mathbb{I}$  is *represented* by a stream transformer  $\hat{f} : \text{SDS} \rightarrow \text{SDS}$  if  $f \circ \sigma = \sigma \circ \hat{f}$ . The coinductive definition of uniform continuity allows us to extract from a constructive proof of the u.c. of a function  $f : \mathbb{I} \rightarrow \mathbb{I}$  an algorithm for a stream transformer  $\hat{f}$  representing  $f$ . Furthermore, we show directly and constructively that the coinductive notion of u.c. is closed under composition. The extracted algorithms are represented by finitely branching non-wellfounded trees which, if executed in a lazy programming language, give rise to memoized algorithms. These trees turn out to be a generalization of the data structure studied in [10], and the extracted program from the proof of closure under composition is a generalization of the tree composing program defined there.

In Section 2 we briefly review inductive and coinductive sets defined by monotone set operators. We give some simple examples, among them a coinductive characterization of the real numbers in the interval  $\mathbb{I}$ . The method of program extraction from proofs involving induction and coinduction is discussed informally, but in some detail, in Section 3. The earlier examples are continued and, for example, a program transforming fast Cauchy representations into signed digit representations is extracted from a coinductive proof. We also show how program extraction can be implemented in the functional programming language Haskell. As Haskell's syntax is very close to the usual mathematical notation for data and functions we hope that also readers not familiar with Haskell will be able to understand the code. In Section 4 the coinductive characterization of real numbers is generalized to real functions, and closure under composition is proven. In Section 5 the positive effect of memoization is demonstrated by a case study on iterated logistic maps.

## 2 Induction and Coinduction

We briefly discuss inductive and coinductive definitions as least and greatest fixed points of monotone set operators and the corresponding induction and coinduction principles. The results in this section are standard and can be found

in many logic and computer science texts. For example in [11] inductive definitions are proof-theoretically analysed, and in [12] least and greatest fixed points are studied in the framework of the modal mu-calculus.

An operator  $\Phi: \mathcal{P}(U) \rightarrow \mathcal{P}(U)$  (where  $U$  is an arbitrary set and  $\mathcal{P}(U)$  is the powerset of  $U$ ) is *monotone* if for all  $X, Y \subseteq U$

$$\text{if } X \subseteq Y, \text{ then } \Phi(X) \subseteq \Phi(Y)$$

A set  $X \subseteq U$  is  $\Phi$ -*closed* (or a pre-fixed point of  $\Phi$ ) if  $\Phi(X) \subseteq X$ . Since  $\mathcal{P}(U)$  is a complete lattice,  $\Phi$  has a least fixed point  $\mu\Phi$  (Knaster-Tarski Theorem). For the sake of readability we will sometimes write  $\mu X.\Phi(X)$  instead of  $\mu\Phi$ .  $\mu\Phi$  can be defined as the least  $\Phi$ -closed subset of  $U$ . Hence we have the *closure principle* for  $\mu\Phi$ ,  $\Phi(\mu\Phi) \subseteq \mu\Phi$  and the *induction principle* stating that for every  $X \subseteq U$ , if  $\Phi(X) \subseteq X$ , then  $\mu\Phi \subseteq X$ . It can easily be shown that  $\mu\Phi$  is even a *fixed point* of  $\Phi$ , i.e.  $\Phi(\mu\Phi) = \mu\Phi$ . For monotone operators  $\Phi, \Psi: \mathcal{P}(U) \rightarrow \mathcal{P}(U)$  we define

$$\Phi \subseteq \Psi \quad :\Leftrightarrow \quad \forall X \subseteq U \Phi(X) \subseteq \Psi(X)$$

It is easy to see that the operation  $\mu$  is *monotone*, i.e. if  $\Phi \subseteq \Psi$ , then  $\mu\Phi \subseteq \mu\Psi$ . Using monotonicity of  $\mu$  one can easily prove, by induction, a principle, called *strong induction*. It says that, if  $\Phi(X \cap \mu\Phi) \subseteq X$ , then  $\mu\Phi \subseteq X$ .

Dual to inductive definitions are *coinductive definitions*. A subset  $X$  of  $U$  is called  $\Phi$ -*coclosed* (or a post-fixed point of  $\Phi$ ) if  $X \subseteq \Phi(X)$ . By duality,  $\Phi$  has a largest fixed point  $\nu\Phi$  which can be defined as the largest  $\Phi$ -coclosed subset of  $U$ . Similarly, all other principles for induction have their coinductive counterparts. To summarise, we have the following principles:

<i>Fixed point</i>	$\Phi(\mu\Phi) = \mu\Phi$ and $\Phi(\nu\Phi) = \nu\Phi$ .
<i>Monotonicity</i>	if $\Phi \subseteq \Psi$ , then $\mu\Phi \subseteq \mu\Psi$ and $\nu\Phi \subseteq \nu\Psi$ .
<i>Induction</i>	if $\Phi(X) \subseteq X$ , then $\mu\Phi \subseteq X$ .
<i>Strong induction</i>	if $\Phi(X \cap \mu\Phi) \subseteq X$ , then $\mu\Phi \subseteq X$ .
<i>Coinduction</i>	if $X \subseteq \Phi(X)$ , then $X \subseteq \nu\Phi$ .
<i>Strong coinduction</i>	if $X \subseteq \Phi(X \cup \nu\Phi)$ , then $X \subseteq \nu\Phi$ .

**Example (natural numbers).** Define  $\Phi: \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{P}(\mathbb{N})$  by

$$\Phi(X) := \{0\} \cup \{y + 1 \mid y \in X\}$$

Then  $\mu\Phi = \mathbb{N} = \{0, 1, 2, \dots\}$ . We consider this as the *definition* of the natural numbers. The induction principle is logically equivalent to the usual zero-successor-induction on  $\mathbb{N}$ : if  $X(0)$  (base) and  $\forall x (X(x) \rightarrow X(x + 1))$  (step), then  $\forall x \in \mathbb{N} X(x)$ . Strong induction weakens the step by restricting  $x$  to the natural numbers:  $\forall x \in \mathbb{N} (X(x) \rightarrow X(x + 1))$ .

**Example (signed digits and the interval  $[-1, 1]$ ).** For every signed digit  $d \in \text{SD}$  we set  $\mathbb{I}_d := [d/2 - 1/2, d/2 + 1/2] = \{x \in \mathbb{R} \mid |x - d/2| \leq 1/2\}$ . Note that  $\mathbb{I}$  is the union of the  $\mathbb{I}_d$  and every sub interval of  $\mathbb{I}$  of length  $\leq 1/2$  is contained in some  $\mathbb{I}_d$ . We define an operator  $\mathcal{J}_0: \mathcal{P}(\mathbb{R}) \rightarrow \mathcal{P}(\mathbb{R})$  by

$$\mathcal{J}_0(X) := \{x \mid \exists d \in \text{SD} (x \in \mathbb{I}_d \wedge 2x - d \in X)\}$$

and set  $C_0 := \nu \mathcal{J}_0$ . Since clearly  $\mathbb{I} \subseteq \mathcal{J}_0(\mathbb{I})$ , it follows, by coinduction, that  $\mathbb{I} \subseteq C_0$ . On the other hand  $C_0 \subseteq \mathbb{I}$ , by the fixed point property. Hence  $C_0 = \mathbb{I}$ . The point of this definition is, that the proof of “ $\mathbb{I} \subseteq \mathcal{J}_0(\mathbb{I})$ ” has an interesting computational content:  $x \in \mathbb{I}$  must be given in such a way that it is possible to find  $d \in \text{SD}$  such that  $x \in \mathbb{I}_d$ . This means that  $d/2$  is a *first approximation* of  $x$ . The computational content of the proof of “ $\mathbb{I} \subseteq C_0$ ”, roughly speaking, iterates the process of finding approximations to  $x$  ad infinitum, i.e. it computes a *signed digit representation* of  $x$  as explained in the introduction, that is, a stream  $a$  of signed digits with  $\sigma(a) = x$ .

**Example (lists, streams and trees).** Let the Scott-domain  $D$  be defined by the recursive domain equation  $D = \{*\} + D \times D$  where “+” denotes the separated sum of domains (see [13] for information on domains). The elements of  $D$  are  $\perp$  (the obligatory least element),  $*$ , and  $(x, y)$  where  $x, y \in D$ . Define  $\text{Times}_* : \mathcal{P}(D) \rightarrow \mathcal{P}(D) \rightarrow \mathcal{P}(D)$  by

$$\text{Times}_*(X)(Y) := \{*\} \cup \{(x, y) \mid x \in X, y \in Y\}$$

Clearly,  $\text{Times}_*$  is monotone in both arguments. For a fixed set  $X \subseteq D$ ,  $\text{List}(X) := \mu(\text{Times}_*(X))$  ( $= \mu Y. \text{Times}_*(X)(Y)$ ) can be viewed as the set of *finite* lists of elements in  $X$  (viewing  $(\cdot, \cdot)$  as the “cons” operation), and  $\text{Stream}(X) := \nu(\text{Times}_*(X))$  ( $= \nu Y. \text{Times}_*(X)(Y)$ ) as the set of *finite or infinite* lists or *streams* of elements in  $X$ . Since  $\mu$  is monotone the operator  $\text{List} : \mathcal{P}(D) \rightarrow \mathcal{P}(D)$  is again monotone. Hence we can define  $\text{Tree} := \nu \text{List} \subseteq D$  which is the set of finitely branching wellfounded or non-wellfounded trees. On the other hand,  $\text{Tree}' := \mu \text{Stream}$  consist of all finitely or infinitely branching wellfounded trees. The point of this example is that the definition of  $\text{Tree}$  is similar to the characterization of uniformly continuous functions from  $\mathbb{I}^n$  to  $\mathbb{I}$  in Section 4, the similarity being the fact that it is a coinductive definition with an inductive definition in its body. The set  $C_0$  of the previous example corresponds to the case  $n = 0$  where the inner inductive definition is trivial.

### 3 Program Extraction from Proofs

In this section we briefly explain how we extract programs from proofs. Rather than giving a technical definition of the method and a rigorous correctness proof (which will be the subject of a separate paper) we explain it by means of simple examples, which hopefully provide a good intuition also for non-experts, and then make some general remarks concerning the computational content of induction and coinduction. The method of program extraction we are using is based on an extension and variation of Kreisel’s *modified realizability* [14]. The *extension* concerns the addition of inductive and coinductive predicates. Realizability for such predicates has been studied previously, in the slightly different context of **q**-realizability by Tatsuta [15]. The *variation* concerns the fact that we are treating the first-order part of the language (i.e. quantification over individuals) in a ‘uniform’ way, that is, realizers do not depend on the individuals quantified over.

This is similar to the common uniform treatment of second-order variables [16]. The argument is that an arbitrary subset of a set is such an abstract (and even vague) entity so that one should not expect an algorithm to depend on it. With a similar argument one may argue that individuals of an abstract mathematical structure (reals, model of set-theory, etc.) are unsuitable as inputs for programs. But which data should a program then depend on? The answer is: on data defined by the ‘propositional skeletons’ of formulas and ‘canonical’ proofs.

**Example (parity).** Let us extract a program from a proof of

$$\forall x (\mathbb{N}(x) \Rightarrow \exists y (x = 2y \vee x = 2y + 1)) \quad (1)$$

where the variable  $x$  ranges over real numbers and the predicate  $\mathbb{N}$  is defined as in the example in Section 2, i.e.  $\mathbb{N}$  is the least set of real numbers such that

$$\mathbb{N}(x) \Leftrightarrow x = 0 \vee \exists y (\mathbb{N}(y) \wedge x = y + 1) \quad (2)$$

The type corresponding to (2) is obtained by the following *type extraction*:

- replace  $\mathbb{N}(t)$  by `Nat` (a name for the data type to be defined),
- replace other atomic formulas by the unit or ‘void’ type `1`,
- delete all quantifiers,
- replace  $\vee$  by  $+$  (disjoint sum) and  $\wedge$  by  $\times$  (cartesian product),
- carry out obvious simplifications (e.g. replacing `Nat`  $\times$  `1` by `Nat`).

Hence `Nat` is the least solution of the equation

$$\text{Nat} = \mathbf{1} + \text{Nat}$$

In Haskell we can define this as

```
data Nat = Zero | Succ Nat      -- data
```

The comment “`-- data`” indicates that we intend to use the recursive data type `Nat` as an inductive data type (or initial algebra). This means that the “total”, or “legal” elements are inductively generated from `Zero` and `Succ`. The natural (domain-theoretic) semantics of `Nat` also contains, for example, an “infinite” element defined recursively by `infty = Succ infty` which is not total in the inductive interpretation of `Nat`. In a coinductive interpretation (usually indicated by the comment `-- codata`) `infty` would count as total. That Haskell does not distinguish between the inductive and the coinductive interpretation is justified by the limit-colimit-coincidence in the domain-theoretic semantics [17].

Applying type extraction to (1) we see that a program extracted from a proof of this formula will have type `Nat`  $\rightarrow$  `1 + 1`. Identifying the two-element type `1 + 1` with the Booleans we get the Haskell signature

```
parity :: Nat -> Bool
```

The definition of `parity` can be extracted from the obvious inductive proof of (1): For the base  $x = 0$ , we take  $y = 0$  to get  $x = 2y$ . In the step,  $x + 1$ , we have, by i.h. some  $y$  with  $x = 2y \vee x = 2y + 1$ . In the first case  $x + 1 = 2y + 1$ , in the second case  $x + 1 = 2(y + 1)$ . The Haskell program extracted from this proof is

```
parity Zero      = True
parity (Succ x) = case parity x of {True -> False ; False -> True}
```

If we wish to compute not only the parity, but as well the rounded down half of  $x$  (i.e. quotient and remainder), we just need to relativise the quantifier  $\exists y$  in  $\textcircled{II}$  to  $\mathbb{N}$  (i.e.  $\forall x (\mathbb{N}(x) \Rightarrow \exists y (\mathbb{N}(y) \wedge (x = 2y \vee x = 2y + 1)))$ ) and use the fact that  $\mathbb{N}$  is closed under the successor operation in the proof. The extracted program is then

```
parity1 :: Nat -> (Nat, Bool)
parity1 Zero      = (Zero, True)
parity1 (Succ x) = case parity1 x of
    {(y, True)  -> (y, False) ;
     (y, False) -> (Succ y, True)}
```

This example shows that we can get meaningful computational content despite ignoring the first-order part of a proof. Moreover, we can fine-tune the amount of computational information we extract from a proof by simple modifications. Note also that in the proofs we used arithmetic operations on the reals and their arithmetic laws without implementing or proving them. Since these laws can be written as equations (or conditional equations) their associated type is void. Hence it is only their truth that matters, allowing us to treat them as ad-hoc axioms without bothering to derive them from basic axioms. Note that a formula that does not contain a disjunction has always a void type and can therefore be taken as an axiom as long as it is true.

The reader might be puzzled by the fact that quantifiers are ignored in the program extraction process. Quantifiers are, of course, *not* ignored in the *specification* of the extracted program, i.e. in the definition of realizability. For example, the statement that the program  $p := \text{parity}$  realizes  $\textcircled{II}$  is expressed by

$$\forall n, x (n \mathbf{r} \mathbb{N}(x) \Rightarrow \exists y (p(n) = \text{True} \wedge x = 2y \vee p(n) = \text{False} \wedge x = 2y + 1))$$

where  $n$  ranges over  $\text{Nat}$ , i.e. the terms  $\text{Zero}$ ,  $\text{Succ Zero}$ ,  $\text{Succ}(\text{Succ Zero})$ ,  $\dots$ , and  $n \mathbf{r} \mathbb{N}(x)$  means that  $n$  realizes  $\mathbb{N}(x)$  which in this case amounts to  $x$  being the value of  $n$  in  $\mathbb{R}$ . The *Soundness Theorem* for realizability states that the program extracted from a proof realizes the proven formula (see e.g. [\[16\]](#), [\[15\]](#) for detailed proofs of soundness for related notions of realizability).

**Example (from Cauchy to signed digits).** In the second example of Section [2](#) we defined the set  $C_0$  as the largest set of real numbers such that

$$C_0(x) \Leftrightarrow \exists d (\text{SD}(d) \wedge \mathbb{I}_d(x) \wedge C_0(2x - d)) \tag{3}$$

Since  $\text{SD}(d)$  is shorthand for  $d = -1 \vee d = 0 \vee d = 1$ , and  $\mathbb{I}_d(x)$  is shorthand for  $|x - d/2| \leq 1/2$ , the corresponding type is the largest solution of the equation

$$\text{SDS} = (\mathbf{1} + \mathbf{1} + \mathbf{1}) \times \text{SDS} \tag{4}$$



Identifying the type  $\mathbf{1} + \mathbf{1} + \mathbf{1}$  with  $\mathbf{SD}$

```
data SD = N | Z | P      -- N = -1 , Z = 0, P = 1
```

we obtain that  $\mathbf{SDS}$  is the type of infinite streams of signed digits, i.e. the largest fixed point of the type operator

```
type J0 alpha = (SD, alpha)
```

which corresponds to the set operator  $\mathcal{J}_0$  which  $C_0$  is the largest fixed point of. Therefore (choosing  $\mathbf{Cons}$  as constructor name)

```
data SDS = Cons (J0 SDS)  -- codata
```

i.e.  $\mathbf{SDS} = \mathbf{Cons} (\mathbf{SD}, \mathbf{SDS})$ .

We wish to extract a program that computes a signed digit representation of  $x \in \mathbb{I}$  from a fast rational Cauchy sequence converging to  $x$ . Set

$$\begin{aligned} \mathbb{Q}(x) &:= \exists n, m, k (\mathbb{N}(n) \wedge \mathbb{N}(m) \wedge \mathbb{N}(k) \wedge x = (n - m)/k) \\ A(x) &:= \forall n (\mathbb{N}(n) \Rightarrow \exists q (\mathbb{Q}(q) \wedge |x - q| \leq 2^{-n})) \end{aligned}$$

Constructively,  $A(x)$  means that there is a fast Cauchy sequence of rational numbers converging to  $x$ .

**Lemma 1.**

$$\forall x (\mathbb{I}(x) \wedge A(x) \Rightarrow C_0(x)) \quad (5)$$

*Proof.* We show  $\mathbb{I} \cap A \subseteq C_0$  by coinduction, i.e. we show  $\mathbb{I} \cap A \subseteq \mathcal{J}_0(\mathbb{I} \cap A)$ . Assume  $\mathbb{I}(x)$  and  $A(x)$ . We have to show (constructively!)  $\mathcal{J}_0(\mathbb{I} \cap A)(x)$ , i.e. we need to find  $d \in \mathbf{SD}$  such that  $x \in \mathbb{I}_d$  and  $2x - d \in \mathbb{I} \cap A$ . Since, clearly  $A(2x - d)$  holds for any  $d \in \mathbf{SD}$ , and  $x \in \mathbb{I}_d$  holds iff  $2x - d \in \mathbb{I}$ , we only need to worry about  $x$  lying in  $\mathbb{I}_d$ . The assumption  $A(x)$ , used with  $n = 2$ , yields a rational  $q$  with  $|x - q| \leq 1/4$ . It is easy to find (constructively!) a signed digit  $d$  such that  $[q - 1/4, q + 1/4] \cap \mathbb{I} \subseteq \mathbb{I}_d$ . For that  $d$  we have  $x \in \mathbb{I}_d$ .

The type corresponding to the predicate  $\mathbb{Q}$  is  $\mathbf{Nat} \times \mathbf{Nat} \times \mathbf{Nat}$ , which we however implement by Haskell's built-in rationals, since it is only the arithmetic operations on rational numbers that matter, whatever the representation. (It is possible - and instructive as an exercise - to extract implementations of the arithmetic operations on rational numbers w.r.t. the representation  $\mathbf{Nat} \times \mathbf{Nat} \times \mathbf{Nat}$  from proofs that  $\mathbb{Q}$  is closed under these operations.) The type of the predicate  $A$  is  $\mathbf{Nat} \rightarrow \mathbf{Rational}$ . The program extracted from the proof of Lemma 1 is

```
cauchy2sd :: (Nat -> Rational) -> SDS
cauchy2sd = coitJ0 step
```

where  $\mathbf{step}$  is the program extracted from the proof of  $\mathbb{I} \cap A \subseteq \mathcal{J}_0(\mathbb{I} \cap A)$ :

```
step :: (Nat -> Rational) -> J0(Nat -> Rational)
step f = (d, f') where
```

```

q = f (Succ (Succ Zero))
d = if q > 1/4 then P else if abs q <= 1/4 then Z else N
f' n = 2 * f (Succ n) - sd2Rational d

```

```

sd2Rational :: SD -> Rational
sd2Rational d = case d of {N -> -1 ; Z -> 0 ; P -> 1}

```

The program `coitJ0` is a polymorphic “coiterator” realizing the coinduction scheme  $X \subseteq \mathcal{J}_0(X) \Rightarrow X \subseteq \nu\mathcal{J}_0$ :

```

coitJ0 :: (alpha -> J0 alpha) -> alpha -> SDS
coitJ0 s x = Cons (mapJ0 (coitJ0 s) (s x))

```

```

mapJ0 :: (alpha -> beta) -> J0 alpha -> J0 beta
mapJ0 f (d,x) = (d,f x)

```

An equivalent definition of `coitJ0` would be

```

coitJ0 s x = Cons (d,coitJ0 s y) where (d,y) = s x

```

We hope that this example and the way it was presented gives enough hints to understand how program extraction from coinductive proofs works in general. In the general case one has a coinductive predicate  $\nu\Phi$  defined from a positive (and therefore monotone) set operator  $\Phi$  ( $\mathcal{J}_0$  in our example), i.e.  $\Phi(X) = \{\mathbf{x} \mid A(X, \mathbf{x})\}$  where  $X$  occurs only positively (in the usual sense) in  $A$ .  $\Phi$  corresponds to a positive type operator `Phi` (`J0` in our example). Due to the positivity of `Phi` one can define `mapPhi :: (alpha -> beta) -> Phi alpha -> Phi beta` (by structural recursion on `Phi alpha`), and from that, recursively, the coiterator

```

coitPhi :: (alpha -> Phi alpha) -> alpha -> Fix
coitPhi s x = Cons (mapPhi (coitPhi s) (s x))

```

where `Fix` is the largest fixed point of `Phi`:

```

data Fix = Cons (Phi Fix) -- codata

```

The program extracted from a coinductive proof of  $X \subseteq \nu\Phi$  is then `coitPhi step` where `step :: alpha -> Phi alpha` is the program extracted from the proof of  $X \subseteq \Phi(X)$  (`alpha` is the type corresponding to the predicate  $X$ ). For inductive proofs the construction is similar: One defines recursively an “iterator”

```

itPhi :: (Phi alpha -> alpha) -> Fix -> alpha
itPhi s (Cons z) = s (mapPhi (itPhi s) z)

```

(where the type `Fix` is now viewed as the *least* fixed point of `Phi`). The program extracted from an inductive proof of  $\mu\Phi \subseteq X$  is now `itPhi step` where `step :: Phi alpha -> alpha` is extracted from the proof of  $\Phi(X) \subseteq X$ .

The above sketched computational interpretations of induction and coinduction and more general recursive schemes can be derived from category-theoretic considerations using the initial algebra/final coalgebra interpretation of least and greatest fixed points (see for example [18,19,20,21]).

## 4 Coinductive Definition of Uniform Continuity

For every  $n$  we define a set  $C_n \subseteq \mathbb{R}^{\mathbb{I}^n}$  for which we will later show that it coincides with the set of uniformly continuous functions from  $\mathbb{I}^n$  to  $\mathbb{I}$ .

In the following we let  $n, m, k, l, i$  range over  $\mathbb{N}$ ,  $p, q$  over  $\mathbb{Q}$ ,  $x, y, z$  over  $\mathbb{R}$ , and  $d, e$  over  $\text{SD}$ . Hence, for example,  $\exists d A(d)$  is shorthand for  $\exists d (\text{SD}(d) \wedge A(d))$ . We define average functions and their inverses

$$\begin{aligned} \text{av}_d: \mathbb{R} &\rightarrow \mathbb{R}, \quad \text{av}_d(x) := \frac{x+d}{2} \\ \text{va}_d: \mathbb{R} &\rightarrow \mathbb{R}, \quad \text{va}_d(x) := 2x-d \end{aligned}$$

Note that  $\text{av}_d[\mathbb{I}] = \mathbb{I}_d$  and hence  $f[\mathbb{I}] \subseteq \mathbb{I}_d$  iff  $(\text{va}_d \circ f)[\mathbb{I}] \subseteq \mathbb{I}$ . We also need extensions of the average functions to  $n$ -tuples

$$\text{av}_{i,d}(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n) := (x_1, \dots, x_{i-1}, \text{av}_d(x_i), x_{i+1}, \dots, x_n)$$

We define an operator  $\mathcal{K}_n: \mathcal{P}(\mathbb{R}^{\mathbb{I}^n}) \rightarrow \mathcal{P}(\mathbb{R}^{\mathbb{I}^n}) \rightarrow \mathcal{P}(\mathbb{R}^{\mathbb{I}^n})$  by

$$\mathcal{K}_n(X)(Y) := \{f \mid \exists d (f[\mathbb{I}^n] \subseteq \mathbb{I}_d \wedge X(\text{va}_d \circ f)) \vee \exists i \forall d Y(f \circ \text{av}_{i,d})\}$$

Since  $\mathcal{K}_n$  is strictly positive in both arguments, we can define an operator  $\mathcal{J}_n: \mathcal{P}(\mathbb{R}^{\mathbb{I}^n}) \rightarrow \mathcal{P}(\mathbb{R}^{\mathbb{I}^n})$  by

$$\mathcal{J}_n(X) := \mu(\mathcal{K}_n(X)) = \mu Y. \mathcal{K}_n(X)(Y)$$

Hence,  $\mathcal{J}_n(X)$  is the set inductively defined by the following two rules:

$$\exists d (f[\mathbb{I}^n] \subseteq \mathbb{I}_d \wedge X(\text{va}_d \circ f)) \Rightarrow \mathcal{J}_n(X)(f) \quad (6)$$

$$\exists i \forall d \mathcal{J}_n(X)(f \circ \text{av}_{i,d}) \Rightarrow \mathcal{J}_n(X)(f) \quad (7)$$

Since, as mentioned in Section 2, the operation  $\mu$  is monotone,  $\mathcal{J}_n$  is monotone as well. Therefore, we can define  $C_n$  as the largest fixed point of  $\mathcal{J}_n$ ,

$$C_n = \nu \mathcal{J}_n = \nu X. \mu Y. \mathcal{K}_n(X)(Y) \quad (8)$$

Note that for  $n = 0$  the second argument  $Y$  of  $\mathcal{K}_n$  becomes a dummy variable, and therefore  $\mathcal{J}_0$  and  $C_0$  are the same as in the corresponding example in Sect. 2.

The type corresponding to the formula  $\mathcal{K}_n(X)(Y)$  is  $\text{SD} \times \alpha + \mathbb{N}_n \times \beta^3$ , where  $\mathbb{N}_n := \{1, \dots, n\}$ . Therefore, the type of  $\mathcal{J}_n(X)$  is  $\mu\beta.\text{SD} \times \alpha + \mathbb{N}_n \times \beta^3$  which is the type of finite ternary branching trees with indices  $i \in \mathbb{N}_n$  attached to the inner nodes and pairs  $(d, x) \in \text{SD} \times \alpha$  attached to the leaves. Consequently, the type of  $C_n$  is  $\nu\alpha.\mu\beta.\text{SD} \times \alpha + \mathbb{N}_n \times \beta^3$  which is the type of non-wellfounded trees obtained by infinitely often stacking the finite trees on top of each other, i.e. replacing in a finite tree each  $x$  in a leaf by another finite tree and repeating ad-infinitum the process in the substituted trees. Alternatively, the elements of this type can be described as non-wellfounded trees without leaves such that

1. each node is either a
  - writing node* labelled with a signed digit and with one subtree, or a
  - reading node* labelled with an index  $i \in \mathbb{N}_n$  and with three subtrees;
2. each path has infinitely many writing nodes.

The interpretation of such a tree as a stream transformer is easy. Given  $n$  signed digit streams  $a_1, \dots, a_n$  as inputs, run through the tree and output a signed digit stream as follows:

1. At a writing node  $(d, t)$  output  $d$  and continue with the subtree  $t$ .
2. At a reading node  $(i, (t_d)_{d \in \text{SD}})$  continue with  $t_d$ , where  $d$  is the head of  $a_i$ , and replace  $a_i$  by its tail.

This interpretation corresponds to the extracted program of a special case of Proposition [1](#) below which shows that the predicates  $C_n$  are closed under composition.

**Lemma 2.** *If  $C_n(f)$ , then  $C_n(f \circ \text{av}_{i,d})$ .*

*Proof.* We fix  $i \in \{1, \dots, n\}$  and  $d \in \text{SD}$  and set

$$D := \{f \circ \text{av}_{i,d} \mid C_n(f)\}$$

We show  $D \subseteq C_n$  by strong coinduction, i.e. we show  $D \subseteq \mathcal{J}_n(D \cup C_n)$ , i.e.  $C_n \subseteq E$  where

$$E := \{f \mid \mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d})\}$$

Since  $C_n = \mathcal{J}_n(C_n)$  it suffices to show  $\mathcal{J}_n(C_n) \subseteq E$ . We prove this by strong induction on  $\mathcal{J}_n(C_n)$ , i.e. we show  $\mathcal{K}_n(C_n)(E \cap \mathcal{J}_n(C_n)) \subseteq E$ . Induction base: Assume  $f[\mathbb{I}^n] \subseteq \mathbb{I}_{d'}$  and  $C_n(\text{va}_{d'} \circ f)$ . We need to show  $E(f)$ , i.e.  $\mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d})$ . By [\(6\)](#) it suffices to show  $(f \circ \text{av}_{i,d})[\mathbb{I}^n] \subseteq \mathbb{I}_{d'}$  and  $(D \cup C_n)(\text{va}_{d'} \circ f \circ \text{av}_{i,d})$ . We have  $(f \circ \text{av}_{i,d})[\mathbb{I}^n] = f[\text{av}_{i,d}[\mathbb{I}^n]] \subseteq f[\mathbb{I}^n] \subseteq \mathbb{I}_{d'}$ . Furthermore,  $D(\text{va}_{d'} \circ f \circ \text{av}_{i,d})$  holds by the assumption  $C_n(\text{va}_{d'} \circ f)$  and the definition of  $D$ . Induction step: Assume, as strong induction hypothesis,  $\forall d' (E \cap \mathcal{J}_n(C_n))(f \circ \text{av}_{i',d'})$ . We have to show  $E(f)$ , i.e.  $\mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d})$ . If  $i' = i$ , then the strong induction hypothesis implies  $\mathcal{J}_n(C_n)(f \circ \text{av}_{i,d})$  which, by the monotonicity of  $\mathcal{J}_n$ , in turn implies  $\mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d})$ , i.e.  $E(f)$ . If  $i' \neq i$ , then  $\forall d' \text{av}_{i',d'} \circ \text{av}_{i,d} = \text{av}_{i,d} \circ \text{av}_{i',d'}$  and therefore, since the strong induction hypothesis implies  $\forall d' E(f \circ \text{av}_{i',d'})$ , we have  $\forall d' \mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d} \circ \text{av}_{i',d'})$ . By [\(7\)](#) this implies  $\mathcal{J}_n(D \cup C_n)(f \circ \text{av}_{i,d})$ , i.e.  $E(f)$ .

**Proposition 1.** *Let  $f: \mathbb{I}^n \rightarrow \mathbb{R}$  and  $g_i: \mathbb{I}^m \rightarrow \mathbb{R}$ ,  $i = 1, \dots, n$ . If  $C_n(f)$  and  $C_m(g_1), \dots, C_m(g_n)$ , then  $C_m(f \circ (g_1, \dots, g_n))$ .*

*Proof.* We prove the proposition by coinduction, i.e. we set

$$D := \{f \circ (g_1, \dots, g_n) \mid C_n(f), C_m(g_1), \dots, C_m(g_n)\}$$

and show that  $D \subseteq \mathcal{J}_m(D)$ , i.e.  $C_n \subseteq E$  where

$$E := \{f \in \mathbb{R}^{\mathbb{I}^n} \mid \forall \mathbf{g} (C_m(\mathbf{g}) \Rightarrow \mathcal{J}_m(D)(f \circ \mathbf{g}))\}$$

and  $C_m(\mathbf{g}) := C_m(g_1) \wedge \dots \wedge C_m(g_n)$ . Since  $C_n = \mathcal{J}_n(C_n)$  it suffices to show  $\mathcal{J}_n(C_n) \subseteq E$ . We do an induction on  $\mathcal{J}_n(C_n)$ , i.e. we show  $\mathcal{K}_n(C_n)(E) \subseteq E$ . Induction base: Assume  $f[\mathbb{I}^m] \subseteq \mathbb{I}_d$ ,  $C_n(\text{va}_d \circ f)$  and  $C_m(\mathbf{g})$ . We have to show  $(f \circ g)[\mathbb{I}^m] \subseteq \mathbb{I}_d$  and  $\mathcal{J}_m(D)(f \circ \mathbf{g})$ . By (6) it suffices to show  $D(\text{va}_d \circ f \circ \mathbf{g})$ . But this holds by the definition of  $D$  and the assumption. Induction step: Assume, as induction hypothesis,  $\forall d E(f \circ \text{av}_{i,d})$ . We have to show  $E(f)$ , i.e.  $C_m \subseteq F$  where

$$F := \{g \in \mathbb{R}^{\mathbb{I}^m} \mid \forall \mathbf{g} (g = g_i \wedge C_m(\mathbf{g}) \Rightarrow \mathcal{J}_m(D)(f \circ \mathbf{g}))\}$$

Since  $C_m \subseteq \mathcal{J}_m(C_m)$  it suffices to show  $\mathcal{J}_m(C_m) \subseteq F$  which we do by a side induction on  $\mathcal{J}_m$ , i.e. we show  $\mathcal{K}_m(C_m)(F) \subseteq F$ . Side induction base: Assume  $g[\mathbb{I}^m] \subseteq \mathbb{I}_d$  and  $C_m(\text{va}_d \circ g)$  and  $C_m(\mathbf{g})$  where  $g = g_i$ . We have to show  $\mathcal{J}_m(D)(f \circ \mathbf{g})$ . Let  $\mathbf{g}'$  be obtained from  $\mathbf{g}$  by replacing  $g_i$  with  $\text{va}_d \circ g$ . By the main induction hypothesis we have  $\mathcal{J}_m(D)(f \circ \text{av}_{i,d} \circ \mathbf{g}')$ . But  $\text{av}_{i,d} \circ \mathbf{g}' = \mathbf{g}$ . Side induction step: Assume  $\forall d F(g \circ \text{av}_{j,d})$  (side induction hypothesis). We have to show  $F(g)$ . Assume  $C_m(\mathbf{g})$  where  $g = g_i$ . We have to show  $\mathcal{J}_m(D)(f \circ \mathbf{g})$ . By (7) it suffices to show  $\mathcal{J}_m(D)(f \circ \mathbf{g} \circ \text{av}_{j,d})$  for all  $d$ . Since the  $i$ -th element of  $\mathbf{g} \circ \text{av}_{j,d}$  is  $g \circ \text{av}_{j,d}$  and, by Lemma 2,  $C_m(\mathbf{g} \circ \text{av}_{j,d})$ , we can apply the side induction hypothesis.

The program extracted from Prop. 11 composes trees (unfortunately there is not enough space to show the extracted Haskell code of this and later examples). The special case where  $m = 0$  interprets a tree in  $C_n$  as an  $n$ -ary stream transformer. The special case  $n = m = 1$  was treated in [10], however, without applications to exact real number computation. The program was ‘guessed’ and then verified, whereas we are able to extract the program from a proof making verification unnecessary. Of course, one could reduce Proposition 11 to the case  $m = n = 1$ , by coding  $n$  streams of single digits into one stream of  $n$ -tuples of digits. But this would lead to less efficient programs, since it would mean that in each reading step *all* inputs are read, even those that might not be needed (for example, the function  $f(x, y) = x/2 + y/100$  certainly should read  $x$  more often than  $y$ ).

## 5 Digital Systems

Now we introduce digital systems which are a convenient tool for obtaining implementations of certain families of u.c. functions.

Let  $(A, <)$  be a wellfounded relation. A *digital system* is a family  $\mathcal{F} = (f_\alpha : \mathbb{I}^n \rightarrow \mathbb{I})_{\alpha \in A}$  such that for all  $\alpha \in A$

$$\exists d (f_\alpha[\mathbb{I}^n] \subseteq \mathbb{I}_d \wedge \exists \beta f_\beta = \text{va}_d \circ f_\alpha) \vee \exists i \forall d \exists \beta < \alpha f_\beta = f_\alpha \circ \text{av}_{i,d}$$

When convenient we identify the family  $\mathcal{F}$  with the set  $\{f_\alpha \mid \alpha \in A\}$ .

**Proposition 2.** *If  $\mathcal{F}$  is a digital system, then  $\mathcal{F} \subseteq C_n$ .*

*Proof.* Let  $\mathcal{F}$  be a digital system. We show  $\mathcal{F} \subseteq C_n$  by coinduction. Hence, we have to show  $\mathcal{J}_n(\mathcal{F})(f_\alpha)$  for all  $\alpha \in A$ . But, looking at the definition of  $\mathcal{J}_n(\mathcal{F})$  and the properties of a digital system, this follows immediately by wellfounded  $<$ -induction on  $\alpha$ .

Since wellfounded induction can be realized by a simple recursive procedure we can extract from the proof of Prop. 2 a program that transforms a (realization of) a digital system into a family of trees realizing its members.

**Example (linear affine functions).** For  $\mathbf{u}, v \in \mathbb{Q}^{n+1}$  define  $f_{\mathbf{u},v}: \mathbb{I}^n \rightarrow \mathbb{R}$  by

$$f_{\mathbf{u},v}(\mathbf{x}) := u_1x_1 + \dots + u_nx_n + v$$

Clearly,  $f_{\mathbf{u},v}[\mathbb{I}^n] = [v - |\mathbf{u}|, v + |\mathbf{u}|]$  where  $|\mathbf{u}| := |u_1| + \dots + |u_n|$ . Hence  $f_{\mathbf{u},v}[\mathbb{I}^n] \subseteq \mathbb{I}$  iff  $|\mathbf{u}| + |v| \leq 1$ , and if  $|\mathbf{u}| \leq 1/4$ , then  $f_{\mathbf{u},v}[\mathbb{I}^n] \subseteq \mathbb{I}_d$  for some  $d$ . Furthermore,  $f_{\mathbf{u},v} \circ \text{av}_{i,d} = f_{\mathbf{u}',v'}$  where  $\mathbf{u}'$  is like  $\mathbf{u}$  except that the  $i$ -th component is halved, and  $v' = v + u_i d/2$ . Hence, if  $i$  was chosen such that  $|u_i| \geq |\mathbf{u}|/n$ , then  $|\mathbf{u}'| \leq q|\mathbf{u}|$  where  $q := 1 - 1/(2n) < 1$ . Therefore, we set  $A := \{\mathbf{u}, v \in \mathbb{Q}^{n+1} \mid |\mathbf{u}| + |v| \leq 1\}$  and define a wellfounded relation  $<$  on  $A$  by

$$\mathbf{u}', v' < \mathbf{u}, v \quad :\Leftrightarrow \quad |\mathbf{u}| \geq 1/4 \wedge |\mathbf{u}'| \leq q|\mathbf{u}|$$

From the above it follows that  $\text{Pol}_{1,n} := (f_{\mathbf{u},v})_{\mathbf{u},v \in A}$  is a digital system. Hence  $\text{Pol}_{1,n} \subseteq C_n$ , by Proposition 2. Program extraction gives us a program that assigns to each tuple of rationals  $\mathbf{u}, w \in A$  a digit implementation of  $f_{\mathbf{u},w}$ .

*Remark.* In [22] it has been shown that the linear affine transformations are exactly the functions that can be represented by a finite automaton. The trees computed by our program generate these automata, simply because for the computation of the tree for  $f_{\mathbf{u},v}$  only finitely many other indices  $\mathbf{u}', v'$  are used, and Haskell will construct the tree by connecting these indices by pointers.

**Example (iterated logistic map).** With a similar proof as for the linear affine maps one can show that all polynomials of degree 2 with rational coefficients mapping  $\mathbb{I}$  to  $\mathbb{I}$  are in  $C_1$ . In particular the function logistic map (transformed to  $\mathbb{I}$ ), defined by  $f_a(x) = a(1 - x^2) - 1$  is in  $C_1$  for each rational number  $a \in [0, 2]$ . Exact computation of iterations of the logistic map on  $[0, 1]$  were studied in [23] and [24]. Our extracted programs are able to compute 100 binary digits of  $f_a^{500}(q)$  for arbitrary choices of  $a, q$  in a few minutes. This compares favourably with the experiments in [24] which are based on the binary signed digit representation as well. In addition, when one carries out this computation for a sequence of values  $q$  that are close together, then the memoizing effect of the tree representation kicks in and one observes a speed up of computation of a factor  $\geq 2$  compared to the non-memoized computation.

An important application of digital systems is the following proof that the predicate  $C_n$  precisely captures uniform continuity. We work with the maximum norm on  $\mathbb{I}^n$  and set  $B_\delta(\mathbf{p}) := \{\mathbf{x} \in \mathbb{I}^n \mid |\mathbf{x} - \mathbf{p}| \leq \delta\}$  for  $\mathbf{p} \in \mathbb{I}^n$ . We also set  $Q := \mathbb{I} \cap \mathbb{Q}$  and let  $\delta, \epsilon$  range over positive rational numbers. Furthermore, we set

$$\text{Box}(\delta, \epsilon, f) :\Leftrightarrow \forall \mathbf{p} \in Q^n \exists q \in Q (f[B_\delta(\mathbf{p})] \subseteq B_\epsilon(q))$$

It is easy to see that  $f: \mathbb{I}^n \rightarrow \mathbb{R}$  is uniformly continuous with  $f[\mathbb{I}^n] \subseteq \mathbb{I}$  iff

$$\forall \delta \exists \epsilon \text{Box}(\delta, \epsilon, f) \tag{9}$$

**Proposition 3.** *For any function  $f: \mathbb{I}^n \rightarrow \mathbb{R}$ ,  $C_n(f)$  iff  $f$  is uniformly continuous and  $f[\mathbb{I}^n] \subseteq \mathbb{I}$ .*

*Proof.* We have to show that  $C_n(f)$  holds iff (9) holds.

For the “if” part we use Prop. 2. Let  $A$  be the set of triples  $(f, m, [d_1, \dots, d_k])$  such that  $f$  satisfies (9),  $\text{Box}(2^{-m}, 1/4, f)$  holds, and  $d_1, \dots, d_k \in \text{SD}$  with  $k < n$  (hence in the case  $n = 1$  the list  $[d_1, \dots, d_k]$  is always empty). Define a well-founded relation  $<$  on  $A$  by

$$(f', m', [d'_1, \dots, d'_{k'}]) < (f, m, [d_1, \dots, d_k]) :\Leftrightarrow m' < m \vee (m' = m \wedge k' > k)$$

For  $\mathbf{d} = [d_1, \dots, d_k]$ , where  $k < n$ , set  $\text{av}_{\mathbf{d}} := \text{av}_{1,d_1} \circ \dots \circ \text{av}_{k,d_k}$ , i.p.  $\text{av}_{\square}$  is the identity function. We show that  $\mathcal{F} := (f \circ \text{av}_{\mathbf{d}})_{(f,m,\mathbf{d}) \in A}$  is a digital system (this is sufficient, because  $f \circ \text{av}_{\square} = f$ ).

Let  $\alpha := (f, m, [d_1, \dots, d_k]) \in A$ .

*Case  $m = 0$ , i.e.  $\text{Box}(1, 1/4, f)$ .* We show that the left disjunct in the definition of a digital system holds. We have  $f[\mathbb{I}^n] = f[\text{B}_1(\mathbf{0})] \subseteq \text{B}_{1/4}(q)$  for some  $q \in Q$ . If  $|q| \leq 1/4$ , choose  $d := 0$ , if  $q > 1/4$ , choose  $d := 1$ , if  $q < -1/4$  choose  $d := -1$ . Then clearly  $f[\mathbb{I}^n] \subseteq \mathbb{I}_d$ , and  $g := \text{va}_d \circ f$  is uniformly continuous and maps  $\mathbb{I}^n$  into  $\mathbb{I}$ . Hence  $(g, m', \square) \in A$  for some  $m'$ .

*Case  $m > 0$ .* We show that the right disjunct in the definition of a digital system holds. Choose  $i := k + 1$ . Let  $d \in \text{SD}$ . If  $k + 1 < n$ , then  $\beta := (f, m, [d_1, \dots, d_k, d]) < \alpha$  and  $f \circ \text{av}_{[d_1, \dots, d_k, d]} = (f \circ \text{av}_{[d_1, \dots, d_k]}) \circ \text{av}_{i,d}$ . If  $k + 1 = n$ , then for  $g := f \circ \text{av}_{[d_1, \dots, d_k, d]}$  we have  $\beta := (g, m - 1, \square) \in A$  because  $\text{av}_{[d_1, \dots, d_k, d]}$  is a contraction with contraction factor  $1/2$ . Clearly,  $\beta < \alpha$ . Furthermore,  $g \circ \text{av}_{\square} = g = (f \circ \text{av}_{[d_1, \dots, d_k]}) \circ \text{av}_{i,d}$ .

For the “only if” part we assume  $C_n(f)$ . Set

$$E_k := \{f : \mathbb{I}^n \rightarrow \mathbb{R} \mid \exists \delta \text{Box}(\delta, 2^{-k}, f)\}$$

For proving (9) it obviously suffices to show  $\forall k (f \in E_k)$ . Hence, it suffices to show  $C_n \subseteq E_k$  for all  $k$ . We proceed by induction on  $k$ .

*Base,  $k = 0$ :* Since  $\text{B}_1(0) = \mathbb{I}$ , we clearly have  $\text{Box}(1, 2^0, f)$  for all  $f \in C_n$ .

*Step,  $k \rightarrow k + 1$ :* Since  $C_n = \mathcal{J}_n(C_n)$  it suffices to show  $\mathcal{J}_n(C_n) \subseteq E_{k+1}$ . We prove this by side induction on  $\mathcal{J}_n(C_n)$ , i.e. we show  $\mathcal{K}_n(C_n)(E_{k+1}) \subseteq E_{k+1}$ . *Side induction base:* Assume  $f[\mathbb{I}^n] \subseteq \mathbb{I}_d$  and  $C_n(\text{va}_d \circ f)$ . By the main induction hypothesis,  $\text{Box}(\delta, 2^{-k}, \text{va}_d \circ f)$  for some  $\delta$ . Hence, clearly,  $\text{Box}(\delta, 2^{-(k+1)}, f)$ . *Side induction step:* Assume, as side induction hypothesis,  $\text{Box}(\delta_d, 2^{-(k+1)}, f \circ \text{av}_{i,d})$  for all  $d \in \text{SD}$ . Setting  $\delta = \min\{\delta_d \mid d \in \text{SD}\}$ , we clearly have  $\text{Box}(\delta/2, 2^{-(k+1)}, f)$ .

*Remark.* The proof of the “if” direction computes a tree for every u.c. function, however, usually not a very good one, since if some input needs to be read, then all inputs are read. Hence, for particular families of u.c. functions one should not use this proof, but rather design a special digital system that reads inputs only when necessary (as done in the case of the linear affine functions).

## 6 Conclusion

We presented a method for extracting from coinductive proofs tree-like data structures that code exact lazy algorithms for real functions. The extraction method is based on a variant of modified realizability that strictly separates the (abstract) mathematical model the proof is about from the data types the extracted program is dealing with. The latter are determined solely by the propositional structure of formulas and proofs. This has the advantage that the abstract mathematical structures do not need to be ‘constructivised’. In addition, formulas that do not contain disjunctions are computationally meaningless and can therefore be taken as axioms as long as they are true. This enormously reduces the burden of formalization and turns - in our opinion - program extraction into a realistic method for the development of nontrivial certified algorithms.

*Further work.* Currently, we are adapting the existing implementation of program extraction in the Minlog proof system [25] to our setting. We are also extending this work to more general situations where the interval  $\mathbb{I}$  and the maps  $av_d$  are replaced by an arbitrary bounded metric space with a system of contractions (see [26] for related work), or even to the non-metric case (for example higher types). These extensions will facilitate the extraction of efficient programs for e.g. analytic functions, parametrised integrals, and set-valued functions.

## Acknowledgements

I would like to thank the anonymous referees for spotting several errors in an earlier version of this paper. Their constructive criticism and valuable suggestions were greatly appreciated.

## References

1. Marcial-Romero, J.R., Escardo, M.H.: Semantics of a sequential language for exact real-number computation. *Theor. Comput. Sci.* 379, 120–141 (2007)
2. Geuvers, H., Niqui, M., Spitters, B., Wiedijk, F.: Constructive analysis, types and exact real numbers. *Math. Struct. Comput. Sci.* 17, 3–36 (2007)
3. Edalat, A., Heckmann, R.: Computing with real numbers: I. The LFT approach to real number computation; II. A domain framework for computational geometry. In: Barthe, G., Dybjer, P., Pinto, L., Saraiva, J. (eds.) *APPSEM 2000*. LNCS, vol. 2395, pp. 193–267. Springer, Heidelberg (2002)
4. Ciaffaglione, A., Di Gianantonio, P.: A certified, corecursive implementation of exact real numbers. *Theor. Comput. Sci.* 351, 39–51 (2006)
5. Bertot, Y.: Affine functions and series with co-inductive real numbers. *Math. Struct. Comput. Sci.* 17, 37–63 (2007)
6. Berger, U., Hou, T.: Coinduction for exact real number computation. *Theory Comput. Sys.* (to appear, 2009)
7. Niqui, M.: Coinductive formal reasoning in exact real arithmetic. *Logical Methods in Computer Science* 4, 1–40 (2008)



8. Schwichtenberg, H.: Realizability interpretation of proofs in constructive analysis. *Theory Comput. Sys.* (to appear, 2009)
9. Berger, U., Seisenberger, M.: Applications of inductive definitions and choice principles to program synthesis. In: Crosilla, L., Schuster, P. (eds.) *From Sets and Types to Topology and Analysis Towards practicable foundations for constructive mathematics*. *Oxford Logic Guides*, vol. 48, pp. 137–148. Oxford University Press, Oxford (2005)
10. Ghani, N., Hancock, P., Pattinson, D.: Continuous functions on final coalgebras. *Electr. Notes in Theoret. Comp. Sci.* 164 (2006)
11. Buchholz, W., Feferman, F., Pohlers, W., Sieg, W.: *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*. *Lecture Notes in Mathematics*, vol. 897. Springer, Berlin (1981)
12. Bradfield, J., Stirling, C.: Modal  $\mu$ -calculi. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) *Handbook of Modal Logic. Studies in Logic and Practical Reasoning*, vol. 3, pp. 721–756. Elsevier, Amsterdam (2007)
13. Gierz, G., Hofmann, K., Keimel, K., Lawson, J., Mislove, M., Scott, D.: *Continuous Lattices and Domains*. *Encyclopedia of Mathematics and its Applications*, vol. 93. Cambridge University Press, Cambridge (2003)
14. Kreisel, G.: Interpretation of analysis by means of constructive functionals of finite types. *Constructivity in Mathematics*, 101–128 (1959)
15. Tatsuta, M.: Realizability of monotone coinductive definitions and its application to program synthesis. In: Jeuring, J. (ed.) *MPC 1998. Lecture Notes in Mathematics*, vol. 1422, pp. 338–364. Springer, Heidelberg (1998)
16. Troelstra, A.: *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. *Lecture Notes in Mathematics*, vol. 344. Springer, Heidelberg (1973)
17. Abramsky, S., Jung, A.: Domain theory. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S.E. (eds.) *Handb. Logic Comput. Sci.*, vol. 3, pp. 1–168. Clarendon Press, Oxford (1994)
18. Malcolm, G.: Data structures and program transformation. *Science of Computer Programming* 14, 255–279 (1990)
19. Hancock, P., Setzer, A.: Guarded induction and weakly final coalgebras in dependent type theory. In: Crosilla, L., Schuster, P. (eds.) *From Sets and Types to Topology and Analysis. Towards Practicable Foundations for Constructive Mathematics*, pp. 115–134. Clarendon Press, Oxford (2005)
20. Abel, A., Matthes, R., Uustalu, T.: Iteration and coiteration schemes for higher-order and nested datatypes. *Theor. Comput. Sci.* 333, 3–66 (2005)
21. Capretta, V., Uustalu, T., Vene, V.: Recursive coalgebras from comonads. *Information and Computation* 204, 437–468 (2006)
22. Konečný, M.: Real functions incrementally computable by finite automata. *Theor. Comput. Sci.* 315, 109–133 (2004)
23. Blanck, J.: Efficient exact computation of iterated maps. *Journal of Logic and Algebraic Programming* 64, 41–59 (2005)
24. Plume, D.: *A Calculator for Exact Real Number Computation*. PhD thesis. University of Edinburgh (1998)
25. Benl, H., Berger, U., Schwichtenberg, H., Seisenberger, M., Zuber, W.: Proof theory at work: Program development in the Minlog system. In: Bibel, W., Schmitt, P. (eds.) *Automated Deduction – A Basis for Applications*. *Applied Logic Series*, vol. II, pp. 41–71. Kluwer, Dordrecht (1998)
26. Scriven, A.: A functional algorithm for exact real integration with invariant measures. *Electron. Notes Theor. Comput. Sci.* 218, 337–353 (2008)

# On the Relation between Sized-Types Based Termination and Semantic Labelling

Frédéric Blanqui<sup>1</sup> and Cody Roux<sup>2</sup>

<sup>1</sup> FIT 3-604, Tsinghua University, Haidian District, Beijing 100084, China  
`frederic.blanqui@inria.fr`

<sup>2</sup> LORIA\*, Pareo team, Campus Scientifique, BP 239, 54506 Vandoeuvre-lès-Nancy,  
Cedex, France  
`cody.roux@loria.fr`

**Abstract.** We investigate the relationship between two independently developed termination techniques. On the one hand, sized-types based termination (SBT) uses types annotated with size expressions and Girard’s reducibility candidates, and applies on systems using constructor matching only. On the other hand, semantic labelling transforms a rewrite system by annotating each function symbol with the semantics of its arguments, and applies to any rewrite system.

First, we introduce a simplified version of SBT for the simply-typed lambda-calculus. Then, we give new proofs of the correctness of SBT using semantic labelling, both in the first and in the higher-order case. As a consequence, we show that SBT can be extended to systems using matching on defined symbols (*e.g.* associative functions).

## 1 Introduction

Sized types were independently introduced by Hughes, Pareto and Sabry [16] and Giménez [11], and were extended to richer type systems, to rewriting and to richer size annotations by various researchers [21,11,2,5,7].

Sized types are types annotated with size expressions. For instance, if  $T$  is the type of binary trees then, for each  $a \in \mathbb{N}$ , a type  $T^a$  is introduced to type the trees of height smaller or equal to  $a$ . In the general case, the size is some ordinal related to the interpretation of types in Girard’s reducibility candidates [12]. However, as suggested in [5], other notions of sizes may be interesting.

These size annotations can then be used to prove the termination of functions by checking that the size of arguments decreases along recursive calls, but this applies to functions defined by using matching on constructor terms only.

At about the same time, semantic labelling was introduced for first-order systems by Zantema [22]. It received a lot of attention in the last years and was recently extended to the higher-order case by Hamana [13].

In contrast with SBT, semantic labelling is not a termination criterion but transforms a system into another one whose termination is equivalent and hopefully simpler to prove. The transformation consists in annotating function symbols with the semantics of their arguments in some model of the rewrite system.

---

\* UMR 7503 CNRS-INPL-INRIA-Nancy2-UHP.

Finding a model may of course be difficult. We will see that the notion of size used in SBT provides such a model.

In this paper, we study the relationship between these two methods. In particular, we give a new proof of the correctness of SBT using semantic labelling. This will enable us to extend SBT to systems using matching on defined symbols.

**Outline.** Section 2 introduces our notations. Section 3 explains what SBT is and Section 4 introduces a simplified version of it. To ease the understanding of the paper, we first present the first-order case which already contains the main ideas, and then consider the higher-order case which requires more knowledge. Hence, in Section 5 (resp. 7), we recall what is semantic labelling in the first (resp. higher) order case and show in Section 6 (resp. 8) that SBT is an instance of it. For lack of space, some proofs are given in the Appendices of 8.

## 2 Preliminaries

**First-order terms.** A *signature*  $\mathcal{F}$  is made of a set  $\mathcal{F}_n$  of *function symbols* of *arity*  $n \in \mathbb{N}$ . Let  $\mathcal{F}$  be the set of all function symbols. Given a set  $\mathcal{X}$  of *variables*, the set of *first-order terms*  $\mathcal{T}(\mathcal{F}, \mathcal{X})$  is defined as usual:  $\mathcal{X} \subseteq \mathcal{T}$ ; if  $f \in \mathcal{F}_n$  and  $\mathbf{t}$  is a sequence  $t_1, \dots, t_n \in \mathcal{T}$  of length  $n = |\mathbf{t}|$ , then  $f(\mathbf{t}) \in \mathcal{T}$ .

An  $\mathcal{F}$ -*algebra*  $\mathcal{M}$  is given by a set  $M$  and, for each symbol  $f \in \mathcal{F}_n$ , a function  $f^{\mathcal{M}} : M^n \rightarrow M$ . Given a valuation  $\mu : \mathcal{X} \rightarrow M$ , the interpretation of a term  $t$  is defined as follows:  $\llbracket x \rrbracket \mu = \mu(x)$  and  $\llbracket f(t_1, \dots, t_n) \rrbracket \mu = f^{\mathcal{M}}(\llbracket t_1 \rrbracket \mu, \dots, \llbracket t_n \rrbracket \mu)$ .

*Positions* are *words* on  $\mathbb{N}$ . We denote by  $\varepsilon$  the *empty* word and by  $p \cdot q$  or  $pq$  the concatenation of  $p$  and  $q$ . Given a term  $t$ , we denote by  $t|_p$  the subterm of  $t$  at position  $p$ , and by  $t[u]_p$  the replacement of this subterm by  $u$ . Let  $\text{Pos}(f, t)$  be the set of the positions of the occurrences of  $f$  in  $t$ .

**Higher-order terms.** The set of (simple) *types* is  $\mathbb{T} = \mathcal{T}(\Sigma)$  where  $\Sigma_0 = \mathcal{B}$  is a set of *base types*,  $\Sigma_2 = \{\Rightarrow\}$  and  $\Sigma_n = \emptyset$  otherwise. The sets of *positive* and *negative positions in a type* are inductively defined as follows:

- $\text{Pos}^+(\mathbf{B}) = \varepsilon$  and  $\text{Pos}^-(\mathbf{B}) = \emptyset$  for each  $\mathbf{B} \in \mathcal{B}$ ,
- $\text{Pos}^\delta(T \Rightarrow U) = 1 \cdot \text{Pos}^{-\delta}(T) \cup 2 \cdot \text{Pos}^\delta(U)$  where  $-- = +$  and  $-+ = -$ .

Let  $\mathcal{X}$  be an infinite set of variables. A typing environment  $\Gamma$  is a map from a finite subset of  $\mathcal{X}$  to  $\mathbb{T}$ . For each type  $T$ , we assume given a set  $\mathcal{F}_T$  of *function symbols of type*  $T$ . The sets  $\Lambda_T(\Gamma)$  of *terms of type*  $T$  in  $\Gamma$  are defined as usual:  $\mathcal{F}_T \subseteq \Lambda_T(\Gamma)$ ; if  $(x, T) \in \Gamma$  then  $x \in \Lambda_T(\Gamma)$ ; if  $t \in \Lambda_U(\Gamma, x : T)$ , then  $\lambda x^T t \in \Lambda_{T \Rightarrow U}(\Gamma)$ ; if  $t \in \Lambda_{U \Rightarrow V}(\Gamma)$  and  $u \in \Lambda_U(\Gamma)$ , then  $tu \in \Lambda_V(\Gamma)$ .

Let  $\mathcal{F}$  (resp.  $\Lambda$ ) be the set of all function symbols (resp. terms). Let  $\mathcal{X}(t)$  be the set of *free variables* of  $t$ . A *substitution*  $\sigma$  is a map from a finite subset of  $\mathcal{X}$  to  $\Lambda$ . We denote by  $\left(\frac{u}{x}\right)$  the substitution mapping  $x$  to  $u$ , and by  $t\sigma$  the application of  $\sigma$  to  $t$ . A term  $t$   $\beta$ -*rewrites* to a term  $u$ , written  $t \rightarrow_\beta u$ , if there is  $p \in \text{Pos}(t)$  such that  $t|_p = (\lambda x^T v)w$  and  $u = t\left[\frac{v^w}{x}\right]_p$ .

A *rewrite rule* is a pair of terms  $l \rightarrow r$  of the same type such that  $\mathcal{X}(r) \subseteq \mathcal{X}(l)$ . A *rewrite system* is a set  $\mathcal{R}$  of rewrite rules. A term  $t$  *rewrites* to a term  $u$ , written  $t \rightarrow_{\mathcal{R}} u$ , if there is  $p \in \text{Pos}(t)$ ,  $l \rightarrow r \in \mathcal{R}$  and  $\sigma$  such that  $t|_p = l\sigma$  and  $u = t[r\sigma]_p$ .

**Constructor systems.** A function symbol  $f$  is either a *constructor symbol* if no rule left-hand side is headed by  $f$ , or a *defined symbol* otherwise. A *pattern* is a variable or a term of the form  $ct$  with  $c$  a constructor symbol and  $t$  patterns. A rewrite system is *constructor* if every rule is of the form  $fl \rightarrow r$  with  $l$  patterns.

As usual, we assume that constructors form a valid inductive structure [6], that is, there is a well-founded quasi-ordering  $\leq_{\mathcal{B}}$  on  $\mathcal{B}$  such that, for each base type  $B$ , constructor  $c : T \Rightarrow B$  and base type  $C$  occurring at position  $p$  in  $T_i$ , either  $C <_{\mathcal{B}} B$  or  $C \simeq_{\mathcal{B}} B$  and  $p \in \text{Pos}^+(T_i)$ . Mendler indeed showed that invalid inductive structures lead to non-termination [18].

Given a constructor  $c : T \Rightarrow B$ , let  $\text{Ind}(c)$  be the set of integers  $i$  such that  $T_i$  contains a base type  $C \simeq_{\mathcal{B}} B$ . A constructor  $c$  with  $\text{Ind}(c) \neq \emptyset$  is said *recursive*.

A constructor  $c : T \Rightarrow B$  is *strictly-positive* if, for each  $i$ , either no base type equivalent to  $B$  occurs in  $T_i$ , or  $T_i$  is of the form  $U \Rightarrow C$  with  $C \simeq_{\mathcal{B}} B$  and no base type equivalent to  $B$  occurring in  $U$ .

SBT applies to constructor systems only. By using semantic labelling, we will prove that it can also be applied to some non-constructor systems.

### 3 Sized-Types Based Termination

We now present a simplified version of the termination criterion introduced in [5], where the first author considers rewrite systems on terms of the Calculus of Algebraic Constructions, a complex type system with polymorphic and dependent types. Here, we restrict our attention to simply-typed  $\lambda$ -terms since there is no extension of semantic labelling to polymorphic and dependent types yet.

This termination criterion is based on the semantics of types in reducibility candidates [12]. An arrow type  $T \Rightarrow U$  is interpreted by the set  $\llbracket T \Rightarrow U \rrbracket = \{v \in \mathcal{T} \mid \forall t \in \llbracket T \rrbracket, vt \in \llbracket U \rrbracket\}$ . A base type  $B$  is interpreted by the fixpoint  $\llbracket B \rrbracket$  of the monotonic function  $F_B(X) = \{v \in \mathcal{SN} \mid \forall \text{ constructor } c : T \Rightarrow B, \forall t, \forall i \in \text{Ind}(c), v \rightarrow^* ct \Rightarrow t_i \in \llbracket T_i \rrbracket_{B \rightarrow X}\}$  on the lattice of reducibility candidates that is complete for set inclusion [6]. This fixpoint, defined by induction on the well-founded quasi-ordering  $\leq_{\mathcal{B}}$  on base types, can be reached by transfinite iteration of  $F_B$  up to some limit ordinal  $\omega_B$  strictly smaller than the first uncountable ordinal  $\aleph$ . This provides us with the following notion of size: the size of a term  $t \in \llbracket B \rrbracket$  is the smallest ordinal  $\text{ob}_B(t) = \alpha < \aleph$  such that  $t \in F_B^\alpha(\perp)$ , where  $\perp$  is the smallest element of the lattice and  $F_B^\alpha$  is the function obtained after  $\alpha$  transfinite iterations of  $F_B$ .

This notion of size, which corresponds to the tree height for patterns, has the following properties: it is well-founded; the size of a pattern is strictly bigger than the size of its subterms; if  $t \rightarrow t'$  then the size of  $t'$  is smaller than (since  $\rightarrow$  may be non confluent) or equal to the size of  $t$ .

SBT consists then in providing a way to syntactically represent the sizes of terms and, given for each function symbol an annotation describing how the size of its output is related to the sizes of its inputs, check that some measure on the sizes of its arguments decreases in each recursive call.

**Size algebra.** Sizes are represented and compared by using a first-order term algebra  $\mathcal{A} = \mathcal{T}(\Sigma, \mathcal{X})$  equipped with an ordering  $\leq_{\mathcal{A}}$  such that:

- $<_{\mathcal{A}}$  is stable by substitution;
- $(\mathfrak{A}, <_{\mathfrak{A}})$ , where  $<_{\mathfrak{A}}$  is the usual ordering on ordinals, is a model of  $(\mathcal{A}, <_{\mathcal{A}})$ :
  - every symbol  $h \in \Sigma_n$  is interpreted by a function  $h^{\mathfrak{A}} : \mathfrak{A}^n \rightarrow \mathfrak{A}$ ;
  - if  $a <_{\mathcal{A}} b$  then  $\llbracket a \rrbracket \mu <_{\mathfrak{A}} \llbracket b \rrbracket \mu$  for each  $\mu : \mathcal{X} \rightarrow \mathfrak{A}$ .

To denote a size that cannot be expressed in  $\mathcal{A}$  (or a size that we do not care about),  $\Sigma$  is extended with a (biggest) nullary element  $\infty$ . Let  $\overline{\mathcal{A}}$  be the extended term algebra in which all terms containing  $\infty$  are identified,  $<_{\overline{\mathcal{A}}} = <_{\mathcal{A}} \cup \{(a, \infty) \mid a \in \mathcal{A}\}$  and  $\leq_{\overline{\mathcal{A}}} = \leq_{\mathcal{A}} \cup \{(a, \infty) \mid a \in \overline{\mathcal{A}}\}$ . Note that such an extension is often used in domain theory but with a least element instead.

**Annotated types.** The set of base types is now all the expressions  $B^a$  such that  $B \in \mathcal{B}$  and  $a \in \overline{\mathcal{A}}$ . The interpretation of  $B^\infty$  (also written  $B$ ) is  $\llbracket B \rrbracket$  and, given  $a \in \mathcal{A}$ , the interpretation of  $B^a$  wrt a size valuation  $\mu : \mathcal{X} \rightarrow \mathfrak{A}$  is the set of terms in  $\llbracket B \rrbracket$  whose size is smaller or equal to  $\llbracket a \rrbracket \mu$ :  $\llbracket B^a \rrbracket \mu = F_B^{\llbracket a \rrbracket \mu}(\perp)$ .

Hence, we assume that every symbol  $f \in \mathcal{F}$  is given an annotated type  $\tau_f^A$  whose size variables, like type variables in ML, are implicitly universally quantified and can be instantiated by any size expression. Hence the typing rule for symbols in Figure 1 allows any size substitution  $\varphi$  to be applied to  $\tau_f^A$ . Subtyping naturally follows from the interpretation of types and the ordering on  $\mathcal{A}$ .

$$\begin{array}{c}
 \frac{\varphi : \mathcal{X} \rightarrow \mathcal{A}}{\Gamma \vdash^s f : \tau_f^A \varphi} \quad \frac{(x, T) \in \Gamma}{\Gamma \vdash^s x : T} \quad \frac{\Gamma, x : T \vdash^s u : U \quad x \notin \Gamma}{\Gamma \vdash^s \lambda x^T u : T \Rightarrow U} \\
 \\
 \frac{\Gamma \vdash^s t : U \Rightarrow V \quad \Gamma \vdash^s u : U}{\Gamma \vdash^s tu : V} \quad \frac{\Gamma \vdash^s t : T \quad T \leq T'}{\Gamma \vdash^s t : T'} \\
 \\
 \frac{a \leq_{\overline{\mathcal{A}}} b}{B^a \leq B^b} \quad \frac{T' \leq T \quad U \leq U'}{T \Rightarrow U \leq T' \Rightarrow U'} \quad \frac{T \leq U \quad U \leq V}{T \leq V}
 \end{array}$$

**Fig. 1.** Type system with size annotations

**Definition 1.** Given a type  $T$ , let  $T^\infty$  be the type obtained by annotating every base type with  $\infty$ , and  $\text{annot}_{\mathcal{B}}^\alpha(T)$  be the type obtained by annotating every base type  $C \simeq_{\mathcal{B}} B$  with  $\alpha$ , and every base type  $C \not\simeq_{\mathcal{B}} B$  with  $\infty$ . Conversely, given an annotated type  $T$ , let  $|T|$  be the type obtained by removing all annotations.

Note that, in contrast to types, terms are unchanged: in  $\lambda x^T u$ ,  $T = T^\infty$ .

Given a size symbol  $h \in \Sigma$ , let  $\text{Mon}^+(h)$  (resp.  $\text{Mon}^-(h)$ ) be the sets of integers  $i$  such that  $h$  is monotonic (resp. anti-monotonic) in its  $i$ -th argument. The sets of *positive* and *negative* positions in an annotated type are:

- $\text{Pos}^-(B^a) = 0 \cdot \text{Pos}^-(a)$  and  $\text{Pos}^+(B^a) = \{\varepsilon\} \cup 0 \cdot \text{Pos}^+(a)$ ,
- $\text{Pos}^-(\alpha) = \emptyset$ ,  $\text{Pos}^+(\alpha) = \varepsilon$ ,  $\text{Pos}^\delta(h(\mathbf{a})) = \bigcup \{i \cdot \text{Pos}^{\varepsilon^\delta}(a_i) \mid i \in \text{Mon}^\varepsilon(h), \varepsilon \in \{-, +\}\}$ .

To ease the expression of termination conditions, for every defined symbol  $f$ ,  $\tau_f^A$  is assumed to be of the form  $\mathbf{P} \Rightarrow \mathbf{B}^{\alpha_f} \Rightarrow \mathbf{B}^{f^A(\alpha_f)}$  with  $|\tau_f^A| = \tau_f$ ,  $\mathcal{X}(\mathbf{P}) = \emptyset$  and  $\mathcal{X}(f^A(\alpha_f)) \subseteq \{\alpha_f\}$  where  $\alpha_f$  are pairwise distinct variables. The arguments of type  $\mathbf{B}$  are the ones whose size will be taken into account for proving termination. The arguments of type  $\mathbf{P}$  are parameters and every rule defining  $f$  must be of the form  $f\mathbf{pl} \rightarrow r$  with  $\mathbf{p} \in \mathcal{X}$ ,  $|\mathbf{p}| = |\mathbf{P}|$  and  $|\mathbf{l}| = |\mathbf{B}|$ .

Moreover, the annotated type of a constructor  $c : T_1 \dots T_n \Rightarrow \mathbf{B}$  is:

$$\tau_c^A = \text{annot}_{\mathbf{B}}^\alpha(T_1) \Rightarrow \dots \Rightarrow \text{annot}_{\mathbf{B}}^\alpha(T_n) \Rightarrow \mathbf{B}^{c^A(\alpha)}$$

with  $c^A(\alpha) = \infty$  if  $c$  is non-recursive, and  $c^A(\alpha) = s(\alpha)$  otherwise, where  $s$  is a monotonic unary symbol interpreted as the ordinal successor and such that  $a <_{\mathcal{A}} s(a)$  for each  $a$ .

**Termination criterion.** We assume given a well-founded quasi-ordering  $\geq_{\mathcal{F}}$  on  $\mathcal{F}$  and, for each function symbol  $f : {}^s \mathbf{T} \Rightarrow \mathbf{B}^{\alpha_f} \Rightarrow \mathbf{B}^{f^A(\alpha_f)}$  and set  $X \in \{\mathcal{A}, \mathfrak{A}\}$ , an ordered domain  $(D_f^X, <_f^X)$  and a function  $\zeta_f^X : X^{|\alpha_f|} \rightarrow D_f^X$  compatible with  $\simeq_{\mathcal{F}}$  (i.e.  $|\alpha_f| = |\alpha_g|$ ,  $D_f^X = D_g^X$ ,  $<_f^X = <_g^X$  and  $\zeta_f^X = \zeta_g^X$  whenever  $f \simeq_{\mathcal{F}} g$ ) and such that  $>_f^{\mathfrak{A}}$  is well-founded and  $\zeta_f^{\mathfrak{A}}(\llbracket \mathbf{a} \rrbracket \mu) <_f^{\mathfrak{A}} \zeta_f^{\mathfrak{A}}(\llbracket \mathbf{b} \rrbracket \mu)$  whenever  $\zeta_f^A(\mathbf{a}) <_f^A \zeta_f^A(\mathbf{b})$  and  $\mu : \mathcal{X} \rightarrow \mathfrak{A}$ .

Usual domains are  $\mathfrak{A}^n$  ordered lexicographically, or the multisets on  $\mathfrak{A}$  ordered with the multiset extension of  $>_{\mathfrak{A}}$ .

**Theorem 1 ([5]).** *Let  $\mathcal{R}$  be a constructor system. The relation  $\rightarrow_{\beta} \cup \rightarrow_{\mathcal{R}}$  terminates if, for each defined  $f : {}^s \mathbf{P} \Rightarrow \mathbf{B}^{\alpha} \Rightarrow \mathbf{B}^{f^A(\alpha)}$  and rule  $f\mathbf{pl} \rightarrow r \in \mathcal{R}$ , there is an environment  $\Gamma$  and a size substitution  $\binom{\mathbf{a}}{\alpha}$  such that:*

- *pattern condition: for each  $\theta$ , if  $\mathbf{p}\theta \in \llbracket \mathbf{P} \rrbracket$  and  $\mathbf{l}\theta \in \llbracket \mathbf{B} \rrbracket$  then there is  $\nu$  such that, for each  $(x, T) \in \Gamma$ ,  $x\theta \in \llbracket T \rrbracket^\nu$  and  $\llbracket \mathbf{a} \rrbracket \nu \leq \text{ob}(\mathbf{l}\theta)$ ;*
- *argument decreasingness:  $\Gamma \vdash_{\mathbf{f}\mathbf{a}}^s r : \mathbf{B}^{f^A(\mathbf{a})}$  where  $\vdash_{\mathbf{f}\mathbf{a}}$  is defined in Figure 2;*
- *size annotations monotonicity:  $\text{Pos}(\alpha, f^A(\alpha)) \subseteq \text{Pos}^+(f^A(\alpha))$ .*

The termination criterion introduced in [5] is not expressed exactly like this. The pattern condition is replaced by syntactic conditions implying the pattern condition, but the termination proof is explicitly based on the pattern condition. This condition means that  $\mathbf{a}$  is a valid representation of the size of  $\mathbf{l}$ , whatever the instantiation of the variables of  $\mathbf{l}$  is, and thus that any recursive call with arguments of size smaller than  $\mathbf{a}$  is admissible. The existence of such a valid syntactic representation depends on  $\mathbf{l}$  and the size annotations of constructors. With the chosen annotations, the condition is not satisfied by some patterns (whose type admits elements of size bigger than  $\omega$ , Appendix A). This suggests to use a more precise annotation for constructors.

The expressive power of the criterion depends on  $\mathcal{A}$ . Taking the size algebra  $\mathcal{A}$  reduced to the successor symbol  $s$  (the decidability of which is proved in [3]) is sufficient to handle every primitive recursive function. As an example, consider the recursor  $\text{rec}_T : \mathbf{O} \Rightarrow T \Rightarrow (\mathbf{O} \Rightarrow T) \Rightarrow ((\mathbf{N} \Rightarrow \mathbf{O}) \Rightarrow (\mathbf{N} \Rightarrow T) \Rightarrow T) \Rightarrow T$  on the type  $\mathbf{O}$  of Brouwer's ordinals whose constructors are  $0 : \mathbf{O}$ ,  $s : \mathbf{O}^\alpha \Rightarrow \mathbf{O}^{s\alpha}$

$\frac{\mathbf{g} <_{\mathcal{F}} \mathbf{f}, \psi : \mathcal{X} \rightarrow \mathcal{A}}{\Gamma \vdash_{\mathbf{f}\mathbf{a}}^s \mathbf{g} : \tau_{\mathbf{g}}^{\mathcal{A}} \psi}$  + variable, abstraction, application and subtyping rules of Fig. [□](#)

$$\frac{\mathbf{g} \simeq_{\mathcal{F}} \mathbf{f} \quad \mathbf{g} :^s \mathbf{U} \Rightarrow \mathbf{C}^{\beta} \Rightarrow \mathbf{C}^{\mathbf{g}^{\mathcal{A}}(\beta)} \quad \Gamma \vdash_{\mathbf{f}\mathbf{a}}^s \mathbf{u} : \mathbf{U} \quad \Gamma \vdash_{\mathbf{f}\mathbf{a}}^s \mathbf{m} : \mathbf{B}^b \quad \zeta_{\mathbf{f}}^{\mathcal{A}}(\mathbf{b}) <_{\mathbf{f}}^{\mathcal{A}} \zeta_{\mathbf{f}}^{\mathcal{A}}(\mathbf{a})}{\Gamma \vdash_{\mathbf{f}\mathbf{a}}^s \mathbf{g}\mathbf{u}\mathbf{m} : \mathbf{C}^{\mathbf{g}^{\mathcal{A}}(\mathbf{b})}}$$

**Fig. 2.** Computability closure

and  $\text{lim} : (\mathbf{N} \Rightarrow \mathbf{O}^{\alpha}) \Rightarrow \mathbf{O}^{\mathbf{s}\alpha}$ , where  $\mathbf{N}$  is the type of natural numbers whose constructors are  $\mathbf{0} : \mathbf{N}$  and  $\mathbf{s} : \mathbf{N}^{\alpha} \Rightarrow \mathbf{N}^{\mathbf{s}\alpha}$ :

$$\begin{aligned} \text{rec}\mathbf{0}uvw &\rightarrow u \\ \text{rec}(\mathbf{s}x)uvw &\rightarrow vx(\text{rec}xuvw) \\ \text{rec}(\text{lim}f)uvw &\rightarrow wf(\lambda n \text{rec}(fn)uvw) \end{aligned}$$

For instance, with  $f : \mathbf{N} \Rightarrow \mathbf{O}^{\alpha}$ , we have  $\text{lim}f : \mathbf{O}^{\mathbf{s}\alpha}$ ,  $fn : \mathbf{O}^{\alpha}$  and  $\mathbf{s}\alpha >_{\mathcal{A}} \alpha$ .

An example of non-simply terminating system satisfying the criterion is the following system defining a division function  $/ : \mathbf{N}^{\alpha} \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}^{\alpha}$  by using a subtraction function  $- : \mathbf{N}^{\alpha} \Rightarrow \mathbf{N} \Rightarrow \mathbf{N}^{\alpha}$ .

$$\begin{aligned} -x\mathbf{0} &\rightarrow x & /0x &\rightarrow \mathbf{0} \\ -0x &\rightarrow \mathbf{0} & /(sx)y &\rightarrow \mathbf{s}(/(-xy)y) \\ -(sx)(sy) &\rightarrow -xy \end{aligned}$$

Indeed, with  $x : \mathbf{N}^x$ , we have  $sx : \mathbf{N}^{\mathbf{s}x}$ ,  $-xy : \mathbf{N}^x$  and  $sx >_{\mathcal{A}} x$ .

## 4 Annotating Constructor Types with a max Symbol

In this section, we simplify the previous termination criterion by annotating constructor types in an algebra made of the following symbols:

- $\mathbf{0} \in \Sigma_0$  interpreted as the ordinal 0;
- $\mathbf{s} \in \Sigma_1$  interpreted as the successor ordinal;
- $\mathbf{max} \in \Sigma_2$  interpreted as the max on ordinals.

For the annotated type of a constructor  $c : T_1 \dots T_n \Rightarrow \mathbf{B}$ , we now take:

$$\tau_c^{\mathcal{A}} = \text{annot}_{\mathbf{B}}^{\alpha_1}(T_1) \Rightarrow \dots \Rightarrow \text{annot}_{\mathbf{B}}^{\alpha_n}(T_n) \Rightarrow \mathbf{B}^{\mathbf{c}_{\mathcal{A}}(\alpha_1, \dots, \alpha_n)}$$

with  $\alpha$  distinct variables,  $\mathbf{c}_{\mathcal{A}}(\alpha) = \mathbf{0}$  if  $c$  is non-recursive, and  $\mathbf{c}_{\mathcal{A}}(\alpha) = \mathbf{s}(\mathbf{max}(\alpha_i \mid i \in \text{Ind}(c)))$  otherwise, where  $\mathbf{max}(\alpha_1, \dots, \alpha_{k+1}) = \mathbf{max}(\alpha_1, \mathbf{max}(\alpha_2, \dots, \alpha_{k+1}))$  and  $\mathbf{max}(\alpha_1) = \alpha_1$ .

This does not affect the correctness of Theorem [□](#) since, in this case too, one can prove that constructors are computable:  $c \in \llbracket \tau_c^{\mathcal{A}} \rrbracket^{\mu}$  for each  $\mu$ .

Moreover, now, both constructors and defined symbols have a type of the form  $\text{annot}_{\mathbf{B}_1}^{\alpha_1}(T_1) \Rightarrow \dots \Rightarrow \text{annot}_{\mathbf{B}_n}^{\alpha_n}(T_n) \Rightarrow \mathbf{B}^{\mathbf{f}^{\mathcal{A}}(\alpha)}$  with  $\alpha$  distinct variables.

This means that a constructor can be applied to any sequence of arguments without having to use subtyping. Indeed, previously, not all constructor applications were possible (take  $cxxy$  with  $c : \mathbf{B}^\alpha \Rightarrow \mathbf{B}^\alpha \Rightarrow \mathbf{b}^{s\alpha}$ ,  $x : \mathbf{B}^x$  and  $y : \mathbf{B}^y$ ) and some constructor applications required subtyping (take  $cx(dx)$  with  $c : \mathbf{B}^\alpha \Rightarrow \mathbf{B}^\alpha \Rightarrow \mathbf{b}^{s\alpha}$ ,  $d : \mathbf{B}^\alpha \Rightarrow \mathbf{B}^{s\alpha}$  and  $x : \mathbf{B}^x$ ).

We can therefore postpone subtyping after typing without losing much expressive power. It follows that every term has a most general type given by a simplified version of the type inference system  $\vdash^i$  of [3] using unification only (see Appendix B).

Moreover, the pattern and monotonicity conditions can always be satisfied by defining, for each symbol  $f :^s \mathbf{P} \Rightarrow \mathbf{B}^\alpha \Rightarrow \mathbf{U}$  and rule  $\mathbf{fpl} \rightarrow r \in \mathcal{R}$ ,  $\mathbf{a}$  as  $\sigma(\mathbf{l})$  where  $\sigma(x) = x$  and  $\sigma(\mathbf{ct}) = \mathbf{c}^A(\sigma(\mathbf{t}))$ , and  $\Gamma$  as the set of pairs  $(x, T)$  such that  $x \in \mathcal{X}(\mathbf{fpl})$  and  $T$  is:

- $P_i$  if  $x = p_i$ ,
- $\mathbf{B}_i^x$  if  $x = l_i$ ,
- $\text{annot}_{\mathbf{B}_i^x}^x(T)$  if  $\mathbf{c}uxv$  is a subterm of  $l_i$  and  $\mathbf{c} : \mathbf{U} \Rightarrow \mathbf{T} \Rightarrow \mathbf{V} \Rightarrow \mathbf{C}$ .

Note that, if  $\Gamma \vdash t : T$  and  $t$  is a non-variable pattern then there is a base type  $\mathbf{B}$  such that  $\Gamma \vdash^i t : \mathbf{B}^{\sigma(t)}$ . So,  $\sigma(t)$  is the most general size of  $t$ .

**Theorem 2.** *Let  $\mathcal{R}$  be a constructor system. The relation  $\rightarrow_\beta \cup \rightarrow_{\mathcal{R}}$  terminates if, for each  $f :^s \mathbf{P} \Rightarrow \mathbf{B}^\alpha \Rightarrow \mathbf{B}^{\mathbf{f}^A(\alpha)}$  and rule  $\mathbf{fpl} \rightarrow r \in \mathcal{R}$ , we have:*

- *argument decreasingness:  $\Gamma \vdash_{\mathbf{f}\mathbf{a}}^i r : \mathbf{B}^a$  and  $a \leq_{\overline{\mathbf{A}}} \mathbf{f}^A(\mathbf{a})$  where  $\Gamma$  and  $\mathbf{a} = \sigma(\mathbf{l})$  are defined just before and  $\vdash_{\mathbf{f}\mathbf{a}}^i$  is the type inference system  $\vdash^i$  [3] (see Appendix B) with function applications restricted as in Figure 2.*

The proof is given in Appendix C. In the following, we say that  $\mathcal{R}$  *SB-terminates* if  $\mathcal{R}$  satisfies the conditions of Theorem 2.

## 5 First-Order Semantic Labelling

Semantic labelling is a transformation technique introduced by Hans Zantema for proving the termination of first-order rewrite systems [22]. It consists in labelling function symbols by using some model of the rewrite system.

Let  $\mathcal{F}$  be a first-order signature and  $\mathcal{M}$  be an  $\mathcal{F}$ -algebra equipped with a partial order  $\leq_{\mathcal{M}}$ . For each  $f \in \mathcal{F}_n$ , we assume given a non-empty poset  $(S^f, \leq_f)$  and a *labelling function*  $\pi_f : M^n \rightarrow S^f$ . Then, let  $\overline{\mathcal{F}}$  be the signature such that  $\overline{\mathcal{F}}_n = \{f_a \mid f \in \mathcal{F}_n, a \in S^f\}$ .

The *labelling* of a term wrt a valuation  $\mu : \mathcal{X} \rightarrow M$  is defined as follows:  $\text{lab}^\mu(x) = x$  and  $\text{lab}^\mu(f(t_1, \dots, t_n)) = f_{\pi_f(\llbracket t_1 \rrbracket_\mu, \dots, \llbracket t_n \rrbracket_\mu)}(\text{lab}^\mu(t_1), \dots, \text{lab}^\mu(t_n))$ .

The fundamental theorem of semantic labelling is then:

**Theorem 3 ([22]).** *Given a rewrite system  $\mathcal{R}$ , an ordered  $\mathcal{F}$ -algebra  $(\mathcal{M}, \leq_{\mathcal{M}})$  and a labelling system  $(S^f, \leq_f, \pi_f)_{f \in \mathcal{F}}$ , the relation  $\rightarrow_{\mathcal{R}}$  terminates if:*



1.  $\mathcal{M}$  is a quasi-model of  $\mathcal{R}$ , that is:
  - for each rule  $l \rightarrow r \in \mathcal{R}$  and valuation  $\mu : \mathcal{X} \rightarrow M$ ,  $\llbracket l \rrbracket \mu \geq_{\mathcal{M}} \llbracket r \rrbracket \mu$ ,
  - for each  $f \in \mathcal{F}$ ,  $f^{\mathcal{M}}$  is monotonic;
2. for each  $f \in \mathcal{F}$ ,  $\pi_f$  is monotonic;
3. the relation  $\rightarrow_{lab(\mathcal{R}) \cup Decr}$  terminates where:
  - $lab(\mathcal{R}) = \{lab^\mu(l) \rightarrow lab^\mu(r) \mid l \rightarrow r \in \mathcal{R}, \mu : \mathcal{X} \rightarrow M\}$ ,
  - $Decr = \{f_a(x_1, \dots, x_n) \rightarrow f_b(x_1, \dots, x_n) \mid f \in \mathcal{F}, a >_f b\}$ .

For instance, by taking  $M = \mathbb{N}$ ,  $0^{\mathcal{M}} = 0$ ,  $s^{\mathcal{M}}(x) = x + 1$ ,  $-^{\mathcal{M}}(x, y) = x$  and  $/^{\mathcal{M}}(x, y) = x$ , and by labelling  $-$  and  $/$  by the semantics of their first argument, we get the following *infinite* system which is easily proved terminating:

$$\begin{array}{ll}
 -_i x 0 \rightarrow x & (i \in \mathbb{N}) \\
 -_0 0x \rightarrow 0 & \\
 -_{i+1}(sx)(sy) \rightarrow -_i xy & (i \in \mathbb{N})
 \end{array}
 \qquad
 \begin{array}{ll}
 /_0 0x \rightarrow 0 & \\
 /_{i+1}(sx)y \rightarrow s(/_i(-ixy)y) & (i \in \mathbb{N})
 \end{array}$$

## 6 First-Order Case

The reader may have already noticed some similarity between semantic labelling and size annotations. We here render it more explicit by giving a new proof of the correctness of SB-termination using semantic labelling.

In the first-order case, the interpretation of a base type does not require transfinite iteration: all sizes are smaller than  $\omega$  and  $\mathfrak{A} = \mathbb{N}$  [6]. Moreover, by taking  $\Gamma(x) = \mathbb{B}^x$  for each  $x$  of type  $\mathbb{B}$ , every term  $t$  has a most general size  $\sigma(t)$  given by its most general type:  $\Gamma \vdash^i t : C^{\sigma(t)}$ . This function  $\sigma$  extends to all terms the function  $\sigma$  defined in the previous section by taking  $\sigma(f(t_1, \dots, t_n)) = f^{\mathcal{A}}(\sigma(t_1), \dots, \sigma(t_n))$  for each defined symbol  $f$ .

**Theorem 4.** *SB-termination implies termination if:*

- $\mathcal{R}$  is finitely branching and the set of constructors of each type  $\mathbb{B}$  is finite;
- for each defined symbol  $f$ ,  $f^{\mathcal{A}}$  and  $\zeta_f^{\mathcal{A}}$  are monotonic.

*Proof.* For the interpretation domain, we take  $M = \mathfrak{A} = \mathbb{N}$  which has a structure of poset with  $\leq_{\mathcal{M}} = \leq_{\mathfrak{A}} = \leq_{\mathbb{N}}$ .

If  $f^{\mathcal{A}}$  is not the constant function equal to  $\infty$  ( $f^{\mathcal{A}} \neq \infty$  for short), which is the case of constructors, then let  $f^{\mathcal{M}}(\mathbf{a}) = \llbracket f^{\mathcal{A}}(\mathbf{a}) \rrbracket \mu$  where  $\mathbf{a}\mu = \mathbf{a}$ .

When  $f^{\mathcal{A}} = \infty$ , we proceed in a way similar to *predictive labelling* [15], a variant of semantic labelling where only the semantics of *usable symbols* need to be given when  $\mathcal{M}$  is a  $\sqcup$ -algebra (all *finite* subsets of  $M$  have a lub wrt  $\leq_{\mathcal{M}}$ ), which is the case of  $\mathbb{N}$ . Here, the notions of usable symbols and rules are not necessary and a semantics can be given to all symbols thanks to the strong assumptions of SB-termination.

Let  $(f, \mathbf{x}) >^{\mathfrak{A}} (g, \mathbf{y})$  if  $f >_{\mathcal{F}} g$  or  $f \simeq_{\mathcal{F}} g$  and  $\zeta_f^{\mathfrak{A}}(\mathbf{x}) >_f^{\mathfrak{A}} \zeta_f^{\mathfrak{A}}(\mathbf{y})$ . The relation  $>^{\mathfrak{A}}$  is well-founded since the relations  $>_{\mathcal{F}}$  and  $>_f^{\mathfrak{A}}$  are well-founded. We then define  $f^{\mathcal{M}}$  by induction on  $>^{\mathfrak{A}}$  by taking  $f^{\mathcal{M}}(\mathbf{a}) = \max(\{0\} \cup \{\llbracket r \rrbracket \mu \mid f\mathbf{l} \rightarrow r \in \mathcal{R}, \mu : \mathcal{X} \rightarrow \mathfrak{A}, \llbracket l \rrbracket \mu \leq \mathbf{a}\})$ . This function is well defined since:

- For each subterm  $\mathbf{gm}$  in  $r$ ,  $(f, \sigma(\mathbf{l})) >^{\mathcal{A}} (g, \sigma(\mathbf{m}))$ . Assume that  $f \simeq_{\mathcal{F}} g$ . Then,  $\sigma(\mathbf{l}) >_{\mathcal{A}} \sigma(\mathbf{m})$ . Hence, for each symbol  $f$  occurring in  $\mathbf{l}$  or  $\mathbf{m}$ ,  $f^{\mathcal{A}} \neq \infty$ . Therefore,  $\llbracket \mathbf{l} \rrbracket \mu = \llbracket \sigma(\mathbf{l}) \rrbracket \mu$ ,  $\llbracket \mathbf{m} \rrbracket \mu = \llbracket \sigma(\mathbf{m}) \rrbracket \mu$  and  $(f, \llbracket \mathbf{l} \rrbracket \mu) >^{\mathfrak{A}} (g, \llbracket \mathbf{m} \rrbracket \mu)$ .
- The set  $\{(f\mathbf{l} \rightarrow r, \mu) \mid f\mathbf{l} \rightarrow r \in \mathcal{R}, \llbracket \mathbf{l} \rrbracket \mu \leq \mathbf{a}\}$  is finite. Indeed, since  $\mathbf{l}$  are patterns and constructors are interpreted by monotonic and strictly extensive functions (i.e.  $c^{\mathcal{A}}(\alpha) \geq_{\mathcal{A}} s(\max(\alpha_i \mid i \in \text{Ind}(c)))$ ),  $\llbracket \mathbf{l} \rrbracket \mu$  is strictly monotonic wrt  $\mu$  and the height of  $\mathbf{l}$ . We cannot have an infinite set of  $\mathbf{l}$ 's of bounded height since, for each base type  $\mathbf{B}$ , the set of constructors of type  $\mathbf{B}$  is finite. And we cannot have an infinite set of  $r$ 's since  $\mathcal{R}$  is finitely branching.

We do not label the constructors, i.e. we take any singleton set for  $S^c$  and the unique (constant) function from  $M^n$  to  $S^c$  for  $\pi_c$ . For any other symbol  $f$ , we take  $S^f = D_f^{\mathfrak{A}}$  which is well-founded wrt  $>_f$ , and  $\pi_f = \zeta_f^{\mathfrak{A}}$ .

1.  $\mathcal{M}$  is a quasi-model of  $\mathcal{R}$ :

- Let  $f :^s \mathbf{P} \Rightarrow \mathbf{B}^{\mathcal{A}} \Rightarrow \mathbf{B}^{f^{\mathcal{A}}}(\alpha)$ ,  $l \rightarrow r \in \mathcal{R}$  with  $l = f\mathbf{pl}$ , and  $\mu : \mathcal{X} \rightarrow M$ . We have  $\llbracket \mathbf{l} \rrbracket \mu = f^{\mathcal{M}}(\mathbf{a})$  where  $\mathbf{a} = \llbracket \mathbf{l} \rrbracket \mu$ . If  $f^{\mathcal{A}} = \infty$ , then  $f^{\mathcal{M}}(\mathbf{a}) = \max(\{0\} \cup \{\llbracket r \rrbracket \mu \mid f\mathbf{l} \rightarrow r \in \mathcal{R}, \mu : \mathcal{X} \rightarrow \mathfrak{A}, \llbracket \mathbf{l} \rrbracket \mu \leq \mathbf{a}\})$  and  $\llbracket \mathbf{l} \rrbracket \mu \geq \llbracket r \rrbracket \mu$ . Assume now that  $f^{\mathcal{A}} \neq \infty$ . Since  $\Gamma \vdash_{f\mathbf{a}} r :^i \mathbf{B}^{\mathcal{A}}$  and  $a \leq_{\overline{\mathcal{A}}} f^{\mathcal{A}}(\mathbf{a})$ , we have  $\sigma(r) = a \leq_{\overline{\mathcal{A}}} f^{\mathcal{A}}(\mathbf{a}) = \sigma(l)$  where  $\mathbf{a} = \sigma(\mathbf{l})$ . By definition of  $\Gamma$  and  $\sigma$ , for each  $i$ ,  $a_i \neq \infty$  ( $\mathbf{a} \neq \infty$  for short). Therefore,  $\sigma(l) \neq \infty$  and  $\sigma(r) \leq_{\mathcal{A}} \sigma(l)$ . Hence,  $\llbracket \mathbf{l} \rrbracket \mu = \sigma(l)\mu \leq_{\mathfrak{A}} \sigma(r)\mu = \llbracket r \rrbracket \mu$  since  $\leq_{\mathfrak{A}}$  is a model of  $\leq_{\mathcal{A}}$ .
  - If  $f$  is a non-recursive constructor, then  $f^{\mathcal{M}}(\mathbf{a}) = 0$  is monotonic. If  $f$  is a recursive constructor, then  $f^{\mathcal{M}}(\mathbf{a}) = \sup\{\mathbf{a}_i \mid i \in \text{Ind}(c)\} + 1$  is monotonic. If  $f^{\mathcal{A}} \neq \infty$ , then  $f^{\mathcal{M}}(\mathbf{a}) = \llbracket f^{\mathcal{A}}(\alpha) \rrbracket \mu$  where  $\alpha\mu = \mathbf{a}$  is monotonic since  $f^{\mathcal{A}}$  is monotonic by assumption. Finally, if  $f^{\mathcal{A}} = \infty$ , then  $f^{\mathcal{M}}(\mathbf{a}) = \max(\{0\} \cup \{\llbracket r \rrbracket \mu \mid f\mathbf{l} \rightarrow r \in \mathcal{R}, \mu : \mathcal{X} \rightarrow \mathfrak{A}, \llbracket \mathbf{l} \rrbracket \mu \leq \mathbf{a}\})$  is monotonic.
2. If  $f$  is a defined symbol, then the function  $\pi_f$  is monotonic by assumption. If  $f$  is a constructor, then the constant function  $\pi_f$  is monotonic too.
3. We now prove that  $\rightarrow_{\text{lab}(\mathcal{R}) \cup \text{Decr}}$  is precedence-terminating (PT), i.e. there is a well-founded relation  $>$  on symbols such that, for each rule  $f\mathbf{l} \rightarrow r \in \text{lab}(\mathcal{R}) \cup \text{Decr}$ , every symbol occurring in  $r$  is strictly smaller than  $f$  [19].

Let  $\mathbf{g}\mathbf{a} < \mathbf{f}\mathbf{b}$  if  $\mathbf{g} <_{\mathcal{F}} \mathbf{f}$  or  $\mathbf{g} \simeq_{\mathcal{F}} \mathbf{f}$  and  $a <_{\mathfrak{A}}^{\mathfrak{A}} b$ . The relation  $>$  is well-founded since both  $>_{\mathcal{F}}$  and  $>_{\mathfrak{A}}^{\mathfrak{A}}$  are well-founded.

*Decr* is clearly PT wrt  $>$ . Let now  $f\mathbf{l} \rightarrow r \in \mathcal{R}$ ,  $\mu : \mathcal{X} \rightarrow M$  and  $\mathbf{g}\mathbf{t}$  be a subterm of  $r$ . The label of  $f$  is  $a = \pi_f(\llbracket \mathbf{l} \rrbracket \mu) = \zeta_f^{\mathfrak{A}}(\llbracket \sigma(\mathbf{l}) \rrbracket \mu)$  and the label of  $\mathbf{g}$  is  $b = \zeta_f^{\mathfrak{A}}(\llbracket \sigma(\mathbf{m}) \rrbracket \mu)$ . By assumption,  $(f, \mathbf{l}) >^{\mathcal{A}} (g, \mathbf{m})$ . Therefore,  $a >_{\mathfrak{A}}^{\mathfrak{A}} b$ .  $\square$

It is interesting to note that we could also have taken  $M = \mathcal{A}$ , assuming that  $<_{\mathfrak{A}}^{\mathcal{A}}$  is stable by substitution ( $\zeta_f^{\mathcal{A}}(\mathbf{a}\theta) <_{\mathfrak{A}}^{\mathcal{A}} \zeta_f^{\mathcal{A}}(\mathbf{b}\theta)$  whenever  $\zeta_f^{\mathcal{A}}(\mathbf{a}) <_{\mathfrak{A}}^{\mathcal{A}} \zeta_f^{\mathcal{A}}(\mathbf{b})$ ). The system labelled with  $\mathcal{A}$  is a syntactic approximation of the system labelled with  $\mathfrak{A}$ . Although less powerful *a priori*, it may be interesting since it provides a *finite* representation of the infinite  $\mathfrak{A}$ -labelled system.

Finally, we see from the proof that the system does not need to be constructor:

**Theorem 5.** *Theorem 4 holds for any (non-constructor) system  $\mathcal{R}$  such that, for each rule  $f\mathbf{l} \rightarrow r \in \mathcal{R}$  with  $f^{\mathcal{A}} = \infty$  and subterm  $\mathbf{gm}$  in  $\mathbf{l}$ :*

- $\mathbf{g}^A$  is monotonic and strictly extensive:  $\mathbf{g}^A(\alpha) \geq_{\mathcal{A}} \mathbf{s}(\max(\alpha_i \mid i \in \text{Ind}(\mathbf{c})))$ ,
- if  $\mathbf{g}^A = \infty$ , then  $\mathbf{g} <_{\mathcal{F}} \mathbf{f}$  or  $\mathbf{g} \simeq_{\mathcal{F}} \mathbf{f}$  and  $\zeta_{\mathbf{f}}^A(\sigma(\mathbf{m})) <_{\mathbf{f}}^A \zeta_{\mathbf{f}}^A(\sigma(\mathbf{l}))$ .

**Example:** assuming that  $\mathbf{A}$  is the  $\Rightarrow$ -type constructor, then the expression  $\mathbf{F}nuv$  represents the set of  $n$ -ary functions from  $u$  to  $v$ .

$$\begin{array}{ll} +0y & \rightarrow y & \mathbf{F}0uv & \rightarrow v \\ +(sx)y & \rightarrow \mathbf{s}(+xy) & \mathbf{F}(sx)uv & \rightarrow \mathbf{A}u(\mathbf{F}xuv) \\ +(+xy)z & \rightarrow +x(+yz) & \mathbf{F}(+xy)uv & \rightarrow \mathbf{F}xu(\mathbf{F}yuv) \end{array}$$

Take  $+^A(x, y) = \zeta_+(x, y) = a = 2x + y + 1$ ,  $\mathbf{F}^A = \infty$  and  $\zeta_{\mathbf{F}}(x, u, v) = x$ . The interpretation of  $\mathbf{F}^{\mathcal{M}}$  is well-defined since  $x < a$  and  $y < a$ . The labelled system that we obtain (where  $b = 2y + z + 1$ ) is precedence-terminating:

$$\begin{array}{ll} +_{y+1}0y & \rightarrow y & \mathbf{F}_00uv & \rightarrow v \\ +_{a+2}(sx)y & \rightarrow \mathbf{s}(+_{a}xy) & \mathbf{F}_{x+1}(sx)uv & \rightarrow \mathbf{A}u(\mathbf{F}_x xuv) \\ +_{2a+z+1}(+_{a}xy)z & \rightarrow +_{2x+b+1}x(+_{b}yz) & \mathbf{F}_a(+_{a}xy)uv & \rightarrow \mathbf{F}_x xu(\mathbf{F}_y yuv) \end{array}$$

## 7 Higher-Order Semantic Labelling

Semantic labelling was extended by Hamana [13] to second-order Inductive Data Type Systems (IDTSs) with higher-order pattern-matching [4]. IDTSs are a typed version of Klop’s Combinatory Reduction Systems (CRSs) [17] whose categorical semantics based on binding algebras and  $\mathcal{F}$ -monoids [10] is studied by the same author and proved complete for termination [14].

The fundamental theorem of higher-order semantic labelling can be stated exactly as in the first-order case, but the notion of model is more involved.

**CRSs and IDTSs.** In CRSs, function symbols have a fixed arity. *Meta-terms* extend terms with the application  $Z(t_1, \dots, t_n)$  of a meta-variable  $Z \in \mathcal{Z}$  of arity  $n$  to  $n$  meta-terms  $t_1, \dots, t_n$ .

An assignment  $\theta$  maps every meta-variable of arity  $n$  to a term of the form  $\lambda x_1.. \lambda x_n t$ . Its application to a meta-term  $t$ , written  $t\theta$ , is defined as follows:

- $x\theta = x$ ,  $(\lambda xt)\theta = \lambda x(t\theta)$  and  $\mathbf{f}(t_1, \dots, t_n)\theta = \mathbf{f}(t_1\theta, \dots, t_n\theta)$ ;
- for  $\theta(Z) = \lambda x_1.. \lambda x_n t$ ,  $Z(t_1, \dots, t_n)\theta = t\{x_1 \mapsto t_1\theta, \dots, x_n \mapsto t_n\theta\}$ .

A rule is a pair of *meta-terms*  $l \rightarrow r$  such that  $l$  is a higher-order pattern [20].

In IDTSs, variables, meta-variables and symbols are equipped with types over a discrete category  $\mathbb{B}$  of base types. However, Hamana only considers *structural meta-terms* where abstractions only appear as arguments of a function symbol, variables are restricted to base types, meta-variables to first-order types and function symbols to second-order types. But, as already noticed by Hamana, this is sufficient to handle any rewrite system (see Section 8). Let  $I_B^{\mathcal{Z}}(\Gamma)$  be the set of structural meta-terms of type  $B$  in  $\Gamma$  whose meta-variables are in  $\mathcal{Z}$ .

**Models.** The key idea of binding algebras [10] is to interpret variables by natural numbers using De Bruijn levels, and to handle bound variables by extending the interpretation to typing environments.

Let  $\mathbb{F}$  be the category whose objects are the finite cardinals and whose arrows from  $n$  to  $p$  are all the functions from  $n$  to  $p$ . Let  $\mathbb{E}$  be the (slice) category of

typing environments whose objects are the maps  $\Gamma : n \rightarrow \mathbb{B}$  and whose arrows from  $\Gamma : n \rightarrow \mathbb{B}$  to  $\Delta : p \rightarrow \mathbb{B}$  are the functions  $\rho : n \rightarrow p$  such that  $\Gamma = \Delta \circ \rho$ .

Given  $\Gamma : n \rightarrow \mathbb{B}$ , let  $\Gamma + B : n + 1 \rightarrow \mathbb{B}$  be the environment such that  $(\Gamma + B)(n) = B$  and  $(\Gamma + B)(k) = \Gamma(k)$  if  $k < n$ .

Let  $\mathbb{M}$  be the functor category  $(\mathbf{Set}^{\mathbb{B}})^{\mathbb{B}}$ . An object of  $\mathbb{M}$  (presheaf) is given by a family of sets  $M_B(\Gamma)$  for every base type  $B$  and environment  $\Gamma$  and, for every base type  $B$  and arrow  $f : \Gamma \rightarrow \Delta$ , a function  $M_B(f) : M_B(\Gamma) \rightarrow M_B(\Delta)$  such that  $M_B(id_\Gamma) = id_{M_B(\Gamma)}$  and  $M_B(f \circ g) = M_B(f) \circ M_B(g)$ . An arrow  $\alpha : M \rightarrow N$  in  $\mathbb{M}$  is a natural transformation, *i.e.* a family of functions  $\alpha_B(\Gamma) : M_B(\Gamma) \rightarrow N_B(\Gamma)$  such that, for each  $\rho : \Gamma \rightarrow \Delta$ ,  $\alpha_B(\Delta) \circ M_B(\rho) = N_B(\rho) \circ \alpha_B(\Gamma)$ .

Given  $M \in \mathbb{M}$ ,  $\Gamma \in \mathbb{E}$  and  $\mathbf{B} \in \mathbb{B}$ , let  $up_\Gamma^{\mathbf{B}}(M) : M(\Gamma) \rightarrow M(\Gamma + \mathbf{B})$  be the arrow equal to  $M(id_\Gamma + 0_\Delta)$  where  $0_\Delta$  is the unique morphism from  $0$  to  $\Delta$ .

An  $\mathcal{X}$  +  $\mathcal{F}$ -algebra  $\mathcal{M}$  is given by a presheaf  $M \in \mathbb{M}$ , an interpretation of variables  $\iota : \mathcal{X} \rightarrow \mathcal{M}$  and, for every symbol  $f : (\mathbf{B}_1 \Rightarrow B_1) \Rightarrow \dots \Rightarrow (\mathbf{B}_n \Rightarrow B_n) \Rightarrow B$  and environment  $\Gamma$ , an arrow  $f^{\mathcal{M}}(\Gamma) : \prod_{i=1}^n M_{B_i}(\Gamma + \mathbf{B}_i) \rightarrow M_B(\Gamma)$ .

The category  $\mathbb{M}$  forms a monoidal category with unit  $\mathcal{X}$  and product  $\bullet$  such that  $(M \bullet N)_B(\Gamma)$  is the set of equivalence classes on the set of pairs  $(t, \mathbf{u})$  with  $t \in M_B(\Delta)$  and  $u_i \in N_{\Delta(i)}(\Gamma)$  for some  $\Delta$ , modulo the equivalence relation  $\sim$  such that  $(t, \mathbf{u}) \sim (t', \mathbf{u}')$  if there is  $\rho : \Delta \rightarrow \Delta'$  for which  $t \in M_B(\Delta)$ ,  $t' = M_B(\rho)(t)$  and  $u'_{\rho(i)} = u_i$ .

To interpret substitutions,  $M$  must be an  $\mathcal{F}$ -monoid, *i.e.* a monoid  $(M, \mu : M^2 \rightarrow M)$  compatible with the structure of  $\mathcal{F}$ -algebra [13] (see Appendix E).

The presheaf  $I^\emptyset$  equipped with the product  $\mu_B(\Gamma)(t, \mathbf{u}) = t\{i \mapsto u_i\}$  (simultaneous substitution) is initial in the category of  $\mathcal{F}$ -monoids [14]. Hence, for each  $\mathcal{F}$ -monoid  $\mathcal{M}$ , there is a unique morphism  $!^{\mathcal{M}} : I^\emptyset \rightarrow \mathcal{M}$ .

**Labelling.** As in the first-order case, for each  $f : (\mathbf{B}_1 \Rightarrow B_1) \Rightarrow \dots \Rightarrow (\mathbf{B}_n \Rightarrow B_n) \Rightarrow B$ , we assume given a non-empty poset  $(S^f, \leq_f)$  for labels and a labelling function  $\pi_f(\Gamma) : \prod_{i=1}^n M_{B_i}(\Gamma + \mathbf{B}_i) \rightarrow S^f$ . Let  $\overline{\mathcal{F}}_n = \{f_a \mid f \in \mathcal{F}_n, a \in S^f\}$ . Note that the set of labelled meta-terms has a structure of  $\mathcal{F}$ -monoid [13].

The labelling of a meta-term wrt a valuation  $\theta : \mathcal{Z} \rightarrow I^\emptyset$  is defined as follows:

- $lab_B^\emptyset(\Gamma)(x) = x$ ;
- $lab_B^\emptyset(\Gamma)(Z(t_1, \dots, t_n)) = Z(lab_B^\emptyset(\Gamma)(t_1), \dots, lab_B^\emptyset(\Gamma)(t_n))$ ;
- for  $f : (\mathbf{B}_1 \Rightarrow B_1) \Rightarrow \dots \Rightarrow (\mathbf{B}_n \Rightarrow B_n) \Rightarrow B$  and  $\Gamma_i = \Gamma, \mathbf{x}_i : \mathbf{B}_i$ ,  
 $lab_B^\emptyset(\Gamma)(f(\lambda \mathbf{x}_1 t_1, \dots, \lambda \mathbf{x}_n t_n)) = f_a(lab_{B_1}^\emptyset(\Gamma_1)(t_1), \dots, lab_{B_n}^\emptyset(\Gamma_n)(t_n))$   
 where  $a = \pi_f(!_{B_1}^{\mathcal{M}}(\Gamma_1)(t_1\theta), \dots, !_{B_n}^{\mathcal{M}}(\Gamma_n)(t_n\theta))$ .

We can now state Hamana's theorem for higher-order semantic labelling.

**Theorem 6 ([13]).** *Given a structural IDTS  $\mathcal{R}$ , an ordered  $\mathcal{F}$ -algebra  $(\mathcal{M}, \leq_{\mathcal{M}})$  and a labelling system  $(S^f, \leq_f, \pi_f)_{f \in \mathcal{F}}$ , the relation  $\rightarrow_{\mathcal{R}}$  terminates if:*

1.  $(\mathcal{M}, \leq_{\mathcal{M}})$  is a quasi-model of  $\mathcal{R}$ , that is:
  - for each  $l \rightarrow r : T \in \mathcal{R}$ ,  $\theta : \mathcal{Z} \rightarrow I^\emptyset$  and  $\Gamma$ ,  $!_B^{\mathcal{M}}(\Gamma)(l\theta) \geq_{M_B(\Gamma)} !_B^{\mathcal{M}}(\Gamma)(r\theta)$ ,
  - for each  $f \in \mathcal{F}$ ,  $f^{\mathcal{M}}$  is monotonic;
2. for each  $f \in \mathcal{F}$ ,  $\pi_f$  is monotonic;

3. the relation  $\rightarrow_{\text{lab}(\mathcal{R}) \cup \text{Decr}}$  terminates, where:

$$\begin{aligned} \text{lab}(\mathcal{R}) &= \{\text{lab}_B^0(\Gamma)(l\theta) \rightarrow \text{lab}_B^0(\Gamma)(r\theta) \mid l \rightarrow r : B \in \mathcal{R}, \theta : \mathcal{Z} \rightarrow I^\emptyset, \Gamma \in \mathbb{E}\}, \\ \text{Decr} &= \{f_a(\dots, \lambda \mathbf{x}_i Z_i(\mathbf{x}_i), \dots) \rightarrow f_b(\dots, \lambda \mathbf{x}_i Z_i(\mathbf{x}_i), \dots) \mid f \in \mathcal{F}, a >_f b\}. \end{aligned}$$

## 8 Higher-Order Case

In order to apply Hamana's higher-order semantic labelling, we first need to translate into a structural IDTS not only the rewrite system  $\mathcal{R}$  but also  $\beta$  itself.

**Translation to structural IDTS.** Following Example 4.1 in [13], the relations  $\beta$  and  $\mathcal{R}$  can be encoded in a structural IDTS as follows.

Let the set of *IDTS base types*  $\mathbb{B}$  be the set  $\mathcal{T}(\Sigma)$  where  $\Sigma_0 = \mathcal{B}$  is the set of base types,  $\Sigma_2 = \{\text{Arr}\}$  and  $\Sigma_n = \emptyset$  otherwise. A simple type  $T$  can then be translated into an IDTS base type  $\langle T \rangle$  by taking  $\langle T \Rightarrow U \rangle = \text{Arr}(\langle T \rangle, \langle U \rangle)$  and  $\langle T \rangle = T$  if  $T \in \mathcal{B}$ . Then, an environment  $\Gamma$  can be translated into an IDTS environment  $\langle \Gamma \rangle$  by taking  $\langle \emptyset \rangle = \emptyset$  and  $\langle x : T, \Gamma \rangle = x : \langle T \rangle, \langle \Gamma \rangle$ . Conversely, let  $|T|$  be the simple type such that  $\langle |T| \rangle = T$ .

Let the set of *IDTS function symbols* be the set  $\mathcal{F}$  made of the symbols  $\langle f \rangle : \langle T_1 \rangle \Rightarrow \dots \Rightarrow \langle T_n \rangle \Rightarrow \mathbb{B}$  such that  $f : T_1 \Rightarrow \dots \Rightarrow T_n \Rightarrow \mathbb{B}$ , and all the symbols  $\lambda_T^U : (T \Rightarrow U) \Rightarrow \text{Arr}(T, U)$  and  $@_T^U : \text{Arr}(T, U) \Rightarrow T \Rightarrow U$  such that  $T$  and  $U$  are IDTS base types. Note that only  $\lambda_T^U$  has a second order type.

A simply-typed  $\lambda$ -term  $t$  such that  $\Gamma \vdash t : T$  can then be translated into an IDTS term  $\langle t \rangle_\Gamma$  such that  $\langle \Gamma \rangle \vdash \langle t \rangle_\Gamma : \langle T \rangle$  as follows:

- $\langle x \rangle_\Gamma = x$ ,
- $\langle \lambda x^T u \rangle_\Gamma = \lambda_{\langle T \rangle}^{\langle U \rangle}(\lambda x \langle u \rangle_{\Gamma, x:T})$  if  $\Gamma, x : T \vdash u : U$ ,
- for  $f : T_1 \Rightarrow \dots \Rightarrow T_n \Rightarrow \mathbb{B}$  and  $U_i = T_{i+1} \Rightarrow \dots \Rightarrow T_n \Rightarrow \mathbb{B}$ ,  
 $\langle ft_1 \dots t_k \rangle_\Gamma = \lambda_{\langle T_{k+1} \rangle}^{\langle U_{k+1} \rangle}(\lambda x_{k+1} \dots \lambda_{\langle T_n \rangle}^{\langle U_n \rangle}(\lambda x_n \langle f \rangle(\langle t_1 \rangle_\Gamma, \dots, \langle t_k \rangle_\Gamma, x_{k+1}, \dots, x_n)) \dots)$ ,
- $\langle tu \rangle_\Gamma = @_{\langle U \rangle}^{\langle V \rangle}(\langle t \rangle_\Gamma, \langle u \rangle_\Gamma)$  if  $\Gamma \vdash t : U \Rightarrow V$ .

A rewrite rule  $l \rightarrow r \in \mathcal{R}$  is then translated into the IDTS rule  $\langle l \rangle \rightarrow \langle r \rangle$  where the free variables of  $l$  are seen as nullary meta-variables, and  $\beta$ -rewriting is translated into the family of IDTS rules  $\langle \beta \rangle = \bigcup_{T, U \in \mathbb{B}} \beta_T^U$  where  $\beta_T^U$  is:

$$@_T^U(\lambda_T^U(\lambda x Z(x)), X) \rightarrow Z(X)$$

where  $Z$  (resp.  $X$ ) is a meta-variable of type  $T \Rightarrow U$  (resp.  $T$ ). Note that only  $\langle \beta \rangle$  uses non-nullary meta-variables.

Then,  $\rightarrow_{\mathcal{R}} \cup \rightarrow_\beta$  terminates iff  $\rightarrow_{\langle \mathcal{R} \rangle \cup \langle \beta \rangle}$  terminates (Appendix F).

**Interpretation domain.** We now define the interpretation domain  $M$  for interpreting  $\langle \beta \rangle \cup \langle \mathcal{R} \rangle$ . First, we interpret environments as arrow types:

- $M_T(\Gamma) = N_{\text{Arr}(\Gamma, T)}$  where:  
 $\text{Arr}(\emptyset, T) = T$  and  $\text{Arr}(\Gamma + U, T) = \text{Arr}(\Gamma, \text{Arr}(U, T))$ .

As explained at the beginning of Section 3, to every base type  $B \in \mathcal{B}$  corresponds a limit ordinal  $\omega_B < \aleph$  that is the number of transfinite iterations of the monotonic function  $F_B$  that is necessary to build the interpretation of  $B$ .

So, a first idea is to take  $N_B = \omega_B$  and the set of functions from  $N_T$  to  $N_U$  for  $N_{Arr(T,U)}$ . But taking all functions creates some problems. Consider for instance the constructor  $\text{lim} : (\mathbb{N} \Rightarrow \mathbb{O}) \Rightarrow \mathbb{O}$ . We expect  $\text{lim}^M(\emptyset)(f) = \text{sup}\{f(n) \mid n \in N_{\mathbb{N}}\} + 1$  to be a valid interpretation, but  $\text{sup}\{f(n) \mid n \in N_{\mathbb{N}}\} + 1$  is not in  $N_{\mathbb{O}}$  for each function  $f$ . We therefore need to restrict  $N_{Arr(T,U)}$  to the functions that correspond to (are realized by) some  $\lambda$ -term.

Hence, let  $N_T = \{x \mid \exists t \in \mathcal{T}, t \vdash_T x\}$  where  $\vdash_T$  is defined as follows:

- $t \vdash_B \mathbf{a} \in \omega_B$  if  $t \in \llbracket B \rrbracket$  and  $o_B(t) \geq \mathbf{a}$ ,
- $v \vdash_{Arr(T,U)} f : N_T \rightarrow N_U$  if  $v \in \llbracket T \rrbracket \Rightarrow \llbracket U \rrbracket$  and  $vt \vdash_U f(x)$  whenever  $t \vdash_T x$ .

Then, we can now check that  $\text{sup}\{f(n) \mid n \in N_{\mathbb{N}}\} + 1 \in N_{\mathbb{O}}$ . Indeed, if there are  $v$  and  $t$  such that  $v \vdash_{Arr(\mathbb{N},\mathbb{O})} f$  and  $t \vdash_{\mathbb{N}} n$ , then  $vt \vdash_{\mathbb{O}} f(n)$  and  $\text{lim}(v) \vdash_{\mathbb{O}} \text{sup}\{f(n) \mid n \in N_{\mathbb{N}}\} + 1 \in N_{\mathbb{O}}$ .

The action of  $M$  on  $\mathbb{E}$ -morphisms is defined as follows. Given  $f : \Gamma \rightarrow \Delta$  with  $\Gamma : n \rightarrow \mathbb{B}$  and  $\Delta : p \rightarrow \mathbb{B}$ , let  $M_T(f) : M_T(\Gamma) \rightarrow M_T(\Delta)$  be the function mapping  $x_0 \in N_{Arr(\Gamma,T)}$ ,  $x_1 \in N_{\Delta(1)}, \dots, x_p \in N_{\Delta(p)}$  to  $x_0(x_{f(1)}, \dots, x_{f(n)})$ .

Finally, the sets  $M_B(\Gamma)$  and  $N_T$  are ordered as follows:

- $x \leq_{M_B(\Gamma)} y$  if  $x \leq_{N_{Arr(\Gamma,B)}} y$  where:
  - $x \leq_{N_B} y$  if  $x \leq y$ ,
  - $f \leq_{N_{Arr(T,U)}} g$  if  $f(x) \leq_{N_U} g(x)$  for each  $x \in N_T$ .

**Interpretation of variables and function symbols.** As one can expect, variables are interpreted by projections:  $\iota_{\Gamma(i)}(\Gamma)(i)(\mathbf{x}) = x_i$ ,  $\lambda_T^U$  by the identity:  $(\lambda_T^U)^M(\Gamma)(f) = f$ , and  $@_T^U$  by the application:  $(@_T^U)^M(\Gamma)(f, x)(\mathbf{y}) = f(\mathbf{y}, x(\mathbf{y}))$ .

One can check that these functions are valid interpretations indeed, *i.e.*  $\iota_{\Gamma(i)}(\Gamma)(i)(\mathbf{x}) \in N_{\Gamma(i)}$  and  $(@_T^U)^M(\Gamma)(f, x)(\mathbf{y}) \in N_U$ .

Moreover, we have  $(@_T^U)^M(\Gamma)(f, x)(\mathbf{x}) = \mu_U(\Gamma)(f, \mathbf{p}\mathbf{x})$  where  $p_i = \iota_{\Gamma(i)}(\Gamma)(i)$  and  $\mu$  is the monoidal product  $\mu_B(\Gamma)(t, u_1 \dots u_n)(\mathbf{x}) = t(u_1(\mathbf{x}), \dots, u_n(\mathbf{x}))$ .

We can then verify that  $\langle \beta \rangle$  is valid if  $(M, \mu)$  is an  $\mathcal{F}$ -monoid, and that  $(M, \mu)$  is an  $\mathcal{F}$ -monoid if, for each  $f$  and  $\Gamma$ ,  $f^M(\Gamma)(\mathbf{x})(\mathbf{y}) = f^M(\emptyset)(x_1(\mathbf{y}), \dots, x_n(\mathbf{y}))$  (Appendix G).

One can see that  $(\lambda_T^U)^M$  and  $(@_T^U)^M$  satisfy this property. Moreover, for each term  $t \in I_T^\emptyset(\Gamma)$ , we have  $!_T^M(x_1 : T_1 \dots x_n : T_n)(t)(\mathbf{a}) = \llbracket t \rrbracket \mu$  where  $x_i \mu = a_i$  and:

$$\begin{aligned} \llbracket x \rrbracket \mu &= \mu(x) & \llbracket @_T^U(v, t) \rrbracket \mu &= \llbracket v \rrbracket \mu(\llbracket t \rrbracket \mu) & \llbracket \lambda_T^U(\lambda x u) \rrbracket \mu &= a \mapsto \llbracket u \rrbracket \mu_x^a \\ \llbracket f(\mathbf{t}) \rrbracket \mu &= f^M(\emptyset)(\llbracket \mathbf{t} \rrbracket \mu) & \llbracket Z(\mathbf{t}) \rrbracket \mu &= \mu(Z)(\llbracket \mathbf{t} \rrbracket \mu) \end{aligned}$$

**Higher-order size algebra.** In the first-order case, the interpretation of the function symbols  $f$  such that  $f^A$  is not the constant function equal to  $\infty$  (which includes constructors) is  $f^M(\mathbf{a}) = \llbracket f^A(\alpha) \rrbracket \mu$  where  $\alpha \mu = \mathbf{a}$ . To be able to do the same thing in the higher-order case, we need the size algebra  $\mathcal{A}$  to be a typed higher-order algebra interpreted in the sets  $N_T$ .

Hence, now, we assume that size expressions are simply-typed  $\lambda$ -terms over a typed signature  $\Sigma$ , and that every function symbol  $f : \tau_f$  is interpreted by  $\infty$  or a size expression  $f^A : \tau_f$ . We then let  $\sigma : \mathcal{T} \rightarrow \overline{\mathcal{A}}$  be the function that replaces in a term every symbol  $f$  by  $f^A$ , all the terms containing  $\infty$  being identified. Hence, for each term  $t$  containing no symbol  $f$  such that  $f^A = \infty$ , we have  $\llbracket t \rrbracket \mu = \llbracket \sigma(t) \rrbracket \mu$ . Finally, we define  $<_{\mathcal{A}}$  as the relation such that  $a <_{\mathcal{A}} b$  if, for each  $\mu$ ,  $\llbracket a \rrbracket \mu <_{\mathfrak{A}} \llbracket b \rrbracket \mu$ .

For instance, for a strictly-positive constructor  $c : \mathbf{T} \Rightarrow \mathbf{B}$  with  $T_i = U_i \Rightarrow B_i$ , we can assume that there is a symbol  $c^A \in \Sigma$  interpreted by the function  $c^{\mathfrak{A}}(\mathbf{x}) = \sup\{x_i \mathbf{y}_i \mid i \in \text{Ind}(c), \mathbf{y}_i \in N_{\langle U_i \rangle}\} + 1$ . Hence, with Brouwer's ordinals, we have  $\sigma(\lim f) = \lim^A f >_{\mathcal{A}} \sigma(fn) = fn$ .

Thus, using such an higher-order size algebra, we can conclude:

**Theorem 7.** *SB-termination implies termination if constructors are strictly-positive and the conditions of Theorems 4 and 5 are satisfied.*

*Proof.* The proof is similar to the first-order case (Theorem 4). We only point out the main differences.

We first check that  $\mathcal{M}$  is a quasi-model. The case of  $\langle \beta \rangle$  is detailed in Appendix G. For  $\langle \mathcal{R} \rangle$ , we use the facts that  $!_B^{\mathcal{M}}(\Gamma)(l\theta) \leq_{M_B(\Gamma)} !_B^{\mathcal{M}}(\Gamma)(r\theta)$  if  $!_B^{\mathcal{M}}(\Gamma)(l\theta)(\mathbf{a}) \leq_{M_B(\emptyset)} !_B^{\mathcal{M}}(\Gamma)(r\theta)(\mathbf{a})$  for each  $\mathbf{a}$ , and that  $!_B^{\mathcal{M}}(\Gamma)(l\theta)(\mathbf{a}) = \llbracket l \rrbracket \theta \mu$  where  $x_i \mu = a_i$ .

We do not label applications and abstractions. And for a defined symbol  $f : \mathbf{B} \Rightarrow B$ , we take  $S^f = \prod_{\Gamma} \prod_{i=1}^n M_{B_i}(\Gamma)$  and  $\pi_f(\Gamma)(\mathbf{x}) = (\Gamma, \mathbf{x})$ .

We now define a well-founded relation on  $S^f$  that we will use for proving some higher-order version of precedence-termination. For dealing with  $\text{lab}(\langle \mathcal{R} \rangle)$ , let  $(\Gamma, \mathbf{x}) >_f^{\mathcal{R}} (\Delta, \mathbf{y})$  if  $\Delta = \Gamma + \Gamma'$  and, for each  $\mathbf{z} \mathbf{z}'$ ,  $\zeta_f(\dots x_i(\mathbf{z}) \dots) >_f^{\mathfrak{A}} \zeta_f(\dots y_i(\mathbf{z} \mathbf{z}') \dots)$ . For dealing with  $\text{lab}(\langle \beta \rangle)$ , let  $(\Gamma, \mathbf{x}) >_f^{\beta} (\Delta, \mathbf{y})$  if  $\Gamma = \Delta + T$  and there is  $e$  such that, for each  $i$  and  $\mathbf{z}$ ,  $x_i(\mathbf{z}, e(\mathbf{z})) = y_i(\mathbf{z})$ . Since  $>_f^{\mathcal{R}} \circ >_f^{\beta}$  is included in  $>_f^{\mathcal{R}} \cup >_f^{\beta} \circ >_f^{\mathcal{R}}$ , the relation  $>_f = >_f^{\mathcal{R}} \cup >_f^{\beta}$  is well-founded [9].

One can easily check that the functions  $\pi_f$  and  $f^{\mathcal{M}}$  are monotonic.

We are now left to prove that  $\rightarrow_{\text{lab}(\langle \beta \rangle) \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates. First, remark that  $\rightarrow_{\text{lab}(\langle \beta \rangle)}$  is included in  $\rightarrow_{\text{Decr}}^* \rightarrow_{\langle \beta \rangle}$ . Indeed, given  $\text{lab}_T(\Gamma)(t) \rightarrow \text{lab}_U(\Gamma)(u_x^t) \in \text{lab}(\langle \beta \rangle)$ , a symbol  $f$  occurring in  $u$  is labelled in  $\text{lab}_U(\Gamma + T)(u)$  by something like  $(\Gamma + T + \Delta, !_B^{\mathcal{M}}(\Gamma + T + \Delta)(\mathbf{v}))$ , and by something like  $(\Gamma + \Delta, !_B^{\mathcal{M}}(\Gamma + \Delta)(\mathbf{v}_x^t))$  in  $\text{lab}_U(\Gamma)(u_x^t)$ . Hence, the relation  $\rightarrow_{\text{lab}(\langle \beta \rangle) \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates if  $\rightarrow_{\langle \beta \rangle \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates.

By translating back IDTS types to simple types and removing the symbols  $\lambda_T^U$  (function  $\lfloor \_ \rfloor$ ), we get a  $\beta$ -IDTS [4] such that  $\rightarrow_{\langle \beta \rangle \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates if  $\rightarrow_{\langle \beta \rangle \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates (Appendix F). Moreover, after [4],  $\rightarrow_{\langle \beta \rangle \cup \text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}}$  terminates if  $|\text{lab}(\langle \mathcal{R} \rangle) \cup \text{Decr}|$  satisfies the *General Schema* (we do not need the results on *solid* IDTSs [13]). This can be easily checked by using the precedence  $>$  on  $\overline{\mathcal{F}}$  such that  $f_a > g_b$  if  $f >_{\mathcal{F}} g$  or  $f \simeq_{\mathcal{F}} g$  and  $a >_f b$ .

**Conclusion.** By studying the relationship between sized-types based termination and semantic labelling, we arrived at a new way to prove the correctness of SBT that enabled us to extend it to non-constructor systems, *i.e.* systems with



matching on defined symbols (*e.g.* associative symbols, Appendix D). This work can be carried on in various directions by considering: richer type structures with polymorphic or dependent types, non-strictly positive constructors, or the inference of size annotations to automate SBT.

**Acknowledgments.** The authors want to thank very much Colin Riba and Andreas Abel for their useful remarks on a previous version of this paper. This work was partly supported by the Bayerisch-Französisches Hochschulzentrum.

## References

1. Abel, A.: Semi-continuous sized types and termination. *Logical Methods in Computer Science* 4(2), 1–33 (2008)
2. Barthe, G., Frade, M.J., Giménez, E., Pinto, L., Uustalu, T.: Type-based termination of recursive definitions. *Mathematical Structures in Computer Science* 14(1), 97–141 (2004)
3. Blanqui, F.: Decidability of type-checking in the calculus of algebraic constructions with size annotations. In: Ong, L. (ed.) *CSL 2005*. LNCS, vol. 3634, pp. 135–150. Springer, Heidelberg (2005)
4. Blanqui, F.: Termination and confluence of higher-order rewrite systems. In: Bachmair, L. (ed.) *RTA 2000*. LNCS, vol. 1833. Springer, Heidelberg (2000)
5. Blanqui, F.: A type-based termination criterion for dependently-typed higher-order rewrite systems. In: van Oostrom, V. (ed.) *RTA 2004*. LNCS, vol. 3091, pp. 24–39. Springer, Heidelberg (2004)
6. Blanqui, F.: Definitions by rewriting in the Calculus of Constructions. *Mathematical Structures in Computer Science* 15(1), 37–92 (2005)
7. Blanqui, F., Riba, C.: Combining typing and size constraints for checking the termination of higher-order conditional rewrite systems. In: Hermann, M., Voronkov, A. (eds.) *LPAR 2006*. LNCS (LNAI), vol. 4246, pp. 105–119. Springer, Heidelberg (2006)
8. Blanqui, F., Roux, C.: On the relation between sized-types based termination and semantic labelling, full version (2009), [www-rocq.inria.fr/~blanqui/](http://www-rocq.inria.fr/~blanqui/)
9. Doornbos, H., von Karger, B.: On the union of well-founded relations. *Logic Journal of the IGPL* 6(2), 195–201 (1998)
10. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: *Proc. of LICS 1999* (1999)
11. Giménez, E.: *Un Calcul de Constructions infinies et son application à la vérification de systèmes communicants*. PhD thesis, ENS Lyon, France (1996)
12. Girard, J.-Y.: *Interprétation fonctionnelle et élimination des coupures dans l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, France (1972)
13. Hamana, M.: Higher-order semantic labelling for inductive datatype systems. In: *Proc. of PPDP 2007* (2007)
14. Hamana, M.: Universal algebra for termination of higher-order rewriting. In: Giesl, J. (ed.) *RTA 2005*. LNCS, vol. 3467, pp. 135–149. Springer, Heidelberg (2005)
15. Hirokawa, N., Middeldorp, A.: Predictive labeling. In: Pfenning, F. (ed.) *RTA 2006*. LNCS, vol. 4098, pp. 313–327. Springer, Heidelberg (2006)
16. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: *Proc. of POPL 1996* (1996)



17. Klop, J.W., van Oostrom, V., van Raamsdonk, F.: Combinatory reduction systems. *Theoretical Computer Science* 121, 279–308 (1993)
18. Mendler, N.P.: *Inductive Definition in Type Theory*. PhD thesis. Cornell University, United States (1987)
19. Middeldorp, A., Ohsaki, H., Zantema, H.: Transforming termination by self-labelling. In: McRobbie, M.A., Slaney, J.K. (eds.) *CADE 1996*. LNCS, vol. 1104. Springer, Heidelberg (1996)
20. Miller, D.: A logic programming language with lambda-abstraction, function variables, and simple unification. In: Schroeder-Heister, P. (ed.) *ELP 1989*. LNCS, vol. 475. Springer, Heidelberg (1991)
21. Xi, H.: Dependent types for program termination verification. In: *Proc. of LICS 2001* (2001)
22. Zantema, H.: Termination of term rewriting by semantic labelling. *Fundamenta Informaticae* 24, 89–105 (1995)

# Expanding the Realm of Systematic Proof Theory

Agata Ciabattoni<sup>1</sup>, Lutz Straßburger<sup>2</sup>, and Kazushige Terui<sup>3</sup>

<sup>1</sup> Technische Universität Wien, Austria

<sup>2</sup> INRIA Saclay–Île-de-France, France

<sup>3</sup> RIMS, Kyoto University, Japan

**Abstract.** This paper is part of a general project of developing a systematic and algebraic proof theory for nonclassical logics. Generalizing our previous work on intuitionistic-substructural axioms and single-conclusion (hyper)sequent calculi, we define a hierarchy on Hilbert axioms in the language of classical linear logic without exponentials. We then give a systematic procedure to transform axioms up to the level  $\mathcal{P}'_3$  of the hierarchy into inference rules in multiple-conclusion (hyper)sequent calculi, which enjoy cut-elimination under a certain condition. This allows a systematic treatment of logics which could not be dealt with in the previous approach. Our method also works as a heuristic principle for finding appropriate rules for axioms located at levels higher than  $\mathcal{P}'_3$ . The case study of Abelian and Łukasiewicz logic is outlined.

## 1 Introduction

Since the axiomatisation of classical propositional logic by Hilbert in 1922, such axiomatic descriptions (nowadays called *Hilbert-systems*) have been successfully used to introduce and characterize logics. Ever since Gentzen’s seminal work it has been an important task for proof theorists to design for these logics deductive systems that admit cut-elimination. The admissibility of cut is crucial to establish important properties of corresponding logics such as consistency, decidability, conservativity, interpolation, and is also the key to make a system suitable for proof search. As designers of deductive systems could never keep pace with the speed of logicians and practitioners coming up with new logics, general tools to automate this design process and extract suitable rules out of axioms would be very desirable. Work in this direction are e.g. [7,20,19].

A general project of systematic and algebraic proof theory for nonclassical logics was recently launched in [3,4] where Hilbert axioms in the language of full Lambek calculus FL (i.e., intuitionistic noncommutative linear logic without exponentials) have been classified into the *substructural hierarchy*  $(\mathcal{P}_n, \mathcal{N}_n)_{n \in \mathbb{N}}$ , with the aim to conquer the whole hierarchy from bottom to top. The work in [3] successfully dealt with the axioms up to level  $\mathcal{N}_2$ . It gave a procedure to transform them into structural rules in the single-conclusion sequent calculus, and algebraically proved (a stronger form of) cut-elimination for FL extended with the generated rules which satisfy the syntactic condition of acyclicity. Then,

[4] expanded the realm to the level  $\mathcal{P}'_3$ , a subclass of  $\mathcal{P}_3$  in the commutative setting, by using the single-conclusion hypersequent calculus [2]. The aim to continue the conquer further faced a serious obstacle: As shown in [3], “strong” cut-elimination for a logical system  $L$  implies that the class of algebras corresponding to  $L$  is closed under completions, whereas certain logics beyond  $\mathcal{P}'_3$  do not admit closure under completions. Typical examples are Abelian logic AL [15,16]—the logic corresponding to compact closed categories—and infinite-valued Łukasiewicz logic  $\mathbb{L}$ , although possessing cut-free hypersequent calculi, see [14].

In this paper, we circumvent the obstacle by shifting from the intuitionistic and single-conclusion setting to the classical and multiple-conclusion one. This causes a *deconstruction* of the hierarchy; certain axioms which resided at high levels are brought down to lower levels, to the reach of our systematic proof theory (Section 3). Generalizing the method in [3,4], Section 4 (resp. Section 5) describe a procedure to transform any  $\mathcal{N}_2$  axiom (resp.  $\mathcal{P}'_3$  axiom) into structural rules in the multiple-conclusion sequent (resp. hypersequent) calculus. The procedure is also applied to obtain logical rules for connectives defined by certain Hilbert axioms. Section 6 outlines a uniform syntactic cut-elimination procedure that works for the generated rules satisfying the acyclicity condition. This allows the systematic introduction of cut-free calculi for logics which cannot be dealt with in the single-conclusion approach, such as 3-valued Łukasiewicz logic and Nelson’s logic. Our method also works as a heuristic principle for finding appropriate rules for axioms located at levels higher than  $\mathcal{P}'_3$ . As a case study, in Section 7 we show how to semi-automatically obtain the cut-free hypersequent calculi for AL and  $\mathbb{L}$ , that have been discovered in [14] by trial and error.

## 2 Preliminaries: Sequents and Hypersequents

We consider formulas to be generated from a set  $\mathcal{V} = \{a, b, c, \dots\}$  of *propositional variables*, their duals  $\mathcal{V}^\perp = \{a^\perp, b^\perp, c^\perp, \dots\}$ , and the constants  $\perp, 1, 0$ , and  $\top$  via the binary connectives  $\wp, \otimes, \oplus$ , and  $\&$ .<sup>1</sup>

$$\mathcal{F} ::= \mathcal{V} \mid \mathcal{V}^\perp \mid \perp \mid 1 \mid 0 \mid \top \mid \mathcal{F} \wp \mathcal{F} \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F} \oplus \mathcal{F} \mid \mathcal{F} \& \mathcal{F} \quad (1)$$

We use  $A, B, C, \dots$  to denote formulas, and we define the negation on formulas via the usual DeMorgan equalities. It follows immediately that  $A^{\perp\perp} = A$  for all  $A$ . We write  $A \multimap B$  for  $A^\perp \wp B$ , and  $A \multimap\multimap B$  for  $(A \multimap B) \& (B \multimap A)$ . For reasons that will become clear later, we will write  $A_{\&1}$  for  $A \& 1$ .

We will also speak about *axiom (schemes)*  $\phi, \psi, \dots$ , which are generated by the same grammar as (1), but starting from *formula variables* instead of propositional variables. By some abuse of notation, we use  $A, B, C, \dots$  to denote formula

<sup>1</sup> We use here the notation used in the linear logic community. The table below gives the translation to the notation used in the substructural logics community.

linear logic:	$\wp$	$\otimes$	$\oplus$	$\&$	$\perp$	$1$	$0$	$\top$
substructural logics:	$\oplus$	$\cdot/\odot$	$\vee$	$\wedge$	$0$	$1$	$\perp$	$\top$

ax $\frac{}{\mathcal{H} \vdash A, A^\perp}$	cut $\frac{\mathcal{H} \mid \vdash \Gamma, A \quad \mathcal{H} \mid \vdash A^\perp, \Delta}{\mathcal{H} \mid \vdash \Gamma, \Delta}$	ew $\frac{\mathcal{H}}{\mathcal{H} \mid \vdash \Gamma}$	ec $\frac{\mathcal{H} \mid \vdash \Gamma \mid \vdash \Gamma}{\mathcal{H} \mid \vdash \Gamma}$
1 $\frac{}{\mathcal{H} \mid \vdash 1}$	$\otimes$ $\frac{\mathcal{H} \mid \vdash \Gamma, A \quad \mathcal{H} \mid \vdash B, \Delta}{\mathcal{H} \mid \vdash \Gamma, A \otimes B, \Delta}$	$\perp$ $\frac{\mathcal{H} \mid \vdash \Gamma}{\mathcal{H} \mid \vdash \Gamma, \perp}$	$\wp$ $\frac{\mathcal{H} \mid \vdash \Gamma, A, B}{\mathcal{H} \mid \vdash \Gamma, A \wp B}$
$\top$ $\frac{}{\mathcal{H} \mid \vdash \Gamma, \top}$	$\&$ $\frac{\mathcal{H} \mid \vdash \Gamma, A \quad \mathcal{H} \mid \vdash \Gamma, B}{\mathcal{H} \mid \vdash \Gamma, A \& B}$	$\oplus_1$ $\frac{\mathcal{H} \mid \vdash \Gamma, A}{\mathcal{H} \mid \vdash \Gamma, A \oplus B}$	$\oplus_2$ $\frac{\mathcal{H} \mid \vdash \Gamma, B}{\mathcal{H} \mid \vdash \Gamma, A \oplus B}$

Fig. 1. Hypersequent system HMALL

variables. We call an axiom  $\phi$  *atomic* if  $\phi = A$  or  $\phi = A^\perp$  for some formula variable  $A$ . By some further abuse of notation, we will use  $A, B, C, \dots$  to denote atomic axioms (positive or negative).

We write  $\oplus_{i=1}^n A_i$ , or simply  $\oplus_i A_i$  to abbreviate  $A_1 \oplus A_2 \oplus \dots \oplus A_n$ , where  $\oplus_{i=1}^n A_i = 0$  if  $n = 0$ , and similarly for the other connectives.

**Definition 1.** A (*single sided*) *sequent* is a finite multiset of formulas, usually written as  $\vdash A_1, \dots, A_n$ . A (*single sided*) *hypersequent*  $\mathcal{H}$  is a finite multiset of sequents written as  $\vdash \Gamma_1 \mid \dots \mid \vdash \Gamma_n$ . The *interpretation*  $(\vdash \Gamma)^I$  of a sequent  $\vdash \Gamma = \vdash A_1, \dots, A_n$  is the formula  $A_1 \wp \dots \wp A_n$ , and  $(\vdash \Gamma)^I = \perp$ , if  $n = 0$ . For a hypersequent  $\mathcal{H} = \vdash \Gamma_1 \mid \dots \mid \vdash \Gamma_n$ , we define  $\mathcal{H}^I = (\vdash \Gamma_1)_{\&1}^I \oplus \dots \oplus (\vdash \Gamma_n)_{\&1}^I$ .

Henceforth we use  $\Gamma, \Delta, \Sigma, \dots$  to denote multisets of formulas, and  $\mathcal{G}, \mathcal{H}, \dots$  to denote hypersequents. We denote by HMALL the hypersequent system shown in Figure 1. With MALL we denote the corresponding sequent system, obtained from HMALL by removing the rules ec and ew, and by dropping the hypersequent context  $\mathcal{H}$  everywhere. In inference rules we will refer to  $\Gamma, \Delta, \Sigma, \dots$  as *multiset variables* (as opposed to the formula variables  $A, B, \dots$ ).

The notation  $\vdash_S A$  (respectively  $\vdash_S \Gamma$  or  $\vdash_S \mathcal{H}$ ) will mean that a formula  $A$  (respectively a sequent  $\vdash \Gamma$  or a hypersequent  $\mathcal{H}$ ) is provable in the system  $S$ .

**Proposition 1.** For any sequent  $\vdash \Gamma$  and hypersequent  $\mathcal{G}$ , we have that

$$\vdash_{\text{HMALL} + \mathcal{G}} \Gamma \quad \text{iff} \quad \vdash_{\text{MALL} + \mathcal{G}^I} \Gamma .$$

*Proof.* For the ‘if’ direction, observe that  $\mathcal{G}^I$  is derivable from  $\mathcal{G}$  in HMALL. For the converse, prove by induction that  $\vdash_{\text{HMALL} + \mathcal{G}} \mathcal{H}$  implies  $\vdash_{\text{MALL} + \mathcal{G}^I} \mathcal{H}^I$ .  $\square$

**Definition 2.** Given two sets of inference rules  $S_1$  and  $S_2$ , we say that  $S_1$  and  $S_2$  are *equivalent* iff (H)MALL +  $S_1$  and (H)MALL +  $S_2$  prove the same sequents. If  $S_1 = \{r\}$  is a singleton, we simply write (H)MALL +  $r$ .

An axiom  $\phi$  is a rule without premises. Thus, the definition above applies also to (sets of) axioms.

**Remark 1.** By moving to the single-sided (multiple conclusion) setting, we do not lose any expressive power of the two-sided single-conclusion setting (i.e. involving sequents of the form  $\Gamma \vdash A$ ). Indeed the latter can faithfully be embedded into the former by using *left/right polarities*, referring to the left and the right side of a two-sided sequent (see, e.g., [8,9] for details). We can then call a MALL formula (or axiom) *intuitionistic*, if it has right polarity. A MALL sequent  $\vdash \Gamma$  is *intuitionistic* iff at most one formula in  $\Gamma$  has right polarity, and all other formulas in  $\Gamma$  have left polarity, and a proof in MALL is called *intuitionistic* if all its lines are intuitionistic sequents. We then have that a formula  $A$  belongs to intuitionistic logic iff its translation  $A^c$  into the classical language is intuitionistic in our sense. Let IMALL denote the usual two-sided sequent calculus for intuitionistic MALL (also known as full Lambek calculus with exchange, FLe), then we have  $\vdash_{\text{IMALL}} \Gamma \vdash A$  iff there is an intuitionistic proof of  $(\Gamma \vdash A)^c$  in MALL. Furthermore, If  $A$  does not contain any occurrences of  $\perp$  or  $\top$ , then  $\vdash_{\text{IMALL}} \Gamma \vdash A$  iff  $\vdash_{\text{MALL}} (\Gamma \vdash A)^c$ . The reason is that MALL is a conservative extension of IMALL without  $\perp$  and  $\top$  [21].

### 3 Substructural Hierarchy

Following [3,4], we define a hierarchy  $(\mathcal{P}_n, \mathcal{N}_n)$  on formulas of MALL. It is based on the *polarities* of the connectives [1], which is also the basis for focusing and linear logic programming [17]. Recall that logical connectives of MALL can be classified into two groups: negative ( $\wp, \&, \perp$ , and  $\top$ ) and positive ( $\oplus, \otimes, 0$ , and  $1$ ), according to the fact that their sequent calculus rules are invertible and non-invertible, respectively.

Let  $\mathcal{A}$  be the set of atomic axioms. The classes  $\mathcal{P}_n$  and  $\mathcal{N}_n$  of positive and negative axioms are defined via the following grammar:

$$\begin{aligned}
 \mathcal{P}_0 &::= \mathcal{A} & \mathcal{P}_{n+1} &::= \mathcal{N}_n \mid \mathcal{P}_{n+1} \otimes \mathcal{P}_{n+1} \mid \mathcal{P}_{n+1} \oplus \mathcal{P}_{n+1} \mid 1 \mid 0 \\
 \mathcal{N}_0 &::= \mathcal{A} & \mathcal{N}_{n+1} &::= \mathcal{P}_n \mid \mathcal{N}_{n+1} \& \mathcal{N}_{n+1} \mid \mathcal{N}_{n+1} \wp \mathcal{N}_{n+1} \mid \top \mid \perp
 \end{aligned}
 \tag{2}$$

We have the following immediate observations:

**Proposition 2.** *Every axiom belongs to some  $\mathcal{P}_n$  and some  $\mathcal{N}_n$ , and for all  $n$ , we have  $\mathcal{P}_n \subseteq \mathcal{P}_{n+1}$  and  $\mathcal{N}_n \subseteq \mathcal{N}_{n+1}$ . Furthermore,  $A \in \mathcal{P}_n$  iff  $A^\perp \in \mathcal{N}_n$ .*

Hence we have a hierarchy, called the *substructural hierarchy* in [3,4], which can be depicted as in Figure 2.

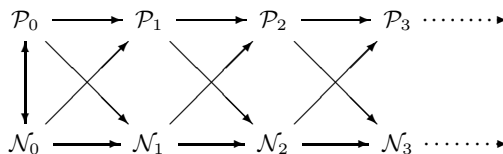


Fig. 2. The Substructural Hierarchy

Axiom (intuitionistic and classical version)		Name	Rule	Class
i:	$A \multimap 1, \perp \multimap A$	weakening	w	$\mathcal{N}_2$
c:	$A \wp 1$			$\mathcal{N}_2$
i:	$A \multimap A \otimes A$	contraction	c	$\mathcal{N}_2$
c:	$A^\perp \wp (A \otimes A)$			$\mathcal{N}_2$
i:	$A \oplus (A \multimap \perp)$	excluded middle	em	$\mathcal{P}_2$
c:	$A \oplus A^\perp$			$\mathcal{P}_1$
i:	$(A \multimap B)_{\&1} \oplus (B \multimap A)_{\&1}$	linearity	com	$\mathcal{P}'_3$
c:	$(A^\perp \wp B)_{\&1} \oplus (B^\perp \wp A)_{\&1}$			$\mathcal{P}'_3$
i:	$((A^{\otimes 2} \multimap B) \& ((B \multimap \perp)^{\otimes 2} \multimap (A \multimap \perp))) \multimap (A \multimap B)$	Nelson axiom	nel	$\mathcal{N}_3$
c:	$((A \otimes A \otimes B) \oplus (B \otimes B \otimes A)) \wp A^\perp \wp B^\perp$			$\mathcal{N}_2$
i:	$A \oplus (A^{\otimes n} \multimap \perp)$	$n$ -excluded middle	$n$ -em	$\mathcal{P}_2$
c:	$A \oplus (A^\perp)^{\wp n}$			$\mathcal{P}_2$

We abbreviate  $A \otimes \dots \otimes A$  ( $n$  times) by  $A^{\otimes n}$  and  $A \wp \dots \wp A$  ( $n$  times) by  $A^{\wp n}$ .

**Fig. 3.** Axioms and their level in the substructural hierarchy

**Remark 2.** In [4] the hierarchy was defined on formulas of IMALL as follows:

$$\begin{aligned}
 \mathcal{P}_0^i &::= \mathcal{A} & \mathcal{P}_{n+1}^i &::= \mathcal{N}_n^i \mid \mathcal{P}_{n+1}^i \otimes \mathcal{P}_{n+1}^i \mid \mathcal{P}_{n+1}^i \oplus \mathcal{P}_{n+1}^i \mid 1 \mid 0 \\
 \mathcal{N}_0^i &::= \mathcal{A} & \mathcal{N}_{n+1}^i &::= \mathcal{P}_n^i \mid \mathcal{N}_{n+1}^i \& \mathcal{N}_{n+1}^i \mid \mathcal{P}_{n+1}^i \multimap \mathcal{N}_{n+1}^i \mid \top \mid \perp,
 \end{aligned}$$

where  $\mathcal{A}$  ranges over positive atomic axioms (without negation). It follows from Remark 1 that the two hierarchies coincide:  $\phi \in \mathcal{P}_n^i$  iff  $\phi^c \in \mathcal{P}_n$ , and  $\phi \in \mathcal{N}_n^i$  iff  $\phi^c \in \mathcal{N}_n$ .

Figure 3 shows some examples of axioms and their class, and Figure 4 shows the corresponding structural rules. How they are obtained will be explained in the course of this paper. Observe we can have the following situation: For a certain intuitionistic axiom  $\phi$  there is an axiom  $\phi'$  located in a lower class of the hierarchy such that  $\phi'$  is not intuitionistic and  $\vdash_{\text{MALL}} \phi \multimap \phi'$ . The use of the classical language also simplifies the following statement, established in [4], which will be used to make syntactic transformations of axioms.

**Proposition 3.** *Every axiom  $\phi \in \mathcal{P}_{n+1}$  is equivalent to an axiom of the form  $\bigoplus_i (\bigotimes_j \psi_{i,j})$  where  $\psi_{i,j} \in \mathcal{N}_n$  for each  $i, j$ . And every axiom  $\phi \in \mathcal{N}_{n+1}$  is equivalent to an axiom of the form  $\&_i (\wp_j \psi_{i,j})$  where  $\psi_{i,j} \in \mathcal{P}_n$  for each  $i, j$ .*

**Definition 3.** An axiom  $\phi$  is called  $\mathcal{N}_2$ -normal if it is of the shape

$$\phi = \wp_k (\bigoplus_i (\bigotimes_j A_{k,i,j})) \quad \text{where each } A_{k,i,j} \text{ is atomic.}$$

It follows immediately from Proposition 3 that any  $\mathcal{N}_2$ -axiom can be transformed into a finite conjunction ( $\&$ ) of  $\mathcal{N}_2$ -normal axioms. As in [4], for dealing with systems having no weakening, we consider a subclass of  $\mathcal{P}_3$  that we call  $\mathcal{P}'_3$ , which is generated by the grammar:

$$\mathcal{P}'_3 ::= \mathcal{N}_2 \& 1 \mid \mathcal{P}'_3 \otimes \mathcal{P}'_3 \mid \mathcal{P}'_3 \oplus \mathcal{P}'_3 \mid 1 \mid 0$$

$\text{w} \frac{\vdash \Gamma}{\vdash \Gamma, \Delta}$	$\text{c} \frac{\vdash \Gamma, \Delta, \Delta}{\vdash \Gamma, \Delta}$	$\text{em} \frac{\vdash \Gamma, \Gamma}{\vdash \Gamma}$	$\text{com} \frac{\vdash \mathcal{H} \Gamma, \Theta \quad \vdash \mathcal{H} \Sigma, \Delta}{\mathcal{H} \vdash \Gamma, \Delta \vdash \Sigma, \Theta}$
$\text{nel} \frac{\vdash \Gamma, \Sigma, \Sigma, \Delta \quad \vdash \Gamma, \Sigma, \Delta, \Delta}{\vdash \Gamma, \Delta, \Sigma}$		$\text{n-em} \frac{\mathcal{H} \vdash \Gamma, \Sigma_1 \quad \dots \quad \mathcal{H} \vdash \Gamma, \Sigma_n}{\mathcal{H} \vdash \Gamma \vdash \Sigma_1, \dots, \Sigma_n}$	

**Fig. 4.** Rules generated from axioms in Figure 3

**Lemma 1.** *The set  $\{\phi_{\&1} \oplus \xi, \psi_{\&1} \oplus \xi\}$  is equivalent (in the sense of Definition 2) to  $(\phi_{\&1} \otimes \psi_{\&1}) \oplus \xi$  as well as to  $(\phi \& \psi)_{\&1} \oplus \xi$ , for any  $\phi, \psi$ , and  $\xi$ .*

*Proof.* Follows from provability in MALL of  $(A_{\&1} \oplus C) \otimes (B_{\&1} \oplus C) \multimap (A_{\&1} \otimes B_{\&1}) \oplus C$  and  $(A_{\&1} \otimes B_{\&1}) \oplus C \multimap (A \& B)_{\&1} \oplus C$ , for all formulas  $A, B$ , and  $C$ . □

**Proposition 4.** *Every axiom  $\phi \in \mathcal{P}'_3$  is equivalent to a finite set  $\{\psi_1, \dots, \psi_n\}$  of axioms such that  $\psi_i = \bigoplus_{j=1}^{m_i} (\xi_{i,j})_{\&1}$  where  $\xi_{i,j}$  is  $\mathcal{N}_2$ -normal for all  $i, j$ .*

*Proof.* We have  $\phi \multimap \bigoplus_j \otimes_k (\&_l \xi_{j,k,l})_{\&1}$  where  $\xi_{j,k,l}$  is  $\mathcal{N}_2$ -normal, so that we can apply Lemma 1. □

## 4 From $\mathcal{N}_2$ -Axioms to Sequent Rules

In this section we provide an algorithm for transforming  $\mathcal{N}_2$  axioms into equivalent sequent calculus rules. Our algorithm extends and simplifies the one introduced in 4 (and in 3, for the noncommutative case) for axioms in  $\mathcal{N}_2^i$  and structural rules. A suitable modification of the procedure also enables us to extend the result to logical rules.

**Lemma 2.** *For any axiom  $\xi$ , the following two sequent rules are equivalent*

$$\frac{\vdash \Sigma_1 \quad \dots \quad \vdash \Sigma_n}{\vdash \Gamma, \xi} \quad \text{and} \quad \frac{\vdash \Sigma_1 \quad \dots \quad \vdash \Sigma_n \quad \vdash \Delta, \xi^\perp}{\vdash \Gamma, \Delta}, \tag{3}$$

where  $\Delta$  is fresh. If  $\xi \in \mathcal{P}_1$ , then the rules in (3) are equivalent to a rule

$$\frac{\vdash \Sigma_1 \quad \dots \quad \vdash \Sigma_n \quad \vdash \Delta, A_{1,1}, \dots, A_{1,k_1} \quad \dots \quad \vdash \Delta, A_{m,1}, \dots, A_{m,k_m}}{\vdash \Gamma, \Delta} \tag{4}$$

where each  $A_{i,j}$  is atomic and  $m, k_1, \dots, k_m \geq 0$ .

*Proof.* The first equivalence is shown in one direction by letting  $\Delta = \xi$  and using the ax-rule and in the other direction by using cut. If  $\xi \in \mathcal{P}_1$  then  $\xi^\perp \in \mathcal{N}_1$ , and hence  $\xi^\perp$  is equivalent to  $\&_i (\wp_j A_{i,j})$ . Then (4) follows by using the (invertible) rules  $\wp, \&, \top$  and  $\perp$ . □

**Theorem 1.** *Every  $\mathcal{N}_2$ -axiom can be transformed into a finite set of equivalent structural sequent rules, whose conclusions consist only of multiset variables.*

*Proof.* Let  $\phi$  be any  $\mathcal{N}_2$ -axiom. By Proposition 3  $\phi$  is equivalent to a finite set  $\{\psi_1, \dots, \psi_n\}$  of  $\mathcal{N}_2$ -normal axioms, which means that each  $\psi_i$  is equivalent to

$$\frac{}{\vdash \xi_{i,1}, \dots, \xi_{i,m_i}} \quad (5)$$

where each  $\xi_{i,j} \in \mathcal{P}_1$ . The claim follows by repeatedly applying Lemma 2.  $\square$

**Example 1.** Applying the above procedure to excluded middle and Nelson axiom respectively (see Figure 3) yields the structural rules:

$$\text{em}' \frac{\vdash \Gamma, A \quad \vdash \Gamma, A^\perp}{\vdash \Gamma} \quad \text{nel}' \frac{\vdash \Gamma, A^\perp, A^\perp, B \quad \vdash \Gamma, A^\perp, B, B \quad \vdash \Delta, A \quad \vdash \Sigma, B^\perp}{\vdash \Gamma, \Delta, \Sigma}$$

These rules will be further transformed in Section 6 into equivalent rules obeying the subformula property.

By suitably adapting the procedure above we show below how to generate logical rules for connectives which are defined via  $\mathcal{N}_2$  axioms.

**Theorem 2.** *Let  $\otimes$  be a connective. Any axiom of the shape  $\phi \wp (A \otimes B)$ , where  $A$  and  $B$  are formula variables and  $\phi \in \mathcal{N}_2$ , is equivalent to a finite set of logical sequent rules for  $\otimes$ .*

*Proof.* By Proposition 3  $\phi \wp (A \otimes B)$  is equivalent to a finite set  $\{\psi_1 \wp (A \otimes B), \dots, \psi_n \wp (A \otimes B)\}$  where each  $\psi_i$  is  $\mathcal{N}_2$ -normal. Hence each  $\psi_i \wp (A \otimes B)$  is equivalent to

$$\frac{}{\vdash \xi_{i,1}, \dots, \xi_{i,m_i}, A \otimes B} \quad (6)$$

where each  $\xi_{i,j} \in \mathcal{P}_1$ . By repeatedly applying Lemma 2 we eliminate all  $\xi_{i,j}$  thus obtaining a logical rule for  $A \otimes B$ .  $\square$

Note that Theorem 2 also applies for  $n$ -ary connectives. In Section 7 we show two examples of the usage of this theorem for the binary case.

## 5 From $\mathcal{P}'_3$ -Axioms to Hypersequent Rules

In this section we show how to obtain structural rules in hypersequent calculus which are equivalent to  $\mathcal{P}'_3$ -axioms ( $\mathcal{P}_3$ , in presence of weakening). Our procedure generalizes the one in 4.

First notice that Lemma 2 can be extended to hypersequent calculus, without having to change the proof (by using ew and ec), as follows:

**Lemma 3.** *For any axiom  $\xi$ , the following hypersequent rules are equivalent*

$$\frac{\mathcal{G}_1 \quad \dots \quad \mathcal{G}_n}{\mathcal{H} | \mathcal{H}' | \vdash \Gamma, \xi} \quad \text{and} \quad \frac{\mathcal{G}_1 \quad \dots \quad \mathcal{G}_n \quad \mathcal{H} | \vdash \Delta, \xi^\perp}{\mathcal{H} | \mathcal{H}' | \vdash \Gamma, \Delta}, \quad (7)$$



where  $\Delta$  is fresh. If  $\xi \in \mathcal{P}_1$ , then the rules in (7) are equivalent to a rule

$$\frac{\mathcal{G}_1 \quad \cdots \quad \mathcal{G}_n \quad \mathcal{H} \mid \vdash \Delta, A_{1,1}, \dots, A_{1,k_1} \quad \cdots \quad \mathcal{H} \mid \vdash \Delta, A_{m,1}, \dots, A_{m,k_m}}{\mathcal{H} \mid \mathcal{H}' \mid \vdash \Gamma, \Delta}, \quad (8)$$

where each  $A_{i,j}$  is atomic and  $m, k_1, \dots, k_m \geq 0$ .

**Definition 4.** A hypersequent structural rule or *hyperstructural rule* is

$$\frac{\mathcal{H} \mid \vdash \Psi_1 \quad \cdots \quad \mathcal{H} \mid \vdash \Psi_n}{\mathcal{H} \mid \vdash \Phi_1 \mid \dots \mid \vdash \Phi_m} \quad (9)$$

where each  $\Phi_i$  and  $\Psi_j$  contains only multiset variables and formula variables.

**Theorem 3.** Every  $\mathcal{P}'_3$ -axiom is equivalent to a finite set of hyperstructural rules where  $\Phi_1, \dots, \Phi_n$  consist of mutually distinct multiset variables.

*Proof.* Let  $\phi$  be a  $\mathcal{P}'_3$ -axiom. By Proposition 4,  $\phi$  is equivalent to a finite set  $\{\psi_1, \dots, \psi_n\}$  of formulas such that  $\psi_i = \bigoplus_j (\chi_{i,j})_{\&1}$  where  $\chi_{i,j}$  is  $\mathcal{N}_2$ -normal for all  $i, j$ . Thus, by Proposition 1 (see Def. 1 and 3), each  $\psi_i$  is equivalent to

$$\vdash \xi_{i,1,1}, \dots, \xi_{i,1,m_{i1}} \mid \dots \mid \vdash \xi_{i,k,1}, \dots, \xi_{i,k,m_{ik}} \quad (10)$$

where each  $\xi_{i,j,l} \in \mathcal{P}_1$ . By presence of the ew-rule, (10) is equivalent to

$$\frac{}{\mathcal{H} \mid \vdash \xi_{i,1,1}, \dots, \xi_{i,1,m_{i1}} \mid \dots \mid \vdash \xi_{i,k,1}, \dots, \xi_{i,k,m_{ik}}} \quad (11)$$

Thus, to each component of (11) we can apply the same procedure as in the proof of Theorem 1, by using Lemma 3 instead of Lemma 2.  $\square$

**Corollary 1.** Every  $\mathcal{P}_3$ -axiom is equivalent to a finite set of hyperstructural rules in presence of weakening.

*Proof.* This follows from Theorem 3 and the fact that  $\vdash_{\text{MALL}+\text{w}} A \circ \multimap A \& 1$ .  $\square$

**Example 2.** The axiom  $A \oplus (A^\perp)^{\otimes n}$  in Figure 3 ( $n$ -excluded middle) is equivalent to the following structural rule, in the presence of weakening

$$\text{n-em}' \frac{\mathcal{H} \mid \vdash \Sigma_1, A^\perp \quad \cdots \quad \mathcal{H} \mid \vdash \Sigma_n, A^\perp \quad \mathcal{H} \mid \vdash \Gamma, A}{\mathcal{H} \mid \vdash \Gamma \mid \vdash \Sigma_1, \dots, \Sigma_n}$$

**Remark 3.** The step from Theorem 1 to Theorem 2 can also be done in the setting of hypersequents. Indeed, suppose that a connective  $\otimes$  appears in an axiom  $\phi$  which is equivalent to a set of rules of the shape

$$\frac{}{\vdash \xi_{1,1}, \dots, \xi_{1,m_1} \mid \dots \mid \vdash \xi_{k,1}, \dots, \xi_{k,m_k}, A \otimes B}$$

We can apply the same procedure as in the proof of Theorem 3 to obtain a set of logical hypersequent rules for  $\otimes$ .

## 6 Rule Completion and Cut-Elimination

Let us take stock of what we achieved so far. Sections 4 and 5 contain procedures to transform axioms up to the class  $\mathcal{P}'_3$  ( $\mathcal{P}_3$ , in presence of  $w$ ) into equivalent (hyper)structural rules. Here we show how they can, provided they are *acyclic*, be transformed into equivalent rules which preserve cut-elimination when added to HMALL. A uniform and constructive cut elimination proof for HMALL extended with these rules is presented.

**Definition 5.** The *cut-closure*  $CUT(r)$  of a (hyper)structural rule  $r$  is the minimal set which contains the premises of  $r$  and it is closed under applications of the cut rule. A rule  $r$  is said to be *cyclic* if for some formula variable  $A$ , we have  $\mathcal{H} \mid \vdash \Gamma, A, A^\perp \in CUT(r)$ . Otherwise  $r$  is *acyclic*.

**Example 3.** The rules  $em'$ ,  $nel'$ , and  $n-em'$  in Examples 1 and 2 are acyclic. On the other hand, the following two rules are cyclic:

$$\text{cancel} \frac{\vdash \Gamma, A, A^\perp}{\vdash \Gamma} \quad \text{and} \quad \text{menace} \frac{\vdash \Gamma, A^\perp, A^\perp \quad \vdash \Delta, A, A}{\vdash \Gamma, \Delta}$$

**Definition 6.** We call a hyperstructural rule  $r$  *completed* if it satisfies the following properties 4:

- *No Formula Variable (NFV)*: The conclusion and all premises of  $r$  contain only multiset variables and hypersequent contexts.
- *Linear Conclusion (LC)*: Each multiset variable occurs at most once in the conclusion of  $r$ .
- *Subformula Property (SP)*: Each multiset variable occurring in the premises of  $r$  also occurs in the conclusion.

The conditions (NFV) and (LC) are crucial for the cut-elimination proof below (see, e.g., 24 for counterexamples when either of them is violated). Condition (SP) ensures that cut-elimination implies the subformula property. The rules generated by our procedures satisfy (NFV) for the conclusion, and (LC) and (SP) for multiset variables. Thus, for transforming them into equivalent completed rules it is enough to remove formula variables from the premises. This is done in the proof of the following theorem, by suitably modifying the “cutting step” of 4.

**Theorem 4.** Any acyclic (hyper)structural rule  $r$  generated by the procedures in Theorems 1 and 3 can be transformed into an equivalent completed rule.

*Proof.* By induction on the number of formula variables in the premises of  $r$ . Let  $A$  be one such variable. We denote by  $\mathcal{G}_A^+$  and  $\mathcal{G}_A^-$  the (subsets of the) premises of  $r$  which contain at least one occurrence of  $A$  and  $A^\perp$ , respectively. If  $\mathcal{G}_A^+ = \emptyset$  we remove  $\mathcal{G}_A^-$ . As  $A$  does not appear in the conclusion of  $r$ , the resulting rule implies the original one by instantiating  $A$  with  $\top$ . The case  $\mathcal{G}_A^- = \emptyset$  is similar. Otherwise, note that  $A^\perp \notin \mathcal{G}_A^+$  and  $A \notin \mathcal{G}_A^-$  by acyclicity; moreover if some hypersequent in  $\mathcal{G}_A^+$  (resp.  $\mathcal{G}_A^-$ ) contains several occurrences

of  $A$  (resp. of  $A^\perp$ ) then no hypersequent in  $\mathcal{G}_A^-$  (resp. in  $\mathcal{G}_A^+$ ) contains more than one occurrence of  $A^\perp$  (resp. of  $A$ ). Hence we may assume w.l.o.g. that  $\mathcal{G}_A^+ = \{\mathcal{H} \vdash \mathcal{Y}_i, A : 1 \leq i \leq m\}$  and  $\mathcal{G}_A^- = \{\mathcal{H} \vdash \Phi_k, A^\perp, \dots, A^\perp : 1 \leq k \leq n\}$ . Let  $\mathcal{G}_A^{\text{cut}} = \{\mathcal{H} \vdash \Phi_k, \mathcal{Y}_{i_1}, \dots, \mathcal{Y}_{i_k} : 1 \leq k \leq n \text{ and } 1 \leq i_1, \dots, i_k \leq m\}$ . Let  $r'$  be the rule obtained by replacing in  $r$  the premises  $\mathcal{G}_A^- \cup \mathcal{G}_A^+$  with  $\mathcal{G}_A^{\text{cut}}$ . We show that  $r'$  is equivalent to  $r$ . The direction  $r' \Rightarrow r$  easily follows by using **cut**. For the other direction, we set  $\tilde{A} = \bigoplus_{i=1}^m \mathcal{Y}_i$ . Clearly  $\vdash_{\text{HMALL}} \mathcal{H} \vdash \mathcal{Y}_i, \tilde{A}$ , for all  $i = 1, \dots, m$ , and for each  $k = 1, \dots, n$ , the hypersequent  $\mathcal{H} \vdash \Phi_k, \tilde{A}^\perp, \dots, \tilde{A}^\perp$  is derivable from  $\mathcal{G}_A^{\text{cut}}$  using the  $\&$ -rule. By applying  $r$  we get the conclusion of  $r'$ . Acyclicity is preserved, the number of formula variables decreased by one.  $\square$

**Example 4.** By applying the procedure in Theorem 4 to the rules in Examples 1 and 2 we obtain the equivalent completed rules **em**, **nel** and **n-em** of Figure 4. **MALL + nel** is the cut-free calculus recently introduced in [12] for constructive logic with strong negation (also known as Nelson's logic). **HMALL + w + n-em** is instead a cut-free calculus for **MALL** extended with weakening and  $n$ -excluded middle (see Figure 3). The latter logic coincides with 3-valued Łukasiewicz logic when  $n = 2$  and with the logic **IMT3** of [5] for  $n = 3$ .

**Conjecture 1.** *In presence of **w**, any (hyper)structural rule generated by Theorems 1 and 3 can be transformed into an equivalent completed rule.*

The construction of the completed rule proceeds similarly as in the proof of Theorem 4. Let  $\hat{\mathcal{G}}_A$  be the cut-closure of  $\mathcal{G}_A^+ \cup \mathcal{G}_A^-$  with cut-formula  $A$  and without  $\mathcal{G}_A^+ \cup \mathcal{G}_A^-$ , and let  $\downarrow \hat{\mathcal{G}}_A$  be the set of minimal elements of  $\hat{\mathcal{G}}_A$  wrt. the application of **w**. Note that if  $r$  is cyclic, then  $\hat{\mathcal{G}}_A$  is infinite, but  $\downarrow \hat{\mathcal{G}}_A$  is finite. Let  $\downarrow \hat{\mathcal{G}}_A^+$  be the set of hypersequents in  $\downarrow \hat{\mathcal{G}}_A$  that do not contain  $A^\perp$ , and let  $\mathcal{G}_A^{\text{cut}}$  be obtained from  $\downarrow \hat{\mathcal{G}}_A^+$  by deleting  $A$  everywhere. Let  $r'$  and  $r''$  be the rules obtained from  $r$  by replacing the premises  $\mathcal{G}_A^+ \cup \mathcal{G}_A^-$  with  $\downarrow \hat{\mathcal{G}}_A$  and  $\mathcal{G}_A^{\text{cut}}$ , respectively. Then it remains to show that  $r$ ,  $r'$ , and  $r''$  are equivalent. (Note that  $r' \Rightarrow r''$  follows by setting  $A = \perp$  and  $A^\perp = 1$  since 1 behaves as  $\top$  in the presence of **w**.)

**Example 5.** By applying the procedure sketched above to **menace**, we get the contraction rule **c**, while **cancel** yields the (contradictory) rule  $\frac{}{\vdash \Gamma}$ . It is easy to see that the obtained rules are equivalent to **menace** and **cancel**, respectively.

Let us write **HMALL**<sup>ext</sup> to denote **HMALL** extended with any set of completed rules. We now outline a syntactic proof of cut-elimination for **HMALL**<sup>ext</sup> (see [4] for a semantic proof in the single-conclusion setting). As usual, the *length*  $|d|$  of a (hyper)sequent derivation  $d$  is the maximal number of inference rules + 1 occurring on any branch of  $d$ . The *complexity*  $|A|$  of a formula  $A$  is the number of occurrences of its connectives. The *cut rank*  $\rho(d)$  of  $d$  is the maximal complexity of the cut-formulas in  $d$  plus 1. Clearly  $\rho(d) = 0$  if  $d$  has no cuts.

**Lemma 4.** *Let  $d_+$  and  $d_-$  be derivations in **HMALL**<sup>ext</sup> such that*

- (i)  $d_+$  is a derivation of  $\mathcal{H} \vdash \Gamma, A$  and  $\rho(d_+) \leq |A|$ , and
- (ii)  $d_-$  is a derivation of  $\mathcal{H} \vdash \Sigma, A^\perp$  and  $\rho(d_-) \leq |A|$ , and

- (iii)  $A$  is a compound formula and one of  $d_+$  or  $d_-$  ends with a logical rule introducing  $A$ .

Then we can find a derivation  $d$  in  $\text{HMALL}^{\text{ext}}$  of  $\mathcal{H} \mid \vdash \Gamma, \Sigma$  with  $\rho(d) \leq |A|$ .

Of course, one could derive  $\mathcal{H} \mid \vdash \Gamma, \Sigma$  by applying cut, but the resulting derivation would then have cut-rank  $|A| + 1$ .

*Proof.* Assume w.l.o.g. that  $d_-$  ends with a logical rule introducing  $A^\perp$ . Consider a derivation  $d'_+$  of  $\mathcal{H} \mid \vdash \Gamma_1, A^{\lambda_1} \mid \dots \mid \vdash \Gamma_n, A^{\lambda_n}$  with  $\rho(d'_+) \leq |A|$  where  $A^\lambda$  stands for  $A, \dots, A$  ( $\lambda$  times). We prove, by induction on  $|d'_+|$ , that one can find a derivation of  $\mathcal{H} \mid \vdash \Gamma_1, \Sigma^{\lambda_1} \mid \dots \mid \vdash \Gamma_n, \Sigma^{\lambda_n}$  with cut-rank  $\leq |A|$ . This is required to deal with internal and external contraction rules. If  $d'_+$  ends in an axiom  $(\text{ax}, 1, \top)$  then we are done. Otherwise, let  $r$  be the last inference rule in  $d'_+$ .

- (a) If  $r$  acts only on  $\mathcal{H}$  or  $r$  is  $\text{ew}$ ,  $\text{ec}$  or  $\perp$  then the claim follows by the induction hypothesis and an application of  $r$ . The same holds when  $r$  is a logical rule which does not introduce a cut formula  $A$ .
- (b) Suppose that  $r$  is an introduction rule for  $A$ . The claim easily follows by applying cut to the premise(s) of  $r$  and to the premise(s) of the last rule applied in  $d_-$  (which is a logical rule introducing  $A^\perp$ ). The cut-formula(s) of the newly introduced cut is (are) the auxiliary formula(s) of  $A$  and therefore the resulting derivation has cut-rank  $\leq |A|$ .
- (c) If  $r$  is any completed rule then the properties of (NFV) and (LC) allow the cut to be shifted upward over the rule premises. The claim follows by the induction hypothesis and an application of  $r$ .  $\square$

**Lemma 5.** *Let  $d_+$  and  $d_-$  be derivations in  $\text{HMALL}^{\text{ext}}$  such that the hypothesis (i) and (ii) of Lemma 4 hold. Then we can find a derivation  $d$  in  $\text{HMALL}^{\text{ext}}$  of  $\mathcal{H} \mid \vdash \Gamma, \Sigma$  with  $\rho(d) \leq |A|$ .*

*Proof.* Proceed similarly to that of Lemma 4. If the last inference rule applied is any rule other than a logical rule introducing a cut-formula  $A$ , the proof proceeds as in cases (a) and (c). Otherwise the claim follows by induction hypothesis, an application of  $r$  and Lemma 4.  $\square$

**Theorem 5 (Cut-elimination).**  *$\text{HMALL}$  extended with any set of completed rules admits cut-elimination.*

*Proof.* Let  $d$  be a derivation in  $\text{HMALL}^{\text{ext}}$  with  $\rho(d) > 0$ . The proof proceeds by a double induction on  $(\rho(d), \# \rho(d))$ , where  $\# \rho(d)$  is the number of cuts in  $d$  with cut-rank  $\rho(d)$ . Consider an uppermost application of cut in  $d$  with cut-rank  $\rho(d)$ . By applying Lemma 5 to its premises either  $\rho(d)$  or  $\# \rho(d)$  decreases.  $\square$

**Remark 4.** Let  $S$  be any set of hypersequents that (1) contain only atomic formulas, (2) are closed under cut, and (3) do not contain any hypersequent of the form  $\mathcal{H} \mid \vdash \Gamma, A, A^\perp$ . Then our cut-elimination proof also allows the elimination of cuts from  $\text{HMALL}^{\text{ext}}$  proofs whose leaves are either axioms  $(\text{ax}, 1, \top)$  or hypersequents belonging to  $S$ . This establishes a stronger form of cut-elimination.

## 7 A Case Study: Abelian and Łukasiewicz Logics

Consider the axiom  $\text{inv} : A \otimes (A \multimap 1)$ . Writing  $A^{-1}$  for  $A \multimap 1$  and noting that  $\text{inv}$  is equivalent to  $1 \multimap (A \otimes A^{-1})$ , one immediately sees that  $\text{inv}$  states that  $A^{-1}$  is the inverse element of  $A$  with unit 1. Adding  $\text{inv}$  to  $\text{IMALL}$  yields a contradiction as  $\vdash_{\text{IMALL}} 0 \otimes 0^{-1} \multimap A$ . However,  $\text{IMALL} + \text{inv}$  without  $\top$  and 0 is consistent. Indeed, the logic  $\text{AL} = \text{IMALL} \setminus \{\top, 0, \perp\} + \text{inv}$  has the *lattice ordered Abelian groups* as models (see, e.g., [18]), and for this reason is called *Abelian logic* [15,16].

Observe that  $\text{inv} \in \mathcal{P}_3$ , but unfortunately  $\text{inv} \notin \mathcal{P}'_3$ . Hence the method developed in [4] does not apply. What can we do?

**Episode 1: The Power of Multiple Conclusion.** First, we move to the classical setting. Let  $\text{MALL}^-$  denote  $\text{MALL}$  without  $\top$  and 0. In  $\text{MALL}^-$ , the axiom  $\text{inv}$  can be derived from two more basic axioms  $\perp \multimap 1$  and  $A \wp B \multimap A \otimes B$ , that we call  $\text{mixax}$  and  $\text{mixinv}$ , respectively. The interesting observation is that these two axioms are not only sufficient, but also necessary:

**Theorem 6.** *For every AL-formula  $A$ , we have that AL proves  $A$  if and only if  $\text{MALL}^- + \text{mixax} + \text{mixinv}$  proves  $A$ .*

*Proof.* ( $\Rightarrow$ ) Observe that  $\text{MALL}^- + \text{mixax} + \text{mixinv}$  proves  $\text{inv}$ . ( $\Leftarrow$ ) Prove by induction that if  $\text{MALL}^- + \text{mixax} + \text{mixinv}$  proves  $\vdash A_1, \dots, A_n$ , then  $\vdash_{\text{AL}} \otimes_i A_i^a$ , where  $A_i^a$  is obtained from  $A_i$  by replacing  $\perp$  by 1,  $\wp$  by  $\otimes$ , and  $a^\perp$  by  $a \multimap 1$ .  $\square$

There are two important observations to make: First,  $\text{mixax}$  and  $\text{mixinv}$  are both in  $\mathcal{N}_2$  and therefore Theorem [1] applies. Second,  $\text{mixinv}$  is not an intuitionistic axiom. Thus, the shift to the multiple conclusion setting is crucial.

**Episode 2: The Structuralization of the Axioms.** Let us now apply Theorem [1] to produce structural rules equivalent to  $\text{mixax}$  and  $\text{mixinv}$ :

$$\text{mixax} : \perp \multimap 1 \quad \rightsquigarrow \quad \frac{}{\vdash 1, 1} \quad \rightsquigarrow \quad \frac{\vdash \Gamma, \perp \quad \vdash \Delta, \perp}{\vdash \Gamma, \Delta} \quad \rightsquigarrow \quad \text{mix} \frac{\vdash \Gamma \quad \vdash \Delta}{\vdash \Gamma, \Delta}$$

$$\text{mixinv} : A \wp B \multimap A \otimes B \quad \rightsquigarrow \quad \frac{}{\vdash A^\perp \otimes B^\perp, A \otimes B} \quad \rightsquigarrow \quad \frac{\vdash \Gamma, A^\perp, B^\perp \quad \vdash \Delta, A, B}{\vdash \Gamma, \Delta}$$

The result for  $\text{mixinv}$  is a cyclic rule. One formula variable (e.g.  $B$ ) can be removed using the procedure in the proof of Theorem [4] thus obtaining the *cancel* rule of Example [3]. However, there is no way to proceed further to obtain any equivalent completed rule.

**Episode 3: Why Not Logical Rules?** Instead of transforming  $\text{mixinv}$  into a structural rule, let us apply Theorem [2] to obtain a new logical rule for the  $\otimes$ -connective. Indeed:

$$\text{mixinv} : A \wp B \multimap A \otimes B \quad \rightsquigarrow \quad \frac{}{\vdash A^\perp \otimes B^\perp, A \otimes B} \quad \rightsquigarrow \quad \otimes' \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \otimes B}$$

The rule  $\otimes$  is then derivable from  $\text{mix}$  and  $\otimes'$ , and can be removed. We now have a system in which only the cut rule does not have the subformula property. However, every attempt to eliminate cut will introduce the *cancel* rule, and every attempt to eliminate the *cancel* rule will introduce cut — a dead end.

**Episode 4: A New Hope.** By inspecting the failed attempts for cut/cancel-elimination, we find a concrete counterexample, that is the excluded middle law  $A \oplus A^\perp$ . It belongs to  $\mathcal{P}_1$ , so why not transforming it into a structural rule? By applying our procedure, we obtain the equivalent rule **em** in Figure 4. Hence we can safely add **em** without changing the logical strength of the system. Do we now have cut and cancel elimination?

**Episode 5: Counterexample Strikes Back.** Unfortunately, no. We find two more counterexamples:  $\perp$  and  $(A \& 1) \oplus (A^\perp \& 1)$ . The former is equivalent to the nullary mix rule  $\text{mix}_0 \frac{}{\vdash}$ , while the latter, which we call **splax**, is in  $\mathcal{P}'_3$ . We can therefore apply Theorems 3 and 4 to obtain a hyperstructural rule:

$$\text{splax} \rightsquigarrow \frac{}{\mathcal{H} \mid \vdash A \mid \vdash A^\perp} \rightsquigarrow \frac{\mathcal{H} \mid \vdash \Gamma, A^\perp \quad \mathcal{H} \mid \vdash \Delta, A}{\mathcal{H} \mid \vdash \Gamma \mid \vdash \Delta} \rightsquigarrow \text{split} \frac{\mathcal{H} \mid \vdash \Gamma, \Delta}{\mathcal{H} \mid \vdash \Gamma \mid \vdash \Delta}$$

This leads us to switch from  $\text{MALL}^-$  to its hypersequent counterpart  $\text{HMALL}^-$ , and accordingly generalize **mix** and  $\otimes'$  to their hypersequent counterparts, which we still call **mix** and  $\otimes'$ . A good news is that the previous rule **em** is redundant in presence of **split**.

**Episode 6: The Return of Cut Elimination.** We have finally arrived at the system  $\text{HAL} = \text{HMALL}^- + \text{mix} + \text{mix}_0 + \otimes' + \text{split}$  for **AL** introduced in 14. A concrete cut-elimination procedure which relies on the invertibility of logical rules is contained in 13. Hence, we have

**Theorem 7.** *HAL admits cut-elimination.*

**The General Pattern.** Our development so far suggests the following heuristics to find a cut-free calculus for a given logic.

1. Convert axioms into structural/logical rules having the subformula property.
2. If we obtain a cut-free system, we are done. Otherwise, find a counterexample  $A$  by inspecting the failure of cut-elimination.
3. If  $A \in \mathcal{N}_2$  or  $A \in \mathcal{P}'_3$ , apply Theorem 1 or 3 accordingly and go to 2. (Otherwise, we get stuck.)

One can think of it analogous to the Knuth-Bendix algorithm for obtaining a confluent rewriting system out of a set of equations. But the analogy is only shallow, since ours is neither complete nor gives rise to a semi-decision procedure.

**Another Example.** A similar situation arises for infinite-valued Łukasiewicz logic **L**. This logic is axiomatized by adding  $((A \multimap B) \multimap B) \multimap ((B \multimap A) \multimap A)$  to **IMALL**. In **L**, the additive disjunction is definable from linear implication:  $((A \multimap B) \multimap B) \multimap A \oplus B$ . Hence the above axiom just states the commutativity of  $\oplus$ . The axiom is in  $\mathcal{N}_3$ , so cannot be dealt with by our general method.

**Episode 7: Defining a New Connective.** It is known that **L** can be faithfully interpreted in Abelian logic. In particular, the Łukasiewicz implication  $A \stackrel{\text{L}}{\multimap} B$  can be defined by  $(A \stackrel{\text{L}}{\multimap} B) \multimap ((A \multimap B) \&_1)$  inside Abelian logic 14. Now, each

of the two directions yields logical rules via Theorem 2 (and the equivalences  $1 \circ \multimap \perp, A \otimes B \circ \multimap A \wp B$ ): one for  $A \stackrel{\mathbf{L}}{\Rightarrow} B$ , and two for its DeMorgan dual  $A \stackrel{\mathbf{L}}{\Leftarrow} B$  2

$$\stackrel{\mathbf{L}}{\Rightarrow} \frac{\vdash \Gamma, A^\perp, B \quad \vdash \Gamma}{\vdash \Gamma, A \stackrel{\mathbf{L}}{\Rightarrow} B} \quad \stackrel{\mathbf{L}}{\Leftarrow} 1 \frac{\vdash \Gamma}{\vdash \Gamma, A \stackrel{\mathbf{L}}{\Leftarrow} B} \quad \stackrel{\mathbf{L}}{\Leftarrow} 2 \frac{\vdash \Gamma, A, B^\perp}{\vdash \Gamma, A \stackrel{\mathbf{L}}{\Leftarrow} B}$$

These are the implication rules of the cut-free calculus for  $\mathbf{L}$  introduced in 14.

## 8 Concluding Remarks

**Relation to Algebraic Completions.** Given a (hyper)sequent calculus  $\mathbf{H}$ , we denote by  $\mathbf{H}^\infty$  its extension with infinitary conjunction  $\&_{i \in I} A_i$  and disjunction  $\oplus_{i \in I} A_i$ , where  $I$  is an arbitrary index set, together with suitable logical rules generalizing  $\&$  and  $\oplus$ . We say that  $\mathbf{H}^\infty$  is *conservative over*  $\mathbf{H}$  if for any set  $\mathcal{S} \cup \{A\}$  of  $\mathbf{H}$ -formulas,  $\mathcal{S} \vdash_{\mathbf{H}^\infty} A$  implies  $\mathcal{S} \vdash_{\mathbf{H}} A$ .

Since cut-elimination is a canonical way to show conservativity, one can expect that if all rules of  $\mathbf{H}$  are “good”, i.e. admit a suitably strong form of cut-elimination, then  $\mathbf{H}^\infty$  is conservative over  $\mathbf{H}$ . The work in 3 proves that any  $\mathcal{N}_2$ -axiom  $\phi$  can be transformed into equivalent acyclic structural rules if and only if  $\text{IMALL}^\infty + \phi$  is conservative over  $\text{IMALL} + \phi$ . We conjecture that the same holds for  $\mathcal{P}'_3$  axioms, both in single and multiple-conclusion settings. Now the question is: do Abelian logic and Łukasiewicz logic admit such a strong form of cut-elimination which imply conservativity?

A negative answer arises from the following two facts:

- (i)  $\mathbf{H}^\infty$  is conservative over  $\mathbf{H}$  if and only if the class  $\mathfrak{V}(\mathbf{H})$  of algebras corresponding to  $\mathbf{H}$  is *closed under completions*, in the sense that any  $V \in \mathfrak{V}(\mathbf{H})$  can be embedded into a complete algebra in  $\mathfrak{V}(\mathbf{H})$  3, Prop. 5.9].
- (ii) Both the class of lattice ordered Abelian groups and the class of MV-algebras (the algebraic semantics of  $\mathbf{L}$ ) are *not* closed under completions, see, e.g., 11].

Therefore, though the calculi of 14 admit cut-elimination, their rules, which we extracted out of  $\mathcal{P}_3$  and  $\mathcal{N}_3$  axioms, are not “good” enough to ensure conservativity. This contrasts with the result for  $\mathcal{N}_2$  axioms (and  $\mathcal{P}'_3$  ones, if our conjecture is true, cf. Remark 4).

**Relation to Categories.** In the category theoretical setting, Abelian logic lives in *compact closed categories*, whose canonical instance is the category of finite dimensional vector spaces over a fixed field. Indeed,  $\text{MALL}^- \setminus \{\&, \oplus_1, \oplus_2\} + \text{mix} + \otimes'$  is the calculus given by 22 which aims to capture morphisms in a freely generated compact closed category via sequent proofs.

On the other hand, Abelian logic (and its hypersequent calculus  $\text{HAL}$ ) also incorporates  $\&$  and  $\oplus$ , which in the world of categories are usually interpreted as binary products and coproducts. Thus a natural question is whether there is

---

<sup>2</sup>  $A \stackrel{\mathbf{L}}{\Leftarrow} B$  corresponds to the left occurrence of  $A \stackrel{\mathbf{L}}{\Rightarrow} B$  in the two-sided sequent calculus (cf. Remark 1), not to be confused with the Łukasiewicz negation of  $A \stackrel{\mathbf{L}}{\Rightarrow} B$ .

a nicely behaved equivalence relation on proofs in HAL such that the equivalence classes are the morphisms in the free compact closed category with binary products and coproducts (as it is achieved by the rule permutations in the sequent calculus for multiplicative linear logic and star-autonomous categories [10]).

Two observations: First, if we add initial and terminal objects, and therefore get all finite products and coproducts, we get a collapse: products and coproducts coincide [6]. In terms of logic we have  $A \& B \cong A \oplus B$ , and therefore an inconsistency. This is not a surprise: we have seen above that adding 0 and  $\top$  to AL makes the logic inconsistent. Second, sequents are category theoretically well studied in the form of polycategories [23], but it has not yet been investigated what *hypersequents* mean in terms of categories.

## References

1. Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation* 2(3), 297–347 (1992)
2. Avron, A.: Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artif. Intell.* 4, 225–248 (1991)
3. Ciabattoni, A., Galatos, N., Terui, K.: Algebraic proof theory for substructural logics: Cut-elimination and completions (2008) (submitted)
4. Ciabattoni, A., Galatos, N., Terui, K.: From axioms to analytic rules in nonclassical logics. In: LICS, pp. 229–240 (2008)
5. Gispert, J., Torrens, A.: Axiomatic extensions of IMT3 logic. *Studia Logica* 81(3), 311–324 (2005)
6. Houston, R.: Finite products are biproducts in a compact closed category. *Journal of Pure and Applied Algebra* 212(2) (2008)
7. Kracht, M.: Power and weakness of the modal display calculus. In: *Proof theory of modal logic*, pp. 93–121. Kluwer, Dordrecht (1996)
8. Lamarche, F.: On the algebra of structural contexts. Accepted at *Mathematical Structures in Computer Science* (2001)
9. Lamarche, F., Retoré, C.: Proof nets for the Lambek-calculus — an overview. In: Abrusci, V.M., Casadio, C. (eds.) *Proceedings of the Third Roma Workshop Proofs and Linguistic Categories*, pp. 241–262. CLUEB, Bologna (1996)
10. Lamarche, F., Straßburger, L.: From proof nets to the free \*-autonomous category. *Logical Methods in Computer Science* 2(4:3), 1–44 (2006)
11. Litak, T., Kowalski, T.: Completions of GBL algebras: negative results. *Algebra Universalis* 58, 373–384 (2008)
12. Metcalfe, G.: A sequent calculus for constructive logic with strong negation as a substructural logic. To appear in *Bulletin of the Section of Logic*
13. Metcalfe, G.: Proof theory for Casari’s comparative logics. *J. Log. Comput.* 16(4), 405–422 (2006)
14. Metcalfe, G., Olivetti, N., Gabbay, D.M.: Sequent and hypersequent calculi for Abelian and Lukasiewicz logics. *ACM Trans. Comput. Log.* 6(3), 578–613 (2005)
15. Meyer, R., Slaney, J.: Abelian logic (from A to Z). In: *Paraconsistent Logic: Essays on the Inconsistent*, pp. 245–288 (1989)
16. Meyer, R., Slaney, J.: Still adorable, Paraconsistency: The Logical Way to the Inconsistent. In: *Proceedings of the world congress on paraconsistency held in Sao Paulo*, pp. 241–260 (2002)



17. Miller, D.: Forum: A multiple-conclusion specification logic. *Theoretical Computer Science* 165, 201–232 (1996)
18. Kowalski, T., Galatos, N., Jipsen, P., Ono, H.: *Residuated Lattices: an algebraic glimpse at substructural logics*. Elsevier, Amsterdam (2007)
19. Negri, S.: Proof analysis in modal logics. *Journal of Philosophical Logic* 34(5-6), 507–544 (2005)
20. Sambin, G., Battilotti, G., Faggian, C.: Basic logic: Reflection, symmetry, visibility. *J. Symb. Log.* 65(3), 979–1013 (2000)
21. Schellinx, H.: Some syntactical observations on linear logic. *Journal of Logic and Computation* 1(4), 537–559 (1991)
22. Shirahata, M.: A sequent calculus for compact closed categories. Unpublished (2000)
23. Szabo, M.E.: Polycategories. *Comm. Alg.* 3, 663–689 (1975)
24. Terui, K.: Which structural rules admit cut elimination? An algebraic criterion. *Journal of Symbolic Logic* 72(3), 738–754 (2007)

# EXPTIME Tableaux for the Coalgebraic $\mu$ -Calculus<sup>\*</sup>

Corina Cîrstea<sup>1</sup>, Clemens Kupke<sup>2</sup>, and Dirk Pattinson<sup>2</sup>

<sup>1</sup> School of Electronics and Computer Science, University of Southampton

<sup>2</sup> Department of Computing, Imperial College London

**Abstract.** The coalgebraic approach to modal logic provides a uniform framework that captures the semantics of a large class of structurally different modal logics, including e.g. graded and probabilistic modal logics and coalition logic. In this paper, we introduce the coalgebraic  $\mu$ -calculus, an extension of the general (coalgebraic) framework with fixpoint operators. Our main results are completeness of the associated tableau calculus and EXPTIME decidability. Technically, this is achieved by reducing satisfiability to the existence of non-wellfounded tableaux, which is in turn equivalent to the existence of winning strategies in parity games. Our results are parametric in the underlying class of models and yield, as concrete applications, previously unknown complexity bounds for the probabilistic  $\mu$ -calculus and for an extension of coalition logic with fixpoints.

## 1 Introduction

The extension of a modal logic with operators for least and greatest fixpoints leads to a dramatic increase in expressive power [1]. The paradigmatic example is of course the modal  $\mu$ -calculus [10]. In the same way that the  $\mu$ -calculus extends the modal logic  $K$ , one can freely add fixpoint operators to any propositional modal logic, as long as modal operators are monotone. Semantically, this poses no problems, and the interpretation of fixpoint formulas can be defined in a standard way in terms of the semantics of the underlying modal logic.

This apparent simplicity is lost once we move from semantics to syntax: completeness and complexity even of the modal  $\mu$ -calculus are all but trivial [204], and  $\mu$ -calculi arising from other monotone modal logics are largely unstudied, with the notable exception of the graded  $\mu$ -calculus [12]. Here, we improve on this situation, not by providing a new complexity result for a specific fixpoint logic, but by providing a generic and uniform treatment of modal fixpoint logics on the basis of *coalgebraic semantics*. This allows for a generic and uniform treatment of a large class of modal logics and replaces the investigation of a concretely given logic with the study of *coherence conditions* that mediate between the axiomatisation and the (coalgebraic) semantics. The use of coalgebras conveniently abstracts the details of a concretely given class of models, which is replaced by the class of coalgebras for a(n unspecified) endofunctor on sets. Specific choices for this endofunctor then yield specific model classes, such as the class of all Kripke frames or probabilistic transition systems. A property such as completeness or complexity of a specific logic is then automatic once the coherence conditions are satisfied. As it turns out, even *the same* coherence conditions that guarantee completeness

---

<sup>\*</sup> Partially supported by grant EP/F031173/1 from the UK EPSRC.

and decidability of the underlying modal logic entail the same properties of the ensuing  $\mu$ -calculus. This immediately provides us with a number of concrete examples: as instances of the generic framework, we obtain not only the known EXPTIME bounds, both for the modal and the graded  $\mu$ -calculus [4][12], but also previously unknown EXPTIME bounds for the probabilistic and monotone  $\mu$ -calculus, and for an extension of coalition logic [15] with fixpoint operators.

Our main technical results are a syntactical characterisation of satisfiability in terms of (non-)existence of closed tableaux and a game-theoretic characterisation of satisfiability that yields an EXPTIME upper bound for the satisfiability problem. Along the way, we establish a small model theorem. We start by describing a parity game that characterizes model checking for the coalgebraic  $\mu$ -calculus. As in the model-checking game for the modal  $\mu$ -calculus (see e.g. [18]), we allow greatest and least fixpoints to be unfolded ad libitum. Truth of a formula in a particular state of a model then follows, if only greatest fixpoints are unfolded infinitely often on the top level along infinite paths. This condition can be captured by a parity condition. The same technique is employed in the construction of tableaux, which we conceptualise as finite directed graphs: closed tableaux witness unsatisfiability of the root formula, provided that along any infinite tableau path one can construct an infinite sequence of formulas (a “trace”) that violates the parity condition. In particular, closed tableaux are finitely represented proofs of the unsatisfiability of the root formula. Soundness of the tableau calculus is established by showing that a winning strategy in the model checking game precludes existence of a closed tableau. An EXPTIME upper bound for decidability is then established with the help of tableau games, where the adversary chooses a tableau rule, and the player claiming satisfiability chooses one conclusion which effectively constructs a path in a tableau. In order to turn this tableau game into a parity game we combine the game board with the transition function of a deterministic parity word automaton. This automaton checks that on any given play, i.e., on any tableau path, there exists no trace that violates the parity condition. We prove adequacy of the tableau game by constructing a satisfying model from a winning strategy in the tableau game, which makes crucial use of the coherence conditions between the axiomatisation and the coalgebraic semantics. This allows us to determine satisfiability of a fixpoint formula by deciding the associated (parity) tableau game, and the announced EXPTIME upper bound follows once we can ensure that legality of moves in the tableau game can be decided in exponential time.

**Related Work.** Our treatment is inspired by [14][19][17], but we note some important differences. In contrast to [14], we use parity games that directly correspond to tableaux, together with parity automata to detect bad traces. Moreover, owing to the generality of the coalgebraic framework, the model construction here needs to super-impose a coalgebra structure on the relation induced by a winning strategy. This construction is necessarily different from [17], since we cannot argue in terms of modal rank in the presence of fixpoints. Coalgebraic fixpoint logics are also treated in [19], where an automata theoretic characterisation of satisfiability is presented. We add to this picture by providing complexity results and a complete tableau calculus. Moreover, we use standard syntax for modal operators, which allows us to subsume for instance the graded  $\mu$ -calculus that cannot be expressed in terms of the  $\nabla$ -operator used in *op.cit.*

## 2 The Coalgebraic $\mu$ -Calculus

To keep our treatment fully parametric in the underlying (modal) logic, we define the syntax of the coalgebraic  $\mu$ -calculus relative to a (fixed) modal similarity type, that is, a set  $\Lambda$  of modal operators with associated arities. Throughout, we fix a denumerable set  $\mathbb{V}$  of propositional variables. We will only deal with formulas in negation normal form and abbreviate  $\overline{\Lambda} = \{\overline{\heartsuit} \mid \heartsuit \in \Lambda\}$  and  $\overline{\mathbb{V}} = \{\overline{p} \mid p \in \mathbb{V}\}$ . The arity of  $\overline{\heartsuit} \in \overline{\Lambda}$  is the same as that of  $\heartsuit$ . The set  $\mathcal{F}(\Lambda)$  of  $\Lambda$ -formulas is given by the grammar

$$A, B ::= p \mid \overline{p} \mid A \vee B \mid A \wedge B \mid \heartsuit(A_1, \dots, A_n) \mid \mu p. A \mid \nu p. A$$

where  $p \in \mathbb{V}$ ,  $\heartsuit \in \Lambda \cup \overline{\Lambda}$  is  $n$ -ary and  $\overline{p}$  does not occur in  $A$  in the last two clauses. The sets of free and bound variables of a formula are defined as usual, in particular  $p$  is bound in  $\mu p. A$  and  $\nu p. A$ . Negation  $\overline{\cdot} : \mathcal{F}(\Lambda) \rightarrow \mathcal{F}(\Lambda)$  is given inductively by  $\overline{\overline{p}} = p$ ,  $\overline{A \wedge B} = \overline{A} \vee \overline{B}$ ,  $\overline{\heartsuit(A_1, \dots, A_n)} = \overline{\heartsuit}(\overline{A_1}, \dots, \overline{A_n})$  and  $\overline{\mu p. A} = \nu p. \overline{A}$  [ $\overline{p} := p$ ] and the dual clauses for  $\vee$  and  $\nu$ . If  $S$  is a set of formulas, then the collection of formulas that arises by prefixing elements of  $S$  by one layer of modalities is denoted by  $(\Lambda \cup \overline{\Lambda})(S) = \{\heartsuit(S_1, \dots, S_n) \mid \heartsuit \in \Lambda \cup \overline{\Lambda} \text{ } n\text{-ary}, S_1, \dots, S_n \in S\}$ . A *substitution* is a mapping  $\sigma : \mathbb{V} \rightarrow \mathcal{F}(\Lambda)$  and  $A\sigma$  is the result of replacing all free occurrences of  $p \in \mathbb{V}$  in  $A$  by  $\sigma(p)$ .

On the semantical side, parametricity is achieved by adopting coalgebraic semantics: formulas are interpreted over  $T$ -coalgebras, where  $T$  is an (unspecified) endofunctor on sets, and we recover the semantics of a large number of logics in the form of specific choices for  $T$ . To interpret the modal operators  $\heartsuit \in \Lambda$ , we require that  $T$  extends to a  $\Lambda$ -structure and comes with a predicate lifting, that is, a natural transformation of type  $[[\heartsuit]] : 2^n \rightarrow 2 \circ T^{\text{op}}$  for every  $n$ -ary modality  $\heartsuit \in \Lambda$ , where  $2 : \text{Set} \rightarrow \text{Set}^{\text{op}}$  is the contravariant powerset functor. In elementary terms, this amounts to assigning a set-indexed family of functions  $([[\heartsuit]]_X : \mathcal{P}(X)^n \rightarrow \mathcal{P}(TX))_{X \in \text{Set}}$  to every  $n$ -ary modal operator  $\heartsuit \in \Lambda$  such that  $(Tf)^{-1} \circ [[\heartsuit]]_X(A_1, \dots, A_n) = [[\heartsuit]]_Y(f^{-1}(A_1), \dots, f^{-1}(A_n))$  for all functions  $f : Y \rightarrow X$ . If  $\heartsuit \in \Lambda$  is  $n$ -ary, we put  $[[\overline{\heartsuit}]]_X(A_1, \dots, A_n) = (TX) \setminus [[\heartsuit]]_X(X \setminus A_1, \dots, X \setminus A_n)$ . We usually denote a structure just by the endofunctor  $T$  and leave the definition of the predicate liftings implicit. A  $\Lambda$ -structure is *monotone* if, for all sets  $X$  we have that  $[[\heartsuit]]_X(A_1, \dots, A_n) \subseteq [[\heartsuit]]_X(B_1, \dots, B_n)$  whenever  $A_i \subseteq B_i$  for all  $i = 1, \dots, n$ .

In the coalgebraic approach, the role of frames is played by  $T$ -coalgebras, i.e. pairs  $(C, \gamma)$  where  $C$  is a (state) set and  $\gamma : C \rightarrow TC$  is a (transition) function. A  $T$ -model is a triple  $(C, \gamma, \sigma)$  where  $(C, \gamma)$  is a  $T$ -coalgebra and  $\sigma : \mathbb{V} \rightarrow \mathcal{P}(C)$  is a valuation (we put  $\sigma(\overline{p}) = C \setminus \sigma(p)$ ). For a monotone  $T$  structure and a  $T$ -model  $M = (C, \gamma, \sigma)$ , the *truth set*  $[[A]]_M$  of a formula  $A \in \mathcal{F}(\Lambda)$  w.r.t.  $M$  is given inductively by

$$\begin{aligned} [[p]]_M &= \sigma(p) & [[\overline{p}]]_M &= C \setminus \sigma(p) & [[\mu p. A]]_M &= \text{LFP}(A_p^M) & [[\nu p. A]]_M &= \text{GFP}(A_p^M) \\ & & & & [[\heartsuit(A_1, \dots, A_n)]_M &= \gamma^{-1} \circ [[\heartsuit]]_C([[A_1]]_M, \dots, [[A_n]]_M) \end{aligned}$$

where  $\text{LFP}(A_p^M)$  and  $\text{GFP}(A_p^M)$  are the least and greatest fixpoint of the monotone mapping  $A_p^M : \mathcal{P}(C) \rightarrow \mathcal{P}(C)$  defined by  $A_p^M(X) = [[A]]_{(C, \gamma, \sigma')}$  with  $\sigma'(q) = \sigma(q)$  for  $q \neq p$  and  $\sigma'(p) = X$ . We write  $M, c \models A$  if  $c \in [[A]]_M$  to denote that  $A$  is satisfied

at  $c$ . A formula  $A \in \mathcal{F}(\Lambda)$  is *satisfiable* w.r.t. a given  $\Lambda$ -structure  $T$  if there exists a  $T$ -model  $M$  such that  $\llbracket A \rrbracket_M \neq \emptyset$ . The mappings  $A_p^M$  are indeed monotone in case of a monotone  $\Lambda$ -structure, which guarantees the existence of fixpoints.

*Example 1.* 1.  $T$ -coalgebras  $(C, \gamma : C \rightarrow \mathcal{P}(C))$  for  $TX = \mathcal{P}(X)$  are Kripke frames. If  $\Lambda = \{\Box\}$  for  $\Box$  unary and  $\overline{\Box} = \Diamond$ ,  $\mathcal{F}(\Lambda)$  are the formulas of the modal  $\mu$ -calculus [10], and the structure  $\llbracket \Box \rrbracket_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\}$  gives its semantics.

2. The syntax of the graded  $\mu$ -calculus [12] is given (modulo an index shift) by the similarity type  $\Lambda = \{\langle n \rangle \mid n \geq 0\}$  where  $\langle n \rangle = [n]$ , and  $\langle n \rangle A$  reads as “ $A$  holds in more than  $n$  successors”. In contrast to *op. cit.* we interpret the graded  $\mu$ -calculus over multigraphs, i.e. coalgebras for the functor  $B$  (below) that extends to a structure via

$$B(X) = \{f : X \rightarrow \mathbb{N} \mid \text{supp}(f) \text{ finite}\} \quad \llbracket \langle n \rangle \rrbracket_X(A) = \{f \in B(X) \mid \sum_{x \in X} f(x) > n\}.$$

where  $\text{supp}(f) = \{x \in X \mid f(x) \neq 0\}$  is the support of  $f$ . Note that this semantics differs from the Kripke semantics for both graded modal logic [7] and the graded  $\mu$ -calculus, but both types of semantics induce the same satisfiability problem: Kripke frames are multigraphs where each edge has multiplicity one, and the unravelling of a multigraph can be turned into a Kripke frame by inserting the appropriate number of copies of each state. Both transformations preserve satisfiability.

3. The probabilistic  $\mu$ -calculus arises from the similarity type  $\Lambda = \{\langle p \rangle \mid p \in [0, 1] \cap \mathbb{Q}\}$  where  $\langle p \rangle = [p]$  and  $\langle p \rangle \phi$  reads as “ $\phi$  holds with probability at least  $p$  in the next state”. The semantics of the probabilistic  $\mu$ -calculus is given by the structure

$$D(X) = \{\mu : X \rightarrow_f [0, 1] \mid \sum_{x \in X} \mu(x) = 1\} \quad \llbracket \langle p \rangle \rrbracket_X(A) = \{\mu \in D(X) \mid \sum_{x \in A} \mu(x) \geq p\}$$

where  $\rightarrow_f$  indicates maps with finite support. Coalgebras for  $D$  are precisely image-finite Markov chains, and the finite model property of the coalgebraic  $\mu$ -calculus that we establish later ensures that satisfiability is independent of image-finite semantics.

4. Formulas of coalition logic over a finite set  $N$  of agents [15] arise via  $\Lambda = \{[C] \mid C \subseteq N\}$ , and are interpreted over game frames, i.e. coalgebras for the functor

$$G(X) = \{(f, (S_i)_{i \in N}) \mid \prod_{i \in N} S_i \neq \emptyset, f : \prod_{i \in N} S_i \rightarrow X\}$$

which is a class-valued functor, which however fits with the subsequent development. We think of  $S_i$  as the set of strategies for agent  $i$  and  $f$  is an outcome function. We read  $[C]A$  reads as “coalition  $C$  can achieve  $A$ ”, which is captured by the lifting

$$\llbracket [C] \rrbracket_X(A) = \{(f, (S_i)_{i \in N}) \in G(X) \mid \exists (s_i)_{i \in C} \forall (s_i)_{i \in N \setminus C} (f((s_i)_{i \in N}) \in A)\}$$

that induces the standard semantics of coalition logic.

5. Finally, the similarity type  $\Lambda = \{\Box\}$  of monotone modal logic [2] has a single unary  $\Box$  (we write  $\overline{\Box} = \Diamond$ ) and interpret the ensuing language over monotone neighbourhood frames, that is, coalgebras for the functor / structure

$$\mathcal{M}(X) = \{Y \subseteq \mathcal{P}(X) \mid Y \text{ upwards closed}\} \quad \llbracket \Box \rrbracket_X(A) = \{Y \in \mathcal{M}(X) \mid A \in Y\}$$

which recovers the standard semantics in a coalgebraic setting [8].

It is readily verified that all structures above are indeed monotone.

### 3 The Model-Checking Game

We start by describing a characterisation of model checking in terms of parity games that generalises [18, Theorem 1, Chapter 6] to the coalgebraic setting. The model-checking game is a variant of the one from [3]. A *parity game* played by  $\exists$  (Éloise) and  $\forall$  (Abelard) is a tuple  $\mathcal{G} = (B_{\exists}, B_{\forall}, E, \Omega)$  where  $B = B_{\exists} \cup B_{\forall}$  is the disjoint union of *positions* owned by  $\exists$  and  $\forall$ , respectively,  $E \subseteq B \times B$  indicates the allowed moves, and  $\Omega : B \rightarrow \omega$  is a (parity) map with finite range. An infinite sequence  $(b_0, b_1, b_2, \dots)$  of positions is called *bad* if  $\max\{k \mid k = \Omega(b_i) \text{ for infinitely many } i \in \omega\}$  is odd.

A *play* in  $\mathcal{G}$  is a finite or infinite sequence of positions  $(b_0, b_1, \dots)$  with the property that  $(b_i, b_{i+1}) \in E$  for all  $i$ , i.e. all moves are legal, and  $b_0$  is the *initial position* of the play. A *full play* is either infinite, or a finite play ending in a position  $b_n$  where  $E[b_n] = \{b \in B \mid (b_n, b) \in E\} = \emptyset$ , i.e. no more moves are possible. A finite play is lost by the player who cannot move, and an infinite play  $(b_0, b_1, \dots)$  is lost by  $\exists$  if  $(b_0, b_1, \dots)$  is bad.

A *strategy* in  $\mathcal{G}$  for a player  $P \in \{\exists, \forall\}$  is a function  $s$  that maps plays that end in a position  $b \in B_P$  of  $P$  to a position  $b' \in B$  such that  $(b, b') \in E$  whenever  $E[b] \neq \emptyset$ . Intuitively, a strategy determines a player's next move, depending on the history of the play, whenever the player has a move available. A strategy for a player  $P \in \{\exists, \forall\}$  is called *history-free* if it only depends on the last position of a play. Formally, a *history-free strategy* for player  $P \in \{\exists, \forall\}$  is a function  $s : B_P \rightarrow B$  such that  $(b, s(b)) \in E$  for all  $b \in B_P$  with  $E[b] \neq \emptyset$ . A play  $(b_0, b_1, \dots)$  is *played according to some strategy*  $s$  if  $b_{i+1} = s(b_0 \dots b_i)$  for all  $i$  with  $b_i \in B_P$ . Similarly a play  $(b_0, b_1, \dots)$  is played according to some *history-free strategy*  $s$  if  $b_{i+1} = s(b_i)$  for all  $i$  with  $b_i \in B_P$ . Finally, we say  $s$  is a *winning strategy* from position  $b \in B$  if  $P$  wins all plays with initial position  $b$  that are played according to  $s$ .

We will use the fact that parity games are history-free determined [5][13] and that winning regions can be decided in  $\text{UP} \cap \text{co-UP}$  [9].

**Theorem 2.** *At every position  $b \in B_{\exists} \cup B_{\forall}$  in a parity game  $\mathcal{G} = (B_{\exists}, B_{\forall}, E, \Omega)$  one of the players has a history-free winning strategy. Furthermore, for every  $b \in B_{\exists} \cup B_{\forall}$ , it can be determined in time  $O\left(d \cdot m \cdot \binom{n}{\lfloor d/2 \rfloor}^{\lfloor d/2 \rfloor}\right)$  which player has a winning strategy from position  $b$ , where  $n$ ,  $m$  and  $d$  are the size of  $B$ ,  $E$  and the range of  $\Omega$ , respectively.*

The model checking game is played on both states and formulas, and only the closure of the initial formula, which is assumed to be clean and guarded, is relevant:

**Definition 3.** *A set  $\Gamma \subseteq \mathcal{F}(A)$  of formulas is closed if  $B \in \Gamma$  whenever  $B$  is a subformula of some  $A \in \Gamma$  and  $A[p := \eta p.A] \in \Gamma$  if  $\eta p.A \in \Gamma$ , where  $\eta \in \{\mu, \nu\}$ . The closure of  $\Gamma$  is the smallest closed set  $\text{Cl}(\Gamma)$  for which  $\Gamma \subseteq \text{Cl}(\Gamma)$ .*

*A formula  $A \in \mathcal{F}(A)$  is guarded if, for all subformulas  $\eta p.B$  of  $A$ ,  $p$  only appears in the scope of a modal operator within  $B$ , and  $A$  is clean if every variable is bound at most once in  $A$ . A set of formulas is clean/guarded if this applies to every element.*

In the model checking game, the unfolding of fixpoint formulas gives rise to infinite plays, and we have to ensure that all infinite plays that cycle on an outermost  $\mu$ -variable

are lost by  $\exists$  (who claims that the formula(s) under consideration are satisfied), as this would correspond to the infinite unfolding of a least fixpoint. This is achieved by the parity function.

**Definition 4.** A parity map for a finite, clean set of formulas  $\Gamma$  is a function  $\Omega : \text{Cl}(\Gamma) \rightarrow \omega$  with finite range for which  $\Omega(A) = 0$  unless  $A$  is of the form  $\eta p.B$ ,  $\eta \in \{\mu, \nu\}$ ,  $\Omega(A)$  is odd (even) iff  $A$  is of the form  $\mu p.B$  ( $\nu p.B$ ), and  $\Omega(\eta_1 p_1.B_1) \leq \Omega(\eta_2 p_2.B_2)$  whenever  $\eta_1 p_1.B_1$  is a subformula of  $\eta_2 p_2.B_2$ , where  $\eta_1, \eta_2 \in \{\mu, \nu\}$ .

It is easy to see that every clean set of formulas admits a parity function.

**Lemma 5.** If  $\Gamma \subseteq \mathcal{F}(\Lambda)$  is finite and clean, then  $\Gamma$  admits a parity function whose range is bounded by the cardinality of  $\text{Cl}(\Gamma)$ .

A parity function for  $\Gamma$  defines the following game:

**Definition 6.** Suppose that  $M = (C, \gamma, \sigma)$  is a  $T$ -model,  $\Gamma \subseteq \mathcal{F}(\Lambda)$  is finite, clean and guarded, and  $\Omega$  is a parity map for  $\Gamma$ . The model checking game  $\text{MG}_\Gamma(M)$  is the parity game whose positions and admissible moves are given in the following table,

Position: $b$	Player	Admissible moves: $E[b]$
$(p, c), c \in \sigma(p)$	$\forall$	$\emptyset$
$(p, c), c \notin \sigma(p)$	$\exists$	$\emptyset$
$(\eta p.A(p), c)$ for $\eta \in \{\mu, \nu\}$	$\exists$	$\{(A[p := \eta p.A(p)], c)\}$
$(A_1 \vee A_2, c)$	$\exists$	$\{(A_1, c), (A_2, c)\}$
$(A_1 \wedge A_2, c)$	$\forall$	$\{(A_1, c), (A_2, c)\}$
$(\heartsuit(A_1, \dots, A_n), c)$	$\exists$	$\{(\heartsuit(A_1, \dots, A_n), (U_1, \dots, U_n)) \mid U_1, \dots, U_n \subseteq C, \gamma(c) \in \llbracket \heartsuit \rrbracket_C(U_1, \dots, U_n)\}$
$(\heartsuit(A_1, \dots, A_n), (U_1, \dots, U_n))$	$\forall$	$\{(A_i, c) \mid 1 \leq i \leq n, c \in U_i\}$

where  $p \in V \cup \bar{V}$ ,  $\heartsuit \in \Lambda \cup \bar{\Lambda}$ ,  $A, A_1, \dots, A_n \in \text{Cl}(\Lambda)$  are  $\Lambda$ -formulas,  $c \in C$  are states and  $U_i \subseteq C$  are state sets. The parity function of  $\text{MG}_\Gamma(M)$  is given by  $\Omega'(A, c) = \Omega(A)$  for  $A \in \text{Cl}(\Gamma)$  and  $c \in C$ , and  $\Omega'(-) = 0$  otherwise.

As any two parity functions for a given set of formulas induce the same winning region for both players, we speak of *the* model checking game given by a set of formulas. The announced generalisation of [18, Theorem 1, Chapter 6] now takes the following form:

**Theorem 7.** For  $\Gamma$  finite, clean and guarded, a  $T$ -model  $M = (C, \gamma, \sigma)$ ,  $A \in \text{Cl}(\Gamma)$  and  $c \in C$ ,  $\exists$  has a winning strategy in  $\text{MG}_\Gamma(M)$  from position  $(A, c)$  iff  $M, c \models A$ .

The model checking game is used to show completeness of associated tableau calculi.

## 4 Tableaux for the Coalgebraic $\mu$ -Calculus

The construction of tableaux for the coalgebraic  $\mu$ -calculus relies on a set of rules that provides the glue between syntax and semantics. As we do not commit to a particular semantics, we exhibit coherence conditions that ensure soundness and completeness.

**Definition 8.** A monotone one-step tableau rule for a similarity type  $\Lambda$  is of the form

$$\frac{\Gamma_0}{\Gamma_1 \quad \dots \quad \Gamma_n}$$

where  $\Gamma_0 \in (\Lambda \cup \overline{\Lambda})(V)$  and  $\Gamma_1, \dots, \Gamma_n \subseteq V$ , every propositional variable occurs at most once in  $\Gamma_0$  and all variables occurring in one of the  $\Gamma_i$ 's ( $i > 0$ ) also occur in  $\Gamma_0$ .

Monotone tableau rules do not contain negated propositional variables, which are not needed to axiomatise (the class of models induced by) monotone  $\Lambda$  structures. The restriction on occurrences of propositional variables is unproblematic, as variables that occur in a conclusion but not in the premise and multiple occurrences of variables in the premise can always be eliminated. The set of one-step tableau rules is a (the only) parameter in the construction of tableaux for coalgebraic fixpoint logics. Example rules are most conveniently presented if we identify a linear inequality  $\sum_i a_i p_i < k$  where  $p_i \in V \cup \overline{V}$  and  $a_i, k \in \mathbb{Q}$  with the set of prime implicants of the boolean function  $(x_i) \mapsto 1$  iff  $\sum a_i \text{sg}(p_i) < k$  where  $\text{sg}(p) = 1$  and  $\text{sg}(\overline{p}) = 0$  for  $p \in V$ , and prime implicants are represented by sets of propositional variables.

*Example 9.* The following rule sets are used for the logics introduced in Example [11](#).

$$\begin{aligned} (K) & \frac{\diamond p_0; \square p_1; \dots; \square p_n}{p_0; p_1; \dots; p_n} & (M) & \frac{\square p; \diamond q}{p; q} \\ (G) & \frac{\langle k_1 \rangle p_1; \dots; \langle k_n \rangle p_n; [l_1] q_1; \dots; [l_m] q_m}{\sum_{j=1}^m s_j \overline{q_j} - \sum_{i=1}^n r_i p_i < 0} \\ (P) & \frac{\langle a_1 \rangle p_1; \dots; \langle a_n \rangle p_n; [b_1] q_1; \dots; [b_m] q_m}{\sum_{j=1}^m s_j \overline{q_j} - \sum_{i=1}^n r_i p_i < k} \\ (C_1) & \frac{[C_1] p_1; \dots; [C_n] p_n}{p_1; \dots; p_n} & (C_2) & \frac{[C_1] p_1; \dots; [C_n] p_n; [\overline{D}] q; [\overline{N}] r_1; \dots; [\overline{N}] r_m}{p_1; \dots; p_n; q; r_1; \dots; r_m} \end{aligned}$$

where  $n, m \in \mathbb{N}$  and sets are represented by ;-separated lists. For the modal and monotone  $\mu$ -calculus, we have all instances of  $(K)$  and  $(M)$ , respectively. The graded  $\mu$ -calculus uses all instances of  $(G)$  for which  $r_i, s_j \in \mathbb{N} \setminus \{0\}$  and  $\sum_{i=1}^n r_i (k_i + 1) \geq 1 + \sum_{j=1}^m s_j l_j$ . The probabilistic  $\mu$ -calculus is axiomatised by instances of  $(P)$  where  $r_i, s_j \in \mathbb{N} \setminus \{0\}$  and  $\sum_{i=1}^n r_i a_i - \sum_{j=1}^m s_j b_j \leq k$  if  $n > 0$  and  $-\sum_{j=1}^m s_j b_j < k$  if  $n = 0$ . Finally, we associate all instances of  $(C_1), (C_2)$  for which the  $C_i$  are disjoint and moreover  $C_i \subseteq D$  in the case of  $(C_2)$ . We note that all rules above are monotone.

Tableaux themselves are formulated in terms of sequents:

**Definition 10.** A  $\Lambda$ -tableau sequent, or just sequent, is a finite set of  $\Lambda$ -formulas. We write  $S(\Lambda)$  for the set of  $\Lambda$ -sequents. If  $\Gamma \in S(\Lambda)$  we write  $S(\Gamma) = \{\Delta \in S(\Lambda) \mid \Delta \subseteq \text{Cl}(\Gamma)\}$  for the set of sequents over the closure of  $\Gamma$ . We identify a formula  $A \in \mathcal{F}(\Lambda)$  with the singleton set  $\{A\}$ , and write  $\Gamma; \Delta = \Gamma \cup \Delta$  for the union of  $\Gamma, \Delta \in S(\Lambda)$  as before. Substitution extends to sequents via  $\Gamma\sigma = \{A\sigma \mid A \in \Gamma\}$ .



The set TR of tableau rules induced by a set R of one-step rules contains the propositional and fixpoint rules, the modal rules (m) and the axiom (rule) below:

$$(\wedge) \frac{\Gamma; A \wedge B}{\Gamma; A; B} \quad (\vee) \frac{\Gamma; A \vee B}{\Gamma; A \quad \Gamma; B} \quad (\text{f}) \frac{\Gamma; \eta p. A}{\Gamma; A[p := \eta p. A]} \quad (\text{m}) \frac{\Gamma_0 \sigma, \Delta}{\Gamma_1 \sigma \dots \Gamma_n \sigma} \quad (\text{Ax}) \frac{\Gamma, A, \bar{A}}{}$$

Here,  $\Gamma, \Delta \in \mathcal{S}(\Lambda)$  range over sequents and  $A, B \in \mathcal{F}(\Lambda)$  over formulas. In (m),  $\Gamma_0/\Gamma_1 \dots \Gamma_n \in \mathbb{R}$  and  $\sigma : V \rightarrow \mathcal{F}(\Lambda)$  is so that  $A\sigma = B\sigma$  only if  $A = B$  for  $A, B \in \Gamma_0$ . An axiom is a premise of (Ax). The sequent  $\Delta$  is called a context of the modal rule  $\Gamma_0 \sigma, \Delta/\Gamma_1 \sigma \dots \Gamma_n \sigma$ , and the context of a non-modal rule is always empty.

We only allow substitutions that do not duplicate literals in the premise of modal rules to ensure decidability, and we require that the set of one-step tableau rules is closed under contraction later. Since fixpoint rules generate infinite paths, we formalise tableaux as finite, rooted graphs. As a consequence, closed tableaux are finitely represented proofs of the unsatisfiability of the root formula.

**Definition 11.** A tableau for a clean, guarded sequent  $\Gamma \in \mathcal{S}(\Lambda)$  is a finite, directed, rooted and labelled graph  $(N, K, R, \ell)$  where  $N$  is the set of nodes,  $K \subseteq N \times N$  is the set of edges,  $R$  is the root node and  $\ell : N \rightarrow \mathcal{S}(\Gamma)$  is a labelling function such that  $\ell(R) = \Gamma$  and, if  $K(n) = \{n' \mid (n, n') \in K\}$ :

- if  $\ell(n)$  is not the premise of a rule in TR, then  $K(n) = \emptyset$ .
- if  $\ell(n)$  is a premise of a rule in TR, then  $\ell(n)/\{\ell(n') \mid n' \in K(n)\} \in \text{TR}$ .

An annotation of a tableau is a mapping  $\alpha : N \rightarrow \mathcal{S}(\Lambda)$  such that  $\alpha(n)$  is a context of the rule  $\ell(n)/\{\ell(n') \mid n' \in K(n)\}$  whenever  $K(n) \neq \emptyset$ .

In other words, tableaux are sequent-labelled graphs where a rule has to be applied at a node if the node label matches a rule premise, no rule may be applied otherwise. The purpose of annotations is to record the weakening steps immediately prior to the applications of modal rules, which is needed for the definition of traces later.

Our goal is to show that a formula  $A \in \mathcal{F}(\Lambda)$  is satisfiable iff no tableau for  $A$  ever closes. In a setting without fixpoints, a tableau is closed iff all leaves are labelled with axioms. Here we also need to consider infinite paths, and ensure that only greatest fixpoints are unfolded infinitely often at the top level of an infinite path. As in [14], this necessitates to consider the set of traces through a given tableau.

**Definition 12.** The set of directional rule names is given by  $\mathbb{N} = \{\vee_l, \vee_r, \wedge, \text{f}, \text{m}\}$ , and an instance of  $\flat \in \mathbb{N}$  is an instance of the  $\vee$ -rule if  $\flat = \vee_l$  or  $\flat = \vee_r$  and an instance of a fixpoint rule/modal rule if  $\flat = \text{f}/\text{m}$ . A trace tile is a triple  $t = (\Delta, \flat, \Delta')$  for  $\Delta, \Delta' \in \mathcal{S}(\Lambda)$  and  $\flat \in \mathbb{N}$ . The trace tile  $t$  is consistent if there exists an instance of  $\flat$  with empty context that has  $\Gamma$  as a premise and  $\Delta$  as one of its conclusions, where  $\Delta$  has to be the left (right) conclusion of the  $\vee$ -rule in case  $\flat = \vee_l$  ( $\flat = \vee_r$ ). A path through a tableau  $\mathbb{T} = (N, K, R, \ell)$  is a finite or infinite sequence of nodes and directional rule names

$$\pi = n_0 \xrightarrow{b_0} n_1 \xrightarrow{b_1} n_2 \xrightarrow{b_2} n_3 \dots$$

such that  $n_{i+1} \in K(n_i)$ ,  $\ell(n_i)/\{\ell(n') \mid n' \in K(n_i)\}$  is an instance of  $\flat$ , and  $(\ell(n_i) \setminus \alpha(n_i), \flat_i, \ell(n_{i+1}))$  is a consistent trace tile. A finite path  $\pi$  is of maximal length if  $K(n) = \emptyset$  for the end node  $n$  of  $\pi$ .

If  $\alpha$  is an annotation for  $\mathbb{T}$ , then an  $\alpha$ -trace through  $\pi$  is a finite or infinite sequence of formulas  $(A_0, A_1, \dots)$  such that  $A_i \in \ell(n_i)$  and  $(A_i, A_{i+1}) \in \text{Tr}(\ell(n_i) \setminus \alpha(n_i), b_i, \ell(n_{i+1}))$  where the relations  $\text{Tr}(\Gamma, b, \Delta)$  induced by trace tiles are given by

- $\text{Tr}((\Gamma, A \vee B), \vee_l, (\Gamma, A)) = \{(A \vee B, A)\} \cup \text{Diag}(\Gamma)$
- $\text{Tr}((\Gamma, A \vee B), \vee_r, (\Gamma, B)) = \{(A \vee B, B)\} \cup \text{Diag}(\Gamma)$
- $\text{Tr}((\Gamma, A \wedge B), \wedge, (\Gamma, A, B)) = \{(A \wedge B, A), (A \wedge B, B)\} \cup \text{Diag}(\Gamma)$
- $\text{Tr}(\Gamma, m, \Delta) = \{(\heartsuit(B_1, \dots, B_k), B_j) \mid \heartsuit(B_1, \dots, B_k) \in \Gamma, B_j \in \Delta, j \leq k\}$
- $\text{Tr}((\Gamma, \eta p.B), f, (\Gamma, B[p := \eta p.B])) = \{\eta p.B, B[p := \eta p.B]\} \cup \text{Diag}(\Gamma)$
- $\text{Tr}(\Gamma, b, \Delta) = \emptyset$  otherwise.

where  $\text{Diag}(X) = \{(x, x) \mid x \in X\}$  is the diagonal on any set  $X$ . Finally, a tableau  $\mathbb{T}$  is closed, if there exists an annotation  $\alpha$  such that the end node of any finite path through  $\mathbb{T}$  of maximal length that starts in the root node is labelled with a tableau axiom and every infinite path starting in the root node carries at least one bad  $\alpha$ -trace with respect to a parity function  $\Omega$  for  $\Gamma$ .

Consistent trace tiles record the premise of a rule and one of its conclusions, together with the directional rule name. Here we require empty context, so that the rule that witnesses consistency of a trace tile is necessarily a substituted one-step rule. This implies that all traces on a tableau path ending in the context of a rule premise terminate.

*Example 13.* If nodes are represented by their labels, then the path

$$A \vee \mu p.B; C \xrightarrow{\vee_r} \mu p.\diamond B; C \xrightarrow{f} \diamond B[p := \mu p.B]; C \dots$$

supports the traces  $(A \vee \mu p.\diamond B, \mu p.\diamond B, \diamond B[p := \mu p.B], \dots)$  and  $(C, C, C, \dots)$ . Note that there is no trace on this path that starts with  $A$ .

Our goal is to show that non-existence of a closed tableau is equivalent to the satisfiability of the root formula. This is where we need coherence conditions between the axiomatisation and the (coalgebraic) semantics. In essence, we require that one-step tableau rules characterise satisfiability of a set of modalised formulas of the form  $\heartsuit(A_1, \dots, A_n)$  purely in terms of the  $\Lambda$ -structure.

**Definition 14.** The interpretation of a propositional sequent  $\Gamma \subseteq V \cup \overline{V}$  with respect to a set  $X$  and a valuation  $\tau : V \rightarrow \mathcal{P}(X)$  is given by  $\llbracket \Gamma \rrbracket_{X, \tau} = \bigcap \{\tau(p) \mid p \in \Gamma\}$ , and the interpretation  $\llbracket \Gamma \rrbracket_{TX, \tau} \subseteq TX$  of a modalised sequent  $\Gamma \subseteq (\Lambda \cup \overline{\Lambda})(V)$  is

$$\llbracket \Gamma \rrbracket_{TX, \tau} = \bigcap \{ \llbracket \heartsuit \rrbracket_X(\tau(p_1), \dots, \tau(p_n)) \mid \heartsuit(p_1, \dots, p_n) \in \Gamma \}.$$

If  $T$  is a  $\Lambda$ -structure, then a set  $R$  of monotone tableau rules for  $\Lambda$  is one-step tableau complete with respect to  $T$  if  $\llbracket \Gamma \rrbracket_{TX, \tau} \neq \emptyset$  iff for all  $\Gamma_0/\Gamma_1, \dots, \Gamma_n \in R$  and all  $\sigma : V \rightarrow V$  with  $\Gamma_0\sigma \subseteq \Gamma$ , there exists  $1 \leq i \leq n$  such that  $\llbracket \Gamma_i\sigma \rrbracket_{X, \tau} \neq \emptyset$ , whenever  $\Gamma \subseteq (\Lambda \cup \overline{\Lambda})(V)$  and  $\tau : V \rightarrow \mathcal{P}(X)$ .

Informally speaking, a set  $R$  of one-step tableau rules is one-step tableau complete if a modalised sequent  $\Gamma$  is satisfiable iff any rule that matches  $\Gamma$  has a satisfiable conclusion.

An adaptation of [16, Theorem 17] to the setting of monotone tableau rules establishes existence of a tableau complete set of monotone rules for monotone  $\Lambda$ -structures.

**Proposition 15.** *Every monotone  $\Lambda$ -structure admits a one-step tableau complete set of monotone tableau rules.*

In our examples, the situation is as follows:

**Proposition 16.** *The rule sets introduced in Example 9 are one-step tableau complete with respect to the corresponding structures defined in Example 7*

With the help of Theorem 7 we can now show that satisfiability precludes the existence of closed tableaux, as a winning strategy for  $\exists$  in the model checking game induces a path through any tableau that contradicts closedness.

**Theorem 17.** *Let  $R$  be a one-step tableau complete set of monotone rules for the modal similarity type  $\Lambda$ , and let  $\Gamma \in S(\Lambda)$  be clean and guarded. If  $\Gamma$  is satisfiable in some model  $M = (C, \gamma, \sigma)$  at state  $c \in C$ , then no closed tableau for  $\Gamma$  exists.*

*Example 18.* Consider the following formula of the coalitional  $\mu$ -calculus

$$[C]\nu X.(p \wedge \overline{[N]}X) \wedge [D]\mu Y.(\overline{p} \vee [D]Y)$$

stating that “coalition  $C$  can achieve that, from the next stage onwards,  $p$  holds irrespective of the strategies used by other agents, and coalition  $D$  can ensure (through suitable strategies used in the long term) that  $\overline{p}$  holds after some finite number of steps”. Here, we assume that  $C, D \subseteq N$  are such that  $C \cap D = \emptyset$ . Define a parity map  $\Omega$  for the above formula by  $\Omega(\nu X.(p \wedge \overline{[N]}X)) = 2$ ,  $\Omega(\mu Y.(\overline{p} \vee [D]Y)) = 1$  and  $\Omega(A) = 0$  otherwise. The unsatisfiability of this formula is witnessed by the following closed tableau:

$$\frac{\frac{\frac{[C]B \wedge [D]A}{[C]B; [D]A}}{B; A}}{p \wedge \overline{[N]}B; A}}{p \wedge \overline{[N]}B; \overline{p} \vee [D]A}}{p; \overline{[N]}B; \overline{p} \vee [D]A}}{p; \overline{[N]}B; \overline{p} \quad p; \overline{[N]}B; [D]A}$$

where  $B = \nu X.(p \wedge \overline{[N]}X)$  and  $A = \mu Y.(\overline{p} \vee [D]Y)$ . Any finite path through this tableau ends in an axiom, and the only infinite path contains the trace

$$[C]B \wedge [D]A, [D]A, A, \overline{A}, \overline{p} \vee [D]A, \overline{p} \vee [D]A, [D]A, A$$

where the overlined sequence is repeated ad infinitum. This trace is bad with respect to  $\Omega$ , as  $\Omega(A) = 1$  and  $A$  is the only fixpoint formula that occurs infinitely often.

## 5 The Tableau Game

We now introduce the tableau game associated to a clean and guarded sequent  $\Gamma$ , and use it to characterise the (non-)existence of closed tableaux in terms of winning strategies in the tableau game. For the entire section, we fix a modal similarity type  $\Lambda$  and

a one-step tableau complete set  $R$  of monotone tableau rules. The idea underlying the tableau game is that  $\forall$  intends to construct a closed tableau for a given set of formulas  $\Gamma$ , while  $\exists$  wants to demonstrate that any tableau constructed by  $\forall$  contains a path  $\pi$  that violates the closedness condition. As (certain) infinite plays of the tableau game correspond to paths through a tableau, an infinite play should be won by  $\exists$  if it does not carry a bad trace, with bad traces being detected with the help of parity word automata.

**Definition 19.** Let  $\Sigma$  be a finite alphabet. A non-deterministic parity  $\Sigma$ -word automaton is a quadruple  $\mathbb{A} = (Q, a_I, \delta : Q \times \Sigma \rightarrow \mathcal{P}(Q), \Omega)$  where  $Q$  is the set of states of  $\mathbb{A}$ ,  $a_I \in Q$  is the initial state,  $\delta$  is the transition function, and  $\Omega : Q \rightarrow \omega$  is a parity function. Given an infinite word  $\gamma = c_0c_1c_2c_3 \dots$  over  $\Sigma$ , a run of  $\mathbb{A}$  on  $\gamma$  is a sequence  $\rho = a_0a_1a_2 \dots \in Q^\omega$  such that  $a_0 = a_I$  and for all  $i \in \omega$  we have  $a_{i+1} \in \delta(a_i, c_i)$ . A run  $\rho$  is accepting if  $\rho$  is not a bad sequence with respect to  $\Omega$ . We say that  $\mathbb{A}$  accepts an infinite  $\Sigma$ -word  $\gamma$  if there exists an accepting run  $\rho$  of  $\mathbb{A}$  on  $\gamma$ . Finally we call  $\mathbb{A}$  deterministic if  $\delta(a, c)$  has exactly one element for all  $(a, c) \in Q \times \Sigma$ . In other words,  $\mathbb{A}$  is deterministic if  $\delta$  is a function of type  $Q \times \Sigma \rightarrow Q$ .

The tableau game uses an automaton over trace tiles (cf. Definition 12) to detect the existence of bad traces through infinite plays.

**Lemma and Definition 20.** Let  $\Gamma \in S(\Lambda)$  be a clean, guarded sequent, and let  $\Sigma_\Gamma$  denote the set of trace tiles  $(\Delta, \flat, \Delta')$  with  $\Delta, \Delta' \in S(\Gamma)$ . There exists a deterministic parity  $\Sigma_\Gamma$ -word automaton  $\mathbb{A}_\Gamma = (Q_\Gamma, a_\Gamma, \delta_\Gamma, \Omega')$  such that  $\mathbb{A}$  accepts an infinite sequence  $(t_0, t_1, \dots) \in \Sigma_\Gamma^\infty$  of trace tiles iff there is no sequence of formulas  $(A_0, A_1, \dots)$  with  $(A_i, A_{i+1}) \in \text{Tr}(t_i)$  which is a bad trace with respect to a parity function for  $\Gamma$ . Moreover, the index of  $\mathbb{A}$  and the cardinality of  $Q$  are bounded by  $p(|\text{Cl}(\Gamma)|)$  and  $2^{p(|\text{Cl}(\Gamma)|)}$  for a polynomial  $p$ , respectively. Such an automaton  $\mathbb{A}$  is called a  $\Gamma$ -parity automaton.

**Definition 21.** Let  $\Gamma \in S(\Lambda)$  be clean and guarded, and let  $\mathbb{A} = (Q, a_\Gamma, \delta, \Omega)$  be a  $\Gamma$ -parity automaton. We denote the set of tableau rules  $\Gamma_0/\Gamma_1, \dots, \Gamma_n \in \text{TR}$  for which  $\Gamma_i \in S(\Gamma)$  by  $\text{TR}_\Gamma$ . The  $\Gamma$ -tableau game is the parity game  $\mathcal{G}_\Gamma = (B_\exists, B_\forall, E, \Omega')$  where  $B_\forall = S(\Gamma) \times Q$ ,  $B_\exists = \text{TR}_\Gamma \times S(\Gamma) \times Q$ , and  $(b_1, b_2) \in E$  if either

- $b_1 = (\Delta, a) \in B_\forall$  and  $b_2 = (r, \Sigma, a) \in B_\exists$  where  $r \in \text{TR}_\Gamma$  has premise  $\Delta$  and  $\Sigma \subseteq \Delta$  is a context of  $r$ , or
- $b_1 = (r, \Sigma, a) \in B_\exists$ ,  $b_2 = (\Delta, a') \in B_\forall$  and there exists  $\flat \in \mathbb{N}$  such that  $r$  is an instance of  $\flat$ ,  $\Delta$  is a conclusion of  $r$ , the trace tile  $t = (\Gamma \setminus \Sigma, \flat, \Delta)$  is consistent where  $\Gamma$  is the premise of  $r$ , and  $a' = \delta(a, t)$ .

The parity function  $\Omega' : (B_\exists \cup B_\forall) \rightarrow \omega$  of  $\mathcal{G}_\Gamma$  is given by  $\Omega'(\Delta, a) = \Omega(a)$  if  $(\Delta, a) \in B_\forall$  and  $\Omega'(r, \Sigma, a) = 0$ .

If not explicitly stated otherwise, we will only consider  $\mathcal{G}_\Gamma$ -plays that start at  $(\Gamma, a_\Gamma)$  where  $a_\Gamma$  is the initial state of the automaton  $\mathbb{A}$ . In particular, we say that a player has a winning strategy in  $\mathcal{G}_\Gamma$  if she/he has a winning strategy in  $\mathcal{G}_\Gamma$  at position  $(\Gamma, a_\Gamma)$ .

The easier part of the correspondence between satisfiability and winning strategies in  $\mathcal{G}_\Gamma$  is proved by constructing a closed tableau based on a winning strategy for  $\forall$ . The notion of trace through a  $\mathcal{G}_\Gamma$ -play is used to show closedness.

**Definition 22.** For a  $\mathcal{G}_\Gamma$ -play

$$\pi = (\Gamma^0, a_0)(r_0, \Sigma_0, a_0)(\Gamma^1, a_1)(r_1, \Sigma_1, a_1) \dots (\Gamma^l, a_l)(r_l, \Sigma_l, a_l) \dots$$

a sequence  $\pi' = \Gamma^0 b_0 \Gamma^1 b_1 \dots \Gamma^l b_l \dots$  of sequents and directional rule names is an underlying path of  $\pi$  if  $r_i$  is a  $b_i$ -rule,  $t_i = (\Gamma^i \setminus \Sigma_i, b_i, \Gamma^{i+1})$  is a consistent trace tile, and  $\delta(a_i, t_i) = a_{i+1}$  for all  $i \in \mathbb{N}$ . A sequence of formulas  $\alpha = A_0 A_1 A_2 \dots \in \mathcal{F}(A)^\infty$  is a trace through  $\pi$  if there exists an underlying path  $\pi' = \Gamma^0 b_0 \Gamma^1 b_1 \Gamma^2 \dots$  of  $\pi$  such that  $(A_i, A_{i+1}) \in \text{Tr}(t_i)$  for all  $i \in \mathbb{N}$ .

An underlying path of a  $\mathcal{G}_\Gamma$ -play assigns directional rule names to the rules used in  $\forall$ 's moves, so that only consistent trace tiles are considered when defining traces.

**Theorem 23.** Let  $\Gamma \in S(A)$  be clean and guarded. If  $\forall$  has a winning strategy in  $\mathcal{G}_\Gamma$ , then  $\Gamma$  has a closed TR-tableau.

The converse of the above theorem is established later as Theorem 26. The challenge there is to construct a model for  $\Gamma$  based on a winning strategy for  $\exists$  in the  $\Gamma$ -tableau game. This crucially requires the set of tableau rules to be closed under contraction.

**Definition 24.** A set  $R$  of monotone one-step rules is closed under contraction, if for all  $\Gamma_0/\Gamma_1, \dots, \Gamma_n \in R$  and all  $\sigma : V \rightarrow V$ , there exists a rule  $\Delta_0/\Delta_1, \dots, \Delta_k \in R$  and a renaming  $\tau : V \rightarrow V$  such that  $A\tau = B\tau$  for  $A, B \in \Delta_0$  implies that  $A = B$ ,  $\Delta_0\tau \subseteq \Gamma_0\sigma$  and, for each  $1 \leq i \leq n$ , there exists  $1 \leq j \leq k$  such that  $\Gamma_i\sigma \subseteq \Delta_j\tau$ .

In other words, instances of one-step rules which duplicate literals in the premise may be replaced by instances for which this is not the case. Under this condition, we prove:

**Theorem 25.** Suppose that  $\Gamma \in S(A)$  is clean and guarded and  $R$  is one-step tableau complete and contraction closed. If  $\exists$  has a winning strategy in  $\mathcal{G}_\Gamma$ , then  $\Gamma$  is satisfiable in a model of size  $\mathcal{O}(2^{p(n)})$  where  $n$  is the cardinality of  $\text{Cl}(\Gamma)$  and  $p$  is a polynomial.

The proof of Theorem 25 constructs a model for  $\Gamma$  out of the game board of  $\mathcal{G}_\Gamma$  using a winning strategy  $f$  for  $\exists$  in  $\mathcal{G}_\Gamma$ . We use one-step tableau completeness to impose a  $T$ -coalgebra structure on the  $\mathcal{G}_\Gamma$ -positions reachable through an  $f$ -conform  $\mathcal{G}_\Gamma$ -play, in such a way that the truth lemma is satisfied. We subsequently equip this  $T$ -coalgebra with a valuation that makes  $\Gamma$  satisfiable in the resulting model. While our construction shares some similarities with the shallow model construction of [17], it is by no means a simple adaptation of loc. cit., as we are dealing with fixpoint formulas and thus cannot employ induction over the modal rank of formulas to construct satisfying models. Our proof of satisfiability is also substantially different from the corresponding proof for the modal  $\mu$ -calculus (cf. [14]) – we show satisfiability by directly deriving a winning strategy for  $\exists$  in the model-checking game from a winning strategy of  $\exists$  in the tableau game.

Putting everything together, we obtain a complete characterisation of satisfiability in the coalgebraic  $\mu$ -calculus.

**Theorem 26.** Suppose that  $\Gamma \in S(A)$  is a clean, guarded sequent and  $R$  is one-step tableau complete and contraction closed. Then  $\Gamma$  is satisfiable iff no tableau for  $\Gamma$  is closed iff  $\exists$  has a winning strategy in the tableau game  $\mathcal{G}_\Gamma$ .

As a by-product, we obtain the following small model property.

**Corollary 27.** *A satisfiable, clean and guarded formula  $A$  is satisfiable in a model of size  $\mathcal{O}(2^{p(n)})$  where  $n$  is the cardinality of  $\text{Cl}(A)$  and  $p$  is a polynomial.*

*Proof.* The statement follows immediately from Theorems [17], [23] and [25] together with the determinacy of two player parity games.

## 6 Complexity

We now show that – subject to a mild condition on the rule set – the satisfiability problem of the coalgebraic  $\mu$ -calculus is decidable in exponential time. By Theorem [26], the satisfiability problem is reducible to the existence of winning strategies in parity games.

To measure the size of a formula, we assume that the underlying similarity type  $\Lambda$  is equipped with a size measure  $s : \Lambda \rightarrow \mathbb{N}$  and measure the size of a formula  $A$  in terms of the number of subformulas counted with multiplicities, adding  $s(\heartsuit)$  for every occurrence of a modal operator  $\heartsuit \in \Lambda$  in  $A$ . In the examples, we code numbers in binary, that is,  $s(\langle k \rangle) = s([k]) = \lceil \log_2 k \rceil$  for the graded  $\mu$ -calculus and  $\langle p/q \rangle = [p/q] = \lceil \log_2 p \rceil + \lceil \log_2 q \rceil$  for the probabilistic  $\mu$ -calculus, and  $s([a_1, \dots, a_k]) = 1$  for coalition logic. The definition of size is extended to sequents by  $\text{size}(\Gamma) = \sum_{A \in \Gamma} \text{size}(A)$  for  $\Gamma \in \mathcal{S}(\Lambda)$  and  $\text{size}(\{\Gamma_1, \dots, \Gamma_n\}) = \sum_{i=1}^n \text{size}(\Gamma_i)$  for sets of sequents. To apply Theorem [26] we need to assume that the formula that we seek to satisfy is both clean and guarded, but this can be achieved in linear time.

**Lemma 28.** *For every formula  $A \in \mathcal{F}(\Lambda)$  we can find an equivalent clean and guarded formula in linear time.*

The proof is a straightforward generalisation of a similar result ([11], Theorem 2.1). To ensure that the size of the game board remains exponential, we encode the set of positions of the game board by strings of polynomial length, measured in the size of the initial sequent, and the rules need to be computationally well behaved. We require:

**Definition 29.** *A set  $R$  of tableau rules is exponentially tractable, if there exists an alphabet  $\Sigma$  and two functions  $f : \mathcal{S}(\Lambda) \rightarrow \mathcal{P}(\Sigma^*)$  and  $g : \Sigma^* \rightarrow \mathcal{P}(\mathcal{S}(\Lambda))$  together with a polynomial  $p$  such that  $|x| \leq p(\text{size}(\Gamma))$  for all  $x \in f(\Gamma)$ ,  $\text{size}(\Delta) \leq p(|y|)$  for all  $\Delta \in g(y)$ , so that, for  $\Gamma \in \mathcal{S}(\Lambda)$ ,*

$$\{g(x) \mid x \in f(\Gamma_0)\} = \{\{\Gamma_1, \dots, \Gamma_n\} \mid \Gamma_0/\Gamma_1, \dots, \Gamma_n \in R\}$$

*and both relations  $x \in f(\Gamma)$  and  $\Gamma \in g(x)$  are decidable in EXPTIME.*

Tractability of the set TR of tableau rules follows from tractability of the substitution instances of rules in R, as the non-modal rules can be encoded easily.

**Lemma 30.** *Suppose R is a set of monotone one-step rules. Then TR is exponentially tractable iff the set  $\{\Gamma_0\sigma/\Gamma_1\sigma, \dots, \Gamma_n\sigma \mid \Gamma_0/\Gamma_1, \dots, \Gamma_n \in R, \forall A, B \in \Gamma_0(A\sigma = B\sigma \implies A = B)\}$  of substituted one-step rules is exponentially tractable.*

Exponential tractability bounds the board of the tableau game and the complexity of both the parity function and the relation determining legal moves.

**Lemma 31.** *Suppose that  $R$  is exponentially tractable. Then every position in the tableau game  $G_\Gamma = (B_\exists, B_\forall, E, \Omega)$  of  $\Gamma \in S(\Lambda)$  can be represented by a string of polynomial length in  $\text{size}(\Gamma)$ . Under this coding, the relation  $(b, b') \in E$  is decidable in exponential time and the parity function  $\Omega$  can be computed in exponential time.*

Together with Lemma 28, we now obtain an EXPTIME upper bound for satisfiability.

**Corollary 32.** *Suppose  $T$  is a monotone  $\Lambda$ -structure and  $R$  is exponentially tractable, contraction closed and one-step tableau complete for  $T$ . Then the problem of deciding whether  $\exists$  has a winning strategy in the tableau game for a clean, guarded sequent  $\Gamma \in S(\Lambda)$  is in EXPTIME. As a consequence, the same holds for satisfiability of  $A \in \mathcal{F}(\Lambda)$ .*

*Proof.* The first assertion follows from Lemma 31 as the problem of deciding the winner in a parity game is exponential only in the size of the parity function of the game (Theorem 2) which is polynomial in the size of  $\Gamma$  (Lemma 20). The second statement now follows with the help of Theorem 26.

*Example 33.* It is easy to see that the rule sets for the modal  $\mu$ -calculus, the coalitional  $\mu$ -calculus and the monotone  $\mu$ -calculus are exponentially tractable, as the number of conclusions of each one-step rule is bounded. To establish exponential tractability for the rule sets for the graded and probabilistic  $\mu$ -calculus, we argue as in [17] where tractability of the (dual) proof rules has been established. We encode the conclusion  $\sum_{i=1}^n r_i a_i < k$  as  $(r_1, a_1, \dots, r_n, a_n, k)$  and Lemma 6.16 of *op. cit.* provides a polynomial bound on the size of the solutions for the linear inequalities that combine conclusion and side condition of both the  $(G)$  and  $(P)$ -rule. This allows us to guess the set of prime implicants of the conclusion in nondeterministic polynomial time, which shows that both rule sets are exponentially tractable. In all cases, contraction closure is immediate.

## 7 Conclusions

We have introduced the coalgebraic  $\mu$ -calculus, a generic and uniform framework for modal fixpoint logics. Our main results are soundness and completeness of tableau calculi, and an EXPTIME upper bound for the satisfiability problem. Concrete instances of the generic approach directly

- reproduce the complexity bound for the modal  $\mu$ -calculus [6], together with the completeness of a slight variant of the tableau calculus presented in [14]
- lead to a new proof of the known EXPTIME bound for the graded  $\mu$ -calculus [12]
- establish previously unknown EXPTIME bounds for the probabilistic  $\mu$ -calculus, for coalition logic with fixpoints and the monotone  $\mu$ -calculus.

We note that these bounds are tight for all logics except possibly the monotone  $\mu$ -calculus, as the modal  $\mu$ -calculus can be encoded into all other logics.

## References

1. Bradfield, J.C.: On the expressivity of the modal mu-calculus. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 479–490. Springer, Heidelberg (1996)
2. Chellas, B.: *Modal Logic*, Cambridge (1980)

3. Cirstea, C., Sadrzadeh, M.: Modular Games for Coalgebraic Fixed Point Logics. In: Adámek, J., Kupke, C. (eds.) *Coalgebraic Methods in Computer Science (CMCS 2008)*. ENTCS, vol. 203 (2008)
4. Emerson, E., Jutla, C.: The complexity of tree automata and logics of programs. In: *Proc. FOCS 1988*, pp. 328–337. IEEE Computer Society Press, Los Alamitos (1988)
5. Emerson, E., Jutla, C.: Tree automata, mu-calculus and determinacy. In: *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science (FoCS 1991)*, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
6. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. *SIAM J. Comput.* 29(1), 132–158 (1999)
7. Fine, K.: In so many possible worlds. *Notre Dame J. Formal Logic* 13, 516–520 (1972)
8. Hansen, H.H., Kupke, C.: A coalgebraic perspective on monotone modal logic. In: Adámek, J., Milius, S. (eds.) *Coalgebraic Methods in Computer Science*. ENTCS, vol. 106, pp. 121–143. Elsevier, Amsterdam (2004)
9. Jurdziński, M.: Small progress measures for solving parity games. In: Reichel, H., Tison, S. (eds.) *STACS 2000*. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
10. Kozen, D.: Results on the propositional  $\mu$ -calculus. *Theoret. Comput. Sci.* 27, 333–354 (1983)
11. Kupferman, O., Vardi, M., Wolper, P.: An automata-theoretic approach to branching-time model checking. *Journal of the ACM* 47(2), 312–360 (2000)
12. Kupferman, O., Sattler, U., Vardi, M.: The complexity of the graded mgr-calculus. In: Voronkov, A. (ed.) *CADE 2002*. LNCS (LNAI), vol. 2392, pp. 423–437. Springer, Heidelberg (2002)
13. Mostowski, A.: Games with forbidden positions. Technical Report 78, Instytut Matematyki, Uniwersytet Gdański, Poland (1991)
14. Niwinski, D., Walukiewicz, I.: Games for the mu-calculus. *Theor. Comput. Sci.* 163(1&2), 99–116 (1996)
15. Pauly, M.: A modal logic for coalitional power in games. *J. Logic Comput.* 12(1), 149–166 (2002)
16. Schröder, L.: A finite model construction for coalgebraic modal logic. In: Aceto, L., Ingólfssdóttir, A. (eds.) *FOSSACS 2006*. LNCS, vol. 3921, pp. 157–171. Springer, Heidelberg (2006)
17. Schröder, L., Pattinson, D.: PSPACE bounds for rank-1 modal logics. *ACM Trans. Compl. Log.* 2(10) (2008) (to appear)
18. Stirling, C.: *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer, Heidelberg (2001)
19. Venema, Y.: Automata and fixed point logics: a coalgebraic perspective. *Inform. Comput.* 204(4), 637–678 (2006)
20. Walukiewicz, I.: Completeness of Kozen’s axiomatisation of the propositional  $\mu$ -calculus. *Inf. Comput.* 157(1-2), 142–182 (2000)



# On the Word Problem for $\Sigma\Pi$ -Categories, and the Properties of Two-Way Communication\* (Extended Abstract)

Robin Cockett<sup>1</sup> and Luigi Santocanale<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Calgary  
robin@ucalgary.ca

<sup>2</sup> Laboratoire d'Informatique Fondamentale de Marseille, Université Aix-Marseille I,  
luigi.santocanale@lif.univ-mrs.fr

**Abstract.** This paper presents a feasible decision procedure for the equality of parallel arrows in the initial category with finite products and coproducts. The algorithm, in particular, handles the “additive units” and demonstrates that the complications introduced by the presence of these units can be managed in an efficient manner.

This problem is directly related to the problem of determining the equivalence between (finite) processes communicating on a two-way channel.

**Keywords:**  $\Sigma\Pi$ -categories, bicategories, word problem, two-way communication, game semantics, proof theory.

## Introduction

This paper presents a feasible decision procedure for the equality of parallel arrows in the initial  $\Sigma\Pi$ -category, that is the initial category with finite sums and products. The algorithm, in particular, handles the “additive units” that is the case when the sum or product is empty (i.e. the initial and final object).

The main contribution of this paper is to demonstrate that the complications introduced into the decision procedure by the presence of these units can be managed in an efficient manner. The efficiency of this decision procedure for free  $\Sigma\Pi$ -categories relies crucially on a number of quite delicate algebraic facts which are peculiar to these categories. These facts, in turn, rely on a key and yet very general observation of André Joyal concerning categories built from free limits and colimits: he observed that such categories are completely characterized by certain “softness” properties. The efficiency of the decision algorithm relies not only on the fact that softness allows the equality relation to be inductively determined but also on the fact that the induced equivalence relations have a very special form. Unravelling the description of these equivalence relations is the technical core of our contribution.

---

\* Research partially supported by the project SOAPDC no. ANR-05-JCJC-0142.

From a categorical perspective the structures we are investigating here are amongst the simplest imaginable. Yet the status of the word problem for these categories has languished in an unsatisfactory state. That equality is decidable follows from an application of standard tools from categorical proof theory [1,2]: these allow, through a Curry-Howard style isomorphism, the free  $\Sigma\Pi$ -category to be viewed as a deductive system for a logic. In [3] the deductive system which corresponds precisely to the usual categorical coherence requirements for products and sums was identified. Furthermore, it was shown there that the resulting system satisfies the cut-elimination property modulo a finite number of “permuting conversions”. This means that the focus of the decision procedure for these categories devolves upon the cut-free terms and the equality between these terms is determined by the permuting conversions.

The cut-free terms, which represent arrows between two given types, are finite in number and this immediately implies that equality is decidable. However, the complexity of this way of deciding equality is exponential because there can be an exponential number of equivalent terms. Thus the question, which remained open, was whether the matter could be decided feasibly. This is of particular interest as these expressions are, in the process world, the analogue of Boolean expressions (whose equality certainly cannot be decided feasibly). The main contribution of this paper is to confirm that there is a polynomial - indeed an efficient - algorithm which settles this question.

There have been a number of contributions towards the goal of this paper. Most, however, involve representation theorems, that is the provision of a faithful functor from some variant of a free  $\Sigma\Pi$ -category into a concrete combinatoric category (in which equality between maps is concrete equality).

For example the results in [4] cover free  $\Sigma\Pi$ -categories in which the initial and final object coincide, and show how these categories can be represented as full subcategories of categories of generalized coherence spaces. In [5,6] a representation of free  $\Sigma\Pi$ -categories *without units* is given into the category of sets and relations. In [7] a representation of multiplicative additive linear logic *without units* (multiplicative or additive) into a combinatoric category of proof-nets is given. As the addition of multiplicative connectives is conservative over the additive fragment this suggests a combinatoric representation for the additive fragment (this was explored in [5] and discovered [8] to be closely related to [4]). All these results, however, work only for the fragment without units – or, more precisely, for the fragment with a common initial and final object. As far we know, there is (currently) no representation theorem into a purely combinatoric setting for the full fragment with distinct additive units.

Units add to the decision problem – and to the representation theory – a non-trivial challenge which is easy to underestimate. In particular, in [3], the first mentioned author of this paper proposed a decision procedure which worked perfectly in the absence of units but fails manifestly in the presence of units. The effect of the additive units on the setting is quite dramatic. In particular, when there are no units (or there is a zero) *all* coproduct injections are monic. However, rather contrarily, in the presence of distinct units this simply is no

longer the case. Furthermore, this can be demonstrated quite simply, consider the following diagram:

$$0 \times 0 \begin{array}{c} \xrightarrow{\pi_1} \\ \xrightarrow{\pi_0} \end{array} 0 \xrightarrow{\sigma_0} 0 + 1$$

As  $0 + 1 \simeq 1$  is a terminal object, there is at most one arrow to it: this makes the above diagram a coequalizer. Yet, the arrows  $\pi_0, \pi_1$  are distinct in any free  $\Sigma\Pi$ -category, as for example they receive distinct interpretations in the opposite of the category of sets and functions.

As logicians and category theorists, we were deeply frustrated by this failure to master the units. The solution we now present for the decision problem, however, was devised only after a much deeper algebraic understanding of the structure of free  $\Sigma\Pi$ -categories had been obtained. The technical observations which underly this development, we believe, should be of interest to logicians and category theorists alike.

It is perhaps worth saying something about why categories with (free) sums and products are of interest. There are various ways to interpret its maps and objects. For example, the objects, thought of as trees with product and sum nodes, can be read as games, in which sum nodes indicate it is the player's turn and product nodes that it is the opponent's turn (with no requirement that play alternates). The maps are then interpreted as being "mediators" between these games which can use the information of one game to determine the play in the other. Their composition is given by hiding the transfer of information which happens through interaction on a middle game [9,10].

This, of course, was essentially the reading originally proposed by Blass [11] who also proposed that the maps should be combinatorial strategies. Abramsky noticed that Blass's composition failed to be associative and, in order to fix this defect, in an influential paper moved attention onto alternating games [12] where morphisms are combinatorial and, furthermore, composition *is* associative.

In retrospect it seems remiss that the obvious alternative for fixing the non-associativity of Blass's composition, namely abandoning the view that maps must be combinatorial, received so little attention. This even though, from a proof theoretic and categorical perspective, it was the more natural direction. Proof theoretically, in this approach, composition becomes the cut-elimination process of the logic of sums and products and this gives rise to an elegant reduction system which is confluent modulo equations – all of which is described in [3].

Our main motivation for studying these categories, however, derives from a slightly different reading in which the maps are viewed as (concurrent) processes which communicate on two-way channels and the objects are (finite) concurrent types or protocols. The protocols govern the interaction on a two-way channel by determining whether a process should be sending or receiving a message (and precisely which messages can be sent or received). This interpretation is more than an idle idea and, for example, the theoretical details have been pursued in [13,14].

This last interpretation is quite compelling as the algebraic results described in this paper suggest a number of not very obvious and even somewhat surprising

properties of communication along such a channel. For example, it is quite possible that a process which is required to send a value could send various different values and yet, semantically, be *the same process*. This is the notion of *indefiniteness* which is central in the business of unraveling the meaning of communication. There are various ways in which this apparently unintuitive situation can arise. For example, it could simply be that the recipient of the communication has stopped listening. It is of course very annoying when this happens but, undeniable, this is an occurrence well within the scope of the human experience of communication.

Proof theoretically and algebraically this all has to do with the behavior of the additive units. The purpose of this paper is to focus on these units and their ramifications in the business of communication. It is certainly true that without the units the theory is purely combinatoric. However, if one is tempted therefore to omit them, it is worth realizing that without units there is simply no satisfactory notion of *finite* communication!

Of course, without the units the theory is not only simpler but a good deal less mathematically interesting. It is to this mathematics we now turn.

In Section 1, we recall the definition of  $\Sigma\Pi$ -categories, and the results of [3]. In Section 2, we introduce Joyal’s notion of softness. In Section 3 we establish that coprojections are weakly disjoint in free  $\Sigma\Pi$ -categories, our first technical result, and list some consequences. This, in particular, leads to considering arrows which factor through a unit – these are the indefinite arrows mentioned above – which play a key role in the decision procedure. In Section 4 we present our second technical observation: if two arrows in  $\text{hom}(X \times Y, A)$  and  $\text{hom}(Y, A + B)$  are definite and are made equal when, respectively, projecting and coprojecting into  $\text{hom}(X \times Y, A + B)$ , this fact is witnessed by a unique “bouncer” in  $\text{hom}(Y, A)$ . Finally in Section 5, we collect our observations and sketch a decision procedure.

## 1 The Construction of Free $\Sigma\Pi$ -Categories

### 1.1 $\Sigma\Pi$ -Categories

The reader might consult [15] for the basic categorical notions used in this paper. Here, an  $\Sigma\Pi$ -category means a category with finite products and coproducts which, furthermore has a chosen presentation for the binary products and terminal object, and for the binary coproducts and initial object.

Recall that a category has *binary products* if, given two objects  $A, B$ , there is an object  $A \times B$ , and natural transformations

$$\begin{aligned} \text{hom}(X, A) \times \text{hom}(X, B) &\xrightarrow{\langle \cdot, \cdot \rangle} \text{hom}(X, A \times B), \\ \text{hom}(X_i, A) &\xrightarrow{\pi_i} \text{hom}(X_0 \times X_1, A), \quad i = 0, 1, \end{aligned}$$

that induce inverse natural bijections:

$$\pi_i(\langle f_0, f_1 \rangle) = f_i, \quad i = 0, 1, \qquad \langle \pi_0(f), \pi_1(f) \rangle = f.$$

A *terminal object* or *empty product* in a category is an object  $1$  such that, for each object  $X$ ,  $\text{hom}(X, 1)$  is a singleton. Recall that a terminal object is unique up to isomorphism and is the unit for the product, as  $X \times 1$  is canonically isomorphic to  $X$ .

The definition of binary sums (coproducts) and of initial object, is obtained by exchanging the roles of left and right objects in the definition of products: a category has *binary sums* if, given two objects  $X, Y$ , there exists a third object  $X + Y$  and natural transformations

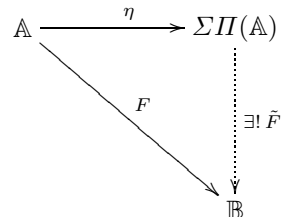
$$\begin{aligned} \text{hom}(X, A) \times \text{hom}(Y, A) &\xrightarrow{\{, \}} \text{hom}(X + Y, A), \\ \text{hom}(X, A_j) &\xrightarrow{\sigma_j} \text{hom}(X, A_0 + A_1), \quad j = 0, 1, \end{aligned}$$

that induce inverse natural bijections:

$$\sigma_j(\{f_0, f_1\}) = f_j, \quad j = 0, 1, \quad \{\sigma_0(f), \sigma_1(f)\} = f.$$

An *initial object*  $0$  is such that, for each object  $A$ ,  $\text{hom}(0, A)$  is a singleton.

A functor between two  $\Sigma\Pi$ -categories  $\mathbb{A}, \mathbb{B}$  is a  $\Sigma\Pi$ -functor if it sends (chosen) products to (chosen) products, and (chosen) coproducts to (chosen) coproducts. The *free  $\Sigma\Pi$ -category over a category  $\mathbb{A}$* , denoted  $\Sigma\Pi(\mathbb{A})$ , has the following property: there is a functor  $\eta : \mathbb{A} \rightarrow \Sigma\Pi(\mathbb{A})$  such that, if  $F : \mathbb{A} \rightarrow \mathbb{B}$  is a functor that “interprets”  $\mathbb{A}$  into a  $\Sigma\Pi$ -category  $\mathbb{B}$ , then there exists a unique  $\Sigma\Pi$ -functor  $\tilde{F} : \Sigma\Pi(\mathbb{A}) \rightarrow \mathbb{B}$  such that  $\tilde{F} \circ \eta = F$ . This is the usual universal property illustrated by the diagram on the right.



The free  $\Sigma\Pi$ -category on  $\mathbb{A}$ , can be “constructed” as follows. Its objects are the types inductively defined by the grammar

$$T = \eta(x) \mid 1 \mid T \times T \mid 0 \mid T + T, \tag{1}$$

where  $x$  is an object of  $\mathbb{A}$ . Then proof-terms are generated according to the deduction system of figure 1. Finally, proof-terms  $t : X \rightarrow A$  are quotiented by means of the least equivalence relation that forces the equivalence classes to satisfy the axioms of a  $\Sigma\Pi$ -category. Of course, while this is a perfectly good specification, we are looking for an effective presentation for  $\Sigma\Pi(\mathbb{A})$ . A first step in this direction comes from the fact the identity-rule as well as the cut-rule can be eliminated from the system:

**Proposition 1** (See [3] Proposition 2.9). *The cut-elimination procedure gives rise to a rewrite system on  $\Sigma\Pi(\mathbb{A})$  that is confluent modulo the set of equations of figure 2.*

From this we obtain an effective description of the category  $\Sigma\Pi(\mathbb{A})$ : the objects are the types generated by the grammar (1), while the arrows are equivalence classes of (identity|cut)-free proof-terms under the least equivalence generated by

$\frac{}{X \xrightarrow{id_X} X} \text{ identity-rule}$	$\frac{X \xrightarrow{f} C \quad C \xrightarrow{g} A}{X \xrightarrow{f;g} A} \text{ cut-rule}$
$\frac{x \xrightarrow{f} y}{\eta(x) \xrightarrow{\eta(f)} \eta(y)} \text{ Generators rule}$	
$\frac{X_i \xrightarrow{f} A}{X_0 \times X_1 \xrightarrow{\pi_i(f)} A} L_i \times$	$\frac{\frac{}{X \xrightarrow{!} 1} R1 \quad X \xrightarrow{f} A \quad X \xrightarrow{g} B}{X \xrightarrow{\langle f, g \rangle} A \times B} R \times$
$\frac{}{0 \xrightarrow{?} A} L0$	$\frac{X \xrightarrow{f} A \quad Y \xrightarrow{g} A}{X + Y \xrightarrow{\{f, g\}} A} L+$
	$\frac{X \xrightarrow{f} A_j}{X \xrightarrow{\sigma_j(f)} A_0 + A_1} R_j +$

**Fig. 1.** The deductive system for  $\Sigma\Pi(\mathbb{A})$

$\pi_i(\langle f, g \rangle) = \langle \pi_i(f), \pi_i(g) \rangle \quad \sigma_j(\{f, g\}) = \{\sigma_j(f), \sigma_j(g)\}$
$\pi_i(\sigma_j(f)) = \sigma_j(\pi_i(f))$
$\{\langle f_{11}, f_{12} \rangle, \langle f_{21}, f_{22} \rangle\} = \{\langle f_{11}, f_{21} \rangle, \langle f_{12}, f_{22} \rangle\}$
$\pi_i(!) = ! \quad \sigma_j(?) = ?$
$\{!, !\} = ! \quad \langle ?, ? \rangle = ?$
$!_0 = ?_1$

**Fig. 2.** The equations on (identity|cut)-free proof-terms

the equations of figure 2. Composition is given by the cut-elimination procedure, which by the above theorem is well defined on equivalence classes.

Our goal in the remainder of the paper is to show that determining whether two (identity|cut)-free proof-terms  $s, t : X \rightarrow A$ , are equivalent (by the equations

of figure 2) can be decided feasibly. The main theoretical tool we shall use is *softness* which we now introduce.

## 2 Softness

In every  $\Sigma\Pi$ -category 1 there exist canonical maps

$$\begin{aligned} \coprod_j \text{hom}(X, A_j) &\longrightarrow \text{hom}(X, \coprod_j A_j), \\ \coprod_i \text{hom}(X_i, A) &\longrightarrow \text{hom}(\coprod_i X_i, A). \end{aligned} \tag{2}$$

We shall be interested in these maps when, in a free  $\Sigma\Pi$ -category  $\Sigma\Pi(\mathbb{A})$ ,  $X = \eta(x)$  and  $A = \eta(a)$  are generators.

In every  $\Sigma\Pi$ -category there also exist canonical commuting diagrams of the form

$$\begin{array}{ccc} \coprod_{i,j} \text{hom}(X_i, A_j) & \longrightarrow & \coprod_j \text{hom}(\coprod_i X_i, A_j) \\ \downarrow & & \downarrow \\ \coprod_i \text{hom}(X_i, \coprod_j A_j) & \longrightarrow & \text{hom}(\coprod_i X_i, \coprod_j A_j) \end{array} \tag{3}$$

The following key theorem holds:

**Theorem 2 (See 3 Theorem 4.8).** *The following properties hold of  $\Sigma\Pi(\mathbb{A})$ :*

1. *The functor  $\eta : \mathbb{A} \rightarrow \Sigma\Pi(\mathbb{A})$  is full and faithful.*
2. *Generators are atomic, that is, the canonical maps of (2) – with  $X = \eta(x)$  and  $A = \eta(a)$  – are isomorphisms.*
3.  *$\Sigma\Pi(\mathbb{A})$  is soft, meaning that the canonical diagrams of (3) are pushouts.*

Moreover, if  $\mathbb{B}$  is a  $\Sigma\Pi$ -category with a functor  $F : \mathbb{A} \rightarrow \mathbb{B}$ , so that the pair  $(F, \mathbb{B})$  satisfies 1,2,3, then the extension  $\hat{F} : \Sigma\Pi(\mathbb{A}) \rightarrow \mathbb{B}$  is an equivalence of categories.

Thus, the structure of the category  $\Sigma\Pi(\mathbb{A})$  is precisely determined by the conditions 1,2,3. We shall spend the next section giving an explicit account of the property of softness. The theorem is a special instance of the more general observations due to Joyal on free bicomplete categories 16,17.

A decision procedure necessarily focuses on the homset  $\text{hom}(X_0 \times X_1, A_0 + A_1)$  which, by Theorem 2, is a certain the pushout. Equivalently, this homset is the colimit of what we shall refer to as the “diagram of cardinals”:

---

<sup>1</sup> As usual in category theory, we shall use the symbols  $\sum$  and  $\coprod$  interchangeably.

$$\begin{array}{ccccc}
 \text{hom}(X_0 \times X_1, A_0) & \xleftarrow{\pi_0} & \text{hom}(X_0, A_0) & \xrightarrow{\sigma_0} & \text{hom}(X_0, A_0 + A_1) \\
 \uparrow \pi_1 & & & & \uparrow \sigma_1 \\
 \text{hom}(X_1, A_0) & & & & \text{hom}(X_0, A_1) \\
 \downarrow \sigma_0 & & & & \downarrow \pi_0 \\
 \text{hom}(X_1, A_0 + A_1) & \xleftarrow{\sigma_1} & \text{hom}(X_1, A_1) & \xrightarrow{\pi_1} & \text{hom}(X_0 \times X_1, A_1)
 \end{array}$$

The explicit way of constructing such a colimit – see [15, §V.2.2] – is to first consider  $S$ , the disjoint sum of the corners:

$$\text{hom}(X_0, A_0 + A_1) + \text{hom}(X_1, A_0 + A_1) + \text{hom}(X_0 \times X_1, A_0) + \text{hom}(X_0 \times X_1, A_1)$$

and then quotient  $S$  by the equivalence relation generated by pairs  $(f, g)$  such that, for some  $h$ ,  $f = \pi_i(h)$  and  $\sigma_j(h) = g$ , as sketched below:

$$\begin{array}{ccc}
 & h \in \text{hom}(X_i, A_j) & \\
 \pi_i \swarrow & & \searrow \sigma_j \\
 f \in \text{hom}(X_0 \times X_1, A_j) & & g \in \text{hom}(X_i, A_0 + A_1)
 \end{array}$$

Thus, for  $f, f' \in S$  we have that  $[f] = [f'] \in \text{hom}(X_0 \times X_1, A_0 + A_1)$  if and only if there is a path (which can be empty) in the diagram of cardinals from  $f$  to  $f'$ , that is a sequence  $f_0 f_1 f_2 \dots f_n$ , where  $f = f_0$ ,  $f_n = f'$ , and for  $i = 0, \dots, n - 1$ ,  $(f_i, f_{i+1})$  or  $(f_{i+1}, f_i)$  is a pair as above.

### 3 Weak Disjointness

A *point* in a  $\Sigma\Pi$ -category is an arrow of the form  $p : 1 \rightarrow A$ . When an object has a point we shall say it is *pointed*. Similarly, a *copoint* is an arrow of the form  $c : X \rightarrow 0$  and an object with a copoint is *copointed*.

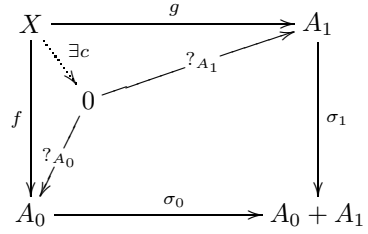
An object of  $\Sigma\Pi(\emptyset)$  can be viewed as a two-player game on a finite tree, with no draw final position. Points then correspond to winning strategies for the player, while copoints correspond to winning strategies for the opponent. Thus, every object of  $\Sigma\Pi(\emptyset)$  either has a point or a copoint but not both.

The first important result for analyzing softness concerns a key relationship between coproduct injections and copoints:



**Theorem 3.** *Coproducts are weakly disjoint in  $\Sigma\Pi(\mathbb{A})$ : if  $f; \sigma_0 = g; \sigma_1 : X \rightarrow A_0 + A_1$ , then there exists a copoint  $c : X \rightarrow 0$  such that  $f = c; ?$  and  $g = c; ?$ .*

*Proof.* We say that a triple  $(X \mid A_0, A_1)$  is good if for every  $f : X \rightarrow A_0$  and  $g : X \rightarrow A_1$  the statement of the Theorem holds. Similarly, we say that a triple  $(X_0, X_1 \mid A)$  is good if, for every  $f : X_0 \rightarrow A$  and  $g : X_1 \rightarrow A$ , the dual statement of the Theorem holds. We prove that every triple is good, by induction on the structural complexity of a triple.



The non trivial induction step arises when considering a triple of the form  $(X_0 \times X_1 \mid A_0, A_1)$  – or the dual case. Here, saying that the equality  $f; \sigma_0 = g; \sigma_1$  holds means that there exists a path  $\phi$  of the form  $f_0 f_1 \dots f_n$  in the diagram of cardinals from  $f = f_0 \in \text{hom}(X_0 \times X_1, A_0)$  to  $g = f_n \in \text{hom}(X_0 \times X_1, A_1)$ , i.e. from northwest to southeast. Moreover, we may assume  $\phi$  to be simple (i.e. there are no repeated maps).

Such path necessarily crosses one of southwest or northeast corners, let us say the latter. This means that, for some  $i = 1, \dots, n - 1$ ,  $f_i \in \text{hom}(X_0, A_0 + A_1)$ , and  $f_{i-1}, f_{i+1}$  are in opposite corners. W.l.o.g. we can assume  $f_{i-1} \in \text{hom}(X_0 \times X_1, A_0)$  and  $f_{i+1} \in \text{hom}(X_0 \times X_1, A_1)$ . Taking into account the definition of an elementary pair, we see that for some  $h \in \text{hom}(X_0, A_0)$  and  $h' \in \text{hom}(X_0, A_1)$  we have  $h; \sigma_0 = f_i = h'; \sigma_1$ . Thus, by the inductive hypothesis on  $(X_0 \mid A_0, A_1)$ , we have  $h = c; ?_{A_0}$  and  $h' = c; ?_{A_1}$ ; in particular the projection  $\pi_0 : X_0 \times X_1 \rightarrow X_0$  is epic, because of the existence of a copoint  $c : X_0 \rightarrow 0$ . Recalling that the path  $\phi$  is simple, we deduce that  $i$  is the only time  $\phi$  visits northeast, i.e. such that  $f_i \in \text{hom}(X_0, A_0 + A_1)$ .

A similar analysis shows that if  $\phi$  crosses a corner, then it visits that corner just once. Thus, we deduce that  $\phi$  does not cross the northwest corner, as  $\phi$  visits the northwest corner at time 0 and a corner may be crossed only at time  $i \in \{1, \dots, n - 1\}$ . Similarly,  $\phi$  does not cross the southeast corner. Also,  $\phi$  cannot visit the southwest corner, as this would imply that at least one of northwest or southeast corners has been crossed.

Putting these considerations together, we deduce that  $\phi$  visits the northwest, northeast, and southeast corners exactly once. That is,  $\phi$  has length 2 and  $i = 1$ . Recalling the definition of elementary pair, we have

$$f = f_0 = \pi_0; h, \quad h; \sigma_0 = f_1, \quad f_1 = h'; \sigma_1, \quad \pi_0; h' = f_2 = g.$$

Considering that  $h = c; ?_{A_0}$  and  $h' = c; ?_{A_1}$ , we deduce that  $f = \pi_0; c; ?_{A_0}$  and  $g = \pi_0; c; ?_{A_1}$ . □

This result may be interpreted from the perspective of a processes: the only way a process can send different messages on a channel without changing its meaning is when the recipient has stopped listening. The consequences of misjudging when the recipient stops listening, of course, is well-understood by school children and adults alike!

There are a number of consequences of this Theorem relevant to the decision procedure. To this end we need to introduce some terminology and some observations. We say that an arrow  $f$  is *pointed* if it factors through a point, i.e. if  $f = !; p$  for some point  $p$ . Similarly, an arrow is *copointed* if it factors through a copoint. Note that an object  $A$  is pointed iff  $! : 0 \rightarrow A$  is pointed and, similarly,  $X$  is copointed iff  $! : X \rightarrow 1$  is copointed. A map which is neither pointed nor copointed is said to be *definite*, otherwise it is said to be *indefinite*.

The following two facts are consequences of the theorem which can be obtained by a careful structural analysis:

**Corollary 4.**

1. *It is possible to decide (and find witnesses) in linear time in the size of a term whether it is pointed or copointed.*
2. *A coproduct injection  $\sigma_0 : A \rightarrow A + B$  is monic iff either  $B$  is not pointed or  $A$  is pointed. In particular  $! : 0 \rightarrow B$  is monic iff  $B$  is not pointed.*

An arrow is a *disconnect* if it is both pointed and copointed: it is easy to see that there is at most one disconnect between any two objects. Furthermore, if an arrow  $f : A \rightarrow B$  is copointed, that is  $f = c; ?$ , and its codomain,  $B$ , is pointed then  $f$  is this unique disconnect. On the other hand, if the codomain  $B$  is not pointed then  $! : 0 \rightarrow B$  is monic and, thus, such an  $f$  corresponds precisely to the copoint  $c$ . These observations allow the equality of indefinite maps, i.e. pointed and copointed, to be decided in linear time.

A further important fact which also follows from [4], in a similar vein to the above, concerns whether a map in  $\Sigma\Pi(\mathbb{A})$  factors through a projection or a coprojection. This can also be decided in linear time on the size of the term. This is by a structural analysis which we now sketch.

Suppose that we wish to determine whether  $f = \sigma_0(f') : A \rightarrow B + C$ . If syntactically  $f$  is  $\sigma_1(f')$  then, as a consequence of Theorem [3], the only way it can factorize is if the map is copointed. However, whether  $f$  is copointed can be determined in linear time on the term by Corollary [4]. The two remaining possibilities are that  $f$  is syntactically  $\{f_1, f_2\}$  or  $\pi_i(f')$ . In the former case, inductively, both  $f_1$  and  $f_2$  have to factorize through  $\sigma_0$ . In the latter case, when the map is not copointed,  $f'$  itself must factorize through  $\sigma_0$ .

There is, at this point, a slight algorithmic subtlety: to determine whether  $f$  can be factorized through a projection it seems that we may have to repeatedly recalculate whether the term is pointed or copointed and this recalculation would, it seems, push us beyond linear time. However, it is not hard to see that this the recalculation can be avoided simply by processing the term initially to include this information into the structure of the term (minimally two extra bits are needed at each node to indicate pointedness and copointedness of the map): subsequently this information would be available at constant cost. The cost of adding this information into the structure of the term is linear and, even better, the cost of maintaining this information, as the term is manipulated, is a constant overhead.

## 4 Bouncing

Given the previous discussion, equality for indefinite terms is understood and so we can focus our attention on definite terms. The main difficulty of the decision procedure concerns equality in the homset  $\text{hom}(X_0 \times X_1, A_0 + A_1)$ . However, the proof of Theorem 3 has revealed an important fact: *if two terms in this homset have a definite denotation, then any path in the diagram of cardinals that witnesses the equality between them cannot cross a corner of the diagram; that is, such a path must bounce backward and forward on one side:*

$$\text{hom}(X_0 \times X_1, A_j) \xleftarrow{\pi_i} \text{hom}(X_i, A_j) \xrightarrow{\sigma_j} \text{hom}(X_i, A_0 + A_1). \quad (4)$$

In other words, in order to understand definite maps we need to study the pushouts of the above spans. Notice that the proof of Theorem 3 also reveals that some simple paths in the diagram of cardinals have bounded length. However, that proof does not provide a bound for the length of paths that bounce on one side. It is the purpose of this section to argue that such a bound does indeed exist and to explore the algorithmic consequences.

We start our analysis by considering a general span  $B \xleftarrow{f} A \xrightarrow{g} C$  of sets and by recalling the construction of its colimit, the pushout  $B +_A C$ . This can be constructed by subdividing  $B$  and  $C$  into the image of  $A$  and the complement of that image. Thus, if  $B = \text{Im}(f) + B'$  and  $C = \text{Im}(g) + C'$  then  $B +_A C = A' + \text{Im}(\rho) + B'$  where  $\rho : A \rightarrow B +_A C$ . The image  $\text{Im}(\rho)$  is the quotient of  $A$  with respect to the equivalence relation witnessed by “bouncing data”; *bouncing data* is a sequence of elements of  $A$ ,  $(a_0, a_1, \dots, a_n)$ , with, for each  $0 \leq i < n$  either  $f(a_i) = f(a_{i+1})$  or  $g(a_i) = g(a_{i+1})$ . Bouncing data,  $(a_0, a_1, \dots, a_n)$ , is said to be *irredundant* if adjacent pairs in the sequence are identified for different reasons. Thus, in irredundant bouncing data if  $f(a_i) = f(a_{i+1})$  then  $f(a_{i+1}) \neq f(a_{i+2})$  and similarly for  $g$ . Redundant bouncing data can always be improved to be irredundant by simply eliding intermediate redundant steps.

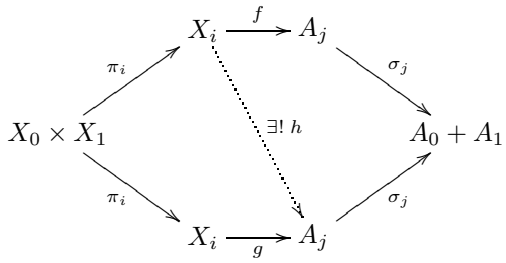
For bouncing data of length 2,  $(a_0, a_1, a_2)$ , we shall write  $a_1 : a_0 \rightsquigarrow a_2$  to indicate  $f(a_0) = f(a_1)$  and  $g(a_1) = g(a_2)$ , and we shall call  $a_1$  a *bouncer* from  $a_0$  to  $a_2$ . The following is a general observation concerning pushouts of sets:

**Proposition 5.** *For any pushout of  $B \xleftarrow{f} A \xrightarrow{g} C$  in sets the following are equivalent:*

1. *If  $a_0, a_n$  are related by some bouncing data, then they are related by bouncing data of length at most 2.*
2. *The equivalence relations generated by  $f$  and  $g$  commute.*
3. *The pushout diagram is a weak pullback, i.e. the comparison map to the pullback is surjective.*

*Moreover, when one of these equivalent conditions holds, the pushout is a pullback iff for every  $a_0$  and  $a_2$  related by bouncing data there is a unique element  $a_1$  such that  $a_1 : a_0 \rightsquigarrow a_2$ .*

Surprisingly, this altogether special situation holds in  $\Sigma\Pi(\mathbb{A})$ . More precisely, we say that the homset  $\text{hom}(X_i, A_j)$  *bounces* if, for each pair of objects  $X_{1-i}, A_{1-j}$ , the span  $\text{[4]}$  has a pushout which makes the homset  $\text{hom}(X_i, A_j)$  the pullback. Intuitively,  $\text{hom}(X_i, A_j)$  bounces if, whenever the upper and lower legs of the diagram on the right are equal (and definite), this is because of a unique bouncer  $h : f \rightsquigarrow g$ , where  $h$  is shown dotted and the fact that it is a bouncer means that the two smaller rectangles commute. Thus we have:



**Theorem 6.** *In  $\Sigma\Pi(\mathbb{A})$  all homsets bounce.*

The Theorem implies that if  $f$  and  $g$  are related by a bouncing path in the diagram of cardinals, then there exists a path of length at most 2 relating them.

The proof of the Theorem  $\text{[6]}$  relies on a tricky structural induction on the pairs  $(X_i, A_j)$ . Rather than presenting it here, we provide the proof for the special case of  $\Sigma\Pi(\emptyset)$ . Here the situation is much simpler, since each object is either pointed or copointed, but not both. We observe first that when there is a map from  $X_i$  to  $A_j$ , if  $X_i$  is pointed then  $A_j$  must be pointed as well and, dually, when  $A_j$  is copointed  $X_i$  must be copointed. As  $X_i$  and  $A_j$  must be either pointed or copointed it follows that  $X_i$  is pointed (respectively copointed) if and only if  $A_j$  is. However, if  $A_j$  is pointed then  $\sigma_j$  is monic so the bouncer  $h$  is forced to be  $f$ . Otherwise, if  $A_j$  is not pointed, then  $A_j$  is copointed and  $X_i$  as well; then  $\pi_i$  is epic and the bouncer  $h$  is forced to be  $g$ .

When  $h \in \{f, g\}$ , say that the bouncers  $h : f \rightsquigarrow g$  is trivial. While  $\Sigma\Pi(\emptyset)$  has only trivial bouncers, the next example shows that this is not in general the case. Let  $k : x \rightarrow a$  be an arbitrary map of  $\mathbb{A}$ , let  $X_0 = (0 \times 0) + \eta(x)$  and  $A_0 = (1 + 1) \times \eta(a)$ , let  $z : 0 \times 0 \rightarrow 1 \times 1$  be the unique disconnect. Recalling that an arrow from a coproduct to a product might be represented as a matrix, define

$$f = \begin{pmatrix} z & \pi_0(\{\}) \\ \sigma_0(\langle \rangle) & \eta(k) \end{pmatrix} \quad h = \begin{pmatrix} z & \pi_1(\{\}) \\ \sigma_0(\langle \rangle) & \eta(k) \end{pmatrix} \quad g = \begin{pmatrix} z & \pi_1(\{\}) \\ \sigma_1(\langle \rangle) & \eta(k) \end{pmatrix}$$

as arrows of the homset  $\text{hom}(X_0, A_0)$ . Then  $h : f \rightsquigarrow g$  is an example of a non-trivial bouncer whenever  $X_1$  is copointed and  $A_1$  is pointed, since then  $f$  and  $h$  are coequalized by  $\sigma_0$  and  $h$  and  $g$  are equalized by  $\pi_0$ . Notice, however, that this example relies crucially on having atomic objects.

We conclude this Section by sketching an algorithm — presented below on the right for  $\Sigma\Pi(\emptyset)$  — which computes whether a term  $f$  of the homset  $\text{hom}(X_0 \times X_1, A_j)$  is equivalent to a term  $g$  of the homset  $\text{hom}(X_i, A_0 + A_1)$  within the pushout of the span  $\text{[4]}$ .

The algorithm tries to lift  $f$  and  $g$  to  $f', g'$  in the homset  $\text{hom}(X_i, A_j)$  and, if successful, it tests for the existence of a bouncer  $h : f' \rightsquigarrow g'$ . Notice that the algorithm is defined by mutual recursion on the general decision procedure `equal`.

```

let equivalent f g =
  let f', g' be such that
    f ≡ σj(f') and g ≡ πi(g')
  in if f', g' do not exist then
    false
  elseif X1-i is copointed then
    equal σj(f') σj(g')
  else (*A1-j is pointed*)
    equal πi(f') πi(g')

```

## 5 The Decision Procedure

We present in Figure 3 a sketch of the decision procedure for  $\Sigma\Pi(\emptyset)$ . The general decision procedure for  $\Sigma\Pi(\mathbb{A})$  – which depends on having a decision procedure for  $\mathbb{A}$  – is considerably complicated by having to construct non-trivial bouncers; we leave that to the full paper.

```

let equal f g = match (dom f, cod g) with
  (0, _) | (_, 1) -> true
| (1, A0 + A1) ->
  let i, f', j, g' be such that
    f ≡ σi(f') and g ≡ σj(g')
  in if i = j then equal f' g' else false
| (Y0 × Y1, 0) -> ... dual
| (_, A0 × A1) ->
  let f0, f1, g0, g1 be such that
    f ≡ ⟨f0, f1⟩ and g ≡ ⟨g0, g1⟩
  in (equal f0 g0) && (equal f1 g1)
| (Y0 + Y1, _) -> ... dual
| (X0 × X1, A0 + A1) ->
  if definite f g then
    match (f, g) with
      (πi(f'), σj(g')) | (σj(g'), πi(f')) -> equivalent f' g'
    | (πi(f'), πi(g')) ->
      let i, g̃ be such that
        πi(g') ≡ σj(g̃)
      in
        if such i, g̃ do not exist then equal f' g'
        else equivalent f' g̃
    | (σi(f'), σi(g')) -> ... dual
    | _ -> false
  else equal_indefinite f g

```

**Fig. 3.** The decision procedure for  $\Sigma\Pi(\emptyset)$

**The procedure.** The procedure starts with two parallel terms  $f, g : X \rightarrow A$  in  $\Sigma\Pi(\emptyset)$ . The first step is to cut-eliminate the terms: this takes linear time on the size of the terms. If  $X$  is initial or  $A$  is final then we are done – there are of course no maps if  $X$  is final and  $A$  is initial. If either  $X$  is a coproduct or  $A$  is a product we can decompose the maps and recursively check the equality of the components. Thus, if  $X = X_1 + X_2$  then  $f = \{\sigma_0; f, \sigma_1; f\}$  and  $g = \{\sigma_0; g, \sigma_1; g\}$ , and then  $f = g$  if and only if  $\sigma_i; f = \sigma_i; g$  for  $i = 0, 1$ .

This reduces the problem to the situation in which the domain of the maps is a product and the codomain is a coproduct. Here we have to consider two cases:

*Indefinite maps.* In section [3](#) we mentioned that in time linear on the size of the maps (which, when terms are cut-eliminated is in turn bounded by the product of the types) one can determine whether the map is pointed (and produce a point) or copointed (and produce a copoint). If both terms are both pointed and copointed then they are the unique disconnect and we are done. If one term is just pointed the other must be just pointed and the points must agree (and dually for being just copointed).

*Definite maps.* When the maps are definite then a first goal is to determine whether the term  $f$  factors through a projection or a coprojection or, indeed, both (i.e.  $f = \sigma_i(f')$  or  $f = \pi_j(f')$ ). These factoring properties, as discussed above, can be determined in linear time with preprocessing. Using these properties – remembering that a path in the diagram of cardinals that relates two definite terms can only move along a side – there are two cases, either they bounce or they do not. If they bounce we can reduce the problem to the case when one term factors through a projection and the other through a coprojection (using **equivalent**). If the terms do not bounce then they both must factor *syntactically* in the same manner so that  $f$  is  $\sigma_0(f')$  and  $g$  is  $\sigma_0(g')$ , then  $f'$  must equal  $g'$ .

**Complexity.** Observe that a simple structural induction yields: *in  $\Sigma\Pi(\emptyset)$  the size of any cut-eliminated term  $t : X \rightarrow A$  is bounded by the product of the sizes of the types and its height is bounded by the sum of the heights of the types.* This means that, for cut-eliminated terms at least, we can use the size of the types to bound the term.

The decision procedure uses a preprocessing sweep to annotate the terms (and the types) with point and copoint information. Then the main equality algorithm either forms a tuple when the codomain is a product (or a cotuple when the domain is a sum) or has to determine whether a term can be factored via a projection or coprojection. Getting this to run in linear time uses the fact that the pointed and copointed information can be retrieved in constant time due to preprocessing.

The other major step in the algorithm, which we have not discussed for the general case, involves finding a bouncer. In the  $\Sigma\Pi(\emptyset)$  case this involves determining which of the projection or coprojection is respectively epic or monic. This, in turn, is determined by the pointedness or copointedness of the components of the type – calculated in the preprocessing stage – and so is constant time.

This means that the algorithm at each node of the term requires processing time bounded by a time proportional to the (maximal) size of the subterm. Such a pattern of processing is bounded by time proportional to the height of the term times the size. We therefore have:

**Proposition 7.** *To decide the equality of two parallel cut-eliminated terms  $t_1, t_2 : A \rightarrow B$  in  $\Sigma\Pi(\emptyset)$  has complexity in  $\mathcal{O}(\text{hgt}(A) + \text{hgt}(B)) \cdot \text{size}(A) \cdot \text{size}(B)$ .*

The analysis of the algorithm for  $\Sigma\Pi(\mathbb{A})$  is more complex and is left to the fuller exposition.

## References

1. Lambek, J.: Deductive systems and categories. I. Syntactic calculus and residuated categories. *Math. Systems Theory* 2, 287–318 (1968)
2. Došen, K.: Cut elimination in categories. *Trends in Logic—Studia Logica Library*, vol. 6. Kluwer Academic Publishers, Dordrecht (1999)
3. Cockett, J.R.B., Seely, R.A.G.: Finite sum-product logic. *Theory Appl. Categ.* 8, 63–99 (2001) (electronic)
4. Hu, H., Joyal, A.: Coherence completions of categories. *Theoret. Comput. Sci.* 227(1-2), 153–184 (1999), *Linear logic, I* (Tokyo, 1996)
5. Hughes, D.J.D.: A canonical graphical syntax for non-empty finite products and sums. Technical report, Stanford University (December 2002), <http://boole.stanford.edu/~dominic/papers>
6. Došen, K., Petrić, Z.: Bicartesian coherence revisited. In: Ognjanovic, Z. (ed.) *Logic in Computer Science, Zbornik Radova*, vol. 12 (2009), arXiv:0711.4961
7. Hughes, D.J.D., van Glabbeek, R.J.: Proof nets for unit-free multiplicative-additive linear logic (extended abstract). In: *LICS*, pp. 1–10. IEEE Computer Society, Los Alamitos (2003)
8. Hu, H.: Contractible coherence spaces and maximal maps. *Electr. Notes Theor. Comput. Sci.* 20 (1999)
9. Joyal, A.: Free lattices, communication and money games. In: *Logic and scientific methods* (Florence, 1995). *Synthese Lib.*, vol. 259, pp. 29–68. Kluwer Acad. Publ, Dordrecht (1997)
10. Santocanale, L.: Free  $\mu$ -lattices. *J. Pure Appl. Algebra* 168(2-3), 227–264 (2002), *Category theory 1999* (Coimbra)
11. Blass, A.: A game semantics for linear logic. *Ann. Pure Appl. Logic* 56(1-3), 183–220 (1992)
12. Abramsky, S., Jagadeesan, R.: Games and full completeness for multiplicative linear logic. *J. of Symb. Logic* 59(2), 543–574 (1994)
13. Cockett, J.R.B., Pastro, C.A.: A language for multiplicative-additive linear logic. *Electr. Notes Theor. Comput. Sci.* 122, 23–65 (2005)
14. Cockett, J.R.B., Pastro, C.A.: The logic of message passing. *Science of Computer Programming* 74(8), 498–533 (2009)
15. Mac Lane, S.: *Categories for the working mathematician*, 2nd edn. *Graduate Texts in Mathematics*, vol. 5. Springer, New York (1998)
16. Joyal, A.: Free bicomplete categories. *C. R. Math. Rep. Acad. Sci. Canada* 17(5), 219–224 (1995)
17. Joyal, A.: Free bicompletion of enriched categories. *C. R. Math. Rep. Acad. Sci. Canada* 17(5), 213–218 (1995)

# Intersection, Universally Quantified, and Reference Types\*

Mariangiola Dezani-Ciancaglini<sup>1</sup>, Paola Giannini<sup>2</sup>, and Simona Ronchi Della Rocca<sup>1</sup>

<sup>1</sup> Dipartimento di Informatica, Univ. di Torino, Italy

[www.di.unito.it](http://www.di.unito.it)

<sup>2</sup> Dipartimento di Informatica, Univ. del Piemonte Orientale, Italy

[www.di.unipmn.it](http://www.di.unipmn.it)

**Abstract.** The aim of this paper is to understand the interplay between intersection, universally quantified, and reference types. Putting together the standard typing rules for intersection, universally quantified, and reference types leads to loss of subject reduction. The problem comes from the invariance of the reference type constructor and the rules of intersection and/or universal quantification elimination, which are subsumption rules. We propose a solution in which types have a kind saying whether the type is (or contains in the case of intersection) a reference type or not. Intersection elimination is limited to intersections not containing reference types, and the reference type constructor can only be applied to closed types. The type assignment is shown to be safe, and when restricted to pure  $\lambda$ -calculus, as expressive as the full standard type assignment system with intersection and universally quantified types.

## Introduction

This paper deals with the problem of understanding the interplay between types built using intersection, universal quantification, and reference type constructors. Reference types, [7] and [15], are an essential tool for typing memory locations and the operations of reading and writing in memory. Parametric polymorphism of universally quantified types, introduced by Girard in [5] and Reynolds in [12], enhances the expressivity of typing in a uniform way. Intersection types, introduced in [2], allow for discrete polymorphism, increase the typability, and in particular give a formal account to overloading. Putting together these type constructs is useful for typing in a significant way a programming language with imperative features. It is well known that reference types must be invariant, since they represent both reading and writing of values, and therefore they should be both covariant and contra-variant [10] [page 198]. On the other hand, the standard intersection and universal quantifier elimination typing rules are subsumption rules, since the intersection of two types is contained in both types, and the instantiation of a universally quantified variable specializes the type.

As already remarked in [3] a naive typing with reference and intersection types may lead to loss of subject reduction as the following example shows. We can derive type Pos for the term

$$(\lambda x. (\lambda y. !x)(x := 0)) \text{ref } 1$$

---

\* Work partially supported by MIUR PRIN'06 EOS DUE, and MIUR PRIN'07 CONCERTO projects. The funding bodies are not responsible for any use that might be made of the results presented here.



by assuming type  $\text{Ref Pos} \wedge \text{Ref Nat}$  for the variable  $x$ . In fact  $\text{ref } 1$  has type  $\text{Ref Pos} \wedge \text{Ref Nat}$  since  $1$  is both  $\text{Pos}$  and  $\text{Nat}$ . By intersection elimination we can use:

- the type  $\text{Ref Nat}$  for  $x$  in the typing of  $x := 0$  getting the type  $\text{Unit}$ ;
- the type  $\text{Ref Pos}$  for  $x$  in the typing of  $!x$  getting the type  $\text{Pos}$ .

Reducing this term starting from the empty memory, with the call-by-value strategy we get

$$\begin{aligned}
 (\lambda x. (\lambda y. !x) (x := 0)) \text{ref } 1 \# \emptyset &\longrightarrow (\lambda x. (\lambda y. !x) (x := 0)) l \# (l = 1) \\
 &\longrightarrow (\lambda y. !l) (l := 0) \# (l = 1) \\
 &\longrightarrow (\lambda y. !l) () \# (l = 0) \\
 &\longrightarrow !l \# (l = 0) \\
 &\longrightarrow 0 \# (l = 0)
 \end{aligned}$$

and  $0$  does not have the type  $\text{Pos}$ . Note that this term evaluates to  $1$  in the memory ( $l = 1$ ) under the call-by-name reduction strategy. So the soundness of typing depends on the evaluation strategy used.

This example is a transcription of an example in [3]. The solution given in [3] is discussed in the conclusion, where it is compared with our proposal.

A variant of this examples shows that also a naive use of universally quantified types may lead to loss of subject reduction. Consider the term:

$$M = ((\lambda x. (\lambda y. !x) (x := \lambda z. 0)) (\text{ref } (\lambda z. 1))) 1$$

We can derive type  $\text{Pos}$  for this term by assuming type  $\forall t. \text{Ref } (t \rightarrow t)$  for the variable  $x$ . In fact  $(\text{ref } (\lambda z. 1))$  has type  $\text{Ref } (\text{Pos} \rightarrow \text{Pos})$  since  $1$  has type  $\text{Pos}$ . By forall elimination we can use:

- the type  $\text{Ref } (\text{Nat} \rightarrow \text{Nat})$  for  $x$  in the typing of  $x := \lambda z. 0$  getting the type  $\text{Unit}$ ;
- the type  $\text{Ref } (\text{Pos} \rightarrow \text{Pos})$  for  $x$  in the typing of  $!x$  getting the type  $\text{Pos} \rightarrow \text{Pos}$ .

Reducing this term starting from the empty memory, with the call-by-value strategy we get

$$\begin{aligned}
 M \# \emptyset &\longrightarrow ((\lambda x. (\lambda y. !x) (x := \lambda z. 0)) l) 1 \# (l = \lambda z. 1) \\
 &\longrightarrow ((\lambda y. !l) (l := \lambda z. 0)) 1 \# (l = \lambda z. 1) \\
 &\longrightarrow ((\lambda y. !l)()) 1 \# (l = \lambda z. 0) \\
 &\longrightarrow !l 1 \# (l = \lambda z. 0) \\
 &\longrightarrow (\lambda z. 0) 1 \# (l = \lambda z. 0) \\
 &\longrightarrow 0 \# (l = \lambda z. 0)
 \end{aligned}$$

and  $0$  does not have the type  $\text{Pos}$ . Note that using the call-by-name reduction strategy we get  $1$ .

As suggested by the above examples, a memory location typed by  $\text{Ref Pos} \wedge \text{Ref Nat}$  must contain values which are both  $\text{Pos}$  and  $\text{Nat}$ , i.e. values of type  $\text{Pos} \wedge \text{Nat}$ . For the case of quantified types a memory location typed by  $\forall t. \text{Ref } (t \rightarrow t)$  must contain a function of type  $\forall t. t \rightarrow t$ . This can be better expressed by typing the memory location with the type  $\text{Ref } (\text{Pos} \wedge \text{Nat})$  in the first case and  $\text{Ref } (\forall t. t \rightarrow t)$  in the second. Therefore, when a value is assigned to it it must have type  $\text{Pos}$  in the first case and  $\forall t. t \rightarrow t$  in the second.

Building on this idea we propose a type system for a  $\lambda$ -calculus with assignment statements and reference/dereference constructors. Intersections and universally quantified types are assigned to terms, via introduction rules, but elimination of intersection

is limited to non reference types, and the `Ref` type constructor can only be applied to closed types. We show safety, i.e. subject reduction and progress, of our type system. Lastly we observe that no expressive power is lost in comparison with the original systems of universally quantified types [5], intersection types [2], and both intersection and universally quantified types [8].

A strongly related paper is [4] which proposes a different type assignment system with both reference and intersection types. We will compare the present solution and that one discussed in the paper [4] in the conclusion.

**Outline of the paper.** Section 1 presents the syntax and reduction rules of the language. Types with their relevant properties are introduced in Section 2 and Section 3 defines the type assignment system and proves its safety. We conclude, in Section 4, by comparing our approach with the ones of [3] and [4], and outlining possible directions for further work.

## 1 Syntax and Reduction Rules

The language  $\Lambda_{imp}$  we are working with is a simplification of the language in [3], which in its turn belongs to the ML-family, the difference being the lack of the *let* construction and of the binary strings. It is well known that the *let* constructor is syntactic sugar [10] [Section 11.5] and in presence of intersection or universally quantified types does not increase the typability of the language, since with either intersection or universally quantified types we can type the translation of *let* in pure  $\lambda$ -calculus [1], [6]. The only data types of  $\Lambda_{imp}$  are the numerals, that is enough for discussing the typing problems shown in the introduction.

Terms of  $\Lambda_{imp}$  are defined by the following grammar:

$$\begin{aligned} M & ::= n \mid x \mid \lambda x.M \mid MM \mid \text{fix } x.M \\ & \quad l \mid \text{ref } M \mid !M \mid M := M \mid () \\ & \quad \text{if } M \text{ then } M \text{ else } M \mid M \text{ op } M \mid \dots \\ n & ::= 0 \mid 1 \mid 2 \mid \dots \\ \text{op} & ::= + \mid \times \mid \dots \end{aligned}$$

where  $x$  ranges over a countable set of variables, and  $l$  ranges over a countable set of *locations*. Free and bound variables are defined as usual. A term is closed if it does not contain free variables. The set of closed terms is denoted by  $\Lambda_{imp}^0$ .

The syntactical constructs with an imperative operational behaviour are the locations, denoting memory addresses, and the operators `ref`, `!`, and `:=`, denoting the operations of allocation, dereferencing, and assignment, whose behaviour is given below. The set of values is the subset of  $\Lambda_{imp}$  defined as follows:

$$V ::= n \mid \lambda x.M \mid l \mid ()$$

The value `()` is the result of the evaluation of an assignment, whose purpose is the side-effect of changing the store. The store is modeled as a finite association between locations and values:

$$\mu ::= \emptyset \mid \mu, (l = V)$$

---

$\mathcal{E}[(\lambda x.M) V] \# \mu$	$\longrightarrow \mathcal{E}[M[V/x]] \# \mu$	$(\beta_V)$
$\mathcal{E}[\text{fix } x.M] \# \mu$	$\longrightarrow \mathcal{E}[M[\text{fix } x.M/x]] \# \mu$	$(\text{fix}R)$
$\mathcal{E}[\text{ref } V] \# \mu$	$\longrightarrow \mathcal{E}[l] \# \mu, (l = V)$	$l \text{ fresh } (\text{ref}R)$
$\mathcal{E}[l] \# \mu, (l = V)$	$\longrightarrow \mathcal{E}[V] \# \mu, (l = V)$	$(\text{loc}R)$
$\mathcal{E}[l := V] \# \mu, (l = V')$	$\longrightarrow \mathcal{E}[(\cdot)] \# \mu, (l = V)$	$(\text{unit}R)$
$\mathcal{E}[\text{if } 0 \text{ then } M \text{ else } N] \# \mu$	$\longrightarrow \mathcal{E}[M] \# \mu$	$(\text{if}ZR)$
$\mathcal{E}[\text{if } n \text{ then } M \text{ else } N] \# \mu$	$\longrightarrow \mathcal{E}[N] \# \mu \quad n \neq 0$	$(\text{if}PR)$
$\mathcal{E}[0 + 0] \# \mu$	$\longrightarrow \mathcal{E}[0] \# \mu$	$(+ZZR)$
$\mathcal{E}[0 + 1] \# \mu$	$\longrightarrow \mathcal{E}[1] \# \mu$	$(+ZOR)$
$\dots$		

---

**Fig. 1.** Reduction Rules

On  $\Lambda_{\text{imp}}$  we consider a call-by-value reduction semantics. The operational semantics is given by defining reductions inside evaluations contexts, that, as usual, are terms with a *hole*,  $[\ ]$ , specifying which subterm must be reduced.

$$\mathcal{E} ::= [\ ] \mid \mathcal{E} M \mid V \mathcal{E} \mid \text{ref } \mathcal{E} \mid !\mathcal{E} \mid \mathcal{E} := M \mid V := \mathcal{E} \mid \text{if } \mathcal{E} \text{ then } M \text{ else } M \mid \mathcal{E} \text{ op } M \mid n \text{ op } \mathcal{E}$$

As one can see the evaluation is left to right and for an application we evaluate both terms. The reduction semantics is given by the sets of rules in Fig. 1 where  $[N/x]$  is the capture free substitution of  $x$  with  $N$ , and  $\mu$  is a store.

## 2 Types and Type Theory

Types,  $\tau, \sigma, \rho$ , are defined by the following syntax:

$$\begin{aligned} \tau, \sigma, \rho &::= \text{Pos} \mid \text{Nat} \mid \text{Unit} \mid t \mid \tau \rightarrow \tau \mid \tau \wedge \tau \mid \forall t : \kappa. \tau \mid \text{Ref } \tau \\ \kappa &::= \mathbf{S} \mid \mathbf{R} \end{aligned}$$

where  $t$  belongs to a countable set of type variables (ranged over by  $t, u, v, w$ ). Kinds (ranged over by  $\kappa$ ) say whether the type is (or contains in the case of intersection) a reference type. The *simple kind*  $\mathbf{S}$  is the kind of types which are constants, arrows or intersections of two types both of kind  $\mathbf{S}$ . The *reference kind*  $\mathbf{R}$  is the kind of types which are references, or intersections of two types at least one of them being of kind  $\mathbf{R}$ . An universally quantified type inherits the kind from the type obtained by erasing the quantification.

The type of natural and positive numbers is denoted respectively by  $\text{Nat}$  and  $\text{Pos}$ ,  $\text{Unit}$  is the type of assignments and  $(\cdot)$ . The arrow constructor,  $\tau \rightarrow \sigma$ , is the type of functions from type  $\tau$  to type  $\sigma$ , and intersection,  $\tau \wedge \sigma$ , is the type of expressions that have both type  $\tau$  and type  $\sigma$ . Universal quantification specifies the kind of the bound variable, since the variable can be replaced only by a type of the same kind. Finally  $\text{Ref } \tau$  is the type of a reference to a value of type  $\tau$ . We assume the following precedence relation between constructs:  $\forall, \text{Ref}, \wedge, \rightarrow$ . As usual  $\rightarrow$  associates to the right. We use  $\forall \vec{t} : \vec{\kappa}. \tau$  as an abbreviation for  $\forall t_1 : \kappa_1 \dots \forall t_n : \kappa_n. \tau$ , where  $n \geq 0$ .

The set of free type variables of a type,  $\mathcal{FV}(\tau)$ , is defined in the usual way. A term without occurrences of free type variables is said *closed*.

---

$\Delta \vdash \text{Pos} :: \mathbf{S}$	$\Delta \vdash \text{Nat} :: \mathbf{S}$	$\Delta \vdash \text{Unit} :: \mathbf{S}$	$\Delta, t : \kappa \vdash t :: \kappa$	
$\Delta \vdash \tau :: \kappa \quad \mathcal{FV}(\tau) = \emptyset$	$\Delta \vdash \tau :: \kappa \quad \Delta \vdash \tau' :: \kappa'$	$\Delta \vdash \tau :: \kappa \quad \Delta \vdash \tau' :: \kappa'$	$\Delta \vdash \tau \wedge \tau' :: \kappa \curlywedge \kappa'$	$\Delta, t : \kappa \vdash \tau :: \kappa'$
$\Delta \vdash \text{Ref } \tau :: \mathbf{R}$	$\Delta \vdash \tau \rightarrow \tau' :: \mathbf{S}$	$\Delta \vdash \tau \wedge \tau' :: \kappa \curlywedge \kappa'$	$\Delta \vdash \forall t : \kappa. \tau :: \kappa'$	

---

**Fig. 2.** Kind Assignment

A *kind environment*  $\Delta$  is an association between type variables and kinds, defined as follows:

$$\Delta ::= \emptyset \mid \Delta, t : \kappa \quad t \notin \text{dom}(\Delta)$$

where  $\text{dom}$  is the environment domain.

We use  $\Delta, \bar{t} : \bar{\kappa}$  as an abbreviation for the kind environment  $\Delta, t_1 : \kappa_1, \dots, t_n : \kappa_n$ , where  $n \geq 0$ .

A type  $\tau$  has kind  $\kappa$  w.r.t.  $\Delta$  if the judgment  $\Delta \vdash \tau :: \kappa$  can be derived from the rules in Fig. 2. Note that only closed types can be arguments of the `Ref` type constructor. As we can see from the rules of Fig. 2 the kind of an arrow is always  $\mathbf{S}$  and the kind of an intersection is  $\mathbf{R}$  if at least one of its types has kind  $\mathbf{R}$ , since we define:

$$\kappa \curlywedge \kappa' = \begin{cases} \mathbf{S} & \text{if } \kappa = \kappa' = \mathbf{S}, \\ \mathbf{R} & \text{otherwise.} \end{cases}$$

We abbreviate  $\Delta \vdash \tau_1 :: \kappa_1, \dots, \Delta \vdash \tau_n :: \kappa_n$ , where  $n \geq 0$ , by  $\Delta \vdash \bar{\tau} :: \bar{\kappa}$ .

In the following we will only consider types to which a kind can be assigned from a suitable environment.

On types, we define a congruence relation,  $\equiv$ , identifying types that denote the same property of terms. The relation  $\equiv$  is the minimal equivalence relation which is a congruence and which satisfies the axioms given in Fig. 3. Regarding intersection we have idempotence, commutativity, associativity, distribution of intersection on the right side of arrows with the same left side, and distribution of `Ref` over intersection. For quantified types we have commutativity of quantification,  $\alpha$ -conversion, the fact that quantifying on a variable not free in a type is irrelevant, and the standard distribution rules for quantifiers on arrow and intersection connectives. We consider types modulo  $\equiv$ , so we write  $\bigwedge_{i \in I} \tau_i$ , and  $\bigwedge_{1 \leq i \leq n} \tau_i$  for denoting  $\tau_1 \wedge \dots \wedge \tau_n$ , where  $I = \{1, \dots, n\}$ , and none of the  $\tau_i$ ,  $1 \leq i \leq n$ , is an intersection.

It is easy to check that if  $\Delta \vdash \tau :: \kappa$  and  $\tau \equiv \sigma$ , then  $\Delta \vdash \sigma :: \kappa$ . It is important to notice that `Ref`  $\tau$  has a kind implies  $\tau$  is closed, so in particular  $\forall t. \text{Ref } \tau \equiv \text{Ref } \tau$ .

In the following it is handy to single out the types whose top quantification is meaningless.

**Definition 1.** A type  $\tau$  is  $\forall$ -top-free if there are no  $t$ ,  $\kappa$ , and  $\sigma$  such that  $\tau \equiv \forall t : \kappa. \sigma$  and  $t \in \mathcal{FV}(\sigma)$ .

For example,  $\forall t. \text{Nat}$  is  $\forall$ -top-free, since  $\forall t. \text{Nat} \equiv \text{Nat}$ . Instead  $\text{Nat} \rightarrow \forall t. t$  is not  $\forall$ -top-free, since  $\text{Nat} \rightarrow \forall t. t \equiv \forall t. \text{Nat} \rightarrow t$ .

---

$\tau \equiv \tau \wedge \tau$	$\sigma \wedge \tau \equiv \tau \wedge \sigma$	$(\tau \wedge \sigma) \wedge \tau' \equiv \tau \wedge (\sigma \wedge \tau')$
$(\sigma \rightarrow \tau) \wedge (\sigma \rightarrow \tau') \equiv \sigma \rightarrow \tau \wedge \tau'$		$\text{Ref}(\tau \wedge \sigma) \equiv \text{Ref} \tau \wedge \text{Ref} \sigma$
$\forall t : \kappa. \forall t' : \kappa'. \tau \equiv \forall t' : \kappa'. \forall t : \kappa. \tau$		
$t \notin \mathcal{FV}(\tau)$	$\Rightarrow$	$\left\{ \begin{array}{l} \forall t' : \kappa. \tau \equiv \forall t : \kappa. \tau[t/t'] \\ \forall t : \kappa. \tau \equiv \tau \\ \forall t : \kappa. (\tau \rightarrow \sigma) \equiv \tau \rightarrow \forall t : \kappa. \sigma \\ \forall t : \kappa. (\tau \wedge \sigma) \equiv \tau \wedge \forall t : \kappa. \sigma \end{array} \right.$

---

**Fig. 3.** The Congruence  $\equiv$  on Types

---

$\Delta \vdash \text{Pos} \leq \text{Nat} (pos)$	$\frac{\Delta \vdash \tau \wedge \sigma :: \mathbf{S}}{\Delta \vdash \tau \wedge \sigma \leq \tau} (\wedge E)$	$\frac{\Delta \vdash \forall t : \kappa. \tau :: \kappa' \quad \Delta \vdash \sigma :: \kappa}{\Delta \vdash \forall t : \kappa. \tau \leq \tau[\sigma/t]} (\forall E)$
$\frac{\Delta \vdash \tau' \leq \tau \quad \Delta \vdash \sigma \leq \sigma'}{\Delta \vdash \tau \rightarrow \sigma \leq \tau' \rightarrow \sigma'} (\rightarrow)$	$\frac{\Delta \vdash \tau \leq \tau' \quad \Delta \vdash \sigma \leq \sigma'}{\Delta \vdash \tau \wedge \sigma \leq \tau' \wedge \sigma'} (\wedge)$	$\frac{\Delta, t : \kappa \vdash \tau \leq \sigma}{\Delta \vdash \forall t : \kappa. \tau \leq \forall t : \kappa. \sigma} (\forall)$
$\frac{\Delta \vdash \tau :: \kappa}{\Delta \vdash \tau \leq \tau} (id)$	$\frac{\Delta \vdash \tau \leq \rho \quad \Delta \vdash \rho \leq \sigma}{\Delta \vdash \tau \leq \sigma} (trans)$	$\frac{\tau \equiv \tau' \quad \Delta \vdash \tau' \leq \sigma' \quad \sigma' \equiv \sigma}{\Delta \vdash \tau \leq \sigma} (congr)$

---

**Fig. 4.** The Preorder Relation  $\leq$  on Types

A preorder relation  $\leq$  is defined on types through the rules shown in Fig. 4. Rule  $(pos)$  says that a positive is also a natural. Rules  $(\wedge E)$  and  $(\forall E)$  are the elimination rules. Note that for eliminating intersection we require that the intersection does not contain reference types. This is a crucial restriction, along with the facts that  $\text{Ref}$  can only be applied to closed types and there is no rule for applying  $\leq$  inside  $\text{Ref}$ , to get subject reduction. Rules  $(\rightarrow)$ ,  $(\wedge)$ , and  $(\forall)$  extend  $\leq$  to the specific constructor, and they are standard. Rule  $(id)$ , and  $(trans)$ , make  $\leq$  a preorder, and rule  $(congr)$  makes  $\leq$  a partial order when we identify congruent types.

Note that  $\Delta \vdash \tau :: \kappa$  and  $\tau \equiv \sigma$  imply both  $\Delta \vdash \tau \leq \sigma$  and  $\Delta \vdash \sigma \leq \tau$ . On the other side  $\Delta \vdash \tau \leq \sigma$  implies  $\Delta \vdash \tau :: \kappa$  and  $\Delta \vdash \sigma :: \kappa$  for some  $\kappa$ .

Weakening holds for kind environments in all the considered judgements, i.e.  $\Delta \vdash \tau :: \kappa$  implies  $\Delta, t : \kappa \vdash \tau :: \kappa$  if  $t \notin \text{dom}(\Delta)$  and similarly for  $\Delta \vdash \tau \leq \sigma$ .

By induction on the definition of  $\leq$  we can show that the preorder is preserved by replacing type variables by types of the same kinds.

**Lemma 1.**  $\Delta, t : \kappa \vdash \tau \leq \sigma$ , and  $\Delta \vdash \rho :: \kappa$  imply  $\Delta \vdash \tau[\rho/t] \leq \sigma[\rho/t]$ .

The next technical lemma is the key tool for proving the subject reduction property in case the used reduction rule is the  $\beta_v$ -rule. It states that if a quantified intersection of arrows is less than the arrow  $\tau \rightarrow \sigma$ , then there are instances of domains and co-domains of some arrows in the intersection which are related by the preorder to  $\tau$  and  $\sigma$ .

**Lemma 2.** *If  $\Delta \vdash \forall \bar{t} : \bar{\kappa}. \bigwedge_{i \in I} (\tau_i \rightarrow \sigma_i) \leq \tau \rightarrow \sigma$ , where  $\sigma_i$  ( $i \in I$ ) and  $\sigma$  are  $\forall$ -top-free, then there are  $\bar{\rho}$ , and  $J \subseteq I$ , such that:*

- $\Delta \vdash \bar{\rho} :: \bar{\kappa}$ ,
- $\Delta \vdash \tau \leq \tau_j[\bar{\rho}/\bar{t}]$  ( $j \in J$ ), and
- $\Delta \vdash \bigwedge_{j \in J} \sigma_j[\bar{\rho}/\bar{t}] \leq \sigma$ .

*Proof.* By induction on the definition of  $\leq$ . In order to prove the result for rule (*trans*) we show the more general assert that follows:

*If  $\tau \equiv \forall \bar{u} : \bar{\kappa}. \bigwedge_{i \in I} (\tau_i \rightarrow \sigma_i)$  where  $\sigma_i$  ( $i \in I$ ) are  $\forall$ -top-free, and  $\Delta \vdash \tau \leq \sigma$ , then there are  $\bar{v}$ ,  $\bar{\kappa}'$ ,  $J$ ,  $\tau'_j$ ,  $\forall$ -top-free  $\sigma'_j$  ( $j \in J$ ),  $\bar{\rho}$  such that:*

- $\sigma \equiv \forall \bar{v} : \bar{\kappa}'. \bigwedge_{j \in J} (\tau'_j \rightarrow \sigma'_j)$
- $\Delta, \bar{v} : \bar{\kappa}' \vdash \bar{\rho} :: \bar{\kappa}$ , and
- for all  $j \in J$  there is  $H_j \subseteq I$  with:
  - $\Delta, \bar{v} : \bar{\kappa}' \vdash \tau'_j \leq \tau_h[\bar{\rho}/\bar{u}]$  for all  $h \in H_j$ , and
  - $\Delta, \bar{v} : \bar{\kappa}' \vdash \bigwedge_{h \in H_j} \sigma_h[\bar{\rho}/\bar{u}] \leq \sigma'_j$ .

The proof is by induction on the derivation of  $\leq$ . We only consider the most difficult case, that is when the statement is the consequent of rule (*trans*). Let  $\Delta \vdash \tau \leq \tau'$  and  $\Delta \vdash \tau' \leq \sigma$  be the premises of the application of rule (*trans*).

By induction hypothesis on  $\tau \leq \tau'$  there are  $\bar{w}$ ,  $\bar{\kappa}''$ ,  $L$ ,  $\tau'_l$ ,  $\forall$ -top-free  $\sigma''_l$  ( $l \in L$ ),  $\bar{\rho}'$  such that:

- (a)  $\tau' \equiv \forall \bar{w} : \bar{\kappa}''. \bigwedge_{l \in L} (\tau'_l \rightarrow \sigma''_l)$
- (b)  $\Delta, \bar{w} : \bar{\kappa}'' \vdash \bar{\rho}' :: \bar{\kappa}$ , and
- (c) for all  $l \in L$  there is  $H'_l \subseteq I$  with:
  - (c.1)  $\Delta, \bar{w} : \bar{\kappa}'' \vdash \tau'_l \leq \tau_h[\bar{\rho}'/\bar{u}]$  for all  $h \in H'_l$ , and
  - (c.2)  $\Delta, \bar{w} : \bar{\kappa}'' \vdash \bigwedge_{h \in H'_l} \sigma_h[\bar{\rho}'/\bar{u}] \leq \sigma''_l$ .

By induction hypothesis on  $\tau' \leq \sigma$  there are  $\bar{v}$ ,  $\bar{\kappa}'$ ,  $J$ ,  $\tau'_j$ ,  $\forall$ -top-free  $\sigma'_j$  ( $j \in J$ ),  $\bar{\rho}''$  such that:

- (a')  $\sigma \equiv \forall \bar{v} : \bar{\kappa}'. \bigwedge_{j \in J} (\tau'_j \rightarrow \sigma'_j)$
- (b')  $\Delta, \bar{v} : \bar{\kappa}' \vdash \bar{\rho}'' :: \bar{\kappa}$ , and
- (c') for all  $j \in J$  there is  $H''_j \subseteq L$  with:
  - (c'.1)  $\Delta, \bar{v} : \bar{\kappa}' \vdash \tau'_j \leq \tau'_h[\bar{\rho}''/\bar{w}]$  for all  $h \in H''_j$ , and
  - (c'.2)  $\Delta, \bar{v} : \bar{\kappa}' \vdash \bigwedge_{h \in H''_j} \sigma''_h[\bar{\rho}''/\bar{w}] \leq \sigma'_j$ .

Note that we can assume that the sets of variables  $\bar{u}$ ,  $\bar{v}$ , and  $\bar{w}$  are fresh and pairwise disjoint. Define  $\bar{\rho} = \bar{\rho}'[\bar{\rho}''/\bar{w}]$  and  $H_j = \bigcup_{k \in H''_j} H'_k$  ( $j \in J$ ). It is easy to verify that:

- $\Delta, \bar{v} : \bar{\kappa}' \vdash \bar{\rho} :: \bar{\kappa}$  (from Lemma [1](#), weakening, (b) and (b')), and
- $H_j \subseteq I$ .

Moreover (c.1) and (b') imply by Lemma [1](#) and weakening that

$$\Delta, \bar{v} : \bar{\kappa}' \vdash \tau'_j[\bar{\rho}]/\bar{u} \leq \tau_h[\bar{\rho}'/\bar{u}][\bar{\rho}''/\bar{w}] \text{ for all } h \in H'_j.$$

Note that  $\tau_h[\bar{\rho}'/\bar{u}][\bar{\rho}''/\bar{w}] = \tau_h[\bar{\rho}/\bar{u}]$  for all  $h \in H_j$  since  $\bar{w}$  cannot occur in  $\tau_h$ . So by (c'.1), and transitivity of  $\leq$  we get for all  $j \in J$ :

$$\Delta, \bar{v} : \bar{\kappa}' \vdash \tau'_j \leq \tau_h[\bar{\rho}/\bar{u}] \text{ for all } h \in H_j.$$

Similarly from (c.2), (b'), Lemma [1](#) and weakening we get

$$\Delta, \bar{v} : \bar{\kappa}' \vdash \bigwedge_{h \in H'_j} \sigma_h[\bar{\rho}/\bar{u}] \leq \sigma''_l[\bar{\rho}''/\bar{w}] \text{ for all } l \in L.$$

---


$$\begin{array}{c}
\Delta; \Sigma; \Gamma \vdash 0 : \text{Nat} \quad (\text{Nat}) \quad \frac{n \neq 0}{\Delta; \Sigma; \Gamma \vdash n : \text{Pos}} \quad (\text{Pos}) \quad \Delta; \Sigma; \Gamma \vdash () : \text{Unit} \quad (\text{Unit}()) \\
\\
\Delta; \Sigma; \Gamma, x : \tau \vdash x : \tau \quad (\text{var}) \quad \Delta; \Sigma, l : \tau; \Gamma \vdash l : \text{Ref } \tau \quad (\text{loc}) \\
\\
\frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad \mathcal{FV}(\tau) = \emptyset}{\Delta; \Sigma; \Gamma \vdash \text{ref } M : \text{Ref } \tau} \quad (\text{Ref } I) \quad \frac{\Delta; \Sigma; \Gamma \vdash M : \text{Ref } \tau}{\Delta; \Sigma; \Gamma \vdash !M : \tau} \quad (\text{Ref } E) \\
\\
\frac{\Delta; \Sigma; \Gamma, x : \tau \vdash M : \sigma}{\Delta; \Sigma; \Gamma \vdash \lambda x. M : \tau \rightarrow \sigma} \quad (\rightarrow I) \quad \frac{\Delta; \Sigma; \Gamma \vdash M : \tau \rightarrow \sigma \quad \Delta; \Sigma; \Gamma \vdash N : \tau}{\Delta; \Sigma; \Gamma \vdash MN : \sigma} \quad (\rightarrow E) \\
\\
\frac{\Delta, t : \kappa; \Sigma; \Gamma \vdash M : \tau \quad t \notin \mathcal{FV}(\Sigma, \Gamma)}{\Delta; \Sigma; \Gamma \vdash M : \forall t : \kappa. \tau} \quad (\forall I) \quad \frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad \Delta; \Sigma; \Gamma \vdash M : \sigma}{\Delta; \Sigma; \Gamma \vdash M : \tau \wedge \sigma} \quad (\wedge I) \\
\\
\frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad \Delta \vdash \tau \leq \sigma}{\Delta; \Sigma; \Gamma \vdash M : \sigma} \quad (\leq) \quad \frac{\Delta; \Sigma; \Gamma \vdash M : \text{Ref } \tau \quad \Delta; \Sigma; \Gamma \vdash N : \tau}{\Delta; \Sigma; \Gamma \vdash M := N : \text{Unit}} \quad (\text{Unit}) \\
\\
\frac{\Delta; \Sigma; \Gamma, x : \tau \vdash M : \tau}{\Delta; \Sigma; \Gamma \vdash \text{fix}_x. M : \tau} \quad (\text{fix}) \quad \frac{\Delta; \Sigma; \Gamma \vdash M : \text{Nat} \quad \Delta; \Sigma; \Gamma \vdash N_1 : \tau \quad \Delta; \Sigma; \Gamma \vdash N_2 : \tau}{\Delta; \Sigma; \Gamma \vdash \text{if } M \text{ then } N_1 \text{ else } N_2 : \tau} \quad (\text{if})
\end{array}$$


---

Fig. 5. The Typing Rules for Terms

This together with (c'.2), using rule ( $\wedge$ ), transitivity of  $\leq$ , and the congruence  $\sigma \wedge \sigma \equiv \sigma$  implies for all  $j \in J$ :

$$\Delta, \bar{v} : \bar{\kappa}^j \vdash \bigwedge_{h \in H_j} \sigma_h[\bar{\rho}/\bar{u}] \leq \sigma_j'. \quad \square$$

### 3 The Typing System

The typing system proves judgements of the shape:

$$\Delta; \Sigma; \Gamma \vdash M : \tau$$

where  $\Delta$  is a kind environment,  $\Sigma$  and  $\Gamma$  are a *store environment* and a *type environment* respectively,  $M$  is a term and  $\tau$  is a type. Store and type environments are defined as follows:

$$\begin{array}{l}
\Sigma ::= \emptyset \mid \Sigma, l : \tau \quad l \notin \text{dom}(\Sigma) \quad \tau \text{ is a closed type} \\
\Gamma ::= \emptyset \mid \Gamma, x : \tau \quad x \notin \text{dom}(\Gamma).
\end{array}$$

A store (type) environment is *well formed* with respect to a kind environment  $\Delta$  if all its predicates have a kind, i.e.,  $\Sigma$  ( $\Gamma$ ) is such that if  $l : \tau \in \Sigma$  ( $x : \tau \in \Gamma$ ) then  $\Delta \vdash \tau :: \kappa$  for some kind  $\kappa$ . When we write a typing judgement  $\Delta; \Sigma; \Gamma \vdash M : \tau$  we always assume that  $\Sigma$  and  $\Gamma$  are well formed with respect to  $\Delta$ .

The typing rules, given in Fig. 5 are standard. We omit the typing rules dealing with arithmetic operators which are obvious. Note that the elimination rules of both  $\wedge$  and  $\forall$  are particular cases of rule ( $\leq$ ).

It is easy to verify that strengthening and weakening for all the environments are admissible rules. Fig. 6 shows these rules, where  $\mathcal{L}(M)$  is the set of locations and  $\mathcal{FV}(M)$  is the set of free variables occurring in  $M$ .

The proof that deductions remain valid under the substitution of type variables by types respecting kinds by induction on deductions is standard.

---

$\frac{\Delta, t : \kappa; \Sigma; \Gamma \vdash M : \tau \quad t \notin \mathcal{TV}(\Sigma, \Gamma, \tau)}{\Delta; \Sigma; \Gamma \vdash M : \tau} \text{ (s}\Delta\text{)}$	$\frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad t \notin \mathcal{TV}(\Sigma, \Gamma, \tau)}{\Delta, t : \kappa; \Sigma; \Gamma \vdash M : \tau} \text{ (w}\Delta\text{)}$
$\frac{\Delta; \Sigma, l : \tau'; \Gamma \vdash M : \tau \quad l \notin \mathcal{L}(M)}{\Delta; \Sigma; \Gamma \vdash M : \tau} \text{ (s}\Sigma\text{)}$	$\frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad l \notin \text{dom}(\Sigma) \quad \Delta \vdash \sigma :: \kappa \quad \text{for some } \kappa \quad \mathcal{TV}(\sigma) = \emptyset}{\Delta; \Sigma, l : \sigma; \Gamma \vdash M : \tau} \text{ (w}\Sigma\text{)}$
$\frac{\Delta; \Sigma; \Gamma, x : \tau' \vdash M : \tau \quad x \notin \mathcal{TV}(M)}{\Delta; \Sigma; \Gamma \vdash M : \tau} \text{ (s}\Gamma\text{)}$	$\frac{\Delta; \Sigma; \Gamma \vdash M : \tau \quad x \notin \text{dom}(\Gamma) \quad \Delta \vdash \sigma :: \kappa \quad \text{for some } \kappa}{\Delta; \Sigma; \Gamma, x : \sigma \vdash M : \tau} \text{ (w}\Gamma\text{)}$

---

Fig. 6. Admissible Rules

**Proposition 1.** *If  $\Delta, t : \kappa; \Sigma; \Gamma \vdash M : \tau$  and  $\Delta \vdash \sigma :: \kappa$ , then*

$$\Delta; \Sigma[\sigma/t]; \Gamma[\sigma/t] \vdash M : \tau[\sigma/t].$$

The type system enjoys a Generation Lemma, which relates the shapes of terms with the shapes of their possible derivations. We omit the obvious points concerning numerals and operators on numerals.

**Lemma 3 (Generation).** *Let  $\Delta; \Sigma; \Gamma \vdash M : \tau$ . Then  $\Delta \vdash \tau \geq \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} \tau_i$ , for some  $I$ ,  $\bar{l}$ ,  $\bar{\kappa}$ ,  $\forall$ -top-free  $\tau_i$  ( $i \in I$ ), and the followings hold, where  $\Delta' = \Delta, \bar{l} : \bar{\kappa}$ :*

1.  $\underline{M} = \underline{x}$  implies that  $x : \sigma \in \Gamma$  for some  $\sigma$  such that  $\sigma \leq \tau$ ;
2.  $\underline{M} = \underline{\lambda x. P}$  implies that there are  $\sigma_i, \rho_i$  ( $i \in I$ ), such that:
  - (a)  $\tau_i = \sigma_i \rightarrow \rho_i$ , and
  - (b)  $\Delta'; \Sigma; \Gamma, x : \sigma_i \vdash P : \rho_i$  ( $i \in I$ );
3.  $\underline{M} = \underline{PN}$  implies that there are  $\sigma_i$  ( $i \in I$ ) such that:
  - (a)  $\Delta'; \Sigma; \Gamma \vdash P : \sigma_i \rightarrow \tau_i$  ( $i \in I$ ), and
  - (b)  $\Delta'; \Sigma; \Gamma \vdash N : \sigma_i$  ( $i \in I$ );
4.  $\underline{M} = \underline{\text{fix } x. N}$  implies that  $\Delta'; \Sigma; \Gamma, x : \tau_i \vdash N : \tau_i$  ( $i \in I$ );
5.  $\underline{M} = \underline{l}$  implies that  $l : \sigma \in \Sigma$ , for some closed  $\sigma$  such that  $\text{Ref } \sigma \equiv \tau$ ;
6.  $\underline{M} = \underline{!N}$  implies that  $\Delta'; \Sigma; \Gamma \vdash N : \text{Ref } \tau_i$  ( $i \in I$ );
7.  $\underline{M} = \underline{\text{ref } N}$  implies that there are closed  $\sigma_i$  such that:
  - (a)  $\tau_i = \text{Ref } \sigma_i$ , and
  - (b)  $\Delta; \Sigma; \Gamma \vdash N : \sigma_i$  ( $i \in I$ );
8.  $\underline{M} = \underline{P := N}$  implies  $\tau \equiv \text{Unit}$ , and for some closed  $\sigma$  we have  $\Delta'; \Sigma; \Gamma \vdash P : \text{Ref } \sigma$  and  $\Delta'; \Sigma; \Gamma \vdash N : \sigma$ ;
9.  $\underline{M} = \underline{()}$  implies  $\tau \equiv \text{Unit}$ ;
10.  $\underline{M} = \underline{\text{if } P \text{ then } N \text{ else } N'}$  implies that  $\Delta'; \Sigma; \Gamma \vdash P : \text{Nat}$  and  $\Delta'; \Sigma; \Gamma \vdash N : \tau_i$  and  $\Delta'; \Sigma; \Gamma \vdash N' : \tau_i$  ( $i \in I$ ).

*Proof.* For all points, the proof is by induction on derivations. We will consider only the case in which the last rule applied is  $(\wedge I)$ , and we will show it for Points 2, 3 and 5. All the other cases are simpler.



2. If the last applied rule is:

$$\frac{\Delta; \Sigma; \Gamma \vdash \lambda x.P : \tau \quad \Delta; \Sigma; \Gamma \vdash \lambda x.P : \tau'}{\Delta; \Sigma; \Gamma \vdash \lambda x.P : \tau \wedge \tau'} (\wedge I)$$

by induction  $\Delta \vdash \tau \geq \forall \bar{t} : \bar{\kappa}. \bigwedge_{i \in I} (\sigma_i \rightarrow \rho_i)$ , and  $\Delta \vdash \tau' \geq \forall \bar{t}' : \bar{\kappa}'. \bigwedge_{j \in J} (\sigma'_j \rightarrow \rho'_j)$ , and  $\Delta, \bar{t} : \bar{\kappa}; \Sigma; \Gamma, x : \sigma_i \vdash P : \rho_i$ , for  $i \in I$  and  $\Delta, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma, x : \sigma'_j \vdash P : \rho'_j$ , for  $j \in J$ . By the monotonicity of  $\leq$  with respect to  $\wedge$  we get

$$\tau \wedge \tau' \geq \forall \bar{t} : \bar{\kappa}. \bigwedge_{i \in I} (\sigma_i \rightarrow \rho_i) \wedge \forall \bar{t}' : \bar{\kappa}'. \bigwedge_{j \in J} (\sigma'_j \rightarrow \rho'_j)$$

and, since types are considered modulo  $\equiv$ , we can assume that  $\bar{t}$  and  $\bar{t}'$  are disjoint, so we have  $\tau \wedge \tau' \geq \forall \bar{t} : \bar{\kappa}. \forall \bar{t}' : \bar{\kappa}'. (\bigwedge_{i \in I} (\sigma_i \rightarrow \rho_i) \wedge \bigwedge_{j \in J} (\sigma'_j \rightarrow \rho'_j))$ . Moreover, by the admissible rule ( $w\Delta$ ), we obtain  $\Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma, x : \sigma_i \vdash P : \rho_i$ , for  $i \in I$  and  $\Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma, x : \sigma'_j \vdash P : \rho'_j$ , for  $j \in J$ .

3. If the last used rule is:

$$\frac{\Delta; \Sigma; \Gamma \vdash PN : \tau \quad \Delta; \Sigma; \Gamma \vdash PN : \tau'}{\Delta; \Sigma; \Gamma \vdash PN : \tau \wedge \tau'} (\wedge I)$$

then by induction,  $\Delta \vdash \tau \geq \forall \bar{t} : \bar{\kappa}. \bigwedge_{i \in I} \tau_i$  and  $\Delta \vdash \tau' \geq \forall \bar{t}' : \bar{\kappa}'. \bigwedge_{j \in J} \tau'_j$ , and:  $\Delta, \bar{t} : \bar{\kappa}; \Sigma; \Gamma \vdash P : \sigma_i \rightarrow \tau_i$  and  $\Delta, \bar{t} : \bar{\kappa}; \Sigma; \Gamma \vdash N : \sigma_i$  ( $i \in I$ )  
 $\Delta, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash P : \sigma'_j \rightarrow \tau'_j$  and  $\Delta, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash N : \sigma'_j$  ( $j \in J$ ).

Then, by ( $w\Delta$ ):

$$\Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash P : \sigma_i \rightarrow \tau_i \quad \text{and} \quad \Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash N : \sigma_i \quad (i \in I)$$

$$\Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash P : \sigma'_j \rightarrow \tau'_j \quad \text{and} \quad \Delta, \bar{t} : \bar{\kappa}, \bar{t}' : \bar{\kappa}'; \Sigma; \Gamma \vdash N : \sigma'_j \quad (i \in J).$$

The proof follows from  $\Delta \vdash \tau \wedge \tau' \geq \forall \bar{t} : \bar{\kappa}. \forall \bar{t}' : \bar{\kappa}'. (\bigwedge_{i \in I} \tau_i \wedge \bigwedge_{j \in J} \tau'_j)$ , since we can assume that  $\bar{t}$  and  $\bar{t}'$  are disjoint.

5. If the last applied rule is:

$$\frac{\Delta; \Sigma; \Gamma \vdash l : \tau \quad \Delta; \Sigma; \Gamma \vdash l : \tau'}{\Delta; \Sigma; \Gamma \vdash l : \tau \wedge \tau'} (\wedge I)$$

by induction  $\tau \equiv \text{Ref } \sigma$  and  $\tau' \equiv \text{Ref } \sigma'$ , and the proof follows from the fact that  $\text{Ref } \sigma \wedge \text{Ref } \sigma' \equiv \text{Ref } (\sigma \wedge \sigma')$ .  $\square$

Note that, without reference types, Points [3](#) and [4](#) of previous lemma hold with  $I$  a singleton set. The restriction on rule ( $\wedge E$ ) is reflected in the necessity of having sets of types of cardinality bigger than 1. For example from  $\{x : (\text{Nat} \rightarrow \text{Nat}) \wedge (\text{Nat} \rightarrow \text{Ref Nat}), y : \text{Nat}, z : (\text{Nat} \rightarrow \text{Nat}) \wedge (\text{Ref Nat} \rightarrow \text{Nat} \rightarrow \text{Nat})\}$  we can derive  $z(xy) : \text{Nat} \wedge (\text{Nat} \rightarrow \text{Nat})$ , but there are no types  $\tau_1, \tau_2$  such that from the same environment we can derive  $z : \tau_1 \rightarrow \tau_2$  and  $xy : \tau_1$ . Instead we have  $\emptyset; \emptyset; \{x : (\sigma \rightarrow \sigma) \wedge (\sigma \rightarrow \tau), y : \sigma, z : (\sigma \rightarrow \sigma) \wedge (\tau \rightarrow \sigma \rightarrow \sigma)\} \vdash z : \sigma \wedge \tau \rightarrow \sigma \wedge (\sigma \rightarrow \tau)$  and  $\emptyset; \emptyset; \{x : (\sigma \rightarrow \sigma) \wedge (\sigma \rightarrow \tau), y : \sigma, z : (\sigma \rightarrow \sigma) \wedge (\tau \rightarrow \sigma \rightarrow \sigma)\} \vdash x : \sigma \wedge \tau$  for all  $\sigma, \tau$  of kind **S**. Similarly we can derive  $\emptyset; \emptyset; \{y : (\text{Ref Nat} \rightarrow \text{Ref Nat}) \wedge (\text{Ref (Nat} \rightarrow \text{Nat)} \rightarrow \text{Ref (Nat} \rightarrow \text{Nat)})\} \vdash \text{fix } x.yx : \text{Ref (Nat} \wedge (\text{Nat} \rightarrow \text{Nat}))$ , but we cannot derive  $\emptyset; \emptyset; \{x : \text{Ref (Nat} \wedge (\text{Nat} \rightarrow \text{Nat})), y : (\text{Ref Nat} \rightarrow \text{Ref Nat}) \wedge (\text{Ref (Nat} \rightarrow \text{Nat)} \rightarrow \text{Ref (Nat} \rightarrow \text{Nat)})\} \vdash yx : \text{Ref (Nat} \wedge (\text{Nat} \rightarrow \text{Nat}))$ .

The typing system enjoys the standard Substitution Property, that can be proved by induction on derivations.

**Lemma 4 (Substitution).** *If  $\Delta; \Sigma; \Gamma, x : \tau \vdash M : \sigma$  and  $\Delta; \Sigma; \Gamma \vdash N : \tau$ , then  $\Delta; \Sigma; \Gamma \vdash M[N/x] : \sigma$ .*

In order to prove subject reduction for our type system we need to show that typing is preserved under the replacement of a type by a smaller one in the type environment.

**Lemma 5.** *Let  $\Delta; \Sigma; \Gamma, x : \sigma \vdash M : \tau$  and  $\Delta \vdash \sigma' \leq \sigma$ . Then  $\Delta; \Sigma; \Gamma, x : \sigma' \vdash M : \tau$ .*

The *agreement* between a store environment and a store is defined as usual [10] [Definition 13.5.1].

**Definition 2.** *We say that a store environment  $\Sigma$  agrees with a store  $\mu$  (notation  $\Sigma \vdash \mu$ ) if:*

- $(l = V) \in \mu$  implies  $l : \tau \in \Sigma$  and  $\Delta; \Sigma; \emptyset \vdash V : \tau$  for some  $\tau$ ;
- $l : \tau \in \Sigma$  implies  $(l = V) \in \mu$  and  $\Delta; \Sigma; \emptyset \vdash V : \tau$  for some  $V$ .

Now we can prove subject reduction.

**Theorem 1 (Subject Reduction).**  *$\Delta; \Sigma; \Gamma \vdash M : \tau$  and  $\Sigma \vdash \mu$  and  $M \# \mu \longrightarrow N \# \mu'$  imply  $\Delta; \Sigma'; \Gamma \vdash N : \sigma$  and  $\Sigma' \vdash \mu'$  for some  $\Sigma' \supseteq \Sigma$ .*

*Proof.*  $M \# \mu \longrightarrow N \# \mu'$  implies that  $M = \mathcal{E}[M']$  and  $N = \mathcal{E}[N']$ , for some evaluation context  $\mathcal{E}$ . The proof is given by induction on  $\mathcal{E}$ . We consider the most interesting cases for  $\mathcal{E} = []$  since the induction cases are straightforward.

If the rule applied is  $(\beta_V)$ , then  $M = (\lambda x.P)V$ , and  $N = P[V/x]$ . From Lemma 3(3), for some  $\bar{l}, \bar{\kappa}, I, \sigma_i$  and  $\forall$ -top-free  $\tau_i$  ( $i \in I$ ),

- (1)  $\Delta \vdash \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} \tau_i \leq \tau$ ,
- (2)  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash \lambda x.P : \sigma_i \rightarrow \tau_i$  ( $i \in I$ ),
- (3)  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash V : \sigma_i$  ( $i \in I$ ),

From Lemma 3(2), and Point (2), for all  $i \in I$ , there are  $\overline{v^{(i)}}$ ,  $\overline{\kappa^{(i)}}$ ,  $H_i$ ,  $\sigma_j^{(i)}$ , and  $\tau_j^{(i)}$

( $j \in H_i$ ) such that  $\sigma_j^{(i)} \rightarrow \tau_j^{(i)}$  is  $\forall$ -top-free and

- (a)  $\Delta, \bar{l} : \bar{\kappa} \vdash \forall \overline{v^{(i)}} : \overline{\kappa^{(i)}}. \bigwedge_{j \in H_i} (\sigma_j^{(i)} \rightarrow \tau_j^{(i)}) \leq \sigma_i \rightarrow \tau_i$ ,
- (b)  $\Delta, \bar{l} : \bar{\kappa}, \overline{v^{(i)}} : \overline{\kappa^{(i)}}; \Sigma; \Gamma, x : \sigma_j^{(i)} \vdash P : \tau_j^{(i)}$  ( $j \in H_i$ ).

Note that  $\sigma_j^{(i)} \rightarrow \tau_j^{(i)}$   $\forall$ -top-free implies  $\tau_j^{(i)}$   $\forall$ -top-free. Then Lemma 2 and Point (a)

imply that there are  $\overline{\rho^{(i)}}$ , and  $J_i \subseteq H_i$ , such that:

- ( $\alpha$ )  $\Delta, \bar{l} : \bar{\kappa} \vdash \overline{\rho^{(i)}} :: \overline{\kappa^{(i)}}$  ( $i \in I$ ),
- ( $\beta$ )  $\Delta, \bar{l} : \bar{\kappa} \vdash \sigma_i \leq \sigma_j^{(i)} [\overline{\rho^{(i)}} / \overline{v^{(i)}}]$  ( $j \in J_i$ ), and
- ( $\gamma$ )  $\Delta, \bar{l} : \bar{\kappa} \vdash \bigwedge_{j \in J_i} \tau_j^{(i)} [\overline{\rho^{(i)}} / \overline{v^{(i)}}] \leq \tau_i$ .

From Points (b), ( $\alpha$ ), and Proposition 1, for all  $j \in J_i$ , we derive

$$\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma, x : \sigma_j^{(i)} [\overline{\rho^{(i)}} / \overline{v^{(i)}}] \vdash P : \tau_j^{(i)} [\overline{\rho^{(i)}} / \overline{v^{(i)}}]$$

and by Point ( $\beta$ ), and Lemma 5 we get:

$$\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma, x : \sigma_i \vdash P : \tau_j^{(i)} [\overline{\rho^{(i)}} / \overline{v^{(i)}}].$$

Applying rules  $(\wedge)$ ,  $(\leq)$  by Point ( $\gamma$ ) we derive

$$\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma, x : \sigma_i \vdash P : \tau_i$$

which by Lemma 4, and Point (3), implies that  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash P[V/x] : \tau_i$  for all  $i \in I$ .

With multiple applications of rule  $(\wedge)$  we get  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash P[V/x] : \bigwedge_{i \in I} \tau_i$ , and then applying many times rule  $(\forall I)$  (note that we can assume  $\bar{l} \notin \mathcal{S}V(\Sigma, \Gamma)$  since we take types modulo  $\equiv$ ) we derive  $\Delta; \Sigma; \Gamma \vdash P[V/x] : \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} \tau_i$ . Finally from Point (1) and rule  $(\leq)$  we conclude  $\Delta; \Sigma; \Gamma \vdash P[V/x] : \tau$ .

If the rule applied is  $(fixR)$ , then  $M = fixx.P$ , and  $N = P[fixx.P/x]$ . From Lemma 3(4), for some  $\bar{l}, \bar{\kappa}, I, \tau_i$  ( $i \in I$ ), we get  $\Delta \vdash \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} \tau_i \leq \tau$  and  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma, x : \tau_i \vdash P : \tau_i$  ( $i \in I$ ). Therefore, rule  $(fix)$  of Fig. 5 implies  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash fixx.P : \tau_i$  ( $i \in I$ ). From the Substitution Lemma 4 we derive  $\Delta, \bar{l} : \bar{\kappa}; \Sigma; \Gamma \vdash P[fixx.P/x] : \tau_i$  ( $i \in I$ ). Applying  $(\wedge I)$ 's,  $(\forall I)$ 's, and  $(\leq)$  we conclude  $\Delta; \Sigma; \Gamma \vdash P[fixx.P/x] : \tau$ .

If the rule applied is  $(refR)$ , then  $M = refV$ ,  $N = l$ , and  $\mu' = \mu, (l = V)$ . From Lemma 3(7), for some  $\bar{l}, \bar{\kappa}, I$ , closed  $\tau_i$  ( $i \in I$ ), we get  $\Delta \vdash \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} Ref \tau_i \leq \tau$  and  $\Delta; \Sigma; \Gamma \vdash V : \tau_i$  ( $i \in I$ ). Let  $\Sigma' = \Sigma, l : \bigwedge_{i \in I} \tau_i$ , we have that  $\Delta; \Sigma'; \Gamma \vdash l : Ref \bigwedge_{i \in I} \tau_i$ . Therefore, since  $\tau_i$  are closed, we have that  $Ref \bigwedge_{i \in I} \tau_i \equiv \forall \bar{l} : \bar{\kappa}. \bigwedge_{i \in I} Ref \tau_i$ . Applying rule  $(\leq)$  we conclude  $\Delta; \Sigma'; \Gamma \vdash l : \tau$ . From  $\Sigma \vdash \mu$  and  $\Delta; \Sigma; \Gamma \vdash V : \bigwedge_{i \in I} \tau_i$  we also get  $\Sigma' \vdash \mu'$ .

If the rule applied is  $(locR)$ , then result derives directly from the fact that  $\Sigma \vdash \mu$ .

If the rule applied is  $(unitR)$ , then  $M = l := V$ ,  $\mu = \mu'', (l = V')$ , and  $N = ()$ ,  $\mu' = \mu'', (l = V)$ . From Lemma 3(8),  $\tau \equiv Unit$ , and for some closed  $\sigma$  we have  $\Delta; \Sigma; \Gamma \vdash l : Ref \sigma$ , and  $\Delta; \Sigma; \Gamma \vdash V : \sigma$ . The typing rule  $(Unit_{()})$  gives  $\Delta; \Sigma; \Gamma \vdash () : Unit$ . From  $\Delta; \Sigma; \Gamma \vdash l : Ref \sigma$ , and Lemma 3(5),  $l : \sigma' \in \Sigma$  for some  $\sigma'$ , and  $Ref \sigma \equiv Ref \sigma'$ , which implies  $\sigma \equiv \sigma'$ . From  $\Sigma \vdash \mu$  in order to show  $\Sigma \vdash \mu'$  we have only to prove that  $\Delta; \Sigma; \Gamma \vdash V : \sigma'$ , which is immediate since  $\Delta; \Sigma; \Gamma \vdash V : \sigma$  and  $\sigma \equiv \sigma'$ .  $\square$

*Remark 1.* Note that the proof of subject reduction for the case of rule  $(\beta_v)$  extends without modifications to rule  $(\beta)$ . Moreover, it is easy to check that the proof works for arbitrary contexts. So we can conclude that subject reduction for our type assignment system holds independently from the used reduction strategy.

In order to prove the progress of our type system we need a Canonical Form Lemma which can be proved in a standard way, see [10], by analyzing the typing rules and the syntax of values.

### Lemma 6 (Canonical Forms).

1.  $\Delta; \Sigma; \emptyset \vdash V : Pos$  implies  $V \in \{1, 2, \dots\}$ .
2.  $\Delta; \Sigma; \emptyset \vdash V : Nat$  implies  $V \in \{0, 1, 2, \dots\}$ .
3.  $\Delta; \Sigma; \emptyset \vdash V : Unit$  implies  $V = ()$ .
4.  $\Delta; \Sigma; \emptyset \vdash V : \tau \rightarrow \sigma$  implies  $V = \lambda x.M$ .
5.  $\Delta; \Sigma; \emptyset \vdash V : Ref \tau$  implies  $V = l$  and  $l : \sigma \in \Sigma$  for some  $\sigma$ .

**Theorem 2 (Progress).** *Let  $M \in \Lambda_{imp}^0$ . Then  $\Delta; \Sigma; \emptyset \vdash M : \sigma$  implies that either  $M$  is a value or for all  $\mu$  such that  $\Sigma \vdash \mu$  we have that  $M \# \mu \longrightarrow N \# \mu'$  for some  $N, \mu'$ .*

*Proof.* The proof is by induction on the derivation  $\Sigma; \Gamma \vdash M : \tau$ .

If the last applied rule is  $(\rightarrow I)$ ,  $(Unit_{()})$ ,  $(loc)$ ,  $(Nat)$ , or  $(Pos)$ , then  $M$  is a value.

If the last applied rule is  $(fix)$ , then  $M$  is immediately reducible.

If the last applied rule is  $(\rightarrow E)$ , then  $M$  is  $NP$ , and  $\Delta; \Sigma; \emptyset \vdash N : \tau \rightarrow \rho$ , and  $\Delta; \Sigma; \emptyset \vdash P : \tau$ .

If  $N$  is a value, then by the Canonical Form Lemma 6(4),  $N = \lambda x.Q$ . If also  $P$  is a value, rule  $(\beta_v)$  applies. Otherwise, by induction hypothesis on  $\Delta; \Sigma; \emptyset \vdash P : \tau$ , for all  $\mu$  such that  $\Sigma \vdash \mu$  we have that  $P \# \mu \longrightarrow P' \# \mu'$  for some  $P'$  and  $\mu'$ . Therefore, for

some  $\mathcal{E}$ ,  $R$  and  $R'$ , we get  $P = \mathcal{E}[R]$  and  $P' = \mathcal{E}[R']$ . Consider the evaluation context  $\mathcal{E}' = (\lambda x.Q)\mathcal{E}$ . We have that  $\mathcal{E}'[R] = M$  and  $\mathcal{E}'[R] \# \mu \longrightarrow \mathcal{E}'[R'] \# \mu'$ .

If  $N$  is not a value, by induction hypothesis on  $\Delta; \Sigma; \emptyset \vdash N : \tau \rightarrow \rho$ , for all  $\mu$  such that  $\Sigma \vdash \mu$  we have that  $N \# \mu \longrightarrow N' \# \mu'$  for some  $N'$  and  $\mu'$ . Therefore, for some  $\mathcal{E}$ ,  $R$  and  $R'$ , we get  $N = \mathcal{E}[R]$  and  $N' = \mathcal{E}[R']$ . Consider the evaluation context  $\mathcal{E}' = \mathcal{E} P$ . We have that  $\mathcal{E}'[R] = M$  and  $\mathcal{E}'[R] \# \mu \longrightarrow \mathcal{E}'[R'] \# \mu'$ .

If the last applied rule is (Unit) then  $M$  is  $N := P$ , and  $\Delta; \Sigma; \emptyset \vdash N : \text{Ref } \tau$  and  $\Delta; \Sigma; \emptyset \vdash P : \tau$ .

If  $N$  is a value, from the Canonical Form Lemma 6(5),  $N = l$ , and  $l : \sigma \in \Sigma$ , for some  $\sigma$ . Moreover, from  $\Sigma \vdash \mu$ , we have that  $(l = V)$  for some  $V$ . If also  $P$  is a value, then rule (unitR) is applicable. Otherwise, if  $P$  is not a value, we apply the induction hypothesis to  $\Delta; \Sigma; \emptyset \vdash P : \tau$ , and derive that for all  $\mu$  such that  $\Sigma \vdash \mu$  we have that  $P \# \mu \longrightarrow P' \# \mu'$  for some  $P'$  and  $\mu'$ . Therefore, for some  $\mathcal{E}$ ,  $R$  and  $R'$ , we get  $P = \mathcal{E}[R]$  and  $P' = \mathcal{E}[R']$ . Consider the evaluation context  $\mathcal{E}' = l := \mathcal{E}$ . We have that  $\mathcal{E}'[R] = M$  and  $\mathcal{E}'[R] \# \mu \longrightarrow \mathcal{E}'[R'] \# \mu'$ .

If  $N$  is not a value, by induction hypothesis on  $\Delta; \Sigma; \emptyset \vdash N : \text{Ref } \tau$ , for all  $\mu$  such that  $\Sigma \vdash \mu$  we have that  $N \# \mu \longrightarrow N' \# \mu'$  for some  $N'$  and  $\mu'$ . Therefore, for some  $\mathcal{E}$ ,  $R$  and  $R'$ , we get  $N = \mathcal{E}[R]$  and  $N' = \mathcal{E}[R']$ . Consider the evaluation context  $\mathcal{E}' = \mathcal{E} := P$ . We have that  $\mathcal{E}'[R] = M$  and  $\mathcal{E}'[R] \# \mu \longrightarrow \mathcal{E}'[R'] \# \mu'$ .

The proof for the cases (Ref E), (Ref I), (if), and (+) are similar.

For rules ( $\wedge I$ ), ( $\forall I$ ) and ( $\leq$ ) the result follows directly by induction.  $\square$

Let us restrict the language to the pure  $\lambda$ -calculus. Then our type assignment system preserves the typability power of intersection types, i.e., it gives types to all and only the strongly normalizing terms. As far as the expressive power is concerned, we can compare our system with System F [5], in its type assignment version [9], with the intersection type assignment system of [11], and with the system defined in [8], where both intersection and universally quantified types are present. Let  $\vdash_F$  denote derivability in the type assignment version of system F [9] and  $\vdash_P$  in the intersection type assignment system of [11]. The system in [8] can give types to all terms, since it contains the universal type  $\omega$ , and a rule that assign  $\omega$  to all terms. Let us consider a restriction of this system, obtained from it by erasing both the type  $\omega$  and the related rule, for whose derivability we use  $\vdash_{MZ}$ . In order to prove that our system preserves the expressive power of  $\vdash_F$ ,  $\vdash_P$  and  $\vdash_{MZ}$ , we define a decorating function  $dec$ , transforming every type in [8] non containing occurrences of  $\omega$  in a type of our system, in the following way:

$$dec(\tau \text{ op } \sigma) = dec(\tau) \text{ op } dec(\sigma) \quad (\text{op} \in \{\rightarrow, \wedge\}) \quad dec(\forall t. \tau) = \forall t : \mathbf{S}. \tau$$

The function  $dec$  can be obviously applied also to the set of System F types and to the set of intersection types, which are proper subsets of the types in [8].

**Theorem 3.** *Let  $M$  be a term of the pure  $\lambda$ -calculus,  $\sigma$  a type,  $\Gamma$  a type environment and  $\Delta$  the kind environment which gives kind  $\mathbf{S}$  to all the type variables occurring in  $\sigma$  and  $\Gamma$ .*

1. *If  $\Gamma \vdash_{MZ} M : \sigma$ , then  $\Delta; \emptyset; \Gamma \vdash M : dec(\sigma)$ .*
2. *If  $\Gamma \vdash_F M : \sigma$ , then  $\Delta; \emptyset; \Gamma \vdash M : dec(\sigma)$ .*
3. *If  $\Gamma \vdash_P M : \sigma$ , then  $\Delta; \emptyset; \Gamma \vdash M : dec(\sigma)$ .*
4.  *$M$  is typable in the system of Fig. 5 if and only if it is strongly normalizing.*

*Proof.* Point 1 is immediate, since the rules of  $\vdash_{MZ}$  are a proper subset of our rules, and also the  $\leq$  relation on types is the same, when reference types are not present.

Points 2 and 3 follow from Point 1, since the rules of  $\vdash_F$  and of  $\vdash_P$  are a proper subsets of the rules of  $\vdash_{MZ}$ .

For Point 4 since all strongly normalising terms are typable in the system of [11] we get from Point 3 that all strongly Normalising terms are typable in our system. The vice versa can be proved by a standard use of the computability technique as done in [11].  $\square$

## 4 Conclusion

In this paper we discuss how to combine intersection, universally quantified and reference types in a meaningful way. The naive use of intersection and universally quantified types is unsound in presence of references, as shown in [3] and in the introduction of this paper. Davies and Pfenning solve the problem by restricting both the definition of the preorder relation  $\leq$  between types, and the type assignment system. In the preorder relation  $\leq$  between types they do not have the standard rules:

$$\begin{array}{ll} (\rightarrow \wedge) & (\tau \rightarrow \sigma) \wedge (\tau \rightarrow \rho) \leq \tau \rightarrow \sigma \wedge \rho \\ (\rightarrow \forall) & \forall t. \tau \rightarrow \sigma \leq \tau \rightarrow \forall t. \sigma \quad t \notin \mathcal{FV}(\tau) \end{array}$$

The type assignment system is restricted in such a way that the intersection and the universal quantification can be introduced just in case the subject is a value. Then the subject reduction property holds, for a call-by-value reduction semantics of terms. As already noticed in [4], while in this way they solve the problem described in the introduction, in the system there are unsound typings. In fact the term  $x := x + 1$  can be typed in their system, extended with the standard typing rule for the sum, through the following derivation:

$$\frac{\frac{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x : \text{Nat} \wedge \text{Ref Nat}}{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x : \text{Ref Nat}} (\leq) \quad \frac{\frac{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x : \text{Nat} \wedge \text{Ref Nat}}{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x : \text{Nat}} (\leq) \quad \frac{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x + 1 : \text{Nat}}{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x + 1 : \text{Nat}} (+)}{\Delta; \emptyset; x : \text{Nat} \wedge \text{Ref Nat} \vdash x := x + 1 : \text{Unit}} (\text{Unit})$$

In [4] a different solution is proposed, for a system with reference and intersection types only, which does restrict neither the definition of the  $\leq$  relation between types nor the type assignment system rules. In this system there cannot be intersections between reference and non-reference types, so, for example,  $\text{Nat} \wedge \text{Ref Nat}$  is not a type. Syntactically, this is realized through a partial intersection operator,  $\cap$ , which applied to two non-reference types returns their intersection, and applied to two reference types commutes with the  $\text{Ref}$  constructor pushing the operator inside the  $\text{Ref}$ . The system is shown to be sound and no expressive power is lost in comparison with the original system [2] [11] of intersection types when we restrict to the terms of pure  $\lambda$ -calculus.

In this paper, we consider a system with intersection, universally quantified and reference types. Our aim is to design a sound system having minimal restrictions. The solution adopted to avoid unsoundness is different from both [3] and [4], and leads to a

more elegant system. The only restriction we impose on types is that the `Ref` constructor can be applied only to closed types. So the quantification on reference types becomes meaningless, since  $\forall t. \text{Ref } \tau$  is equivalent to  $\text{Ref } \tau$ . Regarding intersection types we do not have restrictions. In particular, we may have intersection between reference and non-reference types. With a notion of kind and a kind assignment we keep track of potential reference types. The soundness is reached by limiting the definition of  $\leq$  relation between types in the rule for intersection elimination, which may only be applied if the intersection does not contain reference types. For our types rules  $(\rightarrow \wedge)$  and  $(\rightarrow \forall)$  hold in both directions. As a result, our system enjoys subject reduction independently from the reduction strategy. In fact the critical term  $(\lambda x. (\lambda y. !x)(x := 0))_{\text{ref } 1}$ , showed in the introduction, in our system has only types equivalent to  $\text{Nat}$ , which is the type of both 0 and 1, so the typing is preserved under any reduction strategy. Moreover unsound terms as the one shown before cannot be typed (but their sound versions  $x := !x + 1$  and  $\text{ref } x := x + 1$  are typable).

When restricted to the pure functional part of the language, our typing system has a stronger typability power than the system of [3]. As an example, consider the strongly normalizing pure  $\lambda$ -term  $(\lambda xy. (\lambda z. zz)(xy))(\lambda t. t)$  which is typable in our system (see Theorem 3(4)), while it is not typable in the system of [3]. For typing this term it is necessary to introduce an intersection between two subderivations whose subject is  $xy$ , and in the system of [3] this is not possible, since this subterm is not a value. More precisely, if  $\sigma_1 = (\text{Nat} \rightarrow \text{Nat}) \wedge ((\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Nat} \rightarrow \text{Nat})$  and  $\sigma_2 = \text{Nat} \wedge (\text{Nat} \rightarrow \text{Nat})$ , it is easy to verify that  $\lambda t. t$  has type  $\sigma_1$  and  $\lambda z. zz$  has type  $\sigma_2 \rightarrow \text{Nat}$ . Therefore, in order to type the above term we need to derive  $\Delta; \emptyset; \{x : \sigma_1, y : \sigma_2\} \vdash xy : \sigma_2$ , which requires the application of rule  $(\wedge I)$  to  $xy$ .

The system we define is clearly undecidable, as the subtyping itself is undecidable also when restricted to the types of System F, as proved in [13]. Moreover, type inference for the systems of [5] and [11] is undecidable, as proved in [14] and [11], respectively. Therefore, the present system cannot be proposed for real programming. The interest of this paper is merely foundational, since it explores the difficulties in putting together different type constructs, and formalizes a sound proposal which enhances previous solutions. A future work will be to tailor a proper subsystem of our typing system, possibly using bidirectional type checking as proposed in [3], with the property of being decidable while preserving a good expressive power.

**Acknowledgements.** We gratefully acknowledge fruitful discussions with Frank Pfenning and Betti Venneri. We also thank the referees for their comments.

## References

1. Coppo, M.: An Extended Polymorphic Type System for Applicative Languages. In: Dembinski, P. (ed.) MFCS 1980. LNCS, vol. 88, pp. 194–204. Springer, Heidelberg (1980)
2. Coppo, M., Dezani-Ciancaglini, M.: An Extension of the Basic Functionality Theory for the  $\lambda$ -calculus. *Notre Dame Journal of Formal Logic* 21(4), 685–693 (1980)
3. Davies, R., Pfenning, F.: Intersection Types and Computational Effects. In: Wadler, P. (ed.) ICFP 2000. SIGPLAN Notices, vol. 35(9), pp. 198–208. ACM Press, New York (2000)

4. Dezani-Ciancaglini, M., Ronchi Della Rocca, S.: Intersection and Reference Types. In: Barendsen, E., Capretta, V., Geuvers, H., Niqui, M. (eds.) *Reflections on Type Theory, Lambda Calculus, and the Mind*, pp. 77–86. Radboud University Nijmegen (2007)
5. Girard, J.-Y.: *Interprétation Fonctionnelle et Elimination des Coupures de l'Arithmétique d'Ordre Supérieur*. Thèse d'Etat, Université de Paris VII (1972)
6. Leivant, D.: Polymorphic Type Inference. In: Demers, A., Teitelbaum, T. (eds.) *POPL 1983*, pp. 88–98. ACM Press, New York (1983)
7. Lucassen, J.M.: *Types and Effects: Towards the Integration of Functional and Imperative Programming*. Ph. d. thesis, Massachusetts Institute of Technology (1987)
8. Margaria, I., Zacchi, M.: Principal Typing in a  $\forall\wedge$ -Discipline. *Journal of Logic and Computation* 5(3), 367–381 (1995)
9. Mitchell, J.: Polymorphic Type Inference and Containment. *Information and Computation* 76(2/3), 211–249 (1988)
10. Pierce, B.C.: *Types and Programming Languages*. MIT Press, Cambridge (2002)
11. Pottinger, G.: A Type Assignment for the Strongly Normalizable  $\lambda$ -terms. In: Hindley, R., Seldin, J.P. (eds.) *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pp. 561–577. Academic Press, London (1980)
12. Reynolds, J.C.: Towards a Theory of Type Structure. In: Loecks, J. (ed.) *Colloque sur la Programmation*. LNCS, vol. 19, pp. 408–425. Springer, Heidelberg (1974)
13. Tiuryn, J., Urzyczyn, P.: The Subtyping Problem for Second-Order Types Is Undecidable. *Information and Computation* 179(1), 1–18 (2002)
14. Wells, J.B.: Typability and Type Checking in System F are Equivalent and Undecidable. *Annals of Pure and Applied Logic* 98(1-3), 111–156 (1999)
15. Wright, A.K., Felleisen, M.: A Syntactic Approach to Type Soundness. *Information and Computation* 115, 38–94 (1994)

# Linear Game Automata: Decidable Hierarchy Problems for Stripped-Down Alternating Tree Automata

Jacques Duparc<sup>1</sup>, Alessandro Facchini<sup>1,2,\*</sup>, and Filip Murlak<sup>3,\*\*</sup>

<sup>1</sup> University of Lausanne

<sup>2</sup> University of Bordeaux 1

<sup>3</sup> University of Edinburgh

{jacques.duparc, alessandro.facchini}@unil.ch,

fmurlak@inf.ed.ac.uk

**Abstract.** For deterministic tree automata, classical hierarchies, like Mostowski-Rabin (or index) hierarchy, Borel hierarchy, or Wadge hierarchy, are known to be decidable. However, when it comes to non-deterministic tree automata, none of these hierarchies is even close to be understood. Here we make an attempt in paving the way towards a clear understanding of tree automata. We concentrate on the class of linear game automata (LGA), and prove within this new context, that all corresponding hierarchies mentioned above—Mostowski-Rabin, Borel, and Wadge—are decidable. The class LGA is obtained by taking linear tree automata with alternation restricted to the choice of path in the input tree. Despite their simplicity, LGA recognize sets of arbitrary high Borel rank. The actual richness of LGA is revealed by the height of their Wadge hierarchy:  $(\omega^\omega)^\omega$ .

## 1 Introduction

The Mostowski–Rabin hierarchy, the Borel hierarchy, and the Wadge hierarchy are the most common measures of complexity of recognizable  $\omega$ -languages.

The first one, also known as the index hierarchy, orders languages according to the nesting of positive and negative conditions checked by the recognizing automaton. It has two main versions: weak, relying on finitary conditions (e.g., “ $a$  does occur”); and strong, referring to infinitary conditions (e.g., “ $b$  occurs infinitely often”). It is believed to reflect the inherent computational complexity of the language, and therefore has attracted a lot of attention encouraged by the expectations of the verification community [3,4,10,17,18,19,20].

The classical Borel hierarchy is based on the nesting of countable unions and negations in the set theoretic definition of the language, starting from the simplest (open) sets. It drew attention of automata theorists as early as 1960s [12], and has continued

---

\* Research supported by a grant from the Swiss National Science Foundation, n. 100011-116508: Project “Topological Complexity, Games, Logic and Automata”.

\*\* On leave from the University of Warsaw; partially supported by the Polish government grant no. N206 008 32/0810.



to inspire research efforts ever since, mainly because of its intimate relations with the index hierarchy [9][19][23].

The Wadge hierarchy is an almost ultimate refinement of the Borel hierarchy, defined by the preorder induced on languages by simple (continuous) reductions. It enables precise comparison of different models of computation. What is more powerful: deterministic or weak automata on trees? It is known that there are deterministic languages that are not weakly recognizable and vice versa. How to compare, if not by inclusion? An even more exotic case: deterministic tree languages versus deterministic context free word languages. How to compare trees with words? The Wadge hierarchy makes it possible. The sole heights (huge ordinals) of the Wadge hierarchy restricted to the classes under comparison provide, literally, infinitely more information than other logical techniques [6][8][15][22].

Measuring hardness of recognizable languages of infinite trees is a long standing open problem. Unlike for infinite words, where the understanding is almost complete since Wagner's 1977 paper [25], for trees the only satisfyingly examined case is that of deterministic automata [14][15][16][19][20]. But the deterministic and non-deterministic case differ immensely for trees. The only results obtained for non-deterministic or alternating automata are strictness theorems for various classes [3][4][13][17], and lower bounds for the heights of the hierarchies [7][23]. To the best of our knowledge, the only nontrivial decidability result is that on emptiness [21]. As the empty set and the whole space are the only two sets on the lowest level of the Wadge hierarchy (or the Mostowski–Rabin hierarchy), using emptiness test and the complementation procedure [21] we can decide if a given language is on the first level of the hierarchy or not. Obviously this does not say much about the complexity of the language in question.

This paper intends to change this situation, even if only very slightly for a start. We propose a class of automata having all three hierarchies decidable and capturing a reasonable amount of non-determinism. The class we advocate, *linear game automata* (LGA), is obtained by taking linear automata (a.k.a. very weak automata), that emerged in the verification community, and restricting the alternation to the choice of a path in the input tree. Linear automata capture CTL [11], which is expressive enough for many applications. Though linear game automata are weaker, they retain most alternation related to the branching structure. Evidence for their expressivity is topological: they recognize sets of arbitrarily high finite Borel rank, and their Wadge hierarchy has the height  $(\omega^\omega)^\omega$ , much larger than  $(\omega^\omega)^3 + 3$  for deterministic automata.

As we have already pointed out, these automata are far from capturing the full expressivity of non-deterministic automata, but still, computing the Wadge degree for a given LGA is much more involved than for an  $\omega$ -word automaton and even a deterministic tree automaton. The structural simplicity of LGA might seem to reduce the computation to the decomposition of nested chains, but in fact the alternation (even very weak) makes it much harder. We believe that the notion of game automata is well suited to take us further. Indeed, the next step is to consider weak and then strong game automata. This last class is already quite expressive, as it contains inhabitants of every level of the (strong) index hierarchy and subsumes deterministic languages. Extending decidability to this class would be an important result, though possibly the last one accessible with the tools we are using.

## 2 Preliminaries

### 2.1 Weak Automata

Let  $W$  be a non empty set. A tree over  $\Sigma$  is a partial function  $t : W^* \rightarrow \Sigma$  with a prefix closed domain. A tree is called *full* if  $\text{dom}(t) = W^*$ , and it is called *binary* if  $W = \{0, 1\}$ . Let  $T_\Sigma$  denote the set of full binary trees over  $\Sigma$ . In the sequel we only consider full binary trees. Given  $v \in \text{dom}(t)$ , by  $t.v$  we denote the subtree of  $t$  rooted in  $v$ . We write  $\bar{d}$  to denote the other direction:  $\bar{0} = 1, \bar{1} = 0$ .

A *weak alternating tree automaton*  $A = \langle \Sigma, Q, q_I, \delta, \text{rank} \rangle$  consists of a finite input alphabet  $\Sigma$ , a finite set of states  $Q$  partitioned into the existential states  $Q_\exists$  and the universal states  $Q_\forall$ , an initial state  $q_I$ , a transition relation  $\delta \subseteq Q \times \Sigma \times \{\epsilon, 1, 0\} \times Q$  and a bounded priority function  $\text{rank} : Q \rightarrow \omega$ . Sometimes we write  $q \xrightarrow{\sigma, d} q'$  when  $q' \in \delta(q, \sigma, d)$ . The acceptance is defined in terms of a (weak) parity game.

A *weak parity game* is a two-player game given by  $\langle V, V_0, V_1, E, \text{rank} \rangle$ , where  $V = V_0 \cup V_1$  is the set of vertices,  $E \subseteq V \times V$  is the edge relation, and  $\text{rank} : V \rightarrow \omega$  is the *priority function* with bounded image. A vertex  $v'$  is a successor of a vertex  $v$  if  $(v, v') \in E$ . A play from a vertex  $v_0$  is a path  $v_0 v_1 v_2 \dots$  visited by a token moving along the edges of the graph. If the token is in  $v \in V_i$ , player  $i$  chooses the next location of the token among the successors of  $v$ . We say that player 0 wins a (finite or infinite) play if and only if the greatest priority ever occurring in the play is even.

Consider a weak alternating automaton  $A$  and a tree  $t \in T_\Sigma$ . The automaton  $A$  accepts  $t$  iff Player 0 has a winning strategy in the *weak* parity game  $\mathcal{G}_{A,t}$  defined as:

- $V_0 = \{0, 1\}^* \times Q_\exists, V_1 = \{0, 1\}^* \times Q_\forall$ ,
- the relation  $E = \{((v, p), (vd, q)) : v \in \text{dom}(t), (p, t(v), d, q) \in \delta\}$ ,
- $\text{rank}((v, q)) = \text{rank}(q)$ , for every vertex  $(v, q)$ .

A *path* in  $A$  is a sequence of states and transitions  $q_0 \xrightarrow{\sigma_0, d_0} q_1 \xrightarrow{\sigma_1, d_1} q_2 \dots \dots q_n \xrightarrow{\sigma_n, d_n} q_{n+1}$ . If there is such a path with  $q = q_0$  and  $q' = q_{n+1}$ , we say that  $q'$  is *reachable* from  $q$ . A path is a *loop* if  $q_{n+1} = q_0$ . If there is a loop from a state  $q$ , we say that this state is *looping*. If  $q$  is looping and  $\text{rank}(q)$  is even (resp. odd) we say that the loop in  $q$  is *positive* (resp. *negative*). Finally, we say that a state  $p$  is *replicated* by  $q$  if there is a path  $q \xrightarrow{\sigma_0, d_0} q_1 \dots q_n \xrightarrow{\sigma_n, d_n} p$  and a transition  $q \xrightarrow{\sigma_0, \bar{d}_0} q$ .

### 2.2 Borel Classes and Wadge Reductions

Consider the space  $T_\Sigma$  equipped with the standard Cantor topology (see eg. [19]). Recall that the class of Borel sets of a topological space  $X$  is the closure of the class of open sets of  $X$  by countable unions and complementation. Given  $X$ , the initial finite levels of the Borel hierarchy are defined as follows with  $\Sigma_0^0(X) = \{\emptyset\}$  and  $\Pi_0^0(X) = \{X\}$ .

- $\Sigma_1^0(X)$  is the class of open subsets of  $X$ ,
- $\Pi_n^0(X)$  contains complements of sets from  $\Sigma_n^0(X)$ ,
- $\Sigma_{n+1}^0(X)$  contains countable unions of sets from  $\Pi_n^0(X)$ .

The classes defined above are closed under inverse images of continuous functions. Given a classe  $\mathcal{C}$ , a set  $U$  is called  $\mathcal{C}$ -hard if each set in  $\mathcal{C}$  is an inverse image of  $U$  under some continuous function. If additionally  $U \in \mathcal{C}$ ,  $U$  is said to be  $\mathcal{C}$ -complete. It is well known that every weakly recognizable tree language is a member of a Borel class of finite rank ([7][13]). The rank of a language is the rank of the minimal Borel class the language belongs to. It can be seen as a coarse measure of complexity of languages.

A much finer measure of the topological complexity is the *Wadge degree*. If  $T, U \subseteq T_\Sigma$ , we say that  $T$  is *continuously (or Wadge) reducible* to  $U$ , if there exists a continuous function  $f$  such that  $T = f^{-1}(U)$ . We write  $T \leq_w U$  iff  $T$  is continuously reducible to  $U$ . Thus, given a certain Borel class  $\mathcal{C}$ ,  $T$  is  $\mathcal{C}$ -hard if  $U \leq_w T$  for every  $U \in \mathcal{C}$ . This particular ordering is called the *Wadge ordering*. If  $T \leq_w U$  and  $U \leq_w T$ , then we write  $T \equiv_w U$ . If  $T \leq_w U$  but not  $U \leq_w T$ , then we write  $T <_w U$ . The Wadge hierarchy is the partial order induced by  $<_w$  on the equivalence classes given by  $\equiv_w$ .

Let  $T$  and  $U$  be two arbitrary sets of full binary trees. The *Wadge game*  $\mathcal{W}(T, U)$  is a two-player game (player I and player II). During a play, each player builds a tree, say  $t_I$  and  $t_{II}$ . In each round both players add children to some terminal nodes of their corresponding tree. Player I plays first and Player II is allowed to skip his turn but not forever. Player II wins the game iff  $t_I \in T \Leftrightarrow t_{II} \in U$ . Bill Wadge designed this game precisely in order to obtain a characterisation of continuous reducibility.

**Lemma 1 ([24]).** *Let  $T, U \subseteq T_\Sigma$ . Then  $T \leq_w U$  iff Player II has a winning strategy in the game  $\mathcal{W}(T, U)$ .*

A language  $L$  is called *self dual* if it is equivalent to its complement, otherwise it is called *non self dual*. From Borel determinacy, if  $T, U \subseteq T_\Sigma$  are Borel, then  $\mathcal{W}(T, U)$  is determined. As a consequence, a variant of Martin-Monk’s result shows that  $<_w$  is well-founded. The *Wadge degree* for sets of finite Borel rank is inductively defined by:

- $d_w(\emptyset) = d_w(\emptyset^c) = 1$ ,
- $d_w(L) = \sup\{d_w(K) + 1 : K \text{ non self dual, } K <_w L\}$  for  $L >_w \emptyset$ .

### 2.3 Linear Game Automata

A *linear game automaton* (LGA) is a weak alternating automaton  $A = \langle \Sigma, Q, q_I, \delta, \text{rank} \rangle$  satisfying two special restrictions:

- (game alternation) the transition relation is a total function  $\delta : Q \times \Sigma \rightarrow Q \times Q$ ;
- (linearity) for every loop  $q \xrightarrow{\sigma_0, d_0} q_1 \xrightarrow{\sigma_1, d_1} q_2 \cdots q_n \xrightarrow{\sigma_n, d_n} q$  it holds that  $q_i = q$ , for all  $1 \leq i \leq n$ .

In the remaining of the paper, we often write  $q \xrightarrow{\sigma} q_0, q_1$  if  $\delta(q, \sigma) = (q_0, q_1)$ . Let  $A_q$  denote the automaton obtained from  $A$  by changing the initial state to  $q$ . Without loss of generality, we make the following assumptions:

- there is no trivial state, i.e., if  $q \in Q$  is such that  $A_q \equiv \top$  (resp.  $A_q \equiv \perp$ ), then  $q = \top$  (resp.  $q = \perp$ ),
- there is no trivial transition, i.e., if  $p \in Q_\forall$ , and  $p \xrightarrow{\sigma} q, \perp$ , then  $q = \perp$  (dually for  $p \in Q_\exists$ ).

By convention,  $\top$  is a looping state of even rank, and  $\perp$  is a looping state of odd rank.

LGA are closed under complementation. The usual complementation procedure, that increases the ranks by one and swaps existential and universal states turns LGA into LGA. However, LGA are not closed under union nor intersection. Given  $\sigma \in \Sigma$ , the language  $L_\sigma = \{t \in T_\Sigma : t(0) = t(1) = \sigma\}$  is LGA-recognizable, but  $L_\sigma \cup L_{\sigma'}$  is not.

### 2.4 A Normal Form

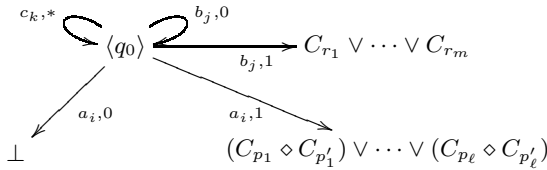
We now provide a useful normal form of LGA. First let us define three operations on tree languages (and tree automata). Let  $L, M$  be tree languages over  $\Sigma$  containing at least two letters,  $a$  and  $b$ . Define *alternative* ( $\vee$ ), *disjunctive product* ( $\diamond$ ), and *conjunctive product* ( $\square$ ) as

$$\begin{aligned} L \vee M &= \{t : t(\varepsilon) = a, t.0 \in L \text{ or } t(\varepsilon) \neq a, t.0 \in M\}, \\ L \diamond M &= \{t : t.0 \in L \text{ or } t.1 \in M\}, \\ L \square M &= \{t : t.0 \in L \text{ and } t.0 \in M\}. \end{aligned}$$

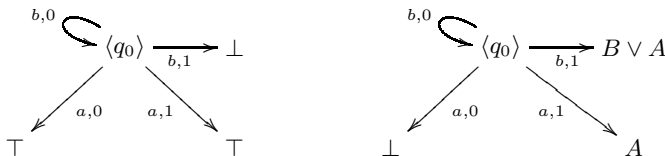
The family of languages recognized by LGAs is closed under these three operations. In particular, the operations have natural counterparts on automata. We write  $A \vee B$  to denote the automaton recognizing  $L(A) \vee L(B)$ , and similarly for  $\diamond$  and  $\square$ . Multifold alternatives are performed from left to right, e.g.,  $A_1 \vee A_2 \vee A_3 \vee A_4 = (((A_1 \vee A_2) \vee A_3) \vee A_4)$ . It is easy to see that these three operations define associative and commutative operations on Wadge equivalence classes.

**Lemma 2.** *Each LGA is Wadge equivalent to an LGA over the alphabet  $\{a, b\}$ .*

*Proof.* We proceed by induction on the number of states. Let  $C$  be an LGA. If  $C$  has only one state, the claim follows trivially. Suppose  $C$  has several states. We may assume w.l.o.g. that its initial state of  $C$ ,  $q_0$ , is existential. Suppose that the transitions of  $C$  starting in  $q_0$  are  $q_0 \xrightarrow{a_i} p_i, p'_i, q_0 \xrightarrow{b_j} q_0, r_i$  and  $q_0 \xrightarrow{c_k} q_0, q_0$  with  $\Sigma = \{a_1, \dots, a_\ell; b_1, \dots, b_m; c_1, \dots, c_n\}$ . Then  $C$  is Wadge equivalent to



By induction hypothesis, there exist automata  $A_i, A'_i, B_j$  over  $\{a, b\}$ , such that  $A_i \equiv_w C_{p_i}, A'_i \equiv_w C_{p'_i}$ , and  $B_j \equiv_w C_{r_j}$ . Let  $A = (A_1 \diamond A'_1) \vee \dots \vee (A_\ell \diamond A'_\ell)$  and  $B = B_1 \vee \dots \vee B_m$ . Further, we see that if  $A \vee B \equiv_w \top$ , then  $C$  is Wadge equivalent to the automaton on the left below and otherwise to the one on the right:



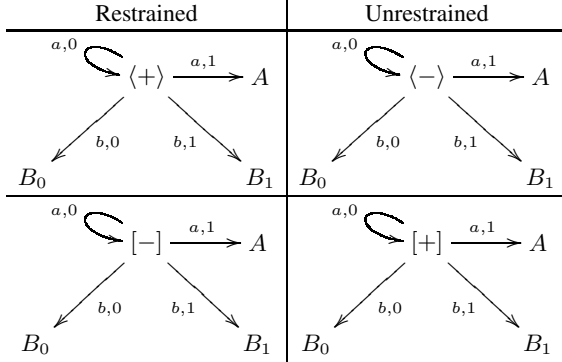
□

From now on we work with automata over  $\{a, b\}$ , unless explicitly stated otherwise.

A looping state  $q$  of an LGA  $A$  is

- *restrained* if it is an existential positive state or a universal negative state,
- *unrestrained* if it is an existential negative state or a universal positive state.

Examining the proof of Lemma 2, we see that in fact, each nontrivial looping state falls into exactly one of the categories shown below (+ means even rank, – means odd rank).



A node  $q$  of each of the above kinds may be seen as an action over triples of LGAs; we denote by  $q(A, B_0, B_1)$  the automaton being the result of the action  $q$  on  $A, B_0, B_1$ , e.g.,  $[+](A, B_0, B_1)$  or  $\langle - \rangle(A, B_0, B_1)$ . Often we use a shorthand  $[\mu](A, B) = [\mu](A, B, \top)$ ,  $\langle \mu \rangle(A, B) = \langle \mu \rangle(A, B, \perp)$  for  $\mu = +$  or  $\mu = -$ .

### 3 Deciding the Borel Hierarchy

#### 3.1 Patterns Menagerie

The basis for the procedure computing the Borel rank of a given LGA-recognizable language is a characterization in terms of difficult patterns. We define  $(0, n)$ -pattern, and  $(1, n + 1)$ -pattern by induction on  $n$ :

- a  $(0, 1)$ -pattern is a negative loop reachable from a positive loop,
- a  $(1, 2)$ -pattern is a positive loop reachable from a negative loop,
- a  $(0, n + 1)$ -pattern is a  $(1, n + 1)$ -pattern replicated by a universal positive node,
- a  $(1, n + 2)$ -pattern is a  $(0, n)$ -pattern replicated by an existential negative node.

We also define canonical automata,  $K_n^\Sigma$  and  $K_n^\Pi$ , corresponding to the patterns:

$$\begin{aligned}
 K_1^\Pi &= [+](\top, \perp, \perp), & K_{n+1}^\Pi &= [+](K_n^\Sigma, \perp, \perp), \\
 K_1^\Sigma &= \langle - \rangle(\perp, \top, \top), & K_{n+1}^\Sigma &= \langle - \rangle(K_n^\Pi, \top, \top).
 \end{aligned}$$

The tree languages recognized by the above canonical automata coincide with the sets used by Skurczyński to prove the existence of weakly recognizable languages of each finite Borel rank.

**Proposition 1 ([23]).** For every  $n > 0$ ,  
 $L(K_n^\Sigma)$  is  $\Sigma_n^0$ -complete and  $L(K_n^\Pi)$  is  $\Pi_n^0$ -complete.

Skurczyński's result follows by straightforward induction from the following easy lemma. For  $v \in \{0, 1\}^*$  and  $U \subseteq T_\Sigma$ , let  $vU = \{t \in T_\Sigma : t.v \in U\}$ .

**Lemma 3.** For each  $n > 0$

1. if  $U_i$  is  $\Sigma_n^0$ -hard for  $i < \omega$ ,  $\bigcap_{i \in \omega} 0^i 1 U_i$  is  $\Pi_{n+1}^0$ -hard;
2. if  $V_i$  is  $\Pi_n^0$ -hard for  $i < \omega$ ,  $\bigcup_{i \in \omega} 0^i 1 V_i$  is  $\Sigma_{n+1}^0$ -hard.

### 3.2 Effective Characterization

Since any Borel class is closed under finite unions and finite intersections, we have:

**Proposition 2.** Let  $K$  be a complete set for some class from  $\bigcup_{1 \leq i < \omega} \{\Sigma_i^0, \Pi_i^0\}$ .  
 For every  $k$ , if  $U_i \leq_w K$  for  $0 \leq i \leq k$ , then

$$(1) \bigcup_{i=0}^k 0^i 1 U_i \leq_w K, \quad (2) \bigcap_{i=0}^k 0^i 1 U_i \leq_w K,$$

and if  $V_i <_w K$  for  $0 \leq i \leq k$ , then

$$(3) \bigcup_{i=0}^k 0^i 1 V_i <_w K, \quad (4) \bigcap_{i=0}^k 0^i 1 V_i <_w K.$$

Analogously, since  $\Sigma_n^0$  is closed under countable unions, and  $\Pi_n^0$  is closed under countable intersections, we obtain the following result.

**Proposition 3.**

1. Let  $K$  be a  $\Sigma_n^0$ -complete set. If for every  $i \in \omega$  it holds that  $U_i \leq_w K$ , then  $\bigcup_{i \in \omega} 0^i 1 U_i \leq_w K$ .
2. Let  $K$  be a  $\Pi_n^0$ -complete set. If for every  $i \in \omega$  it holds that  $U_i \leq_w K$ , then  $\bigcap_{i \in \omega} 0^i 1 U_i \leq_w K$ .

We now apply these properties to characterize the topological power of looping nodes in an LGA.

**Lemma 4.** Let  $A, B_0, B_1, C$  be LGA such that  $C = q(A, B_0, B_1)$ , and  $q$  is a restrained looping node. For  $n \geq 2$

1. if  $L(A), L(B_i) <_w L(K_n^\Sigma)$ , then  $L(C) <_w L(K_n^\Sigma)$ ;
2. if  $L(A), L(B_i) <_w L(K_n^\Pi)$ , then  $L(C) <_w L(K_n^\Pi)$ .

*Proof.* It is enough to prove the first claim, the second follows by duality. Suppose that  $q = \langle + \rangle$ . Let us describe a winning strategy for Player II in  $\mathcal{W}(L(C), L(K_n^\Sigma))$ . If Player I plays  $a$  on the leftmost branch, Player II plays accepting in the subtrees rooted in nodes  $0^i 1$ . If Player I finally plays a  $b$  in the  $k$ th round, Player II switches to playing rejecting in every subtree rooted in  $0^i 1$  for  $i < k$ , and in the subtree rooted in  $0^k$  applies the winning strategy given by Proposition 2(1). Hence,  $L(C) \leq_w L(K_n^\Sigma)$ .

To obtain strictness of the inequality, we describe a winning strategy for Player I in  $\mathcal{W}(L(K_n^\Sigma), L(C))$ . As long as Player II skips or plays  $a$  on the leftmost branch, Player I plays rejecting in the subtrees rooted in  $0^i 1$ . If in the  $k$ th round Player II finally plays

$b$  on the leftmost branch, Player I continues playing rejecting in every subtree rooted in  $0^i 1$  for  $i \leq n$ , and in the subtree rooted in  $0^{n+1}$  applies the winning strategy given by Proposition 2(3).

For  $q = [-]$  the proof is analogous, only uses Proposition 2(2) and (4).  $\square$

**Lemma 5.** *Let  $A, B_0, B_1, C$  be LGA such that  $C = q(A, B_0, B_1)$ , and  $q$  is an unrestrained looping node. Let  $n \geq 2$ . If  $q = \langle -, \rangle$ , then*

1. *if  $L(A) \leq_w L(K_{n-1}^\Sigma)$ , and  $L(B_i) <_w L(K_n^\Sigma)$ , then  $L(C) <_w L(K_n^\Sigma)$ ;*
2. *if  $L(A) \geq_w L(K_{n-1}^\Pi)$ , then  $L(C) \geq_w L(K_n^\Sigma)$ ;*

and if  $q = [+]$ , then

3. *if  $L(A) \leq_w L(K_{n-1}^\Pi)$ , and  $L(B_i) < L(K_n^\Pi)$ , then  $L(C) <_w L(K_n^\Pi)$ ;*
4. *if  $L(A) \geq_w L(K_{n-1}^\Sigma)$ , then  $L(C) \geq_w L(K_n^\Pi)$ .*

*Proof.* Use an argument similar to the proof of Lemma 4 to infer (1) and (3) from Proposition 3, and (2) and (4) from Lemma 3.  $\square$

The main theorem of this part follows from Lemma 4 and Lemma 5 by induction on the structure of the automaton. And as a corollary, we obtain the first decidability result.

**Theorem 1.** *For every  $n$  and every LGA  $A$*

1.  *$L(A)$  is  $\Sigma_n^0$ -hard iff  $A$  contains a  $(1, n+1)$ -pattern;*
2.  *$L(A)$  is  $\Pi_n^0$ -hard iff  $A$  contains a  $(0, n)$ -pattern.*

**Corollary 1.** *The problem of calculating the exact position in the Borel hierarchy of a language recognized by a linear game tree automaton is decidable (in polynomial time if the productive states are given).*

## 4 The Weak Index Hierarchy

### 4.1 Introducing the Hierarchy

The (Mostowski–Rabin) index of an automaton  $A$  is given by  $(i, j) \in \omega \times \omega$ , where  $i$  is the minimal and  $j$  is the maximal value of the priority function rank. Scaling down the priorities if necessary, we can assume that  $i \in \{0, 1\}$  and that for every  $n \in \{i, i+1, \dots, j\}$ , there is a state  $q$  such that  $\text{rank}(q) = n$ . Thus, the indices are elements of  $(\{0, 1\} \times \omega) \setminus (1, 0)$ . Given an index  $(0, j)$  (resp.  $(1, j)$ ), its dual index is  $(1, j+1)$  (resp.  $(0, j-1)$ ).

Consider the partial order on indices of automata given by

$$(i, j) \sqsubseteq (i', j') \quad \text{iff} \quad j - i < j' - i'.$$

Note that this implies that dual indices are incomparable. The hierarchy induced by the partial order  $\sqsubseteq$  is called the *hierarchy of Mostowski–Rabin indices* (or simply the index hierarchy) of the considered class of automata.

For a given class, the hierarchy is said to be strict if there is an automaton at each level that cannot be simulated by any automaton from the same class of lower level. By a result of Bradfield [34], we know that the index hierarchy of alternating tree automata, and therefore the fixpoint hierarchy of the modal  $\mu$ -calculus, is strict. Arnold's proof of

the same result [1] can be adapted to show that the index hierarchy of weak alternating tree automata is also strict. In the latter case we speak of the *weak* index hierarchy.

### 4.2 The Conjecture

In [16] it was conjectured that for weakly recognizable tree languages the weak index hierarchy and the Borel hierarchy coincide, i.e., that a weakly recognizable tree language is in  $\Sigma_n^0$  (resp.  $\Pi_n^0$ ) iff it can be recognized by a weak alternating automaton of index  $(1, n + 1)$  (resp.  $(0, n)$ ). It has long been known that one implication holds.

**Proposition 4 ([13]).** *For every weak alternating automaton with index  $(0, n)$  (resp.  $(1, n + 1)$ ), it holds that  $L(A) \in \Pi_n^0$  (resp.  $L(A) \in \Sigma_n^0$ ).*

It was also proved recently that the conjecture holds when restricted to languages which are in addition deterministically recognizable [16]. We refine this result by showing that the conjecture also holds for languages recognizable by LGA.

### 4.3 Weak Index of LGA-Recognizable Sets

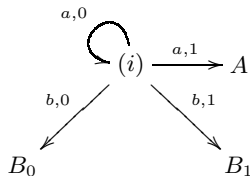
**Theorem 2.** *For languages recognizable by LGA, the Borel hierarchy and the weak index hierarchy coincide.*

*Proof.* For simplicity we assume that all automata are in the normal form. Extending the proof to the general case is easy.

By duality it is enough to consider  $\Pi_n^0$  classes. By Proposition 4 it suffices to show that for each LGA  $C$  with  $L(C) \in \Pi_n^0$  there exists an equivalent weak alternating automaton of index  $(0, n)$ . We proceed by induction on the structure of the automaton.

The case  $n = 0$  is trivial. Suppose that  $n = 1$ . By Theorem 1  $C$  does not contain an accepting loop reachable from a rejecting loop. It is enough to set the rank of all states reachable from odd looping states to 1 and the rank of the remaining states to 0 to obtain an equivalent automaton of index  $(0, 1)$ .

Suppose that  $n \geq 2$ . If the initial state of  $C$  is not looping, the claim follows easily from the induction hypothesis. Suppose that  $q_0$  is a looping node, and  $C$  is of the form



We can treat  $C$  as a weak alternating automaton and transform it into an equivalent one of index  $(0, n)$ . Clearly, it must hold that  $L(A), L(B) \in \Pi_n^0$  and by induction hypothesis we may assume that  $A, B_0, B_1$  have index  $(0, n)$ . If  $i = 0$ , the claim follows trivially. For  $(i) = [1]$ , the equivalent weak automaton of index  $(0, n)$  is shown in Fig. 1(a). To prove the equivalence, observe that the left-hand component checks that finally  $b$  occurs on the leftmost branch, and the right-hand component checks the condition  $A$  until the first  $b$  occurs, and after that checks the conditions  $B_0$  and  $B_1$ .



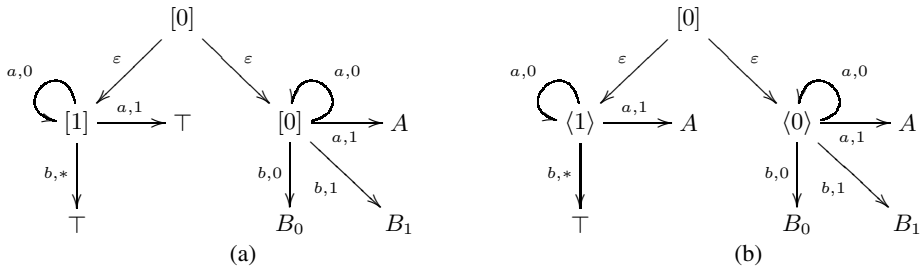


Fig. 1. The equivalent weak automata

Finally, suppose that  $(i) = \langle 1 \rangle$ . By Theorem 1,  $C$  contains  $(1, n + 1)$ -pattern, which implies that  $A$  contains no  $(0, n - 1)$ -pattern. By induction hypothesis we may assume that  $A$  has index  $(1, n)$ . Recall that  $B_0$  and  $B_1$  have index  $(0, n)$ . The corresponding equivalent weak alternating automaton is shown in Fig. 1(b). The left-hand component takes care of the situation, when  $b$  never occurs on the leftmost path. If  $b$  does occur, this component is trivially accepting, but the right-hand component provides the appropriate semantics.  $\square$

Combining Theorem 1 and Theorem 2 we obtain the second decidability result.

**Corollary 2.** *The problem of calculating the exact position in the weak index hierarchy of a language recognized by a LGA is decidable (in polynomial time if the productive states are given).*

## 5 The Wadge Hierarchy

### 5.1 The Difference Hierarchy

For a Borel class  $\Sigma_n^0$ , the finite Hausdorff-Kuratowski, or difference, hierarchy is defined as  $\text{Diff}_1(\Sigma_n) = \Sigma_n$  and  $\text{Diff}_k(\Sigma_n) = \{U \setminus V : U \in \Sigma_n, V \in \text{Diff}_{k-1}(\Sigma_n)\}$ . Let  $\overline{\text{Diff}_k(\Sigma_n)}$  denote the dual class. Recall that this is not the same as  $\text{Diff}_k(\Pi_n)$ . Indeed,  $\text{Diff}_{2k+1}(\Pi_n) = \overline{\text{Diff}_{2k+1}(\Sigma_n)}$  and  $\text{Diff}_{2k}(\Pi_n) = \text{Diff}_{2k}(\Sigma_n)$ . We have

$$\begin{aligned} \text{Diff}_{2k}(\Sigma_n) &= \{U_1 \cap V_1^c \cup \dots \cup U_k \cap V_k^c\}, \\ \text{Diff}_{2k+1}(\Sigma_n) &= \{U_1 \cap V_1^c \cup \dots \cup U_k \cap V_k^c \cup U\}, \\ \overline{\text{Diff}_{2k}(\Sigma_n)} &= \{U_1 \cap V_1^c \cup \dots \cup U_{k-1} \cap V_{k-1}^c \cup U \cup V^c\}, \\ \overline{\text{Diff}_{2k+1}(\Sigma_n)} &= \{U_1 \cap V_1^c \cup \dots \cup U_k \cap V_k^c \cup V^c\}, \end{aligned}$$

where the sets  $U, V, U_i, V_i$  range over  $\Sigma_n$ . From this characterization one easily obtains the following table of the operation  $\diamond$ . For  $n > 0$  let  $S_n(k)$  be a  $\text{Diff}_k(\Sigma_n)$ -complete set, and let  $P_n(k)$  be a  $\overline{\text{Diff}_k(\Sigma_n)}$ -complete set.

**Lemma 6.** For each  $n > 0, i > 0, j \geq 0$

- $S_n(2i) \diamond S_n(2j) \equiv S_n(2i+2j), S_n(2i) \diamond P_n(2j) \equiv P_n(2i+2j)$   
 $P_n(2i) \diamond S_n(2j) \equiv S_n(2i+2j), P_n(2i) \diamond P_n(2j) \equiv P_n(2i+2j-2)$
- $S_n(2i+1) \diamond S_n(2j) \equiv S_n(2i+2j+1), S_n(2i+1) \diamond P_n(2j) \equiv P_n(2i+2j)$   
 $P_n(2i+1) \diamond S_n(2j) \equiv P_n(2i+2j+1), P_n(2i+1) \diamond P_n(2j) \equiv P_n(2i+2j)$
- $S_n(2i+1) \diamond S_n(2j+1) \equiv S_n(2i+2j+1), S_n(2i+1) \diamond P_n(2j+1) \equiv P_n(2i+2j+2)$   
 $P_n(2i+1) \diamond S_n(2j+1) \equiv P_n(2i+2j+2), P_n(2i+1) \diamond P_n(2j+1) \equiv P_n(2i+2j+1).$

The equivalences above, together with closure by  $\diamond$ , immediately provide complete LGA-recognizable languages for  $\text{Diff}_k(\Sigma_n)$  for each  $k, n$ . Building upon this we produce the whole Wadge hierarchy of LGA-recognizable languages.

### 5.2 Bestiarum Vocabulum

For an ordinal  $\alpha$  let  $\exp(\alpha) = \omega_1^\alpha$ . Hence,

$$\exp^{k+1}(\alpha) = \exp(\exp^k(\alpha)) = \underbrace{\omega_1^{\omega_1^{\dots^{\omega_1^\alpha}}}}_{k+1 \text{ times } \omega_1}.$$

Before describing the hierarchy, recall the Wadge degrees of  $\text{Diff}_k(\Sigma_n)$ -complete sets.

**Proposition 5 ([5]).** For each  $k > 0, d_w(S_n(k)) = d_w(P_n(k)) = \exp^n(k)$ .

**Theorem 3.** The family of LGA-recognizable languages contains  $L$  with  $d_w(L) = \beta$  for every  $\beta = \sum_{i=n}^0 \beta_i$ , where each  $\beta_i$  is of the form

$$\beta_i = \exp^i(\omega)\eta + \sum_{p=j}^1 \exp^i(p)k_p$$

with  $\eta < \omega^\omega, k_{2q} \in \{0, 1\}$ , and  $j, k_{2q+1} < \omega$ .

*Proof.* By induction on such ordinals, we provide an automaton  $A_\beta$ , such that  $L(A_\beta)$  is non self dual, and  $d_w(L(A_\beta)) = \beta$ . To make the notation more readable, we use bracketed ordinal  $[\beta]$  to denote the automaton  $A_\beta$ . Since LGA are closed under complementation, when we construct an automaton recognizing a non self dual set of degree  $\beta$ , we also immediately get the automaton  $[\beta]^c$ . We write  $[\beta]^\pm$  for  $[\beta] \vee [\beta]^c$ .

Let us start with the basic building bricks of our construction: the automata  $[1], [\omega^m], [\exp^i(1)]$ , and  $[\exp^i(\omega)\omega^p]$ . Together with these automata we show how to make a step with those ordinals, i.e., how to define the automaton for  $[\alpha + \gamma]$ , once we already have the automaton  $[\alpha]$  and  $\gamma$  is one of the above. Let

$$[1] = \perp, \quad [\alpha + 1] = \langle + \rangle(\perp, [\alpha]^\pm).$$

Note that  $[2] = K_1^\Sigma$ , and  $[2]^c = K_1^H$ . For  $m > 1$  let

$$\begin{aligned} [\omega] &= \langle + \rangle([\!|3], \perp), & [\alpha + \omega] &= \langle + \rangle([\!|3], [\alpha]^\pm), \\ [\omega^m] &= \langle + \rangle([\omega^{m-1} + 1], \perp), & [\alpha + \omega^m] &= \langle + \rangle([\omega^{m-1} + 1], [\alpha]^\pm). \end{aligned}$$

For  $i > 1$  let

$$\begin{aligned} [\exp(1)] &= \langle - \rangle([2]^{\mathbb{G}}, \perp), & [\alpha + \exp(1)] &= \langle - \rangle([2]^{\mathbb{G}}, [\alpha]^{\pm}), \\ [\exp^i(1)] &= \langle - \rangle([\exp^{i-1}(1)]^{\mathbb{G}}, \perp), & [\alpha + \exp^i(1)] &= \langle - \rangle([\exp^{i-1}(1)]^{\mathbb{G}}, [\alpha]^{\pm}). \end{aligned}$$

Note that  $[\exp^i(1)] = K_{i+1}^{\Sigma}$ ,  $[\exp^i(1)]^{\mathbb{G}} \equiv K_{i+1}^{\Pi}$ . For  $p > 0$  let

$$\begin{aligned} [\exp^i(\omega)] &= \langle + \rangle([\exp^i(2)], \perp), \\ [\alpha + \exp^i(\omega)] &= \langle + \rangle([\exp^i(2)], [\alpha]^{\pm}), \\ [\exp^i(\omega)\omega^p] &= \langle + \rangle([\exp^i(\omega)\omega^{p-1} + 1], \perp), \\ [\alpha + \exp^i(\omega)\omega^p] &= \langle + \rangle([\exp^i(\omega)\omega^{p-1} + 1], [\alpha]^{\pm}). \end{aligned}$$

Using the basic building blocks and basic steps defined above we can inductively define automata  $[\sum_{i=n}^1 \gamma_i]$ , such that each  $\delta_i$  is of the form  $\exp^i(\omega)\eta + \exp^i(1)p$  with  $\eta < \omega^\omega$  and  $p < \omega$ .

To define automata for all  $\beta$  described in the statement of the theorem, we need one more kind of bricks and two more kinds of steps. For  $\eta < \omega^\omega$ ,  $1 \leq i < \omega$ , we have:

$$\begin{aligned} [\exp^i(2)] &= [\exp^i(1)] \diamond [\exp^i(1)]^{\mathbb{G}} \\ [\alpha + \exp^i(\omega)\eta + \sum_{p=m}^1 \exp^i(p+2)k_p] &= [\alpha + \exp^i(\omega)\eta + \sum_{p=m}^1 \exp^i(p)k_p] \diamond [\exp^i(2)] \\ [\alpha + \exp^i(\omega)\eta + \sum_{p=m}^1 \exp^i(p+2)k_p + \exp^i(2)] &= [\alpha + \exp^i(\omega)\eta + \sum_{p=m}^{\ell} \exp^i(p)k_p + 1] \diamond [\exp^i(2)]. \end{aligned}$$

Using Lemma 6 and standard Wadge game arguments one can prove that for every ordinal  $\alpha$  from the statement of the theorem,  $[\alpha]$  has Wadge degree  $\alpha$ . □

As a corollary we obtain a lower bound on the height of the hierarchy.

**Corollary 3.** *The LGA hierarchy has height at least  $(\omega^\omega)^\omega = \omega^{\omega^2}$ .*

In the remaining of the paper we prove that the height of the LGA hierarchy is exactly  $(\omega^\omega)^\omega$  and that we can compute the Wadge degree of a language LGA-recognizable.

### 5.3 Two Simple Operations on Sets of Trees

Let us define two more operations on sets of trees. Let  $L, M \subseteq T_\Sigma$ ,  $a, b \in \Sigma$ . We define the set  $L \rightarrow M$  as the set of trees  $t \in T_\Sigma$ , satisfying any of the following conditions:

- $t.1 \in L$  and  $a = t(0^n)$  for all  $n$ ,
- $00^n$  is the first node on the branch  $00^*$  such that  $a \neq t(00^n)$  and  $t.00^{n-1} \in M$ .

A player in charge of  $L \rightarrow M$  is like a player in charge of  $L$  endowed with an extra move, which can be used only once, that erases everything played before. Then he can restart the play being in charge of  $M$ .

The second operation is a generalization of  $\vee$ . Let  $L_n \subseteq T_\Sigma$  for  $n < \omega$ . Define  $\sup_{n < \omega}^- L_n$  as the set of trees  $t \in T_\Sigma$  satisfying the following conditions for some  $k$ :

- $0^k$  is the first node on  $0^*$  labeled with  $b$ ,
- $t.0^k1 \in L_k$ .

Intuitively, a player in charge of  $\sup_{n < \omega}^- L_n$  is given the choice between the  $L_n$ 's. The decision is determined by the number of  $a$ 's played on the leftmost branch of the tree before the first  $b$ . If the player keeps playing  $a$ 's forever on the leftmost branch, the tree will be rejected.

Define also  $\sup_n^+ L_n$  as  $\sup_n^- L_n \cup \{t : \forall_n t(0^n) = a\}$ . The difference from the previous operation is that now, when the player plays  $a$ 's forever on the leftmost branch, the obtained tree is accepted. Note that the operations are dual:

$$\left(\sup_n^+ L_n\right)^c = \sup_n^- (L_n^c)$$

### 5.4 Computing Wadge Degrees

Let  $\Omega$  denote the set of Wadge equivalence classes of languages recognized by the automata  $[\beta], [\beta]^c, [\beta]^\pm$  defined in the proof of Theorem 3. Slightly abusing the notation we write  $[\beta]^-$  for the Wadge equivalence class of  $L([\beta])$ ,  $[\beta]^+$  for the class of  $L([\beta]^c)$ , and  $[\beta]^\pm$  for the class of  $L([\beta]^\pm)$ .

The technical difficulty of the decidability result lies in the following effective closure property (its proof can be found in the appendix).

**Theorem 4.** *For each  $U, V \in \Omega$  it holds that  $U \diamond V, U \vee V$ , and  $\sup_k^+ U^{(k)} \diamond V$  belong to  $\Omega$  and can be effectively computed. The same holds for  $U \rightarrow V$ , if  $U = [\exp^i(1)]^\mu$  for some  $i < \omega$  and  $\mu \in \{+, -\}$ .*

**Theorem 5.** *For each LGA we can calculate effectively the signed degree of the recognized language.*

*Proof.* We proceed by induction on the number of states. Let  $C$  be an LGA. If  $C$  has only one state, it is either totally accepting or totally rejecting. In the first case the signed degree is  $[1]^+$ , in the second case it is  $[1]^-$ . Suppose that  $C$  has more states. By duality we may assume that the initial state  $q_0$  is existential: if it is universal, compute the signed degree for the complement of  $C$ , and return the degree negated. Suppose that  $q_0$  is not looping. By linearity,  $C$  can be represented as in Fig. 2(a) for some automata  $A_0, A_1, B_0, B_1$ , each having less states than  $C$ . Clearly  $L(C) \equiv L(A_0) \diamond L(A_1) \vee L(B_0) \diamond L(B_1)$ . Hence, we can use the induction hypothesis to get the degrees of  $L(C_{q_i})$ , and then Theorem 4 to compute  $d(C) = d(C_{q_1}) \diamond d(C_{q_2}) \vee d(C_{q_3}) \diamond d(C_{q_4})$ .

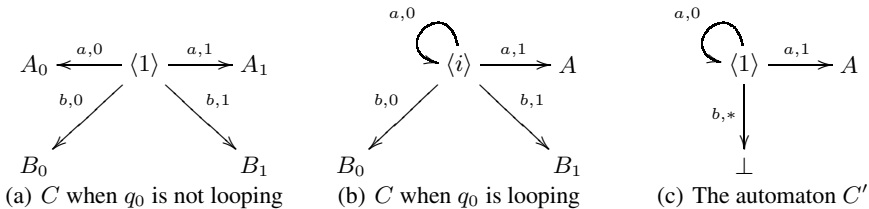


Fig. 2. The automata  $C$  and  $C'$

If  $q_0$  is looping, we can assume w.l.o.g. that  $C$  is of the form shown in Fig. 2(b) with  $i = 0, 1$ . If  $i = 1$ , there exists  $n \in \omega$  such that  $L(A)$  is either  $\Sigma_n^0$ -complete, or in  $\Delta_{n+1}^0 \setminus \Sigma_n^0$ . If  $L(A)$  is  $\Sigma_n$ -complete, by Lemma 5 the language recognized by  $C'$ , defined in Fig. 2(c) is also  $\Sigma_n^0$ -complete. Since  $d(A) = d(C') = [\exp^n(1)]^-$  and  $([\exp^n(1)]^-)^{\langle k \rangle} = [\exp^n(1)]^-$  for each  $k > 0$ , we have  $d(C) = [\exp^n(1)]^- \rightarrow d(B_1) \diamond (B_2) \diamond [\exp^n(1)]^-$ . On the other hand, if  $L(A) \in \Delta_{n+1}^0 \setminus \Sigma_n^0$ , by Theorem 1 and Theorem 2 the language recognized by  $C'$  is  $\Sigma_{n+1}^0$ -complete, and it is easy to see that  $d(C) = [\exp^{n+1}(1)]^- \rightarrow d(B_1) \diamond d(B_2)$ . We conclude by the inductive hypothesis and Theorem 4

If  $i = 0$ , it is straightforward to check that  $d(C) = \sup_k^+ d(A)^{\langle k \rangle} \diamond d(B)$ , and again the claim follows from Theorem 4 and the induction hypothesis.  $\square$

## 6 Conclusion

Alternating tree automata are notorious for the lack of decision procedures for classical hierarchies like the Mostowski-Rabin hierarchy, the Borel hierarchy, or the Wadge hierarchy. The reason for this is that when we move from infinite words to infinite trees, deterministic and non-deterministic modes of computation highly diverge.

We have proposed a novel class of automata capturing an interesting aspect of alternation, and for this class we have proved that all corresponding hierarchies mentioned above are decidable. Moreover we have shown that the weak index and the Borel rank coincide over LGA-recognizable languages.

We have seen that, despite their apparent simplicity, LGA yield a class of languages surprisingly complex from the topological point of view: the height of their Wadge hierarchy is  $(\omega^\omega)^\omega$ . Admittedly, this is much less than the height of the hierarchy for weak alternating automata, which is known to be at least  $\varepsilon_0$  [7], but this was to be expected, as LGA form a very restricted subclass of weak alternating automata. What is surprising however, is that the height of the Wadge hierarchy for LGA is much larger than that for deterministic automata, which was shown in [15] to be  $(\omega^\omega)^3 + 3$ , and the same as for deterministic *push-down* automata on infinite words [6].

**Acknowledgments.** We thank David Janin for initiating this research by showing the interest in the topological complexity of weak alternating automata, Sławek Lasota for bringing linear automata into our attention, Igor Walukiewicz for helpful comments and inspiring discussions, Claire David for space-saving tricks, and the anonymous referees for their suggestions.

## References

1. Arnold, A.: The  $\mu$ -Calculus Alternation-Depth Hierarchy is Strict on Binary Trees. ITA 33(4/5), 329–340 (1999)
2. Arnold, A., Niwiński, D.: Continuous Separation of Game Languages. Fundamenta Informaticae 81(1–3), 19–28 (2008)
3. Bradfield, J.: The Modal  $\mu$ -Calculus Alternation Hierarchy is Strict. Theor. Comput. Sci. 195(2), 133–153 (1998)

4. Bradfield, J.: Simplifying the Modal  $\mu$ -Calculus Alternation Hierarchy. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 39–49. Springer, Heidelberg (1998)
5. Duparc, J.: Wadge Hierarchy and Veblen Hierarchy Part I: Borel Sets of Finite Rank. *J. Symb. Log.* 66(1), 56–86 (2001)
6. Duparc, J.: A Hierarchy of Deterministic Context-Free  $\omega$ -Languages. *Theoret. Comput. Sci.* 290, 1253–1300 (2003)
7. Duparc, J., Murlak, F.: On the Topological Complexity of Weakly Recognizable Tree Languages. In: Csehaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 261–273. Springer, Heidelberg (2007)
8. Finkel, O.: Borel Ranks and Wadge Degrees of  $\omega$ -Context Free Languages. *Mathematical Structures in Computer Science* 16, 813–840 (2006)
9. Hummel, S., Michalewski, H., Niwiński, D.: On the Borel Inseparability of Game Tree Languages. In: Proc. of STACS 2009, pp. 565–576 (2009)
10. Kupferman, O., Safra, S., Vardi, M.: Relating Word and Tree Automata. In: LICS 1996, pp. 322–332 (1996)
11. Kupferman, O., Vardi, M., Wolper, P.: An Automata-Theoretic Approach to Branching-Time Model Checking. *Journal of the ACM* 47(2), 142–155 (1994)
12. Landweber, L.H.: Decision Problems for  $\omega$ -Automata. *Math. Systems Theory* 3, 376–384 (1969)
13. Mostowski, A.W.: Hierarchies of Weak Automata and Weak Monadic Formulas. *Theoret. Comput. Sci.* 83, 323–335 (1991)
14. Murlak, F.: On Deciding Topological Classes of Deterministic Tree Languages. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 428–441. Springer, Heidelberg (2005)
15. Murlak, F.: The Wadge Hierarchy of Deterministic Tree Languages. *Logical Methods in Comput. Sci.* 4(4), Paper 15
16. Murlak, F.: Weak Index vs Borel Rank. In: Proc. STACS 2008, pp. 573–584 (2008)
17. Niwiński, D.: On Fixed Point Clones. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 464–473. Springer, Heidelberg (1986)
18. Niwiński, D., Walukiewicz, I.: Relating Hierarchies of Word and Tree Automata. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 320–331. Springer, Heidelberg (1998)
19. Niwiński, D., Walukiewicz, I.: A Gap Property of Deterministic Tree Languages. *Theor. Comput. Sci.* 303, 215–231 (2003)
20. Niwiński, D., Walukiewicz, I.: Deciding Nondeterministic Hierarchy of Deterministic Tree Automata. *Electr. Notes Theor. Comput. Sci.* 123, 195–208 (2005)
21. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Soc.* 141, 1–35 (1969)
22. Selivanov, V.: Wadge Degrees of  $\omega$ -Languages of Deterministic Turing Machines. *Theoret. Informatics Appl.* 37, 67–83 (2003)
23. Skurczyński, J.: The Borel Hierarchy is Infinite in the Class of Regular Sets of Trees. *Theoret. Comput. Sci.* 112, 413–418 (1993)
24. Wadge, W.W.: Reducibility and Determinateness on the Baire Space. Ph.D. Thesis, Berkeley (1984)
25. Wagner, K.: Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen. *J. Inf. Process. Cybern.* EIK 13, 473–487 (1977)
26. Wagner, K.: On  $\omega$ -Regular Sets. *Inform. and Control* 43, 123–177 (1979)

# Enriching an Effect Calculus with Linear Types

Jeff Egger<sup>1,\*</sup>, Rasmus Ejlers Møgelberg<sup>2,\*\*</sup>, and Alex Simpson<sup>1,\*</sup>

<sup>1</sup> LFCS, School of Informatics, University of Edinburgh, Scotland, UK

<sup>2</sup> IT University of Copenhagen, Copenhagen, Denmark

**Abstract.** We define an *enriched effect calculus* by extending a type theory for computational effects with primitives from linear logic. The new calculus provides a formalism for expressing linear aspects of computational effects; for example, the linear usage of imperative features such as state and/or continuations.

Our main syntactic result is the conservativity of the enriched effect calculus over a basic *effect calculus* without linear primitives (closely related to Moggi's *computational metalanguage*, Filinski's *effect PCF* and Levy's *call-by-push-value*). The proof of this syntactic theorem makes essential use of a category-theoretic semantics, whose study forms the second half of the paper.

Our semantic results include soundness, completeness, the initiality of a syntactic model, and an embedding theorem: every model of the basic effect calculus fully embeds in a model of the enriched calculus. The latter means that our enriched effect calculus is applicable to arbitrary computational effects, answering in the positive a question of Benton and Wadler (LICS 1996).

## 1 Introduction

The *computational metalanguage* was proposed by Moggi [14] as a general metalanguage for ascribing semantics to programming languages with effects, building on his own idea that computational effects can be encapsulated by the mathematical structure of a *strong monad* [13]. The metalanguage extends the simply-typed  $\lambda$ -calculus with a new type constructor  $T$ , where  $TA$  represents a type of all computations that produce values of type  $A$ . Semantically,  $T$  is interpreted as a strong monad that captures the effects that computations may exhibit.

In [4], Benton and Wadler identify a close connection between Moggi's computational metalanguage and Girard's intuitionistic linear logic (ILL) [7]. They show that every model of ILL can be reconstrued as a model of the computational metalanguage, and this determines an interpretation of the computational metalanguage within ILL. However, the models of the computational metalanguage that arise from models of ILL are very special ones: their monads are *commutative*. This means that the interpretation of the computational metalanguage in

---

\* Research supported by an EPSRC research grant EP/F042043/1.

\*\* Research supported by the Danish Agency for Science, Technology and Innovation.

ILL validates an equation that is not always true for effects: the equation that asserts the insensitivity of computational effects to their order of execution.

Many computational effects are *not* commutative, for example: exceptions, state, input/output, and continuations. In [4, §8], Benton and Wadler write:

*“We do not know if it is possible to define a non-commutative linear calculus which corresponds to a wider class of monad models.”*

In this paper, we present such a calculus, the *enriched effect calculus*.

The enriched effect calculus can be viewed as an extension of Moggi’s metalanguage with a judicious selection of type constructors from linear logic. We envisage that this calculus will be applicable to computational scenarios in which the manipulation of computational effects adheres to a discipline of linearity. For example, the usual *state monad*  $S \rightarrow ((-) \times S)$  (where  $S$  is an object of *states*) has a linear counterpart  $S \multimap (!(-) \otimes S)$ , which accounts for the fact that state (unlike values) can neither be duplicated nor discarded, cf. [15]. Similarly, the *continuations monad*  $((-) \rightarrow R) \rightarrow R$  (where  $R$  is a *result type*) has a linear version  $((-) \rightarrow R) \multimap R$ , which enforces the *linear usage* of continuations, a discipline that is ubiquitous amongst structured forms of control [3].

While examples such as the above can be formulated in ILL itself, we believe our enriched effect calculus to be the natural home for them. Indeed, the enriched effect calculus has two main advantages over ILL. First, it is weaker than ILL, and hence applicable more widely (models of ILL are a strict subset of models of the enriched effect calculus). Second, the tight connection between the enriched effect calculus and Moggi’s computational metalanguage, which we establish in this paper, makes the former a natural vehicle for formalising the general phenomenon of interactions between linearity and effects, including the *linear usage of effects* [9]. Detailed applications of the enriched effect calculus to such examples will appear elsewhere, starting with a study of linearly-used continuations in a follow-up paper [5].

In this paper, our enriched effect calculus is defined as an extension of a basic *effect calculus*, which is presented in Section 2. The basic calculus is a simple (and essentially standard) extension of Moggi’s computational metalanguage with a notion of *computation type*, as used in Filinski’s *effect PCF* [6] and in Levy’s *call-by-push-value* [11]. The *enriched effect calculus* is presented in Section 3. There we state our main syntactic theorem: the enriched calculus conservatively extends the basic calculus. Thus the addition of the new linear logic connectives does not interfere with the existing type constructors of the basic calculus.

The second half of the paper, starting with Section 4, is devoted to category-theoretic models. As models of the basic effect calculus we take (an appropriate version of) Levy’s *adjunction models* [12], which are the natural models for calculi based on computation types. As models of the enriched effect calculus, we take adjunction models with the extra structure needed to model the linear connectives. This structure is formulated in terms of *enriched category theory* [10]. Soundness and completeness are addressed in Section 4. In Section 5, morphisms of models are defined and an initiality property is formulated for a syntactic model. Finally, in Section 6, we show that every adjunction model fully embeds



in a model of the enriched effect calculus. It is this theorem that answers Benton and Wadler’s (implicit) question quoted above: the enriched effect calculus is compatible with *any* monad model (any such can be presented as a Levy adjunction model). Furthermore, the embedding theorem is also used to prove the syntactic conservativity of the enriched effect calculus over the basic calculus.

## 2 A Basic Effect Calculus

Moggi’s *computational metalanguage*, [14], extends the simply-typed  $\lambda$ -calculus with new types  $TA$ , which type computations (possibly with effects) that produce values of type  $A$ . The new type has an associated “let” operator, which performs the *Kleisli extension* of a map  $A \rightarrow TB$  to a map  $TA \rightarrow TB$ . This can be seen as a restricted form of elimination rule for the type  $TA$ . Filinski [6] generalises this elimination rule to apply to a wider class of “target” types than those of the form  $TB$ , and develops a calculus for this based on classifying such types as special *computation types* within a broader class of *value types*. Such a generalisation is useful for interpreting call-by-name languages. Its importance has been thoroughly established by Levy, whose *call-by-push-value (CBPV)* paradigm [11] is based on the distinction between computation and value types.

Guided by the above, we define our *effect calculus* as an extension of the simply-typed  $\lambda$ -calculus with a new type constructor, following Moggi, and with a division of types into value types and computation types, following Filinski and Levy. Because we have two classes of types, we assume two classes of type constants. We use  $\alpha, \beta, \dots$  to range over a set of *value type constants*, and  $\underline{\alpha}, \underline{\beta}, \dots$  to range over a disjoint set of *computation type constants*. We then use  $A, B, \dots$  to range over *value types*, and  $\underline{A}, \underline{B}, \dots$  to range over *computation types*, which are specified by the grammar below:

$$\begin{aligned} A &::= \alpha \mid \underline{\alpha} \mid 1 \mid A \times B \mid A \rightarrow B \mid !A \\ \underline{A} &::= \underline{\alpha} \mid 1 \mid \underline{A} \times \underline{B} \mid A \rightarrow \underline{B} \mid !A \end{aligned}$$

By this definition, every computation type is also a value type.

Our notation for type constructors is standard, except that we use the linear exponential notation  $!A$  for Moggi’s monadic type  $TA$ . (The reasons for this non-standard choice will transpire later.) We follow Filinski in making computation types a subclass of value types. Levy, in contrast, keeps computation types and value types separate. He has an operator  $F$  that turns a value type  $A$  into a computation type  $FA$ , and conversely an operator  $U$  that maps a computation type  $\underline{A}$  to a value type  $U\underline{A}$ . Levy’s type  $FA$  corresponds to  $!A$  in our syntax, and his type  $U\underline{A}$  is simply  $\underline{A}$  itself. Three reasons for our choice of omitting  $U$  and subsuming computation types as value types are: the streamlined syntax leads to a very economical type system (see below); there is no loss of information, since one can establish an equivalence between the two systems; and also the term syntax is not cluttered with (inferable) conversions between values and computations. However, for the purposes of the present paper, the main benefit

of our formulation is that our syntax provides a transparent foundation for the extension with linear logic connectives in Section 3.

We mention two other differences from CBPV, as presented in [11]. First, because we are building on the simply-typed  $\lambda$ -calculus, we include a full function space between value types, whereas Levy only includes function spaces as computation types, and, as in the present paper, these are required to have computation type codomains. Second, Levy includes sums of value types as a basic construct, whereas, largely for space reasons, we have omitted them. Since either of these differences could be easily circumvented by making evident minor alterations to our type system (or to CBPV), we believe it is correct to view our effect calculus as essentially CBPV, albeit with a different syntax.

To ease the transition to the linear calculus, we shall build a notion of linearity directly into the typing judgements of the basic effect calculus. This notion has an intuitive motivation. Following Levy [11], we view value types as typing *values*, which are static entities, and we view computation types as typing *computations*, which are dynamic entities. If a term  $t$  has computation type, and contains a parameter  $z$  of computation type then there is a natural notion of  $t$  depending linearly on  $z$ : the execution of the computation  $t$  contains within it exactly one execution of the computation  $z$ . Since computations may perform arbitrary effects including nontermination, such a linear dependency can only hold in general if the execution of  $z$  is the first subcomputation performed in the execution of  $t$ . (If, for example, a computation that diverges were due to be performed before  $z$  then  $z$  might never be executed.) Thus we may rephrase,  $t$  depends linearly on  $z$  if the execution of the computation  $t$  begins with the execution of the computation  $z$ . Accordingly, we have arrived at a notion of  $t$  depending linearly on  $z$  that is similar in spirit to saying that  $t[z]$  is an evaluation context. The situation for value types is fundamentally different. Since values are static, they are reasonably considered as pervasive entities that might be used any number of times. Accordingly, there is no natural notion of a term  $t$  depending linearly on a parameter  $x$  of value type.

The above discussion is intended to give informal motivation for considering type rules for the effect calculus based on two typing judgements:

- $$\begin{aligned} \text{(i)} \quad & \Gamma \mid - \vdash t : \mathbf{A} \\ \text{(ii)} \quad & \Gamma \mid z : \underline{\mathbf{A}} \vdash t : \underline{\mathbf{B}} , \end{aligned}$$

where  $\Gamma$  is a context of value-type assignments to variables. On the right of  $\Gamma$  is a *stoup* (following the terminology of [8]), which may either be empty, as in the case of judgement (i), or may consist of a unique type assignment  $z : \underline{\mathbf{A}}$ , in which case the type on the right of the turnstile is also required to be a computation type, as in (ii). The purpose of judgement (i) is merely to assert that  $t$  has value type  $\mathbf{A}$  in (value) context  $\Gamma$ . Judgement (ii) asserts that  $t$  is a computation of type  $\underline{\mathbf{B}}$  (in context  $\Gamma$ ) which depends linearly on the computation  $z$  of type  $\underline{\mathbf{A}}$ . Note how these two judgement forms correspond to the informal discussion of linearity given above. This discussion also provides some intuitive motivation for the restriction of the linear context to a stoup containing at most

$$\begin{array}{c}
 \frac{}{\Gamma, x:A \mid - \vdash x:A} \qquad \frac{}{\Gamma \mid z:\underline{A} \vdash z:\underline{A}} \qquad \frac{}{\Gamma \mid \Delta \vdash *: 1} \\
 \\
 \frac{\Gamma \mid \Delta \vdash t:A \quad \Gamma \mid \Delta \vdash u:B}{\Gamma \mid \Delta \vdash \langle t, u \rangle: A \times B} \qquad \frac{\Gamma \mid \Delta \vdash t:A \times B}{\Gamma \mid \Delta \vdash \text{fst}(t): A} \qquad \frac{\Gamma \mid \Delta \vdash t:A \times B}{\Gamma \mid \Delta \vdash \text{snd}(t): B} \\
 \\
 \frac{\Gamma, x:A \mid \Delta \vdash t:B}{\Gamma \mid \Delta \vdash \lambda x:A. t: A \rightarrow B} \qquad \frac{\Gamma \mid \Delta \vdash s:A \rightarrow B \quad \Gamma \mid - \vdash t:A}{\Gamma \mid \Delta \vdash s(t): B} \\
 \\
 \frac{\Gamma \mid - \vdash t:A}{\Gamma \mid - \vdash !t: !A} \qquad \frac{\Gamma \mid \Delta \vdash t: !A \quad \Gamma, x:A \mid - \vdash u:\underline{B}}{\Gamma \mid \Delta \vdash \text{let } !x \text{ be } t \text{ in } u: \underline{B}}
 \end{array}$$

**Fig. 1.** Typing rules for the effect calculus

one variable of computation type. Since a linearly-used parameter  $z$  (necessarily of computation type) must be executed first in the execution of  $t$ , it is natural that just one variable can enjoy this property.

The typing rules are given in Figure 1. In them,  $\Delta$  ranges over an arbitrary (possibly empty) stoup, and the rules are only applicable in the case of typing judgements that conform to (i) or (ii) above. The positioning of the stoups  $\Delta$  in the rules can be understood in terms of the intuitive definition of linearity given above. For example, the evaluation behaviour of the terms associated with  $!A$  types can be understood following Moggi [13,14]. In the introduction rule, the term  $!t$  represents the trivial computation that immediately returns the value  $t$  of type  $A$ . There is no space in this for any linear dependency on a subcomputation  $z$ . In the elimination rule, the term  $\text{let } !x \text{ be } t \text{ in } u$  first evaluates the computation  $t$ , binds the result (if any) to  $x$  and then proceeds to evaluate the computation  $u$ . Clearly if  $z$  is evaluated first within  $t$  then it is also evaluated first within  $\text{let } !x \text{ be } t \text{ in } u$ , and this justifies the positioning of  $\Delta$  in the rule. Observe, however, that the following variation on the rule is not legitimised by our interpretation of linearity, and hence is not included in the calculus.

$$\frac{\Gamma \mid - \vdash t: !A \quad \Gamma, x:A \mid \Delta \vdash u: \underline{B}}{\Gamma \mid \Delta \vdash \text{let } !x \text{ be } t \text{ in } u: \underline{B}} \tag{1}$$

The problem here is that any  $z$  in  $\Delta$  is evaluated as part of  $u$ , and this occurs only after  $t$  has been evaluated first. Similar explanations can be given to the other rules. They rely on giving the products a lazy interpretation: components are only evaluated once projected out. (So, e.g., the linearity in the rule for  $1$  is correct because  $1$  is the empty product and  $*$  can never be projected.)

Modulo the differences from CBPV already mentioned above (arbitrary function spaces, no sum types) the type system discussed is essentially just a concise reformulation of CBPV with *complex stacks*, as found in [11,12]. In particular, our judgement form  $\Gamma \mid - \vdash t: A$  corresponds to Levy’s  $\Gamma \vdash^v M: A$ , and our judgement  $\Gamma \mid z:\underline{A} \vdash t: \underline{B}$  corresponds to Levy’s  $\Gamma \mid \underline{A} \vdash^k K: \underline{B}$ .

$\Gamma \mid \Delta \vdash t = * : 1$	if $\Gamma \mid \Delta \vdash t : 1$
$\Gamma \mid \Delta \vdash \text{fst}(\langle t, u \rangle) = t : \mathbf{A}$	if $\Gamma \mid \Delta \vdash t : \mathbf{A}$ and $\Gamma \mid \Delta \vdash t : \mathbf{B}$
$\Gamma \mid \Delta \vdash \text{snd}(\langle t, u \rangle) = u : \mathbf{B}$	if $\Gamma \mid \Delta \vdash t : \mathbf{A}$ and $\Gamma \mid \Delta \vdash t : \mathbf{B}$
$\Gamma \mid \Delta \vdash \langle \text{fst}(t), \text{snd}(t) \rangle = t : \mathbf{A} \times \mathbf{B}$	if $\Gamma \mid \Delta \vdash t : \mathbf{A} \times \mathbf{B}$
$\Gamma \mid \Delta \vdash (\lambda x : \mathbf{A}. t)(u) = t[u/x] : \mathbf{B}$	if $\Gamma, x : \mathbf{A} \mid \Delta \vdash t : \mathbf{B}$ and $\Gamma \mid - \vdash u : \mathbf{A}$
$\Gamma \mid \Delta \vdash \lambda x : \mathbf{A}. (t(x)) = t : \mathbf{A} \rightarrow \mathbf{B}$	if $\Gamma \mid \Delta \vdash t : \mathbf{A} \rightarrow \mathbf{B}$ and $x \notin \Gamma, \Delta$
$\Gamma \mid - \vdash \text{let } !x \text{ be } !t \text{ in } u = u[t/x] : \underline{\mathbf{B}}$	if $\Gamma \mid - \vdash t : \mathbf{A}$ and $\Gamma, x : \mathbf{A} \mid - \vdash u : \underline{\mathbf{B}}$
$\Gamma \mid \Delta \vdash \text{let } !x \text{ be } t \text{ in } u[!x/y] = u[t/y] : \underline{\mathbf{B}}$	if $\Gamma \mid \Delta \vdash t : !\mathbf{A}$ and $\Gamma \mid y : !\mathbf{A} \vdash u : \underline{\mathbf{B}}$

**Fig. 2.** Equality rules for the effect calculus

Once again, our formulation has been chosen both for its economy and to make the extension with linear connectives transparent. Indeed, we have stayed close to linear logic notation (the exception is the use of  $\times$  for product rather than the usual linear  $\&$ ), and our typing rules are simply restrictions, from an arbitrary linear context to a stoup, of the rules for **ILL** in [11]. This, in part, motivates the nonstandard use of  $!\mathbf{A}$  instead of  $T\mathbf{A}$ . The one mismatch here is the illegitimate rule (U) above, which is valid in the context of **ILL**. In spite of this mismatch, it is our belief that the extension of the effect calculus with linear primitives presented below will make it clear that the overlap with linear logic is so strong that the linear notation is helpful more than it is misleading.

In Figure 2, we present rules for equalities between typed terms in the effect calculus. They are to be considered in addition to the expected (typed) congruence and  $\alpha$ -equivalence rules.

### 3 The Enriched Effect Calculus

The enriched effect calculus is obtained by adding a selection of type constructions from linear logic to the effect calculus. As befits the setting, this needs to be done respecting both the distinction between value and computation types, and the interpretation of linearity as a concept related to the latter.

We start with linear function types. In our setting, we have a notion of linearity between computation types only. Thus we add a type  $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$ , internalising the linear dependency of judgements  $\Gamma \mid z : \underline{\mathbf{A}} \vdash t : \underline{\mathbf{B}}$ . In order to have a calculus with a sufficiently wide collection of models (all monad models) it seems essential not to assume that  $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$  is a computation type in general. This restriction is also natural if one thinks of a linear dependency  $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$  as a transformation of computations (much like an evaluation context) rather than a computation itself. The same restriction fits naturally with the stoup-based typing judgements, since allowing a linear function to depend linearly on another parameter would lead to typing rules involving multivariable linear contexts.

In linear logic, linear function space is intimately connected to the tensor product  $\otimes$ , which normally internalises the comma separating types in the linear context of a typing judgement. In our stoup-based system, there is at most

$$\begin{array}{c}
 \frac{\Gamma \mid z:\underline{\mathbf{A}} \vdash t:\underline{\mathbf{B}}}{\Gamma \mid - \vdash \lambda z:\underline{\mathbf{A}}. t:\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}} \qquad \frac{\Gamma \mid - \vdash s:\underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \quad \Gamma \mid \Delta \vdash t:\underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash s[t]:\underline{\mathbf{B}}} \\
 \\
 \frac{\Gamma \mid - \vdash t:\underline{\mathbf{A}} \quad \Gamma \mid \Delta \vdash u:\underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash !t \otimes u:\underline{!\mathbf{A}} \otimes \underline{\mathbf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash s:\underline{!\mathbf{A}} \otimes \underline{\mathbf{B}} \quad \Gamma, x:\underline{\mathbf{A}} \mid z:\underline{\mathbf{B}} \vdash t:\underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash \text{let } !x \otimes z \text{ be } s \text{ in } t:\underline{\mathbf{C}}} \\
 \\
 \frac{\Gamma \mid \Delta \vdash t:\underline{\mathbf{0}}}{\Gamma \mid \Delta \vdash \underline{\text{image}}(t):\underline{\mathbf{A}}} \qquad \frac{\Gamma \mid \Delta \vdash t:\underline{\mathbf{A}}}{\Gamma \mid \Delta \vdash \underline{\text{inl}}(t):\underline{\mathbf{A}} \oplus \underline{\mathbf{B}}} \qquad \frac{\Gamma \mid \Delta \vdash t:\underline{\mathbf{B}}}{\Gamma \mid \Delta \vdash \underline{\text{inr}}(t):\underline{\mathbf{A}} \oplus \underline{\mathbf{B}}} \\
 \\
 \frac{\Gamma \mid \Delta \vdash s:\underline{\mathbf{A}} \oplus \underline{\mathbf{B}} \quad \Gamma \mid x:\underline{\mathbf{A}} \vdash t:\underline{\mathbf{C}} \quad \Gamma \mid y:\underline{\mathbf{B}} \vdash u:\underline{\mathbf{C}}}{\Gamma \mid \Delta \vdash \underline{\text{case}}\ s \text{ of } (\underline{\text{inl}}(x).t; \underline{\text{inr}}(y).u):\underline{\mathbf{C}}}
 \end{array}$$

**Fig. 3.** Additional typing rules for the enriched effect calculus

one type in the linear context (the stoup), and so it seems awkward to implement the usual symmetric  $\otimes$ . Similarly, it is also difficult to find an appropriate  $\otimes$  operation in a sufficiently general class of models. What does work, both syntactically and semantically, is an asymmetric version: for any value type  $\underline{\mathbf{A}}$  and computation type  $\underline{\mathbf{B}}$ , we include a new computation (and hence value) type  $\underline{!\mathbf{A}} \otimes \underline{\mathbf{B}}$ . Note that this is the application of a single primitive binary constructor. The hybrid notation is chosen to maintain consistency with linear logic.

Finally, we include linear coproducts of computation types,  $\underline{\mathbf{A}} \oplus \underline{\mathbf{B}}$  and  $\underline{\mathbf{0}}$ .

The resulting *enriched effect calculus* has types defined by extending the grammar for value and computation types of the effect calculus with the following additional type constructors.

$$\begin{aligned}
 \underline{\mathbf{A}} &::= \dots \mid \underline{\mathbf{A}} \multimap \underline{\mathbf{B}} \mid \underline{!\mathbf{A}} \otimes \underline{\mathbf{B}} \mid \underline{\mathbf{0}} \mid \underline{\mathbf{A}} \oplus \underline{\mathbf{B}} \\
 \underline{\mathbf{A}} &::= \dots \mid \underline{!\mathbf{A}} \otimes \underline{\mathbf{B}} \mid \underline{\mathbf{0}} \mid \underline{\mathbf{A}} \oplus \underline{\mathbf{B}} .
 \end{aligned}$$

The judgement forms for the enriched effect calculus are exactly as for the effect calculus (now using the extended range of types). The additional typing rules are presented in Figure 3. Again, they can be seen to be restrictions of standard intuitionistic linear logic rules, as in 1. The equality theory on terms is extended by the rules in Figure 4.

The restriction that  $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$  is a value type, and the lack of a symmetric tensor have consequences on expressivity that may, at first, seem drastic. An obvious restriction is that linear function space does not iterate: neither  $\underline{\mathbf{A}} \multimap (\underline{\mathbf{B}} \multimap \underline{\mathbf{C}})$  nor  $(\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}) \multimap \underline{\mathbf{C}}$  are allowed. However, it is possible to interleave linear function space with full function space. For example,  $\underline{\mathbf{A}} \multimap (\underline{\mathbf{B}} \rightarrow \underline{\mathbf{C}})$  is a value type, and  $(\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}) \rightarrow \underline{\mathbf{C}}$  is a computation (and hence value) type. Thus, for example, the linearly-used continuations monad,  $((-) \rightarrow \underline{\mathbf{R}}) \multimap \underline{\mathbf{R}}$  is implementable as a monad on value types, for any computation type  $\underline{\mathbf{R}}$  of results. Likewise, the linearly-used state monad  $\underline{\mathbf{S}} \multimap (!(-) \otimes \underline{\mathbf{S}})$ , which makes use of the asymmetric tensor, is implementable for any computation type  $\underline{\mathbf{S}}$  of states. Such examples will be treated in detail in future papers, starting with 5.

$\Gamma \mid \Delta \vdash (\lambda x: \underline{A}. t)[u] = t[u/x]: \underline{B}$	if $\Gamma \mid x: \underline{A} \vdash t: \underline{B}$ and $\Gamma \mid \Delta \vdash u: \underline{A}$
$\Gamma \mid - \vdash \lambda x: \underline{A}. (t[x]) = t: \underline{A} \multimap \underline{B}$	if $\Gamma \mid - \vdash t: \underline{A} \multimap \underline{B}$ and $x \notin \Gamma$
$\Gamma \mid \Delta \vdash \text{let } !x \otimes y \text{ be } !t \otimes s \text{ in } u = u[t, s/x, y]: \underline{C}$	if $\Gamma \mid - \vdash t: \underline{A}$ and $\Gamma \mid \Delta \vdash s: \underline{B}$ and $\Gamma, x: \underline{A} \mid y: \underline{B} \vdash u: \underline{C}$
$\Gamma \mid \Delta \vdash \text{let } !x \otimes y \text{ be } t \text{ in } u[!x \otimes y/z] = u[t/z]: \underline{C}$	if $\Gamma \mid \Delta \vdash t: !\underline{A} \otimes \underline{B}$ and $\Gamma \mid z: !\underline{A} \otimes \underline{B} \vdash u: \underline{C}$
$\Gamma \mid x: \underline{0} \vdash t = \text{image}(x): \underline{A}$	if $\Gamma \mid x: \underline{0} \vdash t: \underline{A}$
$\Gamma \mid \Delta \vdash \text{case inl}(t) \text{ of } (\text{inl}(x). u; \text{inr}(y). u')$ $\quad = u[t/x]: \underline{C}$	if $\Gamma \mid x: \underline{A} \vdash u: \underline{C}$ and $\Gamma \mid y: \underline{B} \vdash u': \underline{C}$ and $\Gamma \mid \Delta \vdash t: \underline{A}$
$\Gamma \mid \Delta \vdash \text{case inr}(t) \text{ of } (\text{inl}(x). u; \text{inr}(y). u')$ $\quad = u'[t/y]: \underline{C}$	if $\Gamma \mid x: \underline{A} \vdash u: \underline{C}$ and $\Gamma \mid y: \underline{B} \vdash u': \underline{C}$ and $\Gamma \mid \Delta \vdash t: \underline{B}$
$\Gamma \mid \Delta \vdash \text{case } t \text{ of } (\text{inl}(x). u[\text{inl}(x)/z]; \text{inr}(y). u[\text{inr}(y)/z])$ $\quad = u[t/z]: \underline{C}$	if $\Gamma \mid \Delta \vdash t: \underline{A} \oplus \underline{B}$ and $\Gamma \mid z: \underline{A} \oplus \underline{B} \vdash u: \underline{C}$

**Fig. 4.** Additional equality rules for the enriched effect calculus

Many of the familiar laws of linear logic transfer to the enriched effect calculus, insofar as they can be expressed. For example:

$\underline{A} \rightarrow \underline{B} \cong !\underline{A} \multimap \underline{B}$	as value types
$!\underline{A} \otimes !\underline{B} \cong !( \underline{A} \times \underline{B} )$	as computation types
$!\underline{A} \otimes (\underline{B} \oplus \underline{C}) \cong (!\underline{A} \otimes \underline{B}) \oplus (!\underline{A} \otimes \underline{C})$	as computation types
$(!\underline{A} \otimes \underline{B}) \multimap \underline{C} \cong \underline{A} \rightarrow (\underline{B} \multimap \underline{C})$ $\quad \cong \underline{B} \multimap (\underline{A} \rightarrow \underline{C})$	as value types
$!1 \otimes \underline{A} \cong \underline{A}$	as computation types,

where the isomorphisms between computation types are themselves linear. The above laws demonstrate that our linear connectives behave just in the way linear logic leads us to expect they should. We take this as justification for our decision to adopt linear logic notation, including the choice of replacing  $TA$  with  $!A$ .

Our main theorem about the enriched effect calculus is that the addition of the new linear type constructions is conservative over the basic effect calculus.

**Theorem 1 (Conservativity).** *Suppose  $\Gamma, \Delta$  and  $A$  contain only types from the basic effect calculus.*

1. *If  $\Gamma \mid \Delta \vdash u: A$  is typable in the enriched effect calculus, then there exists a term  $\Gamma \mid \Delta \vdash t: A$  typable in the basic effect calculus such that  $\Gamma \mid \Delta \vdash t = u: A$  holds in the enriched effect calculus.*
2. *If  $\Gamma \mid \Delta \vdash s: A$  and  $\Gamma \mid \Delta \vdash t: A$  are typable in the basic effect calculus, and  $\Gamma \mid \Delta \vdash s = t: A$  holds in the enriched effect calculus, then  $\Gamma \mid \Delta \vdash s = t: A$  holds in the basic effect calculus.*

Statement 1 can be proved by a standard normalization argument, showing that every term  $t$  is provably equal to one for which every subterm is typed by a subtype of a type in the judgement typing  $t$ . The only proof we know of statement 2 is semantic, and will be given in Section 6.

## 4 Models

Our basic effect calculus is closely related to Levy’s CBPV with stacks. Accordingly, the natural models are given by Levy’s adjunction models [12]. While these are most simply presented as locally-indexed categories, see *op. cit.*, we use a definition based on enriched category theory, since this connects more easily with the models of the enriched effect calculus introduced below.

The idea of enriched category theory is to generalise the notion of category so that, rather than having *sets* of morphisms between pairs of objects, one has *hom-objects* given as objects of a specified “enriching” category  $\mathbf{V}$ . Thus a  $\mathbf{V}$ -enriched category (or  $\mathbf{V}$ -category),  $\mathbf{C}$ , is given by a collection of objects together with, for every pair of objects  $A, B$ , an associated object  $\mathbf{C}(A, B)$  of the category  $\mathbf{V}$ . To make this work, the category  $\mathbf{V}$  needs to be monoidal, and identity maps and an associative composition need to be specified on hom-objects. For space reasons, we refer readers to Kelly’s book [10] for full definitions. Enriched category theory generalises ordinary category theory since an ordinary locally small category is just a **Set**-enriched category.

In this paper, we always consider enrichment with respect to categories  $\mathbf{V}$  that are cartesian closed (we write  $B^A$  or  $[A \rightarrow B]$  for functions spaces), and with the enriching monoidal structure as finite product. Any such category is self-enriched. Recall that for any small category  $\mathbf{V}$ , the Yoneda embedding  $\mathbf{y}_{\mathbf{V}}$  fully and faithfully embeds  $\mathbf{V}$  into the presheaf category  $\widehat{\mathbf{V}} = \mathbf{Set}^{\mathbf{V}^{\text{op}}}$ , and since the latter is cartesian closed,  $\mathbf{V}$  is  $\widehat{\mathbf{V}}$ -enriched.

We say that a  $\mathbf{V}$ -enriched category  $\mathbf{C}$  has  $(\mathbf{V}\text{-})$ powers (Kelly writes *cotensors*) indexed by an object  $A \in \mathbf{V}$  if, for each object  $\underline{B}$  in  $\mathbf{C}$ , there exists an object  $\underline{B}^A$  of  $\mathbf{C}$  such that, for all objects  $\underline{C} \in \mathbf{C}$ , there is an isomorphism:

$$\xi_{A, \underline{B}, \underline{C}}: \mathbf{C}(\underline{C}, \underline{B}^A) \rightarrow \mathbf{C}(\underline{C}, \underline{B})^A \tag{2}$$

in  $\mathbf{V}$ , and such isomorphisms are  $\mathbf{V}$ -natural in  $\underline{C}$ . The dual property is that of having  $(\mathbf{V}\text{-})$ copowers (Kelly writes *tensors*) indexed by  $A$ : for each object  $\underline{B}$  of  $\mathbf{C}$ , there must exist an object  $A \cdot \underline{B}$  of  $\mathbf{C}$  such that there are isomorphisms in  $\mathbf{V}$ ,

$$\psi_{A, \underline{B}, \underline{C}}: \mathbf{C}(A \cdot \underline{B}, \underline{C}) \rightarrow \mathbf{C}(\underline{B}, \underline{C})^A \tag{3}$$

and again this family is  $\mathbf{V}$ -natural in  $\underline{C}$ .

An *enriched* adjunction  $F \dashv U$  between  $\mathbf{V}$ -functors  $F: \mathbf{D} \rightarrow \mathbf{C}$  and  $U: \mathbf{C} \rightarrow \mathbf{D}$  requires the existence of isomorphisms in  $\mathbf{V}$

$$\rho_{A, \underline{B}}: \mathbf{C}(F(A), \underline{B}) \rightarrow \mathbf{D}(A, U(\underline{B})) \tag{4}$$

which are  $\mathbf{V}$ -natural in  $A$  and  $\underline{B}$ , cf. [10].

**Definition 1.** An *effect-calculus model* comprises a (small) cartesian closed category  $\mathbf{V}$ , together with a  $\widehat{\mathbf{V}}$ -enriched category  $\mathbf{C}$  with:  $\widehat{\mathbf{V}}$ -enriched finite products,  $\widehat{\mathbf{V}}$ -powers indexed by representables, and a  $\widehat{\mathbf{V}}$ -adjunction  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$ .

As mentioned above, effect-calculus models are an enriched-category formulation of Levy’s *adjunction models* of CBPV [12], adapted to carry exactly the structure needed to model the effect calculus. Effect-calculus models are closely related to Moggi’s monad-based metalanguage models [14]. Given an effect-calculus model, the composite  $UF$  carries the structure of a strong monad on  $\mathbf{V}$ . Conversely, given a strong monad  $T$  on a cartesian-closed category  $\mathbf{V}$ , the canonical adjunction  $F \dashv U: \mathbf{V}^T \rightarrow \mathbf{V}$  between  $\mathbf{V}$  and the category  $\mathbf{V}^T$  of algebras for the monad, enriches to an effect-calculus model [12, Examples 4.9].

The idea behind effect-calculus models is that  $\mathbf{V}$  is the category of value types and  $\mathbf{C}$  the category of linear maps between computation types. The reason for requiring  $\mathbf{C}$  to be enriched over  $\widehat{\mathbf{V}}$  is to model judgements  $\Gamma \mid z: \underline{A} \vdash t: \underline{B}$  in non-empty contexts  $\Gamma$ , cf. [12]. In the *enriched* effect calculus, however, the presence of the linear function space  $\underline{A} \multimap \underline{B}$  as a value type means that the hom-set  $\mathbf{C}(\underline{A}, \underline{B})$  needs to live as an object of the category  $\mathbf{V}$  of value types itself. This helps to make the definition of an *enriched-effect-calculus model* simpler and more natural than the notion of effect-calculus model.

**Definition 2.** An *enriched-effect-calculus model* comprises a cartesian closed category  $\mathbf{V}$ , together with a  $\mathbf{V}$ -enriched category  $\mathbf{C}$  with:  $\mathbf{V}$ -enriched finite products and coproducts, powers, copowers, and a  $\mathbf{V}$ -adjunction  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$ .

In an obvious way, each enriched-effect-calculus model can be reconstrued as an effect-calculus model. Although the converse, of course, does not hold, we shall show in Section 6 that every effect-calculus model can be fully embedded in an enriched-effect-calculus model.

A rich source of enriched-effect-calculus models is provided by models of ILL. Amongst the various formulations of such models, the most natural for our purposes is that of *linear/nonlinear* model [2], which consists of a cartesian-closed category  $\mathbf{V}$  (the *intuitionistic category*), a symmetric monoidal closed category  $\mathbf{C}$  (the *linear category*), and a symmetric monoidal adjunction  $F \dashv G: \mathbf{C} \rightarrow \mathbf{V}$ . To model the  $\&$  and  $\oplus$  operators of ILL one further requires  $\mathbf{C}$  to have finite products and coproducts.

**Proposition 1.** *Every linear/nonlinear model in which the linear category has finite products and coproducts is an enriched-effect-calculus model.*

**Proof (outline).** Since  $\mathbf{V}$  and  $\mathbf{C}$  are closed categories, they are self-enriched. The functor  $G$ , being monoidal, transports  $\mathbf{C}$ -enriched categories to  $\mathbf{V}$ -enriched categories by application of  $G$  to hom-objects, cf. [10, p.3]. In particular, we can apply this construction to  $\mathbf{C}$  and get a  $\mathbf{V}$ -enriched category  $G_\star(\mathbf{C})$  with hom-objects  $G_\star(\mathbf{C})(\underline{A}, \underline{B}) = G(\underline{A} \multimap \underline{B})$ . The symmetric monoidal adjunction  $F \dashv G$  induces an enriched adjunction  $F \dashv G: G_\star(\mathbf{C}) \rightarrow \mathbf{V}$ . Powers and copowers can be defined respectively as  $\underline{B}^A = F(A) \multimap \underline{B}$  and  $A \cdot \underline{B} = F(A) \otimes \underline{B}$ .  $\square$

We give a uniform treatment of the interpretation of the effect calculus in any effect-calculus model, and the interpretation of the enriched effect calculus in any enriched-effect-calculus model. In either case, a value type  $A$  is interpreted



$$\begin{array}{ll}
 \mathbf{V}[\underline{\alpha}] = U(\mathbf{C}[\underline{\alpha}]) & \\
 \mathbf{V}[1] = 1 & \mathbf{C}[1] = 1 \\
 \mathbf{V}[A \times B] = \mathbf{V}[A] \times \mathbf{V}[B] & \mathbf{C}[A \times B] = \mathbf{C}[A] \times \mathbf{C}[B] \\
 \mathbf{V}[A \rightarrow B] = \mathbf{V}[A] \rightarrow \mathbf{V}[B] & \mathbf{C}[A \rightarrow B] = \mathbf{C}[B]^{\mathbf{V}[A]} \\
 \mathbf{V}[\!|A] = U(\mathbf{C}[\!|A]) & \mathbf{C}[\!|A] = F(\mathbf{V}[A])
 \end{array}$$

**Fig. 5.** Interpretation of effect-calculus types

$$\begin{array}{ll}
 \mathbf{V}[A \multimap B] = \mathbf{C}(\mathbf{C}[A], \mathbf{C}[B]) & \\
 \mathbf{V}[\!|A \otimes B] = U(\mathbf{V}[A] \cdot \mathbf{C}[B]) & \mathbf{C}[\!|A \otimes B] = \mathbf{V}[A] \cdot \mathbf{C}[B] \\
 \mathbf{V}[0] = U(0) & \mathbf{C}[0] = 0 \\
 \mathbf{V}[A \oplus B] = U(\mathbf{C}[A] + \mathbf{C}[B]) & \mathbf{C}[A \oplus B] = \mathbf{C}[A] + \mathbf{C}[B]
 \end{array}$$

**Fig. 6.** Interpretation of enriched-effect-calculus types

as an object  $\mathbf{V}[A]$  of  $\mathbf{V}$ , and each computation type  $\underline{A}$  is interpreted as a pair  $(\mathbf{C}[\underline{A}], s_{\underline{A}})$  where:  $\mathbf{C}[\underline{A}]$  is an object of  $\mathbf{C}$ , and  $s_{\underline{A}}: U(\mathbf{C}[\underline{A}]) \rightarrow \mathbf{V}[\underline{A}]$  is an isomorphism in  $\mathbf{V}$ . The interpretation is determined by specifying objects  $\mathbf{V}[\underline{\alpha}] \in \mathbf{V}$  and  $\mathbf{C}[\underline{\alpha}] \in \mathbf{C}$ , which we assume given. The remainder of the interpretation is defined in Figure 5, for both calculi, and also in Figure 6, for the remaining types of the enriched effect calculus. In the case of the effect calculus only, there is one notational simplification in Figure 5: in the definition of  $\mathbf{C}[A \rightarrow B]$ , the exponent of the power  $\mathbf{C}[B]^{\mathbf{V}[A]}$  should strictly be  $\mathbf{y}(\mathbf{V}[A])$ .

The non-trivial cases in the inductive definition of the isomorphism  $s_{\underline{A}}$  are the cases for unit type, product and function space. Since  $U$  is an enriched right adjoint, it preserves enriched limits and powers. The required isomorphisms are easily calculated from this preservation.

Terms are interpreted differently depending on whether they are typed with empty stoup or not. For  $\Gamma \mid - \vdash t: A$ , the interpretation is a map in  $\mathbf{V}$

$$\mathbf{V}[t]: \mathbf{V}[\Gamma] \rightarrow \mathbf{V}[A]$$

where  $\mathbf{V}[\Gamma]$  is the product of the interpretations of the types in  $\Gamma$ . A typing judgement  $\Gamma \mid z: \underline{A} \vdash t: \underline{B}$  is interpreted as a morphism:

$$\mathbf{C}[t]: \mathbf{V}[\Gamma] \rightarrow \mathbf{C}(\mathbf{C}[\underline{A}], \mathbf{C}[\underline{B}]).$$

For the effect calculus, this morphism is in  $\widehat{\mathbf{V}}$  (and strictly its domain is  $\mathbf{y}(\mathbf{V}[\Gamma])$ ). For the enriched calculus it is in  $\mathbf{V}$  itself. The interpretation of terms is defined by induction on the typing judgement. The details are omitted for lack of space.

**Theorem 2.** *Effect-calculus models are sound and complete with respect to the equational theory of Figure 2. Similarly, enriched-effect-calculus models are sound and complete with respect to the full equational theory of Figures 2 and 4.*

**Proof (outline).** As usual, completeness is proved by constructing a syntactic model. We consider the enriched effect calculus only. The syntactic category  $\mathbf{V}_{\text{Syn}}$  has as objects value types and as morphisms from  $\mathbf{A}$  to  $\mathbf{B}$  terms of the form  $x: \mathbf{A} \mid - \vdash t: \mathbf{B}$  identified up to equality. The  $\mathbf{V}_{\text{Syn}}$ -enriched category  $\mathbf{C}_{\text{Syn}}$  has as objects all computation types, with the  $\mathbf{V}_{\text{Syn}}$ -object of morphisms from  $\underline{\mathbf{A}}$  to  $\underline{\mathbf{B}}$  given by the value type  $\underline{\mathbf{A}} \multimap \underline{\mathbf{B}}$ . The functor  $F_{\text{Syn}}$  maps an object  $\mathbf{A}$  to  $!A$ , and the functor  $U_{\text{Syn}}$  maps  $\underline{\mathbf{A}}$  to itself. The details are routine to verify.  $\square$

## 5 Morphisms of Models

In this section, we define an appropriate notion of morphism between enriched-effect-calculus models<sup>1</sup>. The definition is subtle because it involves an unavoidable change of enrichment. Further intricacies are caused by the mathematically natural choice of requiring morphisms to preserve structure up to isomorphism rather than strictly.

Given two enriched models  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$  and  $F' \dashv U': \mathbf{C}' \rightarrow \mathbf{V}'$ , we need to consider what a structure-preserving morphism amounts to. Such a morphism must include a functor  $S: \mathbf{V} \rightarrow \mathbf{V}'$  that preserves the cartesian-closed structure. The functor  $S$  determines a 2-functor  $S_*$  from the 2-category of  $\mathbf{V}$ -categories to that of  $\mathbf{V}'$ -categories, by application of  $S$  to hom-objects, cf. [10, p.3]. Hence, we obtain a  $\mathbf{V}'$ -enriched adjunction  $S_*(F) \dashv S_*(U): S_*(\mathbf{C}) \rightarrow S_*(\mathbf{V})$ .

Next, observe that  $S$  determines a fully faithful  $\mathbf{V}'$ -functor  $S'$  from  $S_*(\mathbf{V})$  to  $\mathbf{V}'$ . This acts like  $S$  on objects, and its action on hom-objects is given by:

$$S_*(\mathbf{V})(A, B) = S(\mathbf{V}(A, B)) = S[A \rightarrow_{\mathbf{V}} B] \cong [SA \rightarrow_{\mathbf{V}'} SB] = \mathbf{V}'(SA, SB)$$

using the preservation of cartesian-closedness by  $S$ .

A morphism of effect-calculus models must also include a component mapping  $\mathbf{C}$  to  $\mathbf{C}'$ . Having performed the above change of enrichment, this is achieved by requiring a fully faithful  $\mathbf{V}'$ -functor  $T': S_*(\mathbf{C}) \rightarrow \mathbf{C}'$  (the fully faithful property of  $T'$  amounts to the preservation of linear function spaces), together with a  $\mathbf{V}'$ -enriched natural isomorphism  $\alpha: F' S' \Rightarrow T' S_*(F)$  such that the thereby determined (via the adjunctions)  $\mathbf{V}'$ -natural transformation  $\alpha': S' S_*(U) \Rightarrow U' T'$  (we call  $\alpha'$  the *mate* of  $\alpha$ ) is also an isomorphism. Pictorially, we have:

$$\begin{array}{ccc}
 S_*(\mathbf{C}) & \xrightarrow{T'} & \mathbf{C}' \\
 S_*(F) \uparrow \dashv \downarrow S_*(U) & \simeq_{\alpha} & F' \uparrow \dashv \downarrow U' \\
 S_*(\mathbf{V}) & \xrightarrow{S'} & \mathbf{V}'
 \end{array}$$

The data  $(S', T', \alpha)$  constitute a  $\mathbf{V}'$ -enriched *Beck-Chevalley square*, which is the natural notion of non-strict morphism of  $\mathbf{V}'$ -adjunctions.

<sup>1</sup> For lack of space, we do not define morphisms between effect-calculus models.

Finally, we need to ask for  $T'$  to preserve the structure on computation types. Specifically,  $T'$  must preserve finite  $\mathbf{V}'$ -enriched products and coproducts, and  $\mathbf{V}'$ -powers and copowers indexed by objects in the range of  $S$  (such powers exist in  $S_*(\mathbf{C})$  since they are inherited from  $\mathbf{C}$ ).

**Definition 3.** A *morphism of enriched-effect-calculus models* from  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$  to  $F' \dashv U': \mathbf{C}' \rightarrow \mathbf{V}'$ , is given by: a cartesian-closed functor  $S: \mathbf{V} \rightarrow \mathbf{V}'$ , which determines  $S': S_*(\mathbf{V}) \rightarrow \mathbf{V}'$  as above, a fully faithful  $\mathbf{V}'$ -enriched functor  $T': S_*(\mathbf{C}) \rightarrow \mathbf{C}'$ , and a  $\mathbf{V}'$ -natural isomorphism  $\alpha: F' S' \Rightarrow T' S_*(F)$  such that  $(S', T', \alpha)$  form a Beck-Chevalley square and  $T'$  preserves finite  $\mathbf{V}'$ -enriched products and coproducts, and  $\mathbf{V}'$ -powers and copowers indexed by objects in the range of  $S$ .

It can be shown that morphisms are closed under composition.

The main theorem of this section is that the syntactic models constructed in the proof of Theorem 2 enjoy an initiality property with respect to the above notion of morphism. Because the morphisms do not preserve structure strictly, canonical morphisms are unique only up to coherent natural isomorphism. Moreover, establishing that this property indeed holds turns out to be a surprisingly technical undertaking. For lack of space, the definition of coherent natural isomorphism and the proof of theorem below are omitted here.

**Theorem 3.** *Given an enriched-effect-calculus model  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$  and families of objects,  $\mathbf{V}[\alpha]$  in  $\mathbf{V}$  and  $\mathbf{C}[\underline{\alpha}]$  in  $\mathbf{C}$ , indexed by type constants, there exists a morphism of models from the syntactic model  $F_{\text{Syn}} \dashv U_{\text{Syn}}: \mathbf{C}_{\text{Syn}} \rightarrow \mathbf{V}_{\text{Syn}}$  to  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$  that extends the given interpretation of type constants, and this morphism is unique up to coherent natural isomorphism.*

It should be pointed out that, given the way we have defined the syntactic model, the initiality property would not hold if morphisms were required to strictly preserve structure. The reason for this is that our economical syntax makes identifications that do not hold in arbitrary models. While this could be repaired by changing to Levy's syntax, we do not view it as a defect of our approach. The non-strict notion of morphism introduced in this section is the one that is useful in applications; for example, non-strict morphisms are needed in Section 6 below, and also in the companion paper 5.

## 6 Embedding an Effect-Calculus Model in an Enriched Model

The purpose of this section is to establish that every effect-calculus model fully embeds in an enriched-effect-calculus model. It is this result that establishes that enriched-effect-calculus models are, in a precise sense, no less general than effect-calculus models. The important consequence is that the enriched effect calculus is compatible with arbitrary computational effects, whether commutative or not.

**Theorem 4.** *Every (small) effect-calculus model fully embeds in an enriched-effect-calculus model.*

**Proof (outline).** Given an effect-calculus model  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$ , we need to fully embed this in an enriched-effect-calculus model  $F' \dashv U': \mathbf{C}' \rightarrow \mathbf{V}'$ .

Since  $\mathbf{V}$  and  $\mathbf{C}$  are already  $\widehat{\mathbf{V}}$ -enriched, it is natural to define  $\mathbf{V}' = \widehat{\mathbf{V}}$ . Consider  $\mathbf{D} = \mathbf{V}'[\mathbf{C}^{\text{op}}, \mathbf{V}']$ , i.e., the  $\mathbf{V}'$ -category of  $\mathbf{V}'$ -functors from  $\mathbf{C}^{\text{op}}$  to  $\mathbf{V}'$ . It is standard that  $\mathbf{D}$  is complete and cocomplete as a  $\mathbf{V}'$ -category, and hence has all  $\mathbf{V}'$ -powers and copowers. Moreover, the enriched Yoneda functor  $\mathbf{y}': \mathbf{C} \rightarrow \mathbf{D}$  preserves all  $\mathbf{V}'$ -limits that exist in  $\mathbf{V}$ .

$\mathbf{C}'$  will be obtained as a subcategory of  $\mathbf{D}$ . To motivate its definition, we see what goes wrong if we try to use the whole of  $\mathbf{D}$  for  $\mathbf{C}'$ . We first observe that any object  $I$  of  $\mathbf{D}$  determines a  $\mathbf{C}'$ -enriched adjunction  $(-) \cdot I \dashv \mathbf{D}(I, -): \mathbf{D} \rightarrow \mathbf{C}'$ . When  $I$  is chosen to be  $\mathbf{y}'F1$ , there is a  $\mathbf{V}'$ -natural isomorphism  $\alpha': \mathbf{y}U \Rightarrow \mathbf{D}(I, -)\mathbf{y}'$ , because  $U \cong \mathbf{C}(F1, -)$  and both  $\mathbf{y}$  and  $\mathbf{y}'$  preserve powers. However, we do not have a Beck-Chevalley square of adjunctions because the “mate”  $\alpha: ((-) \cdot I)\mathbf{y} \Rightarrow \mathbf{y}'F$  of  $\alpha'$ , has the canonical maps  $\psi_X: \mathbf{y}X \cdot \mathbf{y}'F1 \rightarrow \mathbf{y}'FX$  as its components. which are not generally isomorphisms.

To rectify the situation, we define  $\mathbf{C}'$  to be the full sub- $\mathbf{V}'$ -category of  $\mathbf{D}$  on those objects orthogonal (in the  $\mathbf{V}'$ -enriched sense) to every  $\psi_X$ . One shows that all representables lie in  $\mathbf{D}$  (because  $FX$  enjoys the universal property of  $X \cdot F1$  in  $\mathbf{C}$ ), and that  $\mathbf{C}'$  has all  $\mathbf{V}'$ -limits, and these are calculated as in  $\mathbf{D}$ . Further, by [10, Theorem 6.5],  $\mathbf{C}'$  is a ( $\mathbf{V}'$ -enriched) reflective subcategory of  $\mathbf{D}$ . It follows that  $\mathbf{C}'$  has all  $\mathbf{V}'$ -colimits, in particular copowers and finite coproducts. Thus  $(-) \cdot I \dashv \mathbf{C}'(I, -): \mathbf{D} \rightarrow \mathbf{C}'$  is an enriched-effect-calculus model, where the copowers are calculated in  $\mathbf{V}'$  (by reflecting copowers from  $\mathbf{D}$ ). Moreover,

$$\begin{array}{ccc}
 \mathbf{C} & \xrightarrow{\mathbf{y}'} & \mathbf{C}' \\
 \uparrow F \dashv U & \simeq_{r(\alpha)} & (-) \cdot I \dashv \mathbf{V}'(I, -) \\
 \mathbf{V} & \xrightarrow{\mathbf{y}} & \mathbf{V}'
 \end{array}$$

is now a Beck-Chevalley square of  $\mathbf{V}'$ -adjunctions, since the  $\alpha$  components have been reflected to isomorphisms, and the  $\alpha'$  components remain unchanged.

The above data determines the morphism we require. The functor  $\mathbf{y}: \mathbf{V} \rightarrow \mathbf{V}'$  is cartesian closed. The  $\mathbf{V}'$ -functor  $\mathbf{y}'$  is fully faithful (as in Definition 3, this is a requirement for structure preservation). The Beck-Chevalley property means that the adjunction is preserved. Moreover,  $\mathbf{y}'$  preserves all existing  $\mathbf{V}'$ -limits, in particular finite products and powers indexed by objects of  $\mathbf{V}$  (qua representables). Thus we indeed have appropriate data for a morphism of models. The statement that this morphism is a full embedding amounts to  $\mathbf{y}$  being fully faithful as a  $\mathbf{V}'$ -functor, which is standard.  $\square$

We apply the above embedding to complete the proof of Theorem 11.

**Proof of Theorem 11.2 (outline).** Suppose  $\Gamma \mid \Delta \vdash s, t: A$  are typable in the basic effect calculus and  $\Gamma \mid \Delta \vdash s = t: A$  holds in the enriched effect calculus, then  $\llbracket s \rrbracket = \llbracket t \rrbracket$  holds in all models. Thus, for all models  $F \dashv U: \mathbf{C} \rightarrow \mathbf{V}$  of the

basic effect calculus,  $\llbracket s \rrbracket = \llbracket t \rrbracket$  holds in the model  $F' \dashv U' : \mathbf{C}' \rightarrow \mathbf{V}'$  given by Theorem 4. Since the latter model fully embeds the former,  $\llbracket s \rrbracket = \llbracket t \rrbracket$  holds in every model of the basic effect calculus. By the completeness of such models,  $\Gamma \mid \Delta \vdash s = t : A$  holds in the basic effect calculus.  $\square$

## References

1. Barber, A.: Linear Type Theories, Semantics and Action Calculi. PhD Thesis, University of Edinburgh (1997)
2. Benton, P.N.: A mixed linear and non-linear logic: Proofs, terms and models. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933. Springer, Heidelberg (1995)
3. Berdine, J., O’Hearn, P.W., Reddy, U., Thielecke, H.: Linear continuation-passing. *Higher Order and Symbolic Computation* 15, 181–208 (2002)
4. Benton, P.N., Wadler, P.: Linear logic, monads, and the lambda calculus. In: Proc. 11th Annual Symposium on Logic in Computer Science (1996)
5. Egger, J., Møgelberg, R.E., Simpson, A.: Linearly-used continuations in an enriched effect calculus (in preparation) (2009)
6. Filinski, A.: Controlling Effects. PhD thesis, School of Comp. Sci., CMU (1996)
7. Girard, J.-Y.: Linear logic. *Theoretical Computer Science* 50, 1–102 (1987)
8. Girard, J.-Y.: A new constructive logic: classical logic. *Mathematical Structures in Computer Science* 1, 255–296 (1991)
9. Hasegawa, M.: Linearly used effects: Monadic and CPS transformations into the linear lambda calculus. In: Hu, Z., Rodríguez-Artalejo, M. (eds.) FLOPS 2002. LNCS, vol. 2441, pp. 167–182. Springer, Heidelberg (2002)
10. Kelly, G.M.: Basic Concepts of Enriched Category Theory. London Math. Society Lecture Note Series, vol. 64. Cambridge University Press, Cambridge (1982)
11. Levy, P.B.: Call-by-push-value. A functional/imperative synthesis. In: *Semantic Structures in Computation*. Springer, Heidelberg (2004)
12. Levy, P.B.: Adjunction models for call-by-push-value with stacks. *Theory and Applications of Categories* 14, 75–110 (2005)
13. Moggi, E.: Computational lambda-calculus and monads. In: Proc. 4th Annual Symposium on Logic in Computer Science, pp. 14–23 (1989)
14. Moggi, E.: Notions of computation and monads. *Information and Computation* 93, 55–92 (1991)
15. O’Hearn, P.W., Reynolds, J.C.: From Algol to Polymorphic Linear Lambda-calculus. *Journal of the ACM* 47, 167–223 (2000)

# Degrees of Undecidability in Term Rewriting

Jörg Endrullis<sup>1</sup>, Herman Geuvers<sup>2,3</sup>, and Hans Zantema<sup>2,3</sup>

<sup>1</sup> Vrije Universiteit Amsterdam, The Netherlands

joerg@few.vu.nl

<sup>2</sup> Radboud Universiteit Nijmegen, The Netherlands

herman@cs.ru.nl

<sup>3</sup> Technische Universiteit Eindhoven, The Netherlands

h.zantema@tue.nl

**Abstract.** Undecidability of various properties of first order term rewriting systems is well-known. An undecidable property can be classified by the complexity of the formula defining it. This gives rise to a hierarchy of distinct levels of undecidability, starting from the arithmetical hierarchy classifying properties using first order arithmetical formulas and continuing into the analytic hierarchy, where also quantification over function variables is allowed.

In this paper we consider properties of first order term rewriting systems and classify them in this hierarchy. Most of the standard properties are  $\Pi_2^0$ -complete, that is, of the same level as uniform halting of Turing machines. In this paper we show two exceptions. Weak confluence is  $\Sigma_1^0$ -complete, and therefore essentially easier than ground weak confluence which is  $\Pi_2^0$ -complete.

The most surprising result is on dependency pair problems: we prove this to be  $\Pi_1^1$ -complete, which means that this property exceeds the arithmetical hierarchy and is essentially analytic. A minor variant, dependency pair problems with minimality flag, turns out to be  $\Pi_2^0$ -complete again, just like the original termination problem for which dependency pair analysis was developed.

## 1 Introduction

In classical computability theory a property  $P \subseteq \mathbb{N}$  is called *decidable* iff there exists a Turing machine which for every input  $x \in \mathbb{N}$  outputs 0 if  $x \in P$  and 1 if  $x \notin P$ . The complexity of decidable properties is usually defined in terms of the time (or space) consumption of a Turing machine that decides the property; the respective hierarchies (linear, polynomial, exponential, ...) are well-known. Likewise, but less known, the undecidable properties can be classified into a hierarchy of growing complexity. The arithmetical and the analytical hierarchy establish such a classification of undecidable properties by the complexity of predicate logic formulas that define them, which in turn is defined as the number of quantifier alternations of its prenex normal form.

The *arithmetical hierarchy* is based on first order formulas, that is, quantification is restricted to number quantifiers, function or set quantification is not

allowed; its classes are denoted  $\Pi_n^0$  and  $\Sigma_n^0$  for  $n \in \mathbb{N}$ . The lowest level of the hierarchy, the classes  $\Pi_0^0$  and  $\Sigma_0^0$ , consists of the decidable relations (for which there is a total computable function that decides it). Then the classes  $\Pi_n^0$  and  $\Sigma_n^0$  for  $n \geq 1$  are inductively defined by allowing additional universal and existential quantifiers to define the properties. For example, if  $P(x, y, z)$  is a decidable property, then  $\exists x. P(x, y, z)$  is in  $\Sigma_1^0$  and  $\forall y. \exists x. P(x, y, z)$  is in  $\Pi_2^0$ . In other words, a relation belongs to the class  $\Pi_n^0$  for  $n \in \mathbb{N}$  of the arithmetical hierarchy if it can be defined by a first order formula (in prenex normal form), which has  $n$  quantifiers, starting with a universal quantifier. Likewise a relation is in  $\Sigma_n^0$  if the formula starts with an existential quantifier. The class  $\Sigma_1^0$  consists of *recursively enumerable* (or semi-decidable) relations; the blank tape halting problem is in this class. The initialised uniform halting problem is in the class  $\Pi_2^0$ .

The *analytical hierarchy* continues the classification by second order formulas, allowing for function quantifiers. Its classes are denoted  $\Pi_n^1$  and  $\Sigma_n^1$  for  $n \in \mathbb{N}$ . The lowest level of the analytical hierarchy are the the classes  $\Pi_0^1$  and  $\Sigma_0^1$  which consist of all arithmetical relations. The classes  $\Pi_n^1$  and  $\Sigma_n^1$  for  $n \geq 1$  are defined inductively, each time adding an universal ( $\forall \alpha : \mathbb{N} \rightarrow \mathbb{N}. \varphi$ ) or existential function quantifier ( $\exists \alpha : \mathbb{N} \rightarrow \mathbb{N}. \varphi$ ), respectively. For the current paper we employ only the class  $\Pi_1^1$  of the analytical hierarchy which consists of relations that can be defined by  $\forall \alpha : \mathbb{N} \rightarrow \mathbb{N}. \varphi$  where  $\varphi$  is an arithmetical relation.

*Our Contribution.* We investigate the complexity, of the following properties of first order TRSs:

- confluence (CR),
- weak confluence (WCR),
- finiteness of dependency pair problems [2,5] (DP), and
- finiteness of dependency pair problems with minimality flag [5] ( $DP^{\min}$ ).

In this paper we pinpoint the precise complexities of these properties in terms of the arithmetic and analytic hierarchy. Table 1 provides a classification of various standard properties of TRSs: CR, WCR, DP,  $DP^{\min}$ , termination (SN), ground confluence (grCR), and weak ground confluence (grWCR). We note that  $DP^{\min}$  is only applicable in the uniform variant, see Section 6.

**Table 1.** Degrees of undecidability

	SN	WN	CR	grCR	WCR	grWCR	DP	$DP^{\min}$
uniform	$\Pi_2^0$	$\Pi_2^0$	$\Pi_2^0$	$\Pi_2^0$	$\Sigma_1^0$	$\Pi_2^0$	$\Pi_1^1$	$\Pi_2^0$
single term	$\Sigma_1^0$	$\Sigma_1^0$	$\Pi_2^0$	$\Pi_2^0$	$\Sigma_1^0$	$\Sigma_1^0$	$\Pi_1^1$	–

The contributions of this paper are encircled. The non-encircled uniform properties have been studied in [7] and [12]. For the complexity of these properties for single terms we refer to [3], an extended version of the present paper.

We deepen the study of [12] and find surprisingly that weak ground confluence is harder than weak confluence. While in [12] it has been shown that weak ground

confluence is  $\Pi_2^0$ -complete, we prove that weak confluence (that is, including open terms) is  $\Sigma_1^0$ -complete, a class strictly below  $\Pi_2^0$  in the arithmetical hierarchy. This is an excellent counterexample to a common pitfall for people less familiar with complexity theory: the  $\Pi_2^0$ -hardness of weak confluence on all ground terms does not imply  $\Pi_2^0$ -hardness of weak confluence on the larger set of all terms.

As can be seen in Table 1, the standard properties of TRSs reside within the classes  $\Pi_2^0$  and  $\Sigma_1^0$  of the arithmetical hierarchy (both for the uniform and single term versions). That is, they are of a low degree of undecidability, being at most as hard as the initialised uniform halting problem.

Surprisingly, it turns out that dependency pair problems are of a much higher degree of undecidability: they exceed the whole arithmetical hierarchy and thereby first order predicate logic. In particular we show that dependency pair problems are  $\Pi_1^1$ -complete, a class within the analytical hierarchy with one universal function quantifier. So although dependency pair problems have been invented for proving termination, the complexity of general dependency pair problems is much higher than the complexity of termination itself. This even holds if we restrict to the special format of dependency pairs: dependency pairs are right-linear, all root symbols of left hand sides and right hand sides of dependency pairs are marked, and all other symbols in the dependency pairs and all symbols in the rewrite rules are unmarked. A variant of dependency pair problems again arising from termination problems are dependency pair problems with minimality flag. We show that for this variant the complexity is back to that of termination: it is  $\Pi_2^0$ -complete.

## 2 Preliminaries

### Term Rewriting

We give a brief introduction to term rewriting, we refer to [13] for further reading. A *signature*  $\Sigma$  is a finite set of symbols  $f$  each having a fixed *arity*  $\#(f) \in \mathbb{N}$ . Let  $\Sigma$  be a signature and  $\mathcal{X}$  a countably infinite set of variables such that  $\Sigma \cap \mathcal{X} = \emptyset$ . The *set*  $Ter(\Sigma, \mathcal{X})$  of terms over  $\Sigma$  and  $\mathcal{X}$  is the smallest set satisfying:

- $\mathcal{X} \subseteq Ter(\Sigma, \mathcal{X})$ , and
- $f(t_1, \dots, t_n) \in Ter(\Sigma, \mathcal{X})$  if  $f \in \Sigma$  with arity  $n$  and  $\forall i : t_i \in Ter(\Sigma, \mathcal{X})$ .

We use  $x, y, z, \dots$  to range over variables. The set of positions  $Pos(t) \subseteq \mathbb{N}^*$  of a term  $t \in Ter(\Sigma, \mathcal{X})$  is inductively defined by:  $Pos(f(t_1, \dots, t_n)) = \{\varepsilon\} \cup \{ip \mid 1 \leq i \leq \#(f), p \in Pos(t_i)\}$ , and  $Pos(x) = \{\varepsilon\}$  for variables  $x \in \mathcal{X}$ . We use  $\equiv$  for syntactical equivalence of terms.

A substitution  $\sigma$  is a map  $\sigma : \mathcal{X} \rightarrow Ter(\Sigma, \mathcal{X})$  from variables to terms. For terms  $t \in Ter(\Sigma, \mathcal{X})$  and substitutions  $\sigma$  we define  $t\sigma$  as the result of replacing each  $x \in \mathcal{X}$  in  $t$  by  $\sigma(x)$ . That is,  $t\sigma$  is inductively defined by  $x\sigma := \sigma(x)$  for variables  $x \in \mathcal{X}$  and otherwise  $f(t_1, \dots, t_n)\sigma := f(t_1\sigma, \dots, t_n\sigma)$ . Let  $\square$  be a fresh symbol,  $\square \notin \Sigma \cup \mathcal{X}$ . A *context*  $C$  is a term from  $Ter(\Sigma, \mathcal{X} \cup \{\square\})$  containing precisely one occurrence of  $\square$ . Then  $C[s]$  denotes the term  $C\sigma$  where  $\sigma(\square) = s$  and  $\sigma(x) = x$  for all  $x \in \mathcal{X}$ .



A *term rewriting system (TRS)* over  $\Sigma$ ,  $\mathcal{X}$  is a set  $R$  of finitely many pairs  $\langle \ell, r \rangle \in \text{Ter}(\Sigma, \mathcal{X})$ , called *rewrite rules* and usually written as  $\ell \rightarrow r$ , for which the *left-hand side*  $\ell$  is not a variable ( $\ell \notin \mathcal{X}$ ) and all variables in the *right-hand side*  $r$  occur in  $\ell$  ( $\text{Var}(r) \subseteq \text{Var}(\ell)$ ). Let  $R$  be a TRS. For terms  $s, t \in \text{Ter}(\Sigma, \mathcal{X})$  we write  $s \rightarrow_R t$  if there exists a rule  $\ell \rightarrow r \in R$ , a substitution  $\sigma$  and a context  $C \in \text{Ter}(\Sigma, \mathcal{X} \cup \{\square\})$  such that  $s \equiv C[\ell\sigma]$  and  $t \equiv C[r\sigma]$ ;  $\rightarrow_R$  is the *rewrite relation* induced by  $R$ , and  $\rightarrow_R^*$  denotes the reflexive, transitive closure of  $\rightarrow_R$ .

**Definition 2.1.** Let  $R$  be a TRS and  $t \in \text{Ter}(\Sigma, \mathcal{X})$  a term. Then  $R$  is called

- *strongly normalizing (or terminating) on  $t$* , denoted  $\text{SN}_R(t)$ , if every rewrite sequence starting from  $t$  is finite.
- *confluent (or Church-Rosser) on  $t$* , denoted  $\text{CR}_R(t)$ , if every pair of finite cointial reductions starting from  $t$  can be extended to a common reduct, that is,  $\forall t_1, t_2. t_1 \leftarrow^* t \rightarrow^* t_2 \Rightarrow \exists d. t_1 \rightarrow^* d \leftarrow^* t_2$ .
- *weakly confluent (or weakly Church-Rosser) on  $t$* , denoted  $\text{WCR}_R(t)$ , if every pair of cointial rewrite steps starting from  $t$  can be joined, that is,  $\forall t_1, t_2. t_1 \leftarrow t \rightarrow t_2 \Rightarrow \exists d. t_1 \rightarrow^* d \leftarrow^* t_2$ .

The TRS  $R$  is *strongly normalizing* ( $\text{SN}_R$ ), *confluent* ( $\text{CR}_R$ ) or *weakly confluent* ( $\text{WCR}_R$ ) if the respective property holds on all terms  $t \in \text{Ter}(\Sigma, \mathcal{X})$ .

## Turing Machines

**Definition 2.2.** A *Turing machine*  $M$  is a quadruple  $\langle Q, \Gamma, q_0, \delta \rangle$  consisting of:

- finite set of states  $Q$ ,
- an initial state  $q_0 \in Q$ ,
- a finite alphabet  $\Gamma$  containing a designated symbol  $\square$ , called *blank*, and
- a partial *transition function*  $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ .

A *configuration* of a Turing machine is a pair  $\langle q, \text{tape} \rangle$  consisting of a state  $q \in Q$  and the tape content  $\text{tape} : \mathbb{Z} \rightarrow \Gamma$  such that the carrier  $\{n \in \mathbb{Z} \mid \text{tape}(n) \neq \square\}$  is finite. The set of all configurations is denoted  $\text{Conf}_M$ . We define the relation  $\rightarrow_M$  on the set of configurations  $\text{Conf}_M$  as follows:  $\langle q, \text{tape} \rangle \rightarrow_M \langle q', \text{tape}' \rangle$  whenever:

- $\delta(q, \text{tape}(0)) = \langle q', f, L \rangle$ ,  $\text{tape}'(1) = f$  and  $\forall n \neq 0. \text{tape}'(n+1) = \text{tape}(n)$ , or
- $\delta(q, \text{tape}(0)) = \langle q', f, R \rangle$ ,  $\text{tape}'(-1) = f$  and  $\forall n \neq 0. \text{tape}'(n-1) = \text{tape}(n)$ .

Without loss of generality we assume that  $Q \cap \Gamma = \emptyset$ , that is, the set of states and the alphabet are disjoint. This enables us to denote configurations as  $\langle w_1, q, w_2 \rangle$ , denoted  $w_1^{-1}qw_2$  for short, with  $w_1, w_2 \in \Gamma^*$  and  $q \in Q$ , which is shorthand for  $\langle q, \text{tape} \rangle$  where  $\text{tape}(n) = w_2(n+1)$  for  $0 \leq n < |w_2|$ , and  $\text{tape}(-n) = w_1(n)$  for  $1 \leq n \leq |w_1|$  and  $\text{tape}(n) = \square$  for all other positions  $n \in \mathbb{Z}$ .

The Turing machines we consider are deterministic. As a consequence, final configurations are unique (if they exist), which justifies the following definition.

**Definition 2.3.** Let  $M$  be a Turing machine and  $\langle q, \text{tape} \rangle \in \text{Conf}_M$ . We denote by  $\text{final}_M(\langle q, \text{tape} \rangle)$  the  $\rightarrow_M$ -normal form of  $\langle q, \text{tape} \rangle$  if it exists and undefined, otherwise. Whenever  $\text{final}_M(\langle q, \text{tape} \rangle)$  exists then we say that  $M$  halts on  $\langle q, \text{tape} \rangle$  with final configuration  $\text{final}_M(\langle q, \text{tape} \rangle)$ . Furthermore we say  $M$  halts on  $\text{tape}$  as shorthand for  $M$  halts on  $\langle q_0, \text{tape} \rangle$ .

Turing machines can compute  $n$ -ary functions  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  or relations  $S \subseteq \mathbb{N}^*$ . We need only unary functions  $f_M$  and binary  $>_M \subseteq \mathbb{N} \times \mathbb{N}$  relations.

**Definition 2.4.** Let  $M = \langle Q, \Gamma, q_0, \delta \rangle$  be a Turing machine with  $S, 0 \in \Gamma$ . We define a partial function  $f_M : \mathbb{N} \rightarrow \mathbb{N}$  for all  $n \in \mathbb{N}$  by:

$$f_M(n) = \begin{cases} m & \text{if } \text{final}_M(q_0 S^n 0) = \dots q S^m 0 \dots \\ \text{undefined} & \text{otherwise} \end{cases}$$

and we define the relation  $>_M \subseteq \mathbb{N} \times \mathbb{N}$  by:

$$n >_M m \iff \text{final}_M(0 S^n q_0 S^m 0) = \dots q_0 \dots$$

Here, the functions  $f_M$  are partial since  $M$  may not terminate on certain inputs or  $M$  halts in a state which is not of the form  $\dots q S^m 0 \dots$ . Note that the set  $\{>_M \mid M \text{ halts on all tapes}\}$  is the set of recursive binary relations on  $\mathbb{N}$ .

### The Arithmetic and Analytical Hierarchy

In the introduction we briefly mentioned the arithmetical and analytical hierarchy. We now summarize the main notions and results relevant for this paper. For details see a standard text on mathematical logic, e.g. [11] or [6], which contains more technical results regarding these hierarchies.

**Definition 2.5.** Let  $A \subseteq \mathbb{N}$ . The *set membership problem* for  $A$  is the problem of deciding for given  $a \in \mathbb{N}$  whether  $a \in A$ .

**Definition 2.6.** Let  $A \subseteq \mathbb{N}$  and  $B \subseteq \mathbb{N}$ . Then  $A$  can be many-one reduced to  $B$ , notation  $A \leq_m B$  if there exists a total computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that  $\forall n \in \mathbb{N}. n \in A \iff f(n) \in B$ .

**Definition 2.7.** Let  $B \subseteq \mathbb{N}$  and  $\mathcal{P} \subseteq 2^{\mathbb{N}}$ . Then  $B$  is called  $\mathcal{P}$ -hard if every  $A \in \mathcal{P}$  can be reduced to  $B$ , and  $B$  is  $\mathcal{P}$ -complete whenever additionally  $B \in \mathcal{P}$ .

So a problem  $B$  is  $\mathcal{P}$ -hard if every problem  $A \in \mathcal{P}$  can be reduced to  $B$ : To decide “ $n \in A$ ” we only have to decide “ $f(n) \in B$ ”, where  $f$  is the total computable function that reduces  $A$  to  $B$ .

The classification results in the following sections employ the following well-known lemma, which states that whenever a problem  $A$  can be reduced via a computable function to a problem  $B$ , then  $B$  is at least as hard as  $A$ .

**Lemma 2.8.** *If  $A$  can be reduced to  $B$  and  $A$  is  $\mathcal{P}$ -hard, then  $B$  is  $\mathcal{P}$ -hard.  $\square$*

*Remark 2.9.* Finite lists of natural numbers can be encoded as natural numbers using the well-known Gödel encoding:  $\langle n_1, \dots, n_k \rangle := p_1^{n_1+1} \cdot \dots \cdot p_k^{n_k+1}$ , where  $p_1, \dots, p_k$  are the first  $k$  prime numbers. For this encoding, the length function ( $\text{lth}\langle n_1, \dots, n_k \rangle = k$ ) and the decoding function ( $\text{lth}\langle n_1, \dots, n_k \rangle_i = n_i$  if  $1 \leq i \leq n$ ) are computable and it is decidable if a number is the code of a finite list.

Using the encoding of finite lists of natural numbers, we can encode Turing machines, terms and finite term rewriting systems. Finite rewrite sequences  $\sigma : t_1 \rightarrow \dots \rightarrow t_n$  can be encoded as lists of terms. Then of course a Turing machine can compute the length of  $|\sigma| := n$  of the sequence, every term  $t_1, \dots, t_n$ , in particular the first  $\text{first}(\sigma) := t_1$  and the last term  $\text{last}(\sigma) := t_n$ . Given the TRS as input, a Turing machine can check whether a natural number  $n$  corresponds to a valid rewrite sequence, that is, check  $t_i \rightarrow t_{i+1}$  for every  $i = 1, \dots, (n - 1)$ . Furthermore for a given term  $t$  and  $n \in \mathbb{N}$  it can calculate the set of all reductions of length  $\leq n$  admitted by  $t$  and thereby check properties like ‘all reductions starting from  $t$  have length  $\leq n$ ’ or ‘ $t$  is a normal form’.

An example from term rewriting that we can encode as a problem on natural numbers is (we leave the encoding of terms as numbers implicit),  $s \rightarrow_R t := \exists \ell \rightarrow r \in R. \exists \sigma. \exists C. (s \equiv C[\ell\sigma] \wedge t \equiv C[r\sigma])$ . As all these quantifiers can be bounded by the size of  $s$  and  $t$ , respectively, this amounts to a finite search and is a decidable problem. Note that the fact that the TRS is finite is crucial here.

Undecidable problems can be divided into a hierarchy of increasing complexity, the first part of which is known as the *arithmetical hierarchy*. An example is the problem whether  $t$  reduces in finitely many steps to  $q$ :  $t \rightarrow^*_R q := \exists \langle s_1, \dots, s_n \rangle. (t = s_1 \rightarrow_R \dots \rightarrow_R s_n = q)$ . This problem is undecidable in general and it resides in  $\Sigma_1^0$  which is the class of problems of the form  $\exists x \in \mathbb{N}. P(x, n)$  where  $P(x, n)$  is a decidable problem. (We usually suppress the domain behind the existential quantifier.) Similar to  $\Sigma_1^0$ , we have the class  $\Pi_1^0$ , which is the class of problems of the form  $\forall x \in \mathbb{N}. P(x, n)$  with  $P(x, n)$  a decidable problem. If we continue this procedure, we obtain the classes  $\Sigma_n^0$  and  $\Pi_n$  for every  $n \in \mathbb{N}$ .

**Definition 2.10.** The class  $\Sigma_n^0$  consists of all sets  $A$  that can be defined in form of  $A(k) \iff \exists x_n. \forall x_{n-1}. \dots P(x_1, \dots, x_n, k)$  where  $P$  is a decidable relation. So, there is a sequence of  $n$  alternating quantifiers in front of  $P$ . Likewise  $\Pi_n^0$  is the class of sets of the form  $A(k) \iff \forall x_n. \exists x_{n-1}. \dots P(x_1, \dots, x_n, k)$  where  $P$  is decidable. Then  $\Delta_n^0$  is the intersection of  $\Sigma_n^0$  and  $\Pi_n^0$ , that is,  $\Delta_n^0 := \Sigma_n^0 \cap \Pi_n^0$ .

That this definition is useful is based on the following fact, for which we refer to [8, 6, 11] for a proof and further details.

*Remark 2.11.* Every formula in first order arithmetic is equivalent to a formula in *prenex normal form*, i.e. a formula with all quantifiers on the outside of the formula. Furthermore a sequence of  $\exists$  (or  $\forall$ ) can always be replaced by one  $\exists$  (or one  $\forall$ , respectively) due to the encoding of a finite list of numbers into numbers.

The reason one writes 0 as a superscript is that all quantifiers range over “the lowest type”  $\mathbb{N}$ ; there are no quantifiers of higher types, like  $\mathbb{N} \rightarrow \mathbb{N}$ . So every

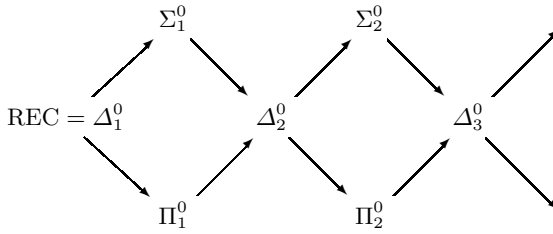


Fig. 1. Arithmetic Hierarchy

arithmetical problem is in one of the classes of Definition 2.10. A natural question is whether all these classes are distinct. A fundamental result in mathematical logic says that they are, see [1], [8] or [6].

The arithmetic hierarchy is usually depicted as in Figure 1, where every arrow denotes a proper inclusion. All classes are closed under bounded quantification: if  $A(n) \Leftrightarrow \exists y < t(n) P(n, y)$  and  $P$  is decidable, then  $A$  is decidable (and similarly for other classes in the hierarchy).

Above the arithmetic hierarchy, we find the *analytical hierarchy*, where we also allow quantification over infinite sequences of numbers (equivalently functions  $f : \mathbb{N} \rightarrow \mathbb{N}$ ). The classes of the analytical hierarchy are denoted  $\Pi_n^1$  and  $\Sigma_n^1$  with  $n \in \mathbb{N}$  where the  $n$  indicates the number of function quantifiers in prenex normal form. That is, first-order quantifiers not counted. As a consequence the lowest classes  $\Pi_0^1$  and  $\Sigma_0^1$  subsume the whole arithmetical hierarchy. As variables ranging over infinite sequences we use  $\alpha, \beta$ , etc. For the analytical hierarchy we can draw a similar diagram as the one in Figure 1: replace  $\Sigma_1^0$  by  $\Sigma_1^1$  etc. To keep the presentation as simple as possible we define only the class  $\Pi_1^1$ .

**Definition 2.12.** The class  $\Pi_1^1$  consists of all sets  $A$  that can be defined in form of  $A(k) \Leftrightarrow \forall \alpha. \varphi$  where  $\varphi$  is a first order formula over decidable predicates.

Note that  $\varphi$  does not need to be in prenex normal form. W.l.o.g. every  $\varphi$  can be converted into an equivalent formula  $\varphi'$  in prenex normal form and  $\forall \alpha. \varphi'$  is still a  $\Pi_1^1$ -formula as first order quantifiers are not counted. For analytical problems we also have all kinds of simplification procedures (analogous to Remark 2.11).

An example of an analytical formula is  $\forall \alpha. \exists x. \alpha(x) \not\rightarrow_R \alpha(x + 1)$ , stating that there exist no infinite rewrite sequences, that is, the rewrite system is SN. This is a  $\Pi_1^1$ -formula.

**Lemma 2.13.** We have the following well-known results:

- (i) the blank tape halting problem  $\{M \mid M \text{ halts on the blank tape}\}$  is  $\Sigma_1^0$ -complete,
- (ii) the totality problem  $\{M \mid M \text{ halts on } q_0 S^n \text{ for every } n \in \mathbb{N}\}$  is  $\Pi_2^0$ -complete,
- (iii) the set  $WF := \{M \mid M \text{ is total and } >_M \text{ is well-founded}\}$  is  $\Pi_1^1$ -complete.

These sets will be the basis for the hardness results in the following sections: we will show that the blank tape halting problem is many-one reducible to WCR and thus conclude that WCR is  $\Sigma_1^0$ -hard. This will be done by effectively giving for every Turing machine  $M$ , a TRS  $R_M$  such that

$$M \text{ halts on the blank tape} \iff \text{WCR}_{R_M}.$$

Similar constructions will be carried out for the other problems that we consider.

To determine if a problem  $A$  is essentially in a certain class  $\mathcal{P}$  (and not lower in the hierarchy), we first prove that  $A$  is  $\mathcal{P}$ -hard and then we show that the property  $A$  can be expressed by formula of  $\mathcal{P}$ .

### 3 Translating Turing Machines

We use the translation of Turing machines  $M$  to TRSs  $R_M$  from [10].

**Definition 3.1.** For every Turing machine  $M = \langle Q, \Gamma, q_0, \delta \rangle$  we define a TRS  $R_M$  as follows. The signature is  $\Sigma = Q \cup \Gamma \cup \{\triangleright\}$  where the symbols  $q \in Q$  have arity 2, the symbols  $f \in \Gamma$  have arity 1 and  $\triangleright$  is a constant symbol, which represents an infinite number of blank symbols. The rewrite rules of  $R_M$  are:

$$\begin{aligned} q(x, f(y)) &\rightarrow q'(f'(x), y) && \text{for every } \delta(q, f) = \langle q', f', R \rangle \\ q(g(x), f(y)) &\rightarrow q'(x, g(f'(y))) && \text{for every } \delta(q, f) = \langle q', f', L \rangle \end{aligned}$$

together with four rules for ‘extending the tape’:

$$\begin{aligned} q(\triangleright, f(y)) &\rightarrow q'(\triangleright, \square(f'(y))) && \text{for every } \delta(q, f) = \langle q', f', L \rangle \\ q(x, \triangleright) &\rightarrow q'(f'(x), \triangleright) && \text{for every } \delta(q, \square) = \langle q', f', R \rangle \\ q(g(x), \triangleright) &\rightarrow q'(x, g(f'(\triangleright))) && \text{for every } \delta(q, \square) = \langle q', f', L \rangle \\ q(\triangleright, \triangleright) &\rightarrow q'(\triangleright, \square(f'(\triangleright))) && \text{for every } \delta(q, \square) = \langle q', f', L \rangle. \end{aligned}$$

We introduce a mapping from terms to configurations to make the connection between the  $M$  and the TRS  $R_M$  precise.

**Definition 3.2.** We define a mapping  $\varphi : \text{Ter}(\Gamma \cup \{\triangleright\}, \emptyset) \rightarrow \Gamma^*$  by:

$$\varphi(\triangleright) := \varepsilon \qquad \varphi(f(t)) := f\varphi(t)$$

for every  $f \in \Gamma$  and  $t \in \text{Ter}(\Gamma \cup \{\triangleright\}, \emptyset)$ . We define the set of (intended) terms:

$$\text{Ter}_M := \{q(s, t) \mid q \in Q, s, t \in \text{Ter}(\Gamma \cup \{\triangleright\}, \emptyset)\}.$$

We define a map  $\Phi : \text{Ter}_M \rightarrow \text{Conf}_M$  by  $\Phi(q(s, t)) := \varphi(s)^{-1}q\varphi(t) \in \text{Conf}_M$ .

**Lemma 3.3.** *Let  $M$  be a Turing machine. Then  $R_M$  simulates  $M$ , that is:*

- (i)  $\forall c \in \text{Conf}_M. \Phi^{-1}(c) \neq \emptyset$ ,
- (ii) for all terms  $s \in \text{Ter}_M$ :  $s \rightarrow_{R_M} t$  implies  $t \in \text{Ter}_M$  and  $\Phi(s) \rightarrow_M \Phi(t)$ , and
- (iii) for all terms  $s \in \text{Ter}_M$ : whenever  $\Phi(s) \rightarrow_M c$  then  $\exists t \in \Phi^{-1}(c). s \rightarrow_{R_M} t$ .

The following is an easy corollary.

**Corollary 3.4.** *For all  $s \in \text{Ter}_M$ :  $\text{SN}_{R_M}(s) \iff M$  halts on  $\Phi(s)$ .*

*Proof.* Induction on item (ii) of Lemma 3.3. □

## 4 Weak Confluence

We show that  $WCR_R$  (uniform) and  $WCR_R(t)$  (for single terms) are  $\Sigma_1^0$ -complete. This result is surprising since (see Table II) usually the uniform property is harder than for single terms: for the uniform property one has to reason about all terms which normally amounts to an additional universal quantifier. Moreover this reveals a remarkable discrepancy in comparison with  $grWCR_R$  which has been shown  $\Pi_2^0$ -complete in [12].

**Theorem 4.1.** *Weak confluence is  $\Sigma_1^0$ -complete, both uniform  $WCR_R$  as well as for single terms  $WCR_R(t)$ .*

*Proof.* For  $\Sigma_1^0$ -hardness we use the blank tape halting problem. Let  $M$  be a Turing machine. We define the TRS  $S$  to consist of the rules of  $R_M$  extended by the following rules:

$$\begin{aligned} & \text{run} \rightarrow T \quad \text{run} \rightarrow q_0(\triangleright, \triangleright) \\ & q(x, f(y)) \rightarrow T \quad \text{for every } f \in \Gamma \text{ such that } \delta(q, f) \text{ is undefined.} \end{aligned}$$

The only critical pair is  $T \leftarrow \text{run} \rightarrow q_0(\triangleright, \triangleright)$ . We have  $q_0(\triangleright, \triangleright) \rightarrow_S^* T$ , if and only if  $M$  halts on the blank tape. By the Critical Pairs Lemma [13]  $WCR_R$  holds if and only if all critical pairs are convergent (can be joined). Hence  $WCR_R$  and  $WCR_R(t)$  (where  $t := \text{run}$ ) are  $\Sigma_1^0$ -hard.

A Turing machine can compute on the input of a TRS  $R$  all (finitely many) critical pairs, and on the input of a TRS  $R$  and a term  $t$  all (finitely many) one step reducts of  $t$ . Therefore it suffices to show that the following problem is in  $\Sigma_1^0$ : decide on the input of a TRS  $S$ ,  $n \in \mathbb{N}$  and terms  $t_1, s_1, \dots, t_n, s_n$  whether for every  $i = 1, \dots, n$  the terms  $t_i$  and  $s_i$  have a common reduct. This property can be described by the following  $\Sigma_1^0$  formula:

$$\begin{aligned} & \exists r \in \mathbb{N}. ((r \text{ is list } r_1, \dots, r_{2 \cdot n} \text{ of length } 2 \cdot n) \\ & \quad \text{and for } i = 1, \dots, n \text{ we have} \\ & \quad (r_{2 \cdot i - 1}, r_{2 \cdot i} \text{ are reductions}) \text{ and } (first(r_{2 \cdot i - 1}) \equiv t_i) \\ & \quad \text{and } (first(r_{2 \cdot i}) \equiv s_i) \text{ and } (last(r_{2 \cdot i - 1}) \equiv last(r_{2 \cdot i})). \quad \square \end{aligned}$$

We remark that the proof also shows  $\Sigma_1^0$ -completeness of weak ground confluence for single terms ( $grWCR_R(t)$ ).

It may be unexpected that  $WCR_R$  is easier than  $grWCR_R$ , so let us add some explanation. In principle we have to check for an infinite number of possibilities,  $t \rightarrow p$  and  $t \rightarrow q$ , whether  $p$  and  $q$  have a common reduct. The essence of the Critical Pairs Lemma (CPL) is that for  $WCR_R$ , it suffices to check a finite set of “overlapping patterns”. For  $grWCR_R$ , this is not enough, because, even if some of the overlapping patterns are not convergent (and thus  $WCR_R$  is false),  $grWCR_R$  may still hold: for ground terms, all overlapping patterns may still be convergent.

As a simplified instance of this situation, consider a confluent term rewriting system with a unary symbol  $F$  that defines the recursive function  $f : \mathbb{N} \rightarrow \mathbb{N}$ .

Now, if we add the rules  $\text{run}(x) \rightarrow 0$  and  $\text{run}(x) \rightarrow F(x)$ , then the rewrite system is not WCR anymore. However, it is grWCR if and only if  $f$  is the zero-function, which is a  $\Pi_2^0$  statement. Note that, this argument does not go through as it stands, because there are some technical subtleties, but it gives the basic intuition why grWCR is “harder” than WCR.

### 5 Confluence

We investigate the complexity of confluence ( $\text{CR}_R$ ). For proving  $\Pi_2^0$ -completeness of confluence one would like to use an extension of  $R_M$  with the following rules:

$$\begin{aligned} \text{run}(x, y) &\rightarrow \top \\ \text{run}(x, y) &\rightarrow q_0(x, y) \\ q(x, f(y)) &\rightarrow \top \qquad \text{for every } f \in \Gamma \text{ with } \delta(q, f) \text{ undefined} \end{aligned}$$

On first glance it seems that  $q_0(s, t) \rightarrow^* \top$  if the Turing machine  $M$  halts on all configurations. However, a problem arises if  $s$  and  $t$  contain variables; e.g. if  $s$  or  $t$  are variables themselves. We solve the problem as follows. For Turing machines  $M$  we define the TRS  $S_M$  to consist of the rules of the TRS  $R_M$  extended by:

$$\begin{aligned} \text{run}(x, \triangleright) &\rightarrow \top & (1) \\ \text{run}(\triangleright, y) &\rightarrow q_0(\triangleright, y) & (2) \\ q(x, f(y)) &\rightarrow \top \qquad \text{for every } f \in \Gamma \text{ with } \delta(q, f) \text{ undefined} & (3) \\ \text{run}(x, S(y)) &\rightarrow \text{run}(S(x), y) & (4) \\ \text{run}(S(x), y) &\rightarrow \text{run}(x, S(y)) . & (5) \end{aligned}$$

Then  $\top$  and  $q_0(\triangleright, s)$  are convertible using the rules (1)–(5) if and only if  $s$  is a ground term of the form  $S^n(\triangleright)$ .

**Theorem 5.1.** *Uniform confluence  $\text{CR}_R$  is  $\Pi_2^0$ -complete.*

*Proof.* For proving  $\Pi_2^0$ -hardness we reduce the totality problem to confluence. Let  $M$  be an arbitrary Turing machine. We consider the TRS  $S_M$  defined above. We employ type introduction [1]: we assign sort  $\gamma_0$  to  $\Gamma \cup \{\triangleright\}$  and sort  $\gamma_1$  to every symbol in  $\{\text{run}, \top\} \cup Q$ ; the obtained many-sorted TRS is confluent if and only if  $S_M$  is. Note that the terms of sort  $\gamma_0$  are normal forms and for terms of  $\gamma_1$  with root symbol  $\neq$  ‘run’ the reduction is deterministic (exhibits no branching). Therefore it suffices to consider the case

$$s_2 \leftarrow (2) s_1 \xleftarrow{* (4)} \text{run}(t_1, t_2) \xrightarrow{* (5)} s_3 \rightarrow (1) \top$$

where  $t_1, t_2 \in \text{Ter}(\Gamma \cup \{\triangleright\}, \mathcal{X})$ . From the existence of such rewrite sequences we conclude that there exists  $n \in \mathbb{N}$  such that  $s_1 \equiv \text{run}(\triangleright, S^n(\triangleright))$ ,  $s_3 \equiv \text{run}(S^n(\triangleright), \triangleright)$ , and  $s_2 \equiv q_0(\triangleright, S^n(\triangleright))$ . On the other hand for every  $n \in \mathbb{N}$  such rewrite sequences exist. As a consequence the TRS  $S_M$  is confluent if and only if  $q_0(\triangleright, S^n(\triangleright)) \rightarrow_S^* \top$

for every  $n \in \mathbb{N}$ , and this holds if and only if  $M$  halts on  $q_0 S^n$  for every  $n \in \mathbb{N}$  by Corollary 3.4. This proves  $\Pi_2^0$ -hardness.

To show that  $\text{CR}_R$  is in  $\Pi_2^0$  let  $R$  be a TRS. Then  $R$  is confluent if and only if the following formula holds:

$$\begin{aligned} \text{CR}_R &\iff \forall t \in \mathbb{N}. \forall r_1, r_2 \in \mathbb{N}. \exists r'_1, r'_2 \in \mathbb{N}. \\ &(((t \text{ is a term}) \text{ and } (r_1, r_2 \text{ are reductions}) \text{ and } t \equiv \text{first}(r_1) \equiv \text{first}(r_2)) \\ &\implies ((r'_1 \text{ and } r'_2 \text{ are reductions}) \\ &\quad \text{and } (\text{last}(r_1) \equiv \text{first}(r'_1)) \text{ and } (\text{last}(r_2) \equiv \text{first}(r'_2)) . \\ &\quad \text{and } (\text{last}(r'_1) \equiv \text{last}(r'_2)))) \end{aligned}$$

By quantifier compression we can simplify the formula such that there is only one universal followed by an existential quantifier.  $\square$

## 6 Dependency Pair Problems

In this section we present the remarkable result that finiteness of dependency pair problems, although invented for proving termination, is of a much higher level of complexity than termination itself: it is  $\Pi_1^1$ -complete, both uniform and for single terms. This only holds for the basic version of dependency pairs; for the version with minimality flag (as arising from TRS termination problems) we show it is of the same level as termination itself. We emphasize that the variant without minimality flag is commonly used: they arise for example from Haskell termination problems [4], and transformations on dependency pair problems that do not preserve the minimality flag.

For relations  $\rightarrow_1, \rightarrow_2$  we write  $\rightarrow_1 / \rightarrow_2$  for  $\rightarrow_2^* \cdot \rightarrow_1$ . For TRSs  $R, S$  instead of  $\text{SN}(\rightarrow_{R,\epsilon} / \rightarrow_S)$  we shortly write  $\text{SN}(R_{\text{top}}/S)$ ; in the literature [5] this is called *finiteness of the dependency pair problem*  $\{R, S\}$ . So  $\text{SN}(R_{\text{top}}/S)$  means that every infinite  $\rightarrow_{R,\epsilon} \cup \rightarrow_S$  reduction, that is,  $R$ -steps are allowed only at the top,  $S$ -steps everywhere, contains only finitely many  $\rightarrow_{R,\epsilon}$  steps.

The motivation for studying this comes from the dependency pair approach [2] for proving termination. There a simple syntactic construction DP is given such that for any TRS  $R$  we have

$$\text{SN}(\text{DP}(R)_{\text{top}}/R) \iff \text{SN}(R).$$

In this way termination of a TRS can be proved by proving  $\text{SN}(R_{\text{top}}/S)$  for suitable TRSs  $R, S$ . This is the basis of nearly all termination proofs for TRSs as they are generated by state-of-the-art termination provers.

The main result of this section is  $\Pi_1^1$ -completeness of dependency pair problems  $\text{SN}(R_{\text{top}}/S)$ , even of  $\text{SN}(S_{\text{top}}/S)$ , for both the uniform and the single term variant, and also of  $\text{SN}(R_{\text{top}}/S)$  restricting to the format in which roots in  $R$  are marked. In the next section we will consider the variant  $\text{SN}(R_{\text{top}}/\text{min } S)$  with minimality flag which only makes sense for the uniform variant, and show that it behaves like normal termination: it is  $\Pi_2^0$ -complete.



For proving  $\Pi_1^1$ -hardness of  $\text{SN}(S_{\text{top}}/S)$  we now adopt Definition [3.1](#), the translation of Turing machines to TRSs. The crucial difference is that every step of the Turing machine ‘produces’ one output pebble ‘ $\bullet$ ’. Thereby we achieve that the TRS  $R_M^\bullet$  is top-terminating even if  $M$  does not terminate.

**Definition 6.1.** For every Turing machine  $M = \langle Q, \Gamma, q_0, \delta \rangle$  we define the TRS  $R_M^\bullet$  as follows. The signature  $\Sigma = Q \cup \Gamma \cup \{\triangleright, \bullet, \top\}$  where  $\bullet$  is a unary symbol,  $\top$  is a constant symbol, and the rewrite rules of  $R_M^\bullet$  are:

$$\ell \rightarrow \bullet(r) \qquad \text{for every } \ell \rightarrow r \in R_M$$

and rules for rewriting to  $\top$  after successful termination:

$$\begin{aligned} q(x, 0(y)) &\rightarrow \top && \text{whenever } \delta(q, S) \text{ is undefined} \\ \bullet(\top) &\rightarrow \top. \end{aligned}$$

Then we obtain the following lemma characterizing  $>_M$  as defined in Definition [2.4](#).

**Lemma 6.2.** *For every Turing machine  $M = \langle Q, \Gamma, q_0, \delta \rangle$  and  $n, m \in \mathbb{N}$  we have  $n >_M m$  if and only if  $q_0(S^n, S^m) \rightarrow^*_{R_M^\bullet} \top$ .* □

Moreover we define an auxiliary TRS  $R_{\text{pickn}}$  for generating a random natural number  $n \in \mathbb{N}$  in the shape of a term  $S^n(0(\triangleright))$ :

**Definition 6.3.** We define the TRS  $R_{\text{pickn}}$  to consist of the following three rules:

$$\text{pickn} \rightarrow c(\text{pickn}) \qquad \text{pickn} \rightarrow \text{ok}(0(\triangleright)) \qquad c(\text{ok}(x)) \rightarrow \text{ok}(S(x)).$$

**Lemma 6.4.** *The TRS  $R_{\text{pickn}}$  has the following properties:*

- $\text{pickn} \rightarrow^* \text{ok}(S^n(0(\triangleright)))$  for every  $n \in \mathbb{N}$ , and
- whenever  $\text{pickn} \rightarrow^* \text{ok}(t)$  for some term  $t$  then  $t \equiv S^n(0(\triangleright))$  for some  $n \in \mathbb{N}$ .

Now we are ready to prove  $\Pi_1^1$ -completeness of dependency pair problems.

**Theorem 6.5.** *Both  $\text{SN}(t, R_{\text{top}}/S)$  and  $\text{SN}(R_{\text{top}}/S)$  are  $\Pi_1^1$ -complete.*

*Proof.* We prove  $\Pi_1^1$ -hardness even for the case where  $R$  and  $S$  coincide. We do this by using that the set  $\text{WF}$  is  $\Pi_1^1$ -complete, that is, checking well-foundedness of  $>_M$ . Let  $M$  be an arbitrary Turing machine. From  $M$  we construct a TRS  $S$  together with a term  $t$  such that:

$$\text{SN}(S_{\text{top}}/S) \iff \text{SN}(t, S_{\text{top}}/S) \iff >_M \text{ is well-founded}.$$

Let  $S$  consist of the rules of  $R_M^\bullet \uplus R_{\text{pickn}}$  together with:

$$\text{run}(\top, \text{ok}(x), \text{ok}(y)) \rightarrow \text{run}(q_0(x, y), \text{ok}(y), \text{pickn}), \tag{6}$$

and define  $t := \text{run}(\top, \text{pickn}, \text{pickn})$ .

As the implication from the first to the second item is trivial, we only have to prove (1)  $\text{SN}(t, S_{\text{top}}/S) \implies >_{\mathbf{M}}$  is well-founded and (2)  $>_{\mathbf{M}}$  is well-founded  $\implies \text{SN}(S_{\text{top}}/S)$ .

(1) Suppose  $\text{SN}(t, S_{\text{top}}/S)$  and assume there is an infinite descending  $>_{\mathbf{M}}$ -sequence:  $n_1 >_{\mathbf{M}} n_2 >_{\mathbf{M}} \dots$ . Then we have:

$$\begin{aligned}
 \text{run}(\mathbf{T}, \text{pickn}, \text{pickn}) &\rightarrow^* \text{run}(\mathbf{T}, \text{ok}(S^{n_1}(0(\triangleright))), \text{ok}(S^{n_2}(0(\triangleright)))) & (*) \\
 &\rightarrow_{S, \epsilon} \text{run}(q_0(S^{n_1}(0(\triangleright))), S^{n_2}(0(\triangleright)), \text{ok}(S^{n_2}(0(\triangleright))), \text{pickn}) \\
 &\rightarrow^* \text{run}(\mathbf{T}, \text{ok}(S^{n_2}(0(\triangleright))), \text{ok}(S^{n_3}(0(\triangleright)))) \\
 &\rightarrow_{S, \epsilon} \dots
 \end{aligned}$$

Note that  $q_0(S^{n_i}(0(\triangleright)), S^{n_{i+1}}(0(\triangleright))) \rightarrow^* \mathbf{T}$  (for all  $i \geq 1$ ) because  $\mathbf{M}$  computes the binary predicate  $>_{\mathbf{M}}$ . So we have an infinite reduction starting from  $t$ , contradicting  $\text{SN}(t, S_{\text{top}}/S)$ . So there is no infinite descending  $>_{\mathbf{M}}$ -sequence.

(2) Suppose that  $>_{\mathbf{M}}$  is well-founded and assume that  $\sigma$  is a rewrite sequence containing infinitely many root steps. Note that (G) is the only candidate for a rule which can be applied infinitely often at the root. Hence all terms in  $\sigma$  have the root symbol  $\text{run}$ . We consider the first three applications of (G) at the root in  $\sigma$ . After the first application the third argument of  $\text{run}$  is  $\text{pickn}$ . Therefore after the second application the second argument of  $\text{run}$  is a reduct of  $\text{pickn}$  and the third is  $\text{pickn}$ . Then before the third application we obtained a term  $t$  whose first argument is  $\mathbf{T}$ , and the second and the third argument are reducts of  $\text{pickn}$ . Observe from  $t$  on the rewrite sequence  $\sigma$  must be of the form as depicted above (E) (c.f. Lemma 6.4) for some  $n_1, n_2, \dots \in \mathbb{N}$ . Then for all  $i \geq 1$ :  $n_i >_{\mathbf{M}} n_{i+1}$  since  $q_0(S^{n_i}(0(\triangleright)), S^{n_{i+1}}(0(\triangleright))) \rightarrow^* \mathbf{T}$ . This contradicts well-foundedness of  $>_{\mathbf{M}}$ .

It remains to prove that both  $\text{SN}(R_{\text{top}}/S)$  and  $\text{SN}(t, R_{\text{top}}/S)$  are in  $\Pi_1^1$ . Let  $R$  and  $S$  be TRSs. Then  $\text{SN}(R_{\text{top}}/S)$  holds if and only if all  $\rightarrow_{R, \epsilon} \cup \rightarrow_S$  reductions contain only a finite number of  $\rightarrow_{R, \epsilon}$  steps. An infinite reduction can be encoded as a function  $\alpha : \mathbb{N} \rightarrow \mathbb{N}$  where  $\alpha(n)$  is the  $n$ -th term of the sequence. We can express the property as follows:

$$\begin{aligned}
 \text{SN}(R_{\text{top}}/S) &\iff \forall \alpha : \mathbb{N} \rightarrow \mathbb{N}. \\
 &((\forall n \in \mathbb{N}. \alpha(n) \text{ rewrites to } \alpha(n+1) \text{ via } \rightarrow_{R, \epsilon} \cup \rightarrow_S) \Rightarrow \\
 &\exists m_0 \in \mathbb{N}. \forall m \geq m_0. \neg(\alpha(m) \text{ rewrites to } \alpha(m+1) \text{ via } \rightarrow_{R, \epsilon})),
 \end{aligned}$$

containing one universal function quantifier in front of an arithmetic formula. Here the predicate ‘ $n$  rewrites to  $m$ ’ tacitly includes a check that both  $n$  and  $m$  indeed encode terms (which establishes no problem for a Turing machine). For the property  $\text{SN}(t, R_{\text{top}}/S)$  we simply add the condition  $t = \alpha(1)$  to restrict the quantification to such rewrite sequences  $\alpha$  that start with  $t$ . Hence  $\text{SN}(R_{\text{top}}/S)$  and  $\text{SN}(t, R_{\text{top}}/S)$  are  $\Pi_1^1$ -complete.  $\square$

By the same argument as in this proof we obtain

$$\text{SN}(R_{\text{top}}/S) \iff \text{SN}(t, R_{\text{top}}/S) \iff >_{\mathbf{M}} \text{ is well-founded}$$

for  $R$  consisting of the single rule  $\text{run}(\top, \text{ok}(x), \text{ok}(y)) \rightarrow \text{run}(q_0(x, y), \text{ok}(y), \text{pickn})$  and  $S$  consisting of the rules of  $R_M^\bullet \uplus R_{\text{pickn}}$ , again for  $t = \text{run}(\top, \text{pickn}, \text{pickn})$ . By considering  $\text{run}$  to be marked and all other symbols to be unmarked, we conclude  $\Pi_1^1$ -hardness and also  $\Pi_1^1$ -completeness for dependency pair problems  $\text{SN}(R_{\text{top}}/S)$  satisfying the standard requirements:

- the root symbols of both  $\ell$  and  $r$  are marked for every rule  $\ell \rightarrow r$  in  $R$ ;
- all other symbols in  $R$  and all symbols in  $S$  are unmarked.

We now sketch how this proof also implies  $\Pi_1^1$ -completeness of the property  $\text{SN}^\infty$  in infinitary rewriting, for its definition and basic observations see [9]. Since in Theorem 6.5 we proved  $\Pi_1^1$ -hardness even for the case where  $R$  and  $S$  coincide, we conclude that  $\text{SN}(S_{\text{top}}/S)$  is  $\Pi_1^1$ -complete. This property  $\text{SN}(S_{\text{top}}/S)$  states that every infinite  $S$ -reduction contains only finitely many root steps. This is the same as the property  $\text{SN}^\omega$  when restricting to finite terms; for the definition of  $\text{SN}^\omega$  see [14] (basically, it states that in any infinite reduction the position of the contracted redex moves to infinity). However, when extending to infinite terms it still holds that for the TRS  $S$  in the proof of Theorem 6.5 the only infinite  $S$ -reduction containing infinitely many root steps is of the shape given in that proof, only consisting of finite terms. So  $\text{SN}^\omega$  for all terms (finite and infinite) is  $\Pi_1^1$ -complete. It is well-known that for left-linear TRSs the properties  $\text{SN}^\omega$  and  $\text{SN}^\infty$  coincide, see e.g. [14]. Since the TRS  $S$  used in the proof of Theorem 6.5 is left-linear we conclude that the property  $\text{SN}^\infty$  for left-linear TRSs is  $\Pi_1^1$ -complete.

## 7 Dependency Pair Problems with Minimality Flag

A variant in the dependency pair approach is the dependency pair problem with minimality flag. Here in the infinite  $\rightarrow_{R,\epsilon} \cup \rightarrow_S$  reductions all terms are assumed to be  $S$ -terminating. This can be defined as follows. On the level of relations  $\rightarrow_1, \rightarrow_2$  we write  $\rightarrow_1 / \min \rightarrow_2 = (\rightarrow_2^* \cdot \rightarrow_1) \cap \rightarrow_{\text{SN}(\rightarrow_2)}$ , where the relation  $\rightarrow_{\text{SN}(\rightarrow_2)}$  is defined to consist of all pairs  $(x, y)$  for which  $x$  and  $y$  are  $\rightarrow_2$ -terminating. For TRSs  $R, S$  instead of  $\text{SN}(\rightarrow_{R,\epsilon} / \min \rightarrow_S)$  we shortly write  $\text{SN}(R_{\text{top}} / \min S)$ . In [5] this is called finiteness of the dependency pair problem  $(R, Q, S)$  with minimality flag; in our setting the middle TRS  $Q$  is empty. Again the motivation for this definition is in proving termination: from [2] we know

$$\text{SN}(\text{DP}(R)_{\text{top}} / \min R) \iff \text{SN}(R).$$

For  $\text{SN}(R_{\text{top}} / \min S)$  it is not clear how to define a single term variant, in particular for terms that are not  $S$ -terminating. In this section we prove that  $\text{SN}(R_{\text{top}} / \min S)$  is  $\Pi_2^0$ -complete. For doing so first we give some lemmas.

**Lemma 7.1.** *Let  $R, S$  be TRSs. Then  $\text{SN}(R_{\text{top}} / \min S)$  holds if and only if*

$$(\rightarrow_{R,\epsilon} \cup \rightarrow_S) \cap \rightarrow_{\text{SN}(\rightarrow_S)}$$

*is terminating.*

*Proof.* By definition  $\text{SN}(R_{\text{top}/\text{min}} S)$  is equivalent to termination of  $(\rightarrow_S^* \cdot \rightarrow_{R,\epsilon}) \cap \rightarrow_{\text{SN}(\rightarrow_S)}$ . Since

$$(\rightarrow_S^* \cdot \rightarrow_{R,\epsilon}) \cap \rightarrow_{\text{SN}(\rightarrow_S)} \subseteq ((\rightarrow_{R,\epsilon} \cup \rightarrow_S) \cap \rightarrow_{\text{SN}(\rightarrow_S)})^+,$$

the ‘if’-part of the lemma follows.

For the ‘only if’-part assume  $(\rightarrow_{R,\epsilon} \cup \rightarrow_S) \cap \rightarrow_{\text{SN}(\rightarrow_S)}$  admits an infinite reduction. If this reduction contains finitely many  $\rightarrow_{R,\epsilon}$ -steps, then this reduction ends in an infinite  $\rightarrow_S$ -reduction, contradicting the assumption that all terms in this reduction are  $S$ -terminating. So this reduction contains infinitely many  $\rightarrow_{R,\epsilon}$ -steps, hence can be written as an infinite  $(\rightarrow_S^* \cdot \rightarrow_{R,\epsilon}) \cap \rightarrow_{\text{SN}(\rightarrow_S)}$  reduction.  $\square$

**Lemma 7.2.** *Let  $R, S$  be TRSs. Then  $\text{SN}(R_{\text{top}/\text{min}} S)$  holds if and only if for every term  $t$  and every  $m \in \mathbb{N}$  there exists  $n \in \mathbb{N}$  such that*

*for every  $n$ -step  $(\rightarrow_{R,\epsilon} \cup \rightarrow_S)$ -reduction  $t = t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_n$  there exists  $i \in [0, n]$  such that  $t_i$  admits an  $m$ -step  $\rightarrow_S$ -reduction.*

*Proof.* Due to Lemma 7.1  $\text{SN}(R_{\text{top}/\text{min}} S)$  is equivalent to finiteness of all  $(\rightarrow_{R,\epsilon} \cup \rightarrow_S)$ -reductions only consisting of  $\rightarrow_S$ -terminating terms. Since  $(\rightarrow_{R,\epsilon} \cup \rightarrow_S)$  is finitely branching, this is equivalent to

for every term  $t$  there exists  $n \in \mathbb{N}$  such that no  $n$ -step  $(\rightarrow_{R,\epsilon} \cup \rightarrow_S)$ -reduction  $t = t_0 \rightarrow t_1 \rightarrow \dots \rightarrow t_n$  exists for which  $t_i$  is  $\rightarrow_S$ -terminating for every  $i \in [0, n]$ .

Since  $\rightarrow_S$  is finitely branching,  $\rightarrow_S$ -termination of  $t_i$  for every  $i \in [0, n]$  is equivalent to the existence of  $m \in \mathbb{N}$  such that no  $t_i$  admits an  $m$ -step  $\rightarrow_S$ -reduction. After removing double negations, this proves equivalence with the claim in the lemma.  $\square$

**Theorem 7.3.** *The property  $\text{SN}(R_{\text{top}/\text{min}} S)$  for given TRSs  $R, S$  is  $\Pi_2^0$ -complete.*

*Proof.*  $\text{SN}(R)$  is  $\Pi_2^0$ -complete and  $\text{SN}(R)$  is equivalent to  $\text{SN}(\text{DP}(R)_{\text{top}/\text{min}} R)$ , so  $\text{SN}(R_{\text{top}/\text{min}} S)$  is  $\Pi_2^0$ -hard. That  $\text{SN}(R_{\text{top}/\text{min}} S)$  is in  $\Pi_2^0$  follows from Lemma 7.2; note that the body of the claim in Lemma 7.2 is recursive.  $\square$

## 8 Conclusion and Future Work

In this paper we have analysed the proof theoretic complexity, in terms of the arithmetic and analytical hierarchy, of standard properties in term rewriting. Extending the work of [12], we observed that not all properties are  $\Pi_2^0$ -complete. In particular, weak confluence turns out to be  $\Sigma_1^0$ -complete, which is a lower class, while dependency pair problems are  $\Pi_1^1$ -complete, being a much higher class. In future work, we will also further study the place in the analytic hierarchy of properties of infinitary rewriting like  $\text{WN}^\infty$ .

## References

1. Aoto, T., Toyama, Y.: Persistency of confluence. *J. Universal Computer Science* 3, 1134–1147 (1997)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. *Theoretical Computer Science* 236, 133–178 (2000)
3. Endrullis, J., Geuvers, H., Zantema, H.: Degrees of Undecidability of TRS Properties (2009), <http://arxiv.org/abs/0902.4723>
4. Giesl, J., Swiderski, S., Schneider-Kamp, P., Thiemann, R.: Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages (invited lecture). In: Pfenning, F. (ed.) *RTA 2006*. LNCS, vol. 4098, pp. 297–312. Springer, Heidelberg (2006)
5. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Baader, F., Voronkov, A. (eds.) *LPAR 2004*. LNCS (LNAI), vol. 3452, pp. 301–331. Springer, Heidelberg (2005)
6. Hinman, P.G.: *Recursion-Theoretic Hierarchies*. Springer, Heidelberg (1978)
7. Huet, G., Lankford, D.: On the uniform halting problem for term rewriting systems. Technical Report 283, IRIA, France, Mars (1978)
8. Rogers Jr., H.: *Theory of recursive functions and effective computability*. McGraw-Hill, New York (1967)
9. Klop, J.W., de Vrijer, R.C.: Infinitary normalization. In: *We Will Show Them! Essays in Honour of Dov Gabbay*, vol. 2, pp. 169–192. College Publications (2005)
10. Klop, J.W.: Term rewriting systems. In: Abramsky, S., Gabbay, M.D., Maibaum, S.E. (eds.) *Handbook of Logic in Computer Science*, vol. 2, pp. 1–116. Oxford University Press, Inc., Oxford (1992)
11. Shoenfield, J.R.: *Mathematical Logic*. Association for Symbolic Logic, by A.K. Peters (1967)
12. Simonsen, J.G.: The  $\Pi_2^0$ -Completeness of Most of the Properties of Rewriting Systems You Care About (and Productivity). In: Treinen, R. (ed.) *RTA 2009*. LNCS, vol. 5595, pp. 335–349. Springer, Heidelberg (2009)
13. *Terese: Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
14. Zantema, H.: Normalization of infinite terms. In: Voronkov, A. (ed.) *RTA 2008*. LNCS, vol. 5117, pp. 441–455. Springer, Heidelberg (2008)

# Upper Bounds on Stream I/O Using Semantic Interpretations

Marco Gaboardi<sup>1</sup> and Romain Péchoux<sup>2,\*</sup>

<sup>1</sup> Dipartimento di Informatica, Università di Torino  
gaboardi@di.unito.it

<sup>2</sup> Computer Science Department, Trinity College, Dublin  
pechouxr@tcd.ie

**Abstract.** This paper extends for the first time semantic interpretation tools to infinite data in order to ensure Input/Output upper bounds on first order Haskell like programs on streams. By I/O upper bounds, we mean temporal relations between the number of reads performed on the input stream elements and the number of output elements produced. We study several I/O upper bounds properties that are of both theoretical and practical interest in order to avoid memory overflows.

## 1 Introduction

Interpretations are a well-established verification tool for proving properties of first order functional programs, term rewriting systems and imperative programs.

In the mid-seventies, Manna and Ness [1] and Lankford [2] introduced polynomial interpretations as a tool to prove the termination of term rewriting systems. The introduction of abstract interpretations [3] has strongly influenced the development of program verification and static analysis techniques. From their introduction, interpretations have been studied with hundreds of variations.

One variation of interest is the notion of quasi-interpretation [4]. It consists in a polynomial interpretation with relaxed constraints (large inequalities, functions over real numbers). Consequently, it no longer applies to termination problems (since well-foundedness is lost) but it allows us to study program complexity in an elegant way. Indeed, the quasi-interpretation of a first order functional program provides an upper bound on the size of the output values in the input size.

The theory of quasi-interpretations has led by now to many theoretical developments [4], for example, characterizations of the classes of functions computable in polynomial time and polynomial space. Moreover, the decidability of finding a quasi-interpretation of a given program has been shown for some restricted class of polynomials [5,6]. This suggests that quasi-interpretations can be interestingly exploited also in practical developments.

Quasi-interpretations have been generalized to sup-interpretations which are intensionally more powerful [7], i.e. sup-interpretations capture the complexity

---

\* The financial support of Science Foundation Ireland and COMPLICE project, ANR BLANC, is gratefully acknowledged.

of strictly more programs than quasi-interpretations do. This notion has led to a characterization of the  $NC^k$  complexity classes [8] which is a complementary approach to characterizations using function algebra presented in [9].

A new theoretical issue is whether interpretations can be used in order to infer resource properties on programs computing over infinite data. Here we approach this problem by considering lazy programs over stream data. Size upper bounds on stream data are meaningless. However, other interesting measures can be considered, e.g. size of stream elements and length of finite parts of streams. Here we consider some I/O properties with regard to such kind of measures. In particular, we would like to be able to obtain relations between the input reads and the output writes of a given program, where a read (or write) corresponds to the computation of a stream element in a function argument (resp. a stream element of the result). In this paper we identify three stream I/O upper bounds properties and we study criteria using interpretations in order to ensure them.

The first criterion, named Length Based I/O Upper Bound (LBUB), ensures that the number of output writes (the output length) is bounded by some function in the number of input reads. This criterion is respected by programs that need to read a finite amount of the input in order to produce a bounded amount of the output. The second criterion, named Size Based I/O Upper Bound (SBUB), ensures that the number of output writes is bounded by some function in the input reads size. It extends the previous criterion to programs where the output writes not only depend on the input structure but also on the input values. Finally, the last criterion, named Synchrony Upper Bound (SUB), ensures upper bounds on the output writes size depending on the input reads size in a synchronous framework, i.e. when the stream functions write exactly one element for one read performed.

The above criteria are interesting since they ensure upper bound properties corresponding to synchrony and asynchrony relations between program I/O. Moreover, besides the particular criteria studied, this work shows that semantic interpretation can be fruitfully exploited in studying programs dealing with infinite data types. Furthermore, we carry out the treatment of stream properties in a purely operational way. This shows that semantic interpretation are suitable for the usual equational reasoning on functional programs. From these conclusions we aim our work to be a new methodology in the study of stream functional languages properties.

**Related works.** Most of the works about stream properties considered stream definability and productivity, a notion dating back to [10]. Several techniques have been developed in order to ensure productivity, e.g. syntactical [10,11,12], data-flows analysis [13,14], type-based [15,16,17,18]. Some of these techniques can be adapted to prove different properties of programs working on streams, e.g. in [17], the authors give different hints on how to use sized types to prove some kind of buffering properties. Unfortunately an extensive treatment using these techniques to prove other properties of programs working on streams is lacking.

**Outline of the paper.** In Section 2, we describe the considered first order stream language and its lazy semantic. In Section 3 we define the semantic interpretations and their basic properties. Then, in Section 4, we introduce the considered properties and criteria to ensure them. Finally, in the last section we conclude.

## 2 Preliminaries

### 2.1 The sHask Language

We consider a first order Haskell-like language, named **sHask**. Let  $\mathcal{X}$ ,  $\mathcal{C}$  and  $\mathcal{F}$  be disjoint sets denoting respectively the set of *variables*, the set of *constructor symbols* and the set of *function symbols*. A **sHask** program is composed of a set of definitions described by the grammar in Table 1, where  $\mathbf{c} \in \mathcal{C}$ ,  $\mathbf{x} \in \mathcal{X}$ ,  $\mathbf{f} \in \mathcal{F}$ . We use the identifier  $\mathbf{t}$  to denote a symbol in  $\mathcal{C} \cup \mathcal{F}$ . Moreover the notation  $\overline{\mathbf{e}}$ , for some identifier  $\mathbf{e}$ , is a short-hand for the sequence  $\mathbf{e}_1, \dots, \mathbf{e}_n$ . As usual, application associates to the left, i.e.  $\mathbf{t} \mathbf{e}_1 \cdots \mathbf{e}_n$  corresponds to the expression  $((\mathbf{t} \mathbf{e}_1) \cdots) \mathbf{e}_n$ . In the sequel we will use the notation  $\mathbf{t} \overline{\mathbf{e}}$  as a short for the application  $\mathbf{t} \mathbf{e}_1 \cdots \mathbf{e}_n$ . The language **sHask** includes a **Case** operator to carry out pattern matching and first order function definitions. All the standard algebraic data types can be considered. Nevertheless, to be more concrete, in what follows we will consider as example three standard data types: numerals, lists and pairs. Analogously to Haskell, we denote by  $0$  and postfix  $+1$  the constructors for numerals, by **nil** and infix  $:$  the constructors for lists and by  $(-, -)$  the constructor for pairs.

Table 1. sHask syntax

$\mathbf{p} ::= \mathbf{x} \mid \mathbf{c} \mathbf{p}_1 \cdots \mathbf{p}_n$	(Patterns)
$\mathbf{e} ::= \mathbf{x} \mid \mathbf{t} \mathbf{e}_1 \cdots \mathbf{e}_n \mid \mathbf{Case} \overline{\mathbf{e}} \text{ of } \overline{\mathbf{p}}_1 \rightarrow \mathbf{e}_1, \dots, \overline{\mathbf{p}}_m \rightarrow \mathbf{e}_m$	(Expressions)
$\mathbf{v} ::= \mathbf{c} \mathbf{e}_1 \cdots \mathbf{e}_n$	(Values)
$\mathbf{d} ::= \mathbf{f} \mathbf{x}_1 \cdots \mathbf{x}_n = \mathbf{e}$	(Definitions)

Between the constants in  $\mathcal{C}$  we distinguish a special *error* symbol **Err** of arity 0 which corresponds to pattern matching failure. In particular, **Err** is treated as the other constructors, so for example pattern matching is allowed on it. The set of **Values** contains the usual lazy values, i.e. expressions with a constructor as the outermost symbol.

In order to simplify our framework, we will put some syntactical restrictions on the shape of the considered programs. We restrict our study to outermost non nested case definitions, this means that no **Case** appears in the  $\mathbf{e}_1, \dots, \mathbf{e}_m$  of a definition of the shape  $\mathbf{f} \overline{\mathbf{x}} = \mathbf{Case} \overline{\mathbf{e}} \text{ of } \overline{\mathbf{p}}_1 \rightarrow \mathbf{e}_1, \dots, \overline{\mathbf{p}}_m \rightarrow \mathbf{e}_m$  and we suppose that the function arguments and case arguments are the same, i.e.  $\overline{\mathbf{x}} = \overline{\mathbf{e}}$ .



The goal of this restriction is to simplify the considered framework. We claim that it is not a severe restriction since every program can be easily transformed in an equivalent one respecting this convention.

Finally, we suppose that all the free variables contained in the expression  $e_i$  of a case expression appear in the patterns  $\overline{p}_i$ , that no variable occurs twice in  $\overline{p}_i$  and that patterns are non-overlapping. It entails that programs are confluent [19].

**Haskell Syntactic Sugar.** In the sequel we use the Haskell-like programming style. An expression of the shape  $f \overline{x} = \text{Case } \overline{x} \text{ of } \overline{p}_1 \rightarrow e_1, \dots, \overline{p}_k \rightarrow e_k$  will be written as a set of definitions  $f \overline{p}_1 = e_1, \dots, f \overline{p}_k = e_k$ . Moreover, we adopt the standard Haskell convention for the parenthesis, e.g. we use  $f (x + 1) 0$  to denote  $((f(x + 1))0)$ .

## 2.2 sHask Type System

Similarly to Haskell, we are interested only in well typed expressions. For simplicity, we consider only programs dealing with lists that do not contain other lists and we assure this property by a typing restriction similar to the one of [18].

**Definition 1.** *The basic and value types are defined by the following grammar:*

$$\begin{aligned} \sigma &::= \alpha \mid \text{Nat} \mid \sigma \times \sigma && \text{(basic types)} \\ A &::= a \mid \sigma \mid A \times A \mid [\sigma] && \text{(value types)} \end{aligned}$$

where  $\alpha$  is a basic type variable,  $a$  is a value type variable,  $\text{Nat}$  is a constant type representing natural numbers,  $\times$  and  $[\ ]$  are type constructors. The set of types contains elements of the shape  $A_1 \rightarrow (\dots \rightarrow (A_n \rightarrow A))$ , for every  $n \geq 0$ .

Notice that the above definition can be extended to standard algebraic data types. In the sequel, we use  $\sigma, \tau$  to denote basic types and  $A, B$  to denote value types. As in Haskell, there is restricted polymorphism, i.e. a basic type variable  $\alpha$  and a value type variable  $a$  represent every basic type and respectively every value type. As usual,  $\rightarrow$  associates to the right, i.e. the notation  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow A$  corresponds to the type  $A_1 \rightarrow (\dots \rightarrow (A_n \rightarrow A))$ . Moreover, for notational convenience, we will use  $\overrightarrow{A} \rightarrow B$  as an abbreviation for  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$  throughout the paper.

In what follows, we will be particularly interested in studying expressions of type  $[\sigma]$ , for some  $\sigma$ , i.e. the type of finite and infinite lists over  $\sigma$ , in order to study stream properties. Every function and constructor symbol  $\mathfrak{t}$  of arity  $n$  come equipped with a type  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow A$ . Well typed symbols, patterns and expressions are defined using the type system in Table 2. Note that the symbol **Err** can be typed with every value type  $A$  in order to get type preservation in the evaluation mechanism. Moreover, it is worth noting that the type system, in order to allow only first order function definitions, assigns types to constant and function symbols, but only value types to expressions.

As usual, we use  $::$  to denote typing judgments, e.g.  $0 :: \text{Nat}$  denotes the fact that  $0$  has type  $\text{Nat}$ . A well typed definition is a function definition where we can assign the same value type  $A$  both to its left-hand and right-hand sides.

**Table 2.** sHask type system

---

$\frac{}{x :: A}$ (Var)	$\frac{\bar{e} :: \bar{A} \quad \bar{p}_1 :: \bar{A} \quad \cdots \quad \bar{p}_m :: \bar{A} \quad e_1 :: A \quad \cdots \quad e_m :: A}{\text{Case } \bar{e} \text{ of } \bar{p}_1 \rightarrow e_1, \dots, \bar{p}_m \rightarrow e_m :: A}$ (Case)
$\frac{}{t :: A_1 \rightarrow \cdots \rightarrow A_n \rightarrow A}$ (Tb)	$\frac{t :: A_1 \rightarrow \cdots \rightarrow A_n \rightarrow A \quad e_1 :: A_1 \quad \cdots \quad e_n :: A_n}{t \ e_1 \ \cdots \ e_n :: A}$ (Ts)

---

**Stream terminology.** In this work, we are specifically interested in studying stream properties. Since both finite lists over  $\sigma$  and streams over  $\sigma$  can be typed with type  $[\sigma]$ , we pay attention to particular classes of function working on  $[\sigma]$ , for some  $\sigma$ . Following the terminology of [14], a function symbol  $f$  is called a *stream function* if it is a symbol of type  $f :: [\sigma] \rightarrow \vec{\tau} \rightarrow [\sigma]$ .

*Example 1.* Consider the following programs:

merge ::  $[\alpha] \rightarrow [\alpha] \rightarrow [\alpha \times \alpha]$       nat ::  $\text{Nat} \rightarrow [\text{Nat}]$   
 merge  $(x : xs) (y : ys) = (x, y) : (\text{merge } xs \ ys)$       nat  $x = x : (\text{nat } (x + 1))$

merge and nat are two examples of stream functions

### 2.3 sHask Lazy Operational Semantics

We define a lazy operational semantics for the sHask language. The lazy semantics we give is an adaptation of the one in [20] to our first order Haskell-like

**Table 3.** sHask lazy operational semantics

---

$\frac{c \in \mathcal{C}}{c \ e_1 \ \cdots \ e_n \Downarrow c \ e_1 \ \cdots \ e_n}$ (val)	$\frac{e\{e_1/x_1, \dots, e_n/x_n\} \Downarrow v \quad f \ x_1 \ \cdots \ x_n = e}{f \ e_1 \ \cdots \ e_n \Downarrow v}$ (fun)
$\frac{\text{Case } e^1 \text{ of } p_1^1 \rightarrow \dots \rightarrow \text{Case } e^m \text{ of } p_1^m \rightarrow d_1 \Downarrow v \quad v \neq \mathbf{Err}}{\text{Case } \bar{e} \text{ of } \bar{p}_1 \rightarrow d_1, \dots, \bar{p}_n \rightarrow d_n \Downarrow v}$ (cb)	
$\frac{\text{Case } e^1 \text{ of } p_1^1 \rightarrow \dots \rightarrow \text{Case } e^m \text{ of } p_1^m \rightarrow d_1 \Downarrow \mathbf{Err} \quad \text{Case } \bar{e} \text{ of } \bar{p}_2 \rightarrow d_2, \dots, \bar{p}_n \rightarrow d_n \Downarrow v}{\text{Case } \bar{e} \text{ of } \bar{p}_1 \rightarrow d_1, \dots, \bar{p}_n \rightarrow d_n \Downarrow v}$ (c)	
$\frac{e \Downarrow c \ e_1 \ \cdots \ e_n \quad \text{Case } e_1 \text{ of } p_1 \rightarrow \dots \rightarrow \text{Case } e_n \text{ of } p_n \rightarrow d \Downarrow v}{\text{Case } e \text{ of } c \ p_1 \ \cdots \ p_n \rightarrow d \Downarrow v}$ (pm)	
$\frac{e \Downarrow v \quad v \neq c \ e_1 \ \cdots \ e_n}{\text{Case } e \text{ of } c \ p_1 \ \cdots \ p_n \rightarrow d \Downarrow \mathbf{Err}}$ (pm <sub>e</sub> )	$\frac{e'\{e/x\} \Downarrow v}{\text{Case } e \text{ of } x \rightarrow e' \Downarrow v}$ (pm <sub>b</sub> )

---

language, where we do not consider sharing for simplicity. The semantics is defined by the rules of Table 3.

The computational domain is the set of **Values**. **Values** are particular expressions with a constructor symbol at the outermost position. Note that in particular **Err** is a value corresponding to pattern matching errors. As usual in lazy semantics, the evaluation does not explore the entire expression and stops once the requested information is found. The intended meaning of the notation  $e \Downarrow v$  is that the expression  $e$  *evaluates* to the value  $v \in \mathbf{Values}$ .

*Example 2.* Consider again the program defined in Example 1. It is easy to verify that:  $\mathbf{nat} \ 0 \Downarrow 0 : (\mathbf{nat} \ (0 + 1))$  and  $\mathbf{merge} \ (\mathbf{nat} \ 0) \ \mathbf{nil} \Downarrow \mathbf{Err}$ .

### 2.4 Preliminary Notions

We are interested in studying stream properties by operational finitary means, for this purpose, we introduce some useful programs and notions.

First, we define the usual Haskell take and indexing programs **take** and **!!** which return the first  $n$  elements of a list and the  $n$ -th element of a list, respectively. As in Haskell, we use infix notation for **!!**.

$$\begin{array}{ll} \mathbf{take} \ :: \ \mathbf{Nat} \rightarrow [\alpha] \rightarrow [\alpha] & \mathbf{!!} \ :: \ [\alpha] \rightarrow \mathbf{Nat} \rightarrow \alpha \\ \mathbf{take} \ 0 \quad \mathbf{s} \quad = \quad \mathbf{nil} & (\mathbf{x} : \mathbf{xs}) \mathbf{!!} \ 0 \quad = \quad \mathbf{x} \\ \mathbf{take} \ (\mathbf{x} + 1) \ \mathbf{nil} \quad = \quad \mathbf{nil} & (\mathbf{x} : \mathbf{xs}) \mathbf{!!} \ (\mathbf{y} + 1) = \mathbf{xs} \mathbf{!!} \ \mathbf{y} \\ \mathbf{take} \ (\mathbf{x} + 1) \ (\mathbf{y} : \mathbf{ys}) = \mathbf{y} : (\mathbf{take} \ \mathbf{x} \ \mathbf{ys}) \end{array}$$

Second, we define a program **eval** that forces the (possibly diverging) full evaluation of expressions to fully evaluated values, i.e. values with no function symbols. We define **eval** for every value type **A** as:

$$\begin{array}{l} \mathbf{eval} \ :: \ \mathbf{A} \rightarrow \mathbf{A} \\ \mathbf{eval} \ (\mathbf{c} \ e_1 \ \dots \ e_n) = \hat{\mathbf{C}} \ (\mathbf{eval} \ e_1) \ \dots \ (\mathbf{eval} \ e_n) \end{array}$$

where  $\hat{\mathbf{C}}$  is a program representing the *strict* version of the primitive constructor  $\mathbf{c}$ . For example in the case where  $\mathbf{c}$  is  $+ 1$  we can define  $\hat{\mathbf{C}}$  as the program  $\mathbf{succ} \ :: \ \mathbf{Nat} \rightarrow \mathbf{Nat}$  defined by:

$$\begin{array}{l} \mathbf{succ} \ 0 \quad = \ 0 + 1 \\ \mathbf{succ} \ (\mathbf{x} + 1) = (\mathbf{x} + 1) + 1 \end{array}$$

A set of fully evaluated values of particular interest is the set  $\mathbf{N} = \{\underline{n} \mid \underline{n} = \underbrace{((\dots(0 + 1)\dots) + 1)}_{n \text{ times}}\}$  and  $\underline{n} \ :: \ \mathbf{Nat}\}$  of *canonical numerals*.

Then, we define a program **lg** that returns the number of elements in a finite partial list:

$$\begin{array}{l} \mathbf{lg} \ :: \ [\alpha] \rightarrow \mathbf{Nat} \\ \mathbf{lg} \ \mathbf{nil} \quad = \quad \underline{0} \\ \mathbf{lg} \ \mathbf{Err} \quad = \quad \underline{0} \\ \mathbf{lg} \ (\mathbf{x} : \mathbf{xs}) = (\mathbf{lg} \ \mathbf{xs}) + 1 \end{array}$$

*Example 3.* In order to illustrate the behaviour of  $\text{lg}$  consider the expression  $((\text{succ } 0) : (\text{nil} !! 0))$ . We have  $\text{eval}(\text{lg } ((\text{succ } 0) : (\text{nil} !! 0))) \Downarrow \underline{1}$ .

Finally we introduce a notion of *size* for expressions:

**Definition 2 (Size).** *The size of an expression  $e$  is defined as*

$$\begin{aligned} |e| &= 0 && \text{if } e \text{ is a variable or a symbol of arity } 0 \\ |e| &= \sum_{i \in \{1, \dots, n\}} |e_i| + 1 && \text{if } e = \mathfrak{t} \ e_1 \ \dots \ e_n, \ \mathfrak{t} \in \mathcal{C} \cup \mathcal{F}. \end{aligned}$$

Note that for each  $\underline{n} \in \mathbb{N}$  we have  $|\underline{n}| = n$ . Throughout the paper,  $F(\bar{e})$  denotes the componentwise application of  $F$  to the sequence  $\bar{e}$ , i.e.  $F(e_1, \dots, e_n) = F(e_1), \dots, F(e_n)$ . For example, given a sequence  $\bar{s} = s_1, \dots, s_n$ , we will use the notation  $|\bar{s}|$  for  $|s_1|, \dots, |s_n|$ .

### 3 Interpretations

In this section, we introduce the interpretation terminology. The interpretations we consider are inspired by the notions of quasi-interpretation [4] and super-interpretation [7] and are used as a main tool in order to ensure stream properties. They basically consist in assignments over non negative real numbers following the terminology of [21]. Throughout the paper,  $\geq$  and  $>$  denote the natural ordering on real numbers and its restriction.

**Definition 3 (Assignment).** *An assignment of a symbol  $\mathfrak{t} \in \mathcal{F} \cup \mathcal{C}$  of arity  $n$  is a function  $(\mathfrak{t}) : (\mathbb{R}^+)^n \rightarrow \mathbb{R}^+$ . For each variable  $x \in \mathcal{X}$ , we define  $(x) = X$ , with  $X$  a fresh variable ranging over  $\mathbb{R}^+$ . A program assignment is an assignment  $(-)$  defined for each symbol of the program. An assignment is (weakly) monotonic if for any symbol  $\mathfrak{t}$ ,  $(\mathfrak{t})$  is an increasing (not necessarily strictly) function with respect to each variable, that is for every symbol  $\mathfrak{t}$  and all  $X_i, Y_i$  of  $\mathbb{R}^+$  such that  $X_i \geq Y_i$ , we have  $(\mathfrak{t})(\dots, X_i, \dots) \geq (\mathfrak{t})(\dots, Y_i, \dots)$ .*

Notice that assignments are not defined on the **Case** construct since we only apply assignments to expressions without **Case**.

An assignment  $(-)$  can be extended to expressions canonically. Given an expression  $\mathfrak{t} \ e_1 \ \dots \ e_n$  with  $m$  variables, its assignment is a function  $(\mathbb{R}^+)^m \rightarrow \mathbb{R}^+$  defined by:

$$(\mathfrak{t} \ e_1 \ \dots \ e_n) = (\mathfrak{t})(\langle e_1 \rangle, \dots, \langle e_n \rangle)$$

*Example 4.* The function  $(-)$  defined by  $(\text{merge})(U, V) = U + V$ ,  $((-, -))(U, V) = U + V + 1$  and  $(:)(X, XS) = X + XS + 1$  is a monotonic assignment of the program **merge** of example [1].

Now we define the notion of additive assignments which guarantees that the size of a fully evaluated value is bounded by its assignment.

**Definition 4 (Additive assignment).** An assignment of a symbol  $\mathbf{c}$  of arity  $n$  is additive if:

$$\begin{aligned} \llbracket \mathbf{c} \rrbracket (X_1, \dots, X_n) &= \sum_{i=1}^n X_i + \alpha_{\mathbf{c}}, \text{ with } \alpha_{\mathbf{c}} \geq 1 && \text{if } n > 0, \\ \llbracket \mathbf{c} \rrbracket &= 0 && \text{otherwise.} \end{aligned}$$

The assignment  $\llbracket - \rrbracket$  of a program is called additive assignment if each constructor symbol of  $\mathcal{C}$  has an additive assignment.

**Definition 5 (Interpretation).** A program admits an interpretation  $\llbracket - \rrbracket$  if  $\llbracket - \rrbracket$  is a monotonic assignment such that for each definition of the shape  $\mathbf{f} \vec{\mathbf{p}} = \mathbf{e}$  we have  $\llbracket \mathbf{f} \vec{\mathbf{p}} \rrbracket \geq \llbracket \mathbf{e} \rrbracket$ .

Notice that if  $\llbracket \mathbf{t} \rrbracket$  is a subterm function (i.e.  $\forall i \in \{1, n\} \llbracket \mathbf{t} \rrbracket (X_1, \dots, X_n) \geq X_i$ ), for every symbol  $\mathbf{t}$ , then the considered interpretation is called a quasi-interpretation in the literature (used for inferring upper bounds on values). Moreover, if  $\llbracket \mathbf{t} \rrbracket$  is a polynomial over natural numbers and the inequalities are strict then  $\llbracket - \rrbracket$  is called a polynomial interpretation (used for showing program termination).

*Example 5.* The assignment of example 4 is an additive interpretation of the program `merge`. Indeed, we have:

$$\begin{aligned} \llbracket \text{merge } (\mathbf{x} : \mathbf{x}\mathbf{s}) (\mathbf{y} : \mathbf{y}\mathbf{s}) \rrbracket &= \llbracket \text{merge} \rrbracket (\llbracket \mathbf{x} : \mathbf{x}\mathbf{s} \rrbracket, \llbracket \mathbf{y} : \mathbf{y}\mathbf{s} \rrbracket) && \text{By canonical extension} \\ &= \llbracket \mathbf{x} : \mathbf{x}\mathbf{s} \rrbracket + \llbracket \mathbf{y} : \mathbf{y}\mathbf{s} \rrbracket && \text{By definition of } \llbracket \text{merge} \rrbracket \\ &= \llbracket \mathbf{x} \rrbracket + \llbracket \mathbf{x}\mathbf{s} \rrbracket + \llbracket \mathbf{y} \rrbracket + \llbracket \mathbf{y}\mathbf{s} \rrbracket + 2 && \text{By definition of } \llbracket \cdot \rrbracket \\ &= \llbracket (\mathbf{x}, \mathbf{y}) : \text{merge } \mathbf{x}\mathbf{s} \mathbf{y}\mathbf{s} \rrbracket && \text{Using the same reasoning} \end{aligned}$$

Let  $\rightarrow$  be the rewrite relation induced by giving an orientation from left to right to the definitions and let  $\rightarrow^*$  be its transitive and reflexive closure. We start by showing some properties on monotonic assignments.

**Proposition 1.** Given a program admitting the interpretation  $\llbracket - \rrbracket$ , then for every closed expression  $\mathbf{e}$  such that  $\mathbf{e} \rightarrow^* \mathbf{d}$ , we have:  $\llbracket \mathbf{e} \rrbracket \geq \llbracket \mathbf{d} \rrbracket$

*Proof.* The proof is by induction on the derivation length [22]. □

**Corollary 1.** Given a program admitting the interpretation  $\llbracket - \rrbracket$ , then for every closed expression  $\mathbf{e}$  such that  $\mathbf{e} \Downarrow \mathbf{v}$ , we have:  $\llbracket \mathbf{e} \rrbracket \geq \llbracket \mathbf{v} \rrbracket$

*Proof.* The lazy semantics is just a particular rewrite strategy. □

**Corollary 2.** Given a program admitting the interpretation  $\llbracket - \rrbracket$ , then for every closed expression  $\mathbf{e}$  such that `eval`  $\mathbf{e} \Downarrow \mathbf{v}$ , we have:  $\llbracket \mathbf{e} \rrbracket \geq \llbracket \mathbf{v} \rrbracket$

*Proof.* By induction on the structure of expressions. □

It is important to relate the size of an expression and its interpretation.

**Lemma 1.** Given a program having an interpretation  $\llbracket - \rrbracket$  then there is a function  $G : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that for each expression  $\mathbf{e}$ :  $\llbracket \mathbf{e} \rrbracket \leq G(|\mathbf{e}|)$

*Proof.* By induction on the structure of expressions. □

## 4 Bounded I/O Properties and Criteria

In this section, we define distinct stream properties related to time and space and criteria using interpretations to ensure them. A naive approach would be to consider a time unit to be either a stream input read (the evaluation of a stream element in a function argument) or a stream output write (the evaluation of a stream element of the result). Since most of the interesting programs working on streams are non-terminating, this approach fails. In fact, we need a more concrete notion of what time should be. Consequently, by time we mean relations between input reads and output writes, that is the ability of a program to return a certain amount of elements in the output stream (that is to perform some number of writes) when fed with some input stream elements.

### 4.1 Length Based I/O Upper Bound (LBUB)

We focus on the relations that provide upper bounds on output writes. We here consider structural relations, that depend on the stream structure but not on the value of its elements. We point out an interesting property giving bounds on the number of generated outputs by a function in the length of the inputs. As already stressed, the complete evaluation of stream expressions does not terminate. So, in order to deal with streams by finitary means, we ask, inspired by the well known Bird and Wadler *Take Lemma* [23], the property to hold on all the finite fragments of the streams that produce a result.

**Definition 6.** A stream function  $\mathbf{f} :: \overrightarrow{[\sigma]} \rightarrow \overrightarrow{\tau} \rightarrow [\sigma]$  has a length based I/O upper bound if there is a function  $F : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that for every expression  $\mathbf{s}_i :: [\sigma_i]$  and for every expression  $\mathbf{e}_i :: \tau_i$ , we have that:

$$\forall \underline{n}_i \in \mathbb{N}, \text{ s.t. } \text{eval}(\text{lg}(\mathbf{f}(\overrightarrow{\text{take } \underline{n} \ \mathbf{s}}) \overrightarrow{\mathbf{e}})) \Downarrow \underline{m}, F(\max(|\underline{n}|, |\underline{e}|)) \geq |\underline{m}|$$

where  $\overrightarrow{\text{take } \underline{n} \ \mathbf{s}}$  is a short for  $(\text{take } \underline{n}_1 \ \mathbf{s}_1) \cdots (\text{take } \underline{n}_m \ \mathbf{s}_m)$ .

Let us illustrate the length based I/O upper bound property by an example:

*Example 6.* The function `merge` of example [1] has a length based I/O upper bound. Indeed, consider  $F(X) = X$ , given two finite lists  $\mathbf{s}, \mathbf{s}'$  of size  $n, n'$  such that  $n \leq n'$ , we know that  $\text{eval}(\text{lg}(\text{merge } \mathbf{s} \ \mathbf{s}'))$  evaluates to an expression  $\underline{m}$  such that  $m = n$ . Consequently, given two stream expressions  $\mathbf{e}$  and  $\mathbf{e}'$  such that  $\text{eval}(\text{take } \underline{n}' \ \mathbf{e}') \Downarrow \mathbf{s}'$  and  $\text{eval}(\text{take } \underline{n} \ \mathbf{e}) \Downarrow \mathbf{s}$ , we have:

$$F(\max(|\underline{n}|, |\underline{n}'|)) = F(\max(n, n')) = n' \geq n = |\underline{m}|$$

### 4.2 A Criterion for Length Based I/O Upper Bound

We here give a criterion ensuring that a given stream has a length based I/O upper bound. For simplicity, in the following sections, we suppose that the considered programs do not use the programs `lg` and `take`.

**Definition 7.** A program is LBUB if it admits an interpretation  $\llbracket - \rrbracket$  which satisfies  $\llbracket +1 \rrbracket(X) = X + 1$  and which is additive but on the constructor symbol : where  $\llbracket \cdot \rrbracket$  is defined by  $\llbracket \cdot \rrbracket(X, Y) = Y + 1$ .

We start by showing some basic properties of LBUB programs.

**Lemma 2.** Given a LBUB program, for every  $\underline{n} :: \text{Nat}$  we have  $\llbracket \underline{n} \rrbracket = \lfloor \underline{n} \rfloor$ .

*Proof.* By an easy induction on canonical numerals. □

**Lemma 3.** Given a LBUB program wrt the interpretation  $\llbracket - \rrbracket$ , the interpretation can be extended to the program  $\text{lg}$  by  $\llbracket \text{lg} \rrbracket(X) = X$ .

*Proof.* We check that the inequalities hold for every equation in the definition of  $\text{lg}$ . □

**Lemma 4.** Given a LBUB program wrt the interpretation  $\llbracket - \rrbracket$ , the interpretation can be extended to the program  $\text{take}$  by  $\llbracket \text{take} \rrbracket(N, L) = N$ .

*Proof.* We check that the inequalities hold for every equation in the definition of  $\text{take}$ . □

**Theorem 1.** If a program is LBUB then each stream function in it has a length based I/O upper bound.

*Proof.* Given a LBUB program, if  $\text{eval}(\text{lg}(\overrightarrow{\text{f}(\text{take } \underline{n} \text{ s}) \overrightarrow{\text{e}}})) \Downarrow \underline{m}$  then we know that  $\llbracket \text{lg}(\overrightarrow{\text{f}(\text{take } \underline{n} \text{ s}) \overrightarrow{\text{e}}}) \rrbracket \geq \lfloor \underline{m} \rfloor$ , by Corollary 2. By Lemma 3, we obtain that  $\llbracket \overrightarrow{\text{f}(\text{take } \underline{n} \text{ s}) \overrightarrow{\text{e}}} \rrbracket \geq \lfloor \underline{m} \rfloor$ . By Lemma 4, we know that  $\llbracket \overrightarrow{\text{f}(\text{take } \underline{n} \text{ s}) \overrightarrow{\text{e}}} \rrbracket = \llbracket \text{f} \rrbracket(\lfloor \underline{n} \rfloor, \llbracket \overrightarrow{\text{e}} \rrbracket)$ . Applying Lemma 2, we obtain  $\lfloor \underline{m} \rfloor = \lfloor \underline{m} \rfloor \leq \llbracket \text{f} \rrbracket(\lfloor \underline{n} \rfloor, \llbracket \overrightarrow{\text{e}} \rrbracket)$ . Finally, by Lemma 1, we know that there is a function  $G : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that  $\lfloor \underline{m} \rfloor \leq \llbracket \text{f} \rrbracket(\lfloor \underline{n} \rfloor, G(\llbracket \overrightarrow{\text{e}} \rrbracket))$ . □

*Example 7.* The  $\text{merge}$  program of example 1 admits the following additive interpretation  $\llbracket \text{merge} \rrbracket(X, Y) = \max(X, Y)$ ,  $\llbracket (-, -) \rrbracket(X, Y) = X + Y + 1$  together with  $\llbracket \cdot \rrbracket(X, Y) = Y + 1$ . Consequently, it is LBUB and, defining  $F(X) = \llbracket \text{merge} \rrbracket(X, X)$  we know that for any two finite lists  $s_1$  and  $s_2$  of length  $m_1$  and  $m_2$ , we have that if  $\text{eval}(\text{lg}(\text{merge } s_1 \text{ } s_2)) \Downarrow \underline{m}$  then  $F(\max(m_1, m_2)) \geq \lfloor \underline{m} \rfloor$  (i.e. we are able to exhibit a precise upper bound).

### 4.3 Size Based I/O Upper Bound (SBUB)

The previous criterion guarantees an interesting homogeneous property on stream data. However, a wide class of stream programs with bounded relations between input reads and output writes do not enjoy it. The reason is just that some programs do not only take into account the structure of the input it reads, but also its value. We here point out a generalization of the LBUB property by considering an upper bound depending on the size of the stream expressions.

*Example 8.* Consider the following motivating example:

<code>append</code> :: $[\alpha] \rightarrow [\alpha] \rightarrow [\alpha]$	<code>upto</code> :: $\text{Nat} \rightarrow [\text{Nat}]$
<code>append</code> (x : xs) ys = x : (append xs ys)	<code>upto</code> 0 = nil
<code>append</code> nil ys = ys	<code>upto</code> (x + 1) = (x + 1) : (upto x)
<code>extendupto</code> :: $[\text{Nat}] \rightarrow [\text{Nat}]$	
<code>extendupto</code> (x : xs) = append (upto x) (extendupto xs)	

The program `extendupto` has no length based I/O upper bound because for each number  $\underline{n}$  it reads, it performs  $n$  output writes (corresponding to a decreasing sequence from  $\underline{n}$  to  $\underline{1}$ ).

Now we introduce a new property dealing with size, that allows us to overcome this problem.

**Definition 8.** A stream function  $f :: \overrightarrow{[\sigma]} \rightarrow \overrightarrow{\tau} \rightarrow [\sigma]$  has a size based I/O upper bound if there is a function  $F : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that, for every stream expression  $s_i :: [\sigma_i]$  and for expression  $e_i :: \tau_i$ , we have that:

$$\forall \underline{n}_i \in \mathbb{N}, \text{ s.t. } \text{eval}(\text{lg}(f(\overrightarrow{\text{take } \underline{n}_i s_i} \overrightarrow{e_i}))) \Downarrow \underline{m}, F(\max(|\overline{s}|, |\overline{e}|)) \geq \underline{m}$$

where  $\overrightarrow{\text{take } \underline{n}_i s_i}$  is a short for  $(\text{take } \underline{n}_1 s_1) \cdots (\text{take } \underline{n}_m s_m)$ .

*Example 9.* Since the program of example [8](#) performs  $n$  output writes for each number  $\underline{n}$  it reads, it has a size based I/O upper bound.

Notice that this property informally generalizes the previous one, i.e. a size based I/O upper bounded program is also length based I/O program, just because size always bounds the length. But the length based criterion is still relevant for two reasons, first it is uniform (input reads and output writes are treated in the same way), second it provides more accurate upper bounds.

#### 4.4 A Criterion for Size Based I/O Upper Bound

We give a criterion ensuring that a stream has a size based I/O upper bound.

**Definition 9.** A program is SBUB if it admits an additive interpretation  $(|-)$  such that  $(+1)(X) = X + 1$  and  $(:)(X, Y) = X + Y + 1$ .

**Lemma 5.** Given a SBUB program wrt the interpretation  $(|-)$ , the interpretation can be extended to the program `lg` by  $(\text{lg})(X) = X$ .

*Proof.* We check that the inequalities hold for every definition of `lg`. □

**Lemma 6.** Given a SBUB program wrt the interpretation  $(|-)$ , the interpretation can be extended to the program `take` by  $(\text{take})(N, L) = L$ .

*Proof.* We check that the inequalities hold for every definition of `take`. □



**Theorem 2.** *If a program is SBUB then each stream function in it has a size based I/O upper bound.*

*Proof.* Given a SBUB program, then if  $\text{eval}(\text{lg}(\overrightarrow{\text{f}(\text{take } \underline{n} \text{ s})} \overrightarrow{\text{e}})) \Downarrow \underline{m}$ , for some stream function  $\text{f}$ , then  $\langle \text{lg}(\overrightarrow{\text{f}(\text{take } \underline{n} \text{ s})} \overrightarrow{\text{e}}) \rangle \geq \langle \underline{m} \rangle$ , by Corollary 2. By Lemma 5, we obtain that  $\langle \text{f}(\overrightarrow{\text{take } \underline{n} \text{ s})} \overrightarrow{\text{e}}) \rangle \geq \langle \underline{m} \rangle$ . By Lemma 6, we know that  $\langle \text{f}(\overrightarrow{\text{take } \underline{n} \text{ s})} \overrightarrow{\text{e}}) \rangle = \langle \text{f} \rangle(\langle \overline{\text{s}} \rangle, \langle \overline{\text{e}} \rangle)$ . Applying Lemma 2 (which still holds because the interpretation of  $+1$  remains unchanged), we obtain  $\langle \underline{m} \rangle = \langle \underline{m} \rangle \leq \langle \text{f} \rangle(\langle \overline{\text{s}} \rangle, \langle \overline{\text{e}} \rangle)$ . Finally, by Lemma 1, we know that there is a function  $G : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that  $\langle \underline{m} \rangle \leq \langle \text{f} \rangle(G(\langle \overline{\text{s}} \rangle), G(\langle \overline{\text{e}} \rangle))$ .  $\square$

*Example 10.* The program `extendupto` of example 8 admits the following additive interpretation  $\langle \text{nil} \rangle = \langle 0 \rangle = 0$ ,  $\langle \text{append} \rangle(X, Y) = X + Y$ ,  $\langle \text{upto} \rangle(X) = \langle \text{extendupto} \rangle(X) = 2 \times X^2$  together with  $\langle +1 \rangle(X) = X + 1$  and  $\langle ! \rangle(X, Y) = X + Y + 1$ . Consequently, it is SBUB and, defining  $F(X) = \langle \text{extendupto} \rangle(X)$  we know that for any finite list  $\text{s}$  of size  $n$ , if  $\text{eval}(\text{lg}(\text{extendupto } \text{s})) \Downarrow \underline{m}$  then  $F(n) \geq \langle \underline{m} \rangle$ , i.e. we are able to exhibit a precise upper bound. However notice that, as already mentioned, the bound is less tight than in previous criterion. The reason for that is just that size is an upper bound rougher than length.

### 4.5 Synchrony Upper Bound (SUB)

Sometimes we would like to be more precise about the computational complexity of the program. In this case a suitable property would be synchrony, i.e. a finite part of the input let produce a finite part of the output. Clearly not every stream enjoys this property. Synchrony between stream Input and Output is a non-trivial question that we have already tackled in the previous subsections by providing some upper bounds on the length of finite output stream parts. In this subsection, we consider the problem in a different way: we restrict ourselves to synchronous streams and we adapt the interpretation methodology in order to give upper bound on the output writes size with respect to input reads size. In the sequel it will be useful to have the following program:

```
1Ind :: Nat → [α] → [α]
1Ind x xs = (xs !! x) : nil
```

We start to define the meaning of synchrony between input stream reads and output stream writes:

**Definition 10.** *A stream function  $\text{f} :: [\sigma] \rightarrow \overrightarrow{\tau} \rightarrow [\sigma]$  is said to be of type “Read, Write” if for every expression  $\text{s}_i :: \sigma_i$ ,  $\text{e}_i :: \tau_i$  and for every  $\underline{n} \in \mathbb{N}$ :*

$$\text{If } \text{eval}(\text{1Ind } \underline{n} (\text{f } \overrightarrow{\text{s}} \overrightarrow{\text{e}})) \Downarrow \text{v} \text{ and } \text{eval}(\text{f}(\overrightarrow{\text{1Ind } \underline{n} \text{ s}}) \overrightarrow{\text{e}}) \Downarrow \text{v}' \text{ then } \text{v} = \text{v}'$$

This definition provides a one to one correspondence between input stream reads and output stream writes, because the stream function needs one input read in order to generate one output write and conversely, we know that it will not generate more than one output write (otherwise the two fully evaluated values cannot be matched).

*Example 11.* The following is an illustration of a “Read, Write” program:

```
sadd :: [Nat] → [Nat] → [Nat]
sadd (x : xs) (y : ys) = (add x y) : (sadd xs ys)

add :: Nat → Nat → Nat
add (x + 1) (y + 1) = ((add x y) + 1) + 1
add (x + 1) 0      = x + 1
add 0 (y + 1)     = y + 1
```

In this case, we would like to say that for each integer  $n$ , the size of the  $n$ -th output stream element is equal to the sum of the two  $n$ -th input stream elements.

**Definition 11.** A “Read, Write” stream function  $f :: \vec{[\sigma]} \rightarrow \vec{\tau} \rightarrow [\sigma]$  has a synchrony upper bound if there is a function  $F : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that for every expression  $s_i :: [\sigma_i]$ ,  $e_i :: \tau_i$  and for every  $\underline{n} \in \mathbb{N}$ :

$$\text{If } \text{eval}(s_i !! \underline{n}) \Downarrow w_i \text{ and } \text{eval}((f \vec{s} \vec{e}) !! \underline{n}) \Downarrow v \text{ then } F(\max(|\vec{w}|, |\vec{e}|)) \geq |v|$$

Another possibility would have been to consider a  $n$  to  $m$  correspondence between inputs and outputs. However such correspondences can be studied with slight changes over the one-one correspondence.

#### 4.6 A Criterion for Synchrony Upper Bound

We begin to put some syntactical restriction on the considered programs so that each stream function symbol is “Read, Write” with respect to this restriction.

**Definition 12.** A stream function  $f :: \vec{[\sigma]} \rightarrow \vec{\tau} \rightarrow [\sigma]$  is synchronously restricted if it can be written (and maybe extended) by definitions of the shape:

$$\begin{array}{l} f (x_1 : xs_1) \cdots (x_n : xs_n) \vec{p} = \text{hd} : (f xs_1 \cdots xs_n \vec{p}) \\ f \quad \text{nil} \quad \cdots \quad \text{nil} \quad \vec{p} = \quad \text{nil} \end{array}$$

where  $xs_1, \dots, xs_n$  do not appear in the expression  $\text{hd}$ .

Now we may show the following lemma.

**Lemma 7.** Every synchronously restricted function is “Read, Write”.

*Proof.* By induction on numerals. □

**Definition 13.** A program is SUB if it is synchronously restricted and admits an additive interpretation  $(|-)$  but on  $:$  where  $(:|)$  is defined by  $(:|)(X, Y) = X$ .

Fully evaluated values, i.e. values  $v$  containing only constructor symbols, have the following remarkable property.

**Lemma 8.** Given an additive assignment  $(|-)$ , for each fully evaluated value  $v$ :  $|v| \leq (|v|)$ .

*Proof.* By structural induction on fully evaluated values. □

**Theorem 3.** *If a program is SUB then each stream function in it admits a synchrony upper bound.*

*Proof.* By Lemma 7, we can restrict our attention to a “Read, Write” stream function  $f :: [\vec{\sigma}] \rightarrow \vec{\tau} \rightarrow [\sigma]$  s.t.  $\forall \underline{n} \in \mathbb{N}$ , we both have  $\text{eval}((f \vec{s} \vec{e}) !! \underline{n}) \Downarrow v$  and  $\text{eval}(s_i !! \underline{n}) \Downarrow w_i$ .

We firstly prove that  $\text{eval}(f(\overline{w : \text{nil}}) \vec{e}) \Downarrow v : \text{nil}$ . By assumption, we have  $\text{eval}((f \vec{s} \vec{e}) !! \underline{n}) \Downarrow v$ , so in particular we also have  $\text{eval}(((f \vec{s} \vec{e}) !! \underline{n}) : \text{nil}) \Downarrow v : \text{nil}$ . By definition of “Read, Write” function,  $\text{eval}((f((\vec{s} !! \underline{n}) : \text{nil}) \vec{e}) \Downarrow v : \text{nil})$  and since by assumption  $\text{eval}(s_i !! \underline{n}) \Downarrow w_i$  we can conclude  $\text{eval}(f(\overline{w : \text{nil}}) \vec{e}) \Downarrow v : \text{nil}$ . By Corollary 2, we have  $(f(\overline{w : \text{nil}}) \vec{e}) \geq (v : \text{nil})$ . By definition of SUB programs, we know that  $(\cdot)(X, Y) = X$  and, consequently,  $(f)((\overline{w : \text{nil}}), (\vec{e})) = (f)((\overline{w}), (\vec{e})) \geq (v : \text{nil}) = (v)$ . By Lemma 8, we have  $(f)((\overline{w}), (\vec{e})) \geq |v|$ . Finally, by Lemma 11, there exists a function  $G : \mathbb{R}^+ \rightarrow \mathbb{R}^+$  such that  $(f)(G(|\overline{w}|), G(|\vec{e}|)) \geq |v|$ . We conclude by taking  $F(X) = (f)(G(\overline{X}), G(\vec{X}))$ . □

*Example 12.* The program of example 11 is synchronously restricted (it can be extended in such a way) and admits the following additive interpretation  $(0) = 0$ ,  $(+1)(X) = X + 1$ ,  $(\text{add})(X, Y) = X + Y$ ,  $(\text{sadd})(X, Y) = X + Y$  and  $(\cdot)(X, Y) = X$ . Consequently, the program is SUB and admits a synchrony upper bound. Moreover, taking  $F(X) = (\text{sadd})(X, X)$ , we know that if the k-th input reads evaluate to numbers  $\underline{n}$  and  $\underline{m}$  then  $F(\max(|\underline{m}|, |\underline{n}|))$  is an upper bound on the k-th output size.

## 5 Conclusion

In this paper, we have applied interpretation methods for the first time to a lazy functional stream language, obtaining several criteria ensuring bound properties on the input read and output write elements of a program working on stream data. This shows that interpretations are a valid tool to well ensure stream function properties. Many interesting properties should be investigated, in particular memory leaks and overflows [17,18]. These questions are strongly related to the notions we have tackled in this paper. For example, consider the following:

<code>odd :: [α] → [α]</code>	<code>memleak :: [α] → [α × α]</code>
<code>odd (x : y : xs) = x : (odd xs)</code>	<code>memleak s = merge (odd s) s</code>

The evaluation of an expression `memleak s` leads to a memory leak. Indeed, `merge` reads one stream element on each of its arguments in order to output one element and `odd` needs to read two stream input elements of `s` in order to output one element whereas `s` just makes one output for one input read. Consequently, there is a factor 2 of asynchrony between the two computations on `s`. Which means that `merge` needs to read  $s_n$  and  $s_{2 \times n}$  (where  $s_i$  is the i-th element of

s) in order to compute the  $n$ -th output element. From a memory management perspective, it means that all the elements between  $s_n$  and  $s_{2 \times n}$  have to be stored, leading the memory to a leak. We think interpretations could help the programmer to prevent such "bad properties" of programs. Moreover, we think that interpretations can be also exploited in the study of stream definitions, in particular in the context of stream productivity, but we leave this subject for further researchs.

## References

1. Manna, Z., Ness, S.: On the termination of Markov algorithms. In: Third hawaii international conference on system science, pp. 789–792 (1970)
2. Lankford, D.: On proving term rewriting systems are noetherien. tech. rep (1979)
3. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of ACM POPL 1977, pp. 238–252 (1977)
4. Bonfante, G., Marion, J.Y., Moyon, J.Y.: Quasi-interpretations, a way to control resources. TCS (accepted)
5. Amadio, R.: Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae* 65(1-2) (2005)
6. Bonfante, G., Marion, J.Y., Moyon, J.Y., Péchoux, R.: Synthesis of quasi-interpretations. In: LCC 2005, LICS Workshop (2005), <http://hal.inria.fr/>
7. Marion, J.Y., Péchoux, R.: Sup-interpretations, a semantic method for static analysis of program resources. In: ACM TOCL 2009 (accepted, 2009), <http://tocl.acm.org/>
8. Marion, J.-Y., Péchoux, R.: A Characterization of NCK by First Order Functional Programs. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 136–147. Springer, Heidelberg (2008)
9. Bonfante, G., Kahle, R., Marion, J.-Y., Oitavem, I.: Recursion schemata for  $NC^k$ . In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 49–63. Springer, Heidelberg (2008)
10. Dijkstra, E.W.: On the productivity of recursive definitions. EWD749 (1980)
11. Sijtsma, B.: On the productivity of recursive list definitions. *ACM TOPLAS* 11(4), 633–649 (1989)
12. Coquand, T.: Infinite objects in type theory. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 62–78. Springer, Heidelberg (1994)
13. Wadge, W.: An extensional treatment of dataflow deadlock. *TCS* 13, 3–15 (1981)
14. Endrullis, J., Grabmayer, C., Hendriks, D., Ishihara, A., Klop, J.W.: Productivity of Stream Definitions. In: Csuhanj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 274–287. Springer, Heidelberg (2007)
15. Telford, A., Turner, D.: Ensuring streams flow. In: Johnson, M. (ed.) AMAST 1997. LNCS, vol. 1349, pp. 509–523. Springer, Heidelberg (1997)
16. Buchholz, W.: A term calculus for (co-) recursive definitions on streamlike data structures. *Annals of Pure and Applied Logic* 136(1-2), 75–90 (2005)
17. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: Proceedings of ACM POPL 1996, pp. 410–423 (1996)

18. Frankau, S., Mycroft, A.: Stream processing hardware from functional language specifications. In: Proceeding of IEEE HICSS-36 (2003)
19. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM* 27(4), 797–821 (1980)
20. Launchbury, J.: A natural semantics for lazy evaluation. In: Proceedings of POPL 1993, pp. 144–154 (1993)
21. Dershowitz, N.: Orderings for term-rewriting systems. *TCS* 17(3), 279–301 (1982)
22. Marion, J.Y., Péchoux, R.: Characterizations of polynomial complexity classes with a better intensionality. In: Proceedings ACM PPDP 2008, pp. 79–88 (2008)
23. Bird, R., Wadler, P.: *Introduction to Functional Programming*. Prentice-Hall, Englewood Cliffs (1988)

# Craig Interpolation for Linear Temporal Languages<sup>\*</sup>

Amélie Gheerbrant<sup>1</sup> and Balder ten Cate<sup>2</sup>

<sup>1</sup> ILLC, Universiteit van Amsterdam

`a.gheerbrant@uva.nl`

<sup>2</sup> INRIA and ENS Cachan

`balder.tencate@uva.nl`

**Abstract.** We study Craig interpolation for fragments and extensions of propositional linear temporal logic (PLTL). We consider various fragments of PLTL obtained by restricting the set of temporal connectives and, for each of these fragments, we identify its smallest extension that has Craig interpolation. Depending on the underlying set of temporal operators, this extension turns out to be one of the following three logics: the fragment of PLTL having only the Next operator; the extension of PLTL with a fixpoint operator  $\mu$  (known as linear time  $\mu$ -calculus); the fixpoint extension of the “Until-only” fragment of PLTL.

**Keywords:** Propositional Linear Temporal Logic, Craig Interpolation, Linear Time  $\mu$ -Calculus.

## 1 Introduction

Craig’s interpolation theorem in classical model theory dates back from the late fifties [7]. It states that if a first-order formula  $\phi$  (semantically) entails another first-order formula  $\psi$ , then there is an *interpolant* first-order formula  $\theta$ , such that every non-logical symbol in  $\theta$  occurs both in  $\phi$  and  $\psi$ ,  $\phi$  entails  $\theta$  and  $\theta$  entails  $\psi$ . The key idea of the Craig interpolation theorem is to relate different logical theories via their common non-logical vocabulary. In his original paper, Craig presents his work as a generalization of Beth’s definability theorem, according to which implicit (semantic) definability is equivalent to explicit (syntactic) definability. Indeed, Beth’s definability theorem follows from Craig’s interpolation theorem, but the latter is more general.

From the point of view of applications in computer science, interpolation is often a desirable property of a logic. For instance, in fields such as automatic reasoning and software development, interpolation is related to modularization [1, 10], a property which allows systems or specifications to be developed efficiently by first building component subsystems (or modules). Interpolation for

---

<sup>\*</sup> We are grateful to Alexandru Baltag for helpful comments and to Frank Wolter for first raising the question. The first author was supported by a GLoRiClass fellowship of the European Commission (Research Training Fellowship MEST-CT-2005-020841) and the second author by the Netherlands Organization for Scientific Research (NWO) grant 639.021.508 and by ERC Advanced Grant Webdam on Foundation of Web data management.

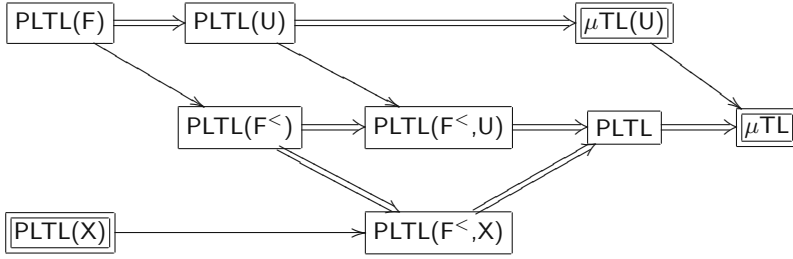


Fig. 1. Hierarchy of temporal languages

temporal logics is also an increasingly important topic. Temporal logics in general are widely used in systems and software verification, and interpolation has proven to be useful for building efficient model-checkers [8]. This is particularly true of a strong form of Craig interpolation known as uniform interpolation, which is quite rare in modal logic, but that the modal  $\mu$ -calculus satisfies (see [9]), whereas most temporal logics lack even Craig interpolation (see [17]).

We study Craig interpolation for fragments and extensions of propositional linear temporal logic (PLTL). We use the framework of [4] and work with a general notion of *abstract temporal language* which allows us to consider a general notion of extension of such languages. We consider different sets of temporal connectives and, for each, identify the smallest extension of the fragment of PLTL with these temporal connectives that has Craig interpolation. Depending on the set of temporal connectives, the resulting logic turns out to be either the fragment of PLTL with only the Next operator, or the extension of PLTL with a fixpoint operator  $\mu$  (known as linear time  $\mu$ -calculus), or the fixpoint extension of the fragment of PLTL with only the Until operator (which is the stutter-invariant fragment of linear time  $\mu$ -calculus). The diagram in Figure 1 summarises our results. A simple arrow linking two languages means that the first one is an extension of the second one and a double arrow means that, furthermore, every extension of the first one having Craig interpolation is an extension of the second one. Temporal languages with Craig interpolation (in fact, uniform interpolation) are represented in a double frame. Thus we have for instance that  $\mu\text{TL}(\text{U})$  is the least expressive extension of PLTL(F) with Craig interpolation.

**Outline of the paper:** In Section 2, we introduce a general notion of *abstract temporal language*. We then introduce PLTL, some of its natural fragments and its fixpoint extension known as linear time  $\mu$ -calculus ( $\mu\text{TL}$ ).

Section 3 contains some technical results that are used in subsequent sections. One of these relates projective definability in PLTL to definability in the fixpoint extension  $\mu\text{TL}$ . Another result relates in a similar way PLTL(U) and  $\mu\text{TL}(\text{U})$ . Along the way, we show that  $\mu\text{TL}(\text{U})$  is the stutter invariant fragment of  $\mu\text{TL}$ . Stutter-invariance is a property that is argued by some authors [16] to be natural and desirable for a temporal logic. Roughly, a temporal logic is stutter-invariant if it cannot detect the addition of identical copies of a state.

In Section 4, we give three positive interpolation results. Among the fragments of PLTL obtained by restricting the set of temporal operators, we show that only one (the “Next-only” fragment) has Craig interpolation. In fact, this fragment satisfies a stronger form of interpolation, called uniform interpolation. The logics  $\mu\text{TL}$  and  $\mu\text{TL}(\text{U})$  also have uniform interpolation.

Section 5 completes the picture by showing that  $\mu\text{TL}$  and  $\mu\text{TL}(\text{U})$  are the least extensions of  $\text{PLTL}(F)$  and  $\text{PLTL}(F^<)$ , respectively, with Craig interpolation.

## 2 Preliminaries

### 2.1 Abstract Temporal Languages

We will be dealing with a variety of temporal languages. They are all interpreted in structures consisting of a set of worlds (or, time points), a binary relation intuitively representing temporal precedence, and a valuation of proposition letters. In this section, we give an abstract model theoretic definition of temporal languages.

A *flow of time*, or *frame*, is a structure  $\mathcal{T} = (W, <)$ , where  $W$  is a non-empty set of worlds and  $<$  is a binary relation on  $W$ . We will focus here on  $\mathbf{T}_\omega$ , the class of linear orders of order type  $\omega$ , i.e., frames  $(D, <)$  that are isomorphic to  $(\mathbb{N}, <)$ , where  $\mathbb{N}$  is the set of natural numbers with the natural ordering. We will also freely use  $\leq$  to denote the reflexive closure of  $<$ .

By a *propositional signature* we mean a finite non-empty set of propositional letters  $\sigma = \{p_i \mid i \in I\}$ . A *pointed  $\sigma$ -structure* is a structure  $\mathfrak{M} = (\mathcal{T}, V, w)$  where  $\mathcal{T} = (W, R)$  is a frame,  $V : \sigma \rightarrow \wp(W)$  a valuation and  $w \in W$  a world. The class of all pointed  $\sigma$ -structures is denoted by  $\text{Str}[\sigma]$  and we call them  $\sigma$ -structures for short. Furthermore, for any class of frames  $\mathbf{T}$ ,  $\text{Str}_{\mathbf{T}}[\sigma]$  will denote the class of  $\sigma$ -structures of which the underlying frame belongs to  $\mathbf{T}$ . Let  $\sigma \subseteq \tau$  be propositional signatures. Given a  $\tau$ -structure  $\mathfrak{M} = (\mathcal{T}, V, w)$ , we define its  $\sigma$ -*reduct*  $\mathfrak{M} \upharpoonright \sigma$  as the  $\sigma$ -structure  $(\mathcal{T}, V \upharpoonright \sigma, w)$  where  $V \upharpoonright \sigma$  is the restriction of the valuation to the propositional letters in  $\sigma$ . We call  $\mathfrak{M}$  a  $\tau$ -*expansion* of  $\mathfrak{M} \upharpoonright \sigma$ . We also write  $K \upharpoonright \sigma$  for  $\{\mathfrak{M} \upharpoonright \sigma \mid \mathfrak{M} \in K\}$ . Let  $(\mathcal{T}, V, w)$  be a  $\sigma$ -structure and  $A \subseteq W$  a subset of its domain. By  $V[A/p]$ , we will refer to the valuation  $V$  extended with  $V(p) = A$  ( $p$  being a fresh proposition letter). We will refer to the corresponding  $\sigma \cup \{p\}$ -expansion of  $(\mathcal{T}, V, w)$  by  $(\mathcal{T}, V[A/p], w)$ .

**Definition 1 (Abstract temporal language).** An abstract temporal language (*temporal language* for short) is a pair  $\mathcal{L} = (\mathcal{L}, \models_{\mathcal{L}})$ , where  $\mathcal{L} : \sigma \mapsto \mathcal{L}[\sigma]$  is a map from propositional signatures to sets of objects that we call formulas and  $\models_{\mathcal{L}}$  is a relation between formulas and pointed structures satisfying the following conditions, for all propositional signatures  $\sigma, \tau$ :

1. **Expansion property.** If  $\sigma \subseteq \tau$  then  $\mathcal{L}[\sigma] \subseteq \mathcal{L}[\tau]$ . Furthermore, for all  $\phi \in \mathcal{L}[\sigma]$  and  $\mathfrak{M} \in \text{Str}[\tau]$ ,  $\mathfrak{M} \models_{\mathcal{L}} \phi$  iff  $\mathfrak{M} \upharpoonright \sigma \models_{\mathcal{L}} \phi$ . If  $\mathfrak{M} \in \text{Str}[\sigma]$  and  $\mathfrak{M} \models_{\mathcal{L}} \phi$ , then  $\phi \in \mathcal{L}[\sigma]$ .



2. **Closure under uniform substitution.** For all  $\psi \in \mathcal{L}[\sigma]$ ,  $p \notin \sigma$  and  $\phi \in \mathcal{L}[\sigma \cup \{p\}]$ , there is a formula of  $\mathcal{L}[\sigma]$ , which we will denote by  $\phi[p/\psi]$ , such that for every  $(\mathcal{T}, V, w) \in \text{Str}[\sigma]$  the following holds:

$$(\mathcal{T}, V, w) \models_{\mathcal{L}} \phi[p/\psi] \text{ iff } (\mathcal{T}, V', w) \models_{\mathcal{L}} \phi$$

where  $V' = V[\{w \mid (\mathcal{T}, V, w) \models_{\mathcal{L}} \psi\}/p]$ .

3. **Negation property.** For each  $\phi \in \mathcal{L}[\sigma]$  there is a formula of  $\mathcal{L}[\sigma]$ , which we will denote by  $\neg\phi$ , s.t. for all  $\mathfrak{M} \in \text{Str}[\sigma]$ ,  $\mathfrak{M} \models_{\mathcal{L}} \neg\phi$  iff  $\mathfrak{M} \not\models_{\mathcal{L}} \phi$ .

For any class of frames  $\mathbf{T}$ ,  $\models_{\mathcal{L}, \mathbf{T}}$  will denote the restriction of  $\models_{\mathcal{L}}$  to pointed structures based on  $\mathbf{T}$ . For  $\phi \in \mathcal{L}[\sigma]$ , we will use  $\text{Mod}^{\sigma}(\phi)$  as shorthand for  $\{\mathfrak{M} \in \text{Str}[\sigma] \mid \mathfrak{M} \models_{\mathcal{L}, \mathbf{T}} \phi\}$  and  $\text{Mod}_{\mathbf{T}}^{\sigma}(\phi)$  when restricting to a frame class  $\mathbf{T}$ . Whenever this is clear from the context, we will be omitting superscript and subscripts in  $\text{Mod}_{\mathbf{T}}^{\sigma}(\phi)$  and  $\models_{\mathcal{L}, \mathbf{T}}$ . We say that a class of pointed structures  $\mathbf{K} \subseteq \text{Str}_{\mathbf{T}}[\sigma]$  is *definable* in an abstract temporal language  $\mathcal{L}$  (relative to the frame class  $\mathbf{T}$ ) if there is a  $\mathcal{L}$ -formula  $\phi$  such that for every  $(\mathcal{T}, V, w) \in \text{Str}_{\mathbf{T}}[\sigma]$ ,  $(\mathcal{T}, V, w) \models \phi$  iff  $(\mathcal{T}, V, w) \in \mathbf{K}$ .

**Definition 2 (Extension of a temporal language).** Let  $\mathcal{L}_1 = (\mathcal{L}_1, \models_{\mathcal{L}_1})$ ,  $\mathcal{L}_2 = (\mathcal{L}_2, \models_{\mathcal{L}_2})$  be temporal languages.  $\mathcal{L}_2$  extends  $\mathcal{L}_1$  (notation:  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ ) if for all  $\sigma$ , for all  $\phi \in \mathcal{L}_1[\sigma]$ , there exists  $\phi^* \in \mathcal{L}_2[\sigma]$  such that  $\text{Mod}_{\sigma}(\phi) = \text{Mod}_{\sigma}(\phi^*)$ . Also, whenever  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , we say that  $\mathcal{L}_1$  is a *fragment* of  $\mathcal{L}_2$ . Whenever restricting attention to a frame class  $\mathbf{T}$  we write  $\mathcal{L}_1 \subseteq_{\mathbf{T}} \mathcal{L}_2$ .

The following notion is related to existential second-order quantification over propositional letters. Allowing such a form of quantification in a given temporal language indeed amounts to considering its projective classes. It is a classical notion in abstract modal theory and it will be useful in the context of  $\Delta$ -interpolation (see Definition 15).

**Definition 3 (Projective class).** Let  $\sigma$  be a propositional signature,  $\mathbf{T}$  a frame class and let  $K \subseteq \text{Str}_{\mathbf{T}}[\sigma]$ . Then  $K$  is a projective class of a temporal language  $\mathcal{L}$  relative to  $\mathbf{T}$  if there is a  $\phi \in \mathcal{L}[\tau]$  with  $\tau \supseteq \sigma$  a propositional signature, such that  $K = \text{Mod}(\phi) \upharpoonright \sigma$ .

**Lemma 1.** *Let  $\mathbf{T}$  be a frame class. If  $\mathcal{L}_1 \subseteq_{\mathbf{T}} \mathcal{L}_2$ , then every projective class of  $\mathcal{L}_1$  relative to  $\mathbf{T}$  is also a projective class of  $\mathcal{L}_2$  relative to  $\mathbf{T}$ .*

**Definition 4 (Entailment).** Let  $\mathcal{L}$  be a temporal language,  $\sigma$  a propositional signature,  $\mathbf{T}$  a frame class and  $\phi, \psi \in \mathcal{L}[\sigma]$ . We say that  $\phi$  entails  $\psi$  in  $\mathcal{L}$  over  $\mathbf{T}$  and write  $\phi \models_{\mathcal{L}, \mathbf{T}} \psi$  if for any  $(\mathcal{T}, V, w) \in \text{Str}_{\mathbf{T}}[\sigma]$ , whenever  $(\mathcal{T}, V, w) \models_{\mathcal{L}, \mathbf{T}} \phi$ , then also  $(\mathcal{T}, V, w) \models_{\mathcal{L}, \mathbf{T}} \psi$ .

## 2.2 Propositional Linear Temporal Logic

Recall that  $\mathbf{T}_{\omega}$  denotes the linear orders of order type  $\omega$ . We now introduce the syntax and semantics of PLTL, following the terminology of [11].

**Definition 5 (PLTL).** Let  $\sigma$  be a propositional signature. The set of formulas  $\text{PLTL}[\sigma]$  is defined inductively, as follows:

$$\phi, \psi := At \mid \top \mid \neg\phi \mid \phi \wedge \psi \mid \phi \rightarrow \psi \mid \phi \vee \psi \mid X\phi \mid F\phi \mid F^<\phi \mid \phi U \psi$$

where  $At \in \sigma$ . We use  $\mathbf{G}$  and  $\mathbf{G}^<$  as shorthand for respectively  $\neg\mathbf{F}\neg$  and  $\neg\mathbf{F}^<\neg$ . The relation  $\models_{\text{PLTL}}$  between PLTL-formulas and structures  $(\mathcal{T}, V, w)$  is defined as follows (we only list the clauses of the temporal operators, the others are as in the case of classical propositional logic):

- $(\mathcal{T}, V, w) \models_{\text{PLTL}} X\phi$  iff there exists  $w'$  such that  $w < w'$ , there is no  $w''$  such that  $w < w'' < w'$  and  $(\mathcal{T}, V, w') \models \phi$
- $(\mathcal{T}, V, w) \models_{\text{PLTL}} F\phi$  iff there exists  $w'$  such that  $w \leq w'$  and  $(\mathcal{T}, V, w') \models \phi$
- $(\mathcal{T}, V, w) \models_{\text{PLTL}} F^<\phi$  iff there exists  $w'$  such that  $w < w'$  and  $(\mathcal{T}, V, w') \models \phi$
- $(\mathcal{T}, V, w) \models_{\text{PLTL}} \phi U \psi$  iff there exists  $w'$  such that  $w \leq w'$ ,  $(\mathcal{T}, V, w') \models \psi$  and for all  $w''$  such that  $w \leq w'' < w'$ ,  $(\mathcal{T}, V, w'') \models \phi$

While the above definition in principle applies to arbitrary pointed structures, the intended semantics will be, of course, in terms of structures based on frames in  $\mathbf{T}_\omega$ , and in what follows we will always restrict attention to such frames.

We define fragments  $\text{PLTL}(\mathcal{O})$  of PLTL by allowing in their syntax only a subset  $\mathcal{O} \subseteq \{X, F^<, F, U\}$  of temporal operators. Note that  $\text{PLTL}(U, X)$  has the same expressive power as PLTL, because  $F\phi$  can be defined as  $\top U \phi$  and  $F^<\phi$  as  $X(\top U \phi)$ . The same holds of  $\text{PLTL}(F^<, X)$  and  $\text{PLTL}(F^<, X, F)$ , as  $F\phi$  can be defined as  $\phi \vee F^<\phi$ . Nevertheless, it is known (see [15]), that  $\phi U \psi$  can be defined neither in  $\text{PLTL}(F)$  nor in  $\text{PLTL}(F^<, X)$ . Also  $X\phi$  and  $F^<\phi$  can be defined neither in  $\text{PLTL}(U)$  nor in  $\text{PLTL}(F)$  (we will see why later on in this paper, once we introduce the notion of stutter-invariance).

### 2.3 Linear Time $\mu$ -Calculus

A way of increasing the expressive power of temporal languages is to add a fixpoint operator. On arbitrary structures, adding to PLTL the least fixpoint operator  $\mu$  gives the  $\mu$ -calculus (see for instance [9]). Here, the class of intended structures for  $\mu$ -calculus is restricted to those based on  $\mathbf{T}_w$  and the resulting restricted temporal language is called  $\mu\text{TL}$  (see for instance [14]).

**Definition 6 ( $\mu\text{TL}$ ).** Let  $\sigma$  be a propositional signature, and let  $\mathcal{V} = \{x_1, x_2, \dots\}$  be a disjoint countably infinite stock of *propositional variables*. We define  $\mu\text{TL}[\sigma]$  as the set of all formulas *without free variables* that are generated by the following inductive definition:

$$\phi, \psi, \xi := At \mid \top \mid \neg\phi \mid \phi \wedge \psi \mid \phi \rightarrow \psi \mid \phi \vee \psi \mid X\phi \mid F\phi \mid F^<\phi \mid \phi U \psi \mid \mu x_i. \xi$$

where  $At \in \sigma \cup \mathcal{V}$  and, in the last clause,  $x_i$  occurs only positively in  $\xi$  (i.e., within the scope of an even number of negations). We will use  $\nu x_i. \phi(x_i)$  as shorthand for  $\neg \mu x_i. \neg \phi(\neg x_i)$ . The relation  $\models_{\mu\text{TL}}$  is defined between  $\mu\text{TL}$ -formulas and pointed structures  $(\mathcal{T}, V, w)$  where  $\mathcal{T} \in \mathbf{T}_\omega$ . In order to define it inductively, we use an auxiliary assignment to interpret formulas with free variables. The assignment  $g$  maps each free variable of  $\phi$  to a set of worlds. We let  $g[x \mapsto A]$  be the assignment which differ from  $g$  only by assigning  $A$  to  $x$  and we only recall:

- $(\mathcal{T}, V, w) \models_{\mu\text{PLTL}} x_i [g]$  iff  $w \in g(x_i)$
- $(\mathcal{T}, V, w) \models_{\mu\text{PLTL}} \mu x.\phi [g]$  iff  $\forall A \subseteq W$ , if  $\{v \mid (\mathcal{T}, V, v) \models_{\mu\text{TL}} \phi [g[x \mapsto A]]\} \subseteq A$ , then  $w \in A$

To understand this, consider a  $\mu\text{TL}$ -formula  $\phi(x)$  and a structure  $(\mathcal{T}, V, w)$  together with a valuation  $g$ . This formula induces an operator  $F^\phi$  taking a set  $A \subseteq W$  to the set  $\{v : (\mathcal{T}, V, v) \models_{\mu\text{TL}} \phi(x) [g[x \mapsto A]]\}$ .  $\mu\text{TL}$  is concerned with least fixpoints of such operators. If  $\phi(x)$  is positive in  $x$ , the operator  $F^\phi$  is monotone, i.e.,  $x \subseteq y$  implies  $F^\phi(x) \subseteq F^\phi(y)$ . Monotone operators  $F^\phi$  always have a least fixpoint, defined as the intersection of all their prefixpoints:  $\bigcap \{A \subseteq W : \{v : (\mathcal{T}, V, v) \models \phi(x) [g[x \mapsto A]]\} \subseteq A\}$  (see [3]). The formula  $\mu x.\phi(x)$  denotes this least fixpoint.

It is easy to see that, for formulas without free variables, the assignment is irrelevant, and therefore  $\models_{\mu\text{TL}}$  defines a binary relation between (the set of sentences of)  $\mu\text{TL}$  and pointed structures. In this way,  $\mu\text{TL}$  is an abstract modal language in the sense of Definition [1].

As before, we define a fragment  $\mu\text{TL}(\mathcal{O})$  for each  $\mathcal{O} \subseteq \{X, F^<, F, U\}$ .  $\mu\text{TL}(X)$  already has the full expressive power of  $\text{TL}$ , since  $\phi U \psi$  can be defined by  $\mu y.(\psi \vee (\phi \wedge Xy))$ ,  $F^<\phi$  by  $\mu y.(X\phi \vee Xy)$  and  $F\phi$  by  $\mu y.(\phi \vee Xy)$ . Another fragment of particular interest will be  $\mu\text{TL}(U)$ . In  $\mu\text{TL}(U)$ , we can still define  $F\phi$  in the usual way by  $\top U \phi$ , but we will see that  $X\phi$  and  $F^<\phi$  are not definable.

### 3 Projective Definability versus Definability with Fixpoints

In this section, we discuss two results that relate projective definability in languages without fixpoint operators to explicit definability in the corresponding language with fixpoint operators. Along the way, we also show that  $\mu\text{TL}(U)$  is the stutter-invariant fragment of  $\mu\text{TL}$ . These results will be put to use in Section [4] and [5].

**Theorem 1.** *Let  $\sigma$  be a propositional signature. For any  $K \subseteq \text{Str}_{\mathbf{T}_\omega}[\sigma]$ , the following are equivalent:*

1.  $K$  is a projective class of  $\text{PLTL}(F^<, X)$  relative to  $\mathbf{T}_\omega$
2.  $K$  is definable by a  $\mu\text{TL}$  sentence relative to  $\mathbf{T}_\omega$

*Proof (Sketch).* One direction follows from the fact that  $\mu\text{TL}$  is expressively complete for MSO on  $\mathbf{T}_\omega$  (see [3, 18]). For the other direction, the main idea is that the existence of an accepting run of a Büchi automaton can be projectively defined by means of a PLTL-formula using only the  $F^<$  and  $X$  operators (this is a refinement of a similar result for MSO, see [20]).  $\square$

Below, we will show a similar theorem linking projective definability in  $\text{PLTL}(U)$  (which was shown in [12, 19] to be the stutter-invariant fragment of  $\text{PLTL}$ ) to definability in  $\mu\text{TL}(U)$ , which we show here to be the stutter-invariant fragment of linear time  $\mu$ -calculus. Before stating this second result, we first define stuttering. Intuitively, a stuttering of a linearly ordered structure  $\mathfrak{M}$  is a structure obtained from  $\mathfrak{M}$  by replacing each world by a non-empty finite sequence of worlds, all satisfying the same proposition letters.

**Definition 7 (Stuttering).** Let  $\sigma$  be a propositional signature and  $\mathfrak{M} = ((W, <), V, w)$ ,  $\mathfrak{M}' = ((W', <), V', w')$  be in  $Str_{\mathbf{T}_\omega}[\sigma]$ . We say that  $\mathfrak{M}'$  is a stuttering of  $\mathfrak{M}$  if and only if there is a surjective function  $s : W' \rightarrow W$  such that

1.  $s(w') = w$
2. for every  $w_i, w_j \in W'$ ,  $w_i < w_j$  implies  $s(w_i) \leq s(w_j)$
3. for every  $w_i \in W'$  and  $p \in \sigma$ ,  $w_i \in V'(p)$  iff  $s(w_i) \in V(p)$

Some notation will be useful later on. For any  $w \in W$ , we let  $s^{-1}(w) = \{w' \in W' \mid s(w') = w\}$ . We also extend  $s$  and  $s^{-1}$  to subsets of  $W'$  in the following way: for any  $A' \subseteq W'$ ,  $A \in W$ , we let  $s(A') = \{s(v') \mid v' \in A'\}$  and  $s^{-1}(A) = \bigcup_{v \in A} s^{-1}(v)$ .

**Lemma 2.** *Let  $\mathfrak{M} = ((W, <), V, w)$ ,  $\mathfrak{M}' = ((W', <), V', w')$  be in  $Str_{\mathbf{T}_\omega}[\sigma]$  and  $\mathfrak{M}'$  be a stuttering of  $\mathfrak{M}$ , then the following hold:*

1.  $\forall v' \in W', \forall A' \subseteq W'$  such that  $v' \in A'$  implies  $s^{-1}(s(v')) \subseteq A'$ :

$((W', <), V'[A'/p], v')$  is a stuttering of  $((W, <), V[s(A')/p], s(v'))$

2.  $\forall v \in W, \forall A \subseteq W, \forall v' \in s^{-1}(v)$ :

$((W', <), V'[s^{-1}(A)/p], v')$  is a stuttering of  $((W, <), V[A/p], v)$

**Definition 8 (Stutter-Invariant Class of Pointed Structures).** Let  $\sigma$  be a propositional signature and  $\mathbf{K} \subseteq Str_{\mathbf{T}_\omega}[\sigma]$ . Then  $\mathbf{K}$  is a stutter-invariant class relative to  $\mathbf{T}_\omega$  iff for every  $\mathfrak{M} \subseteq Str_{\mathbf{T}_\omega}[\sigma]$  and for every stuttering  $\mathfrak{M}'$  of  $\mathfrak{M}$ ,  $\mathfrak{M} \in \mathbf{K} \Leftrightarrow \mathfrak{M}' \in \mathbf{K}$ .

**Definition 9 (Stutter-free Pointed Structure).** We say that a pointed structure  $\mathfrak{M}$  is *stutter-free* whenever for all  $\mathfrak{M}'$  such that  $\mathfrak{M}$  is a stuttering of  $\mathfrak{M}'$ ,  $\mathfrak{M}'$  is isomorphic to  $\mathfrak{M}$ .

Only stutter-invariant classes of structures in  $Str_{\mathbf{T}_\omega}[\sigma]$  are definable in  $PLTL(\mathbf{U})$  and  $\mu TL(\mathbf{U})$ . This is known for  $PLTL(\mathbf{U})$  (see [12, 19]), but it also holds for  $\mu TL(\mathbf{U})$ .

**Proposition 1.** *Let  $\sigma$  be a propositional signature. For every  $\mu TL(\mathbf{U})$ -sentence  $\phi$  in signature  $\sigma$ ,  $Mod(\phi)$  is stutter-invariant.*

*Proof.* By induction on the sentence complexity. For the sake of the induction, we can use expanded  $\sigma$ -structures as in classical model theory. Hence we consider two base cases, one for propositional letters and one for propositional variables. The propositional letter case is clear. We handle the propositional variable case  $x_i$  similarly, except that we use  $\sigma$ -models expanded with the value of  $x_i$  (i.e., models considered together with a partial auxiliary valuation, so that  $x_i$  can be seen as a sentence). The induction hypothesis says that for any propositional signature  $\sigma$  and  $\mu TL(\mathbf{U})$ -sentence  $\phi$  of complexity  $n$  in signature  $\sigma$ ,  $Mod(\phi)$  is a stutter-invariant invariant class. Now consider the case were  $\phi$  is of complexity  $n + 1$ . We handle the Boolean connectives and the  $\mathbf{U}$  operator as in the  $PLTL(\mathbf{U})$

case. Now suppose  $\phi := \mu x.\psi(x)$ . We want to show that for every  $\mathfrak{M} \subseteq \text{Str}_{\mathbf{T}}[\sigma]$  and for every stuttering  $\mathfrak{M}'$  of  $\mathfrak{M}$ :

$$\mathfrak{M} = ((\langle, W), V, w) \in \text{Mod}(\mu x.\psi(x)) \Leftrightarrow \mathfrak{M}' = ((\langle, W'), V', w') \in \text{Mod}(\mu x.\psi(x))$$

For the left to right direction, suppose  $((W, \langle), V, w) \models \mu x.\psi(x)$ , i.e.,  $\forall A \subseteq W$ , if  $\{v \mid ((W, \langle), V[A/p], v) \models \psi(p)\} \subseteq A$ , then  $w \in A$ . Consider  $A' \subseteq W'$  such that  $\{v \mid ((W', \langle), V'[A'/p], v) \models \psi(p)\} \subseteq A'$ . We want to show that  $w' \in A'$ . Let us first show that  $v' \in A'$  implies  $s^{-1}(s(v')) \subseteq A'$ . For every  $v' \in A'$ , we have that  $((W', \langle), V'[A'/p], v') \models \psi(p)$ . Now by induction hypothesis for any  $v \in s^{-1}(s(v'))$ ,  $((W', \langle), V'[A'/p], v) \models \psi(p)$  and by hypothesis on  $A'$ ,  $v \in A'$ . It follows from this property of  $A'$  that  $\mathfrak{M}'$  being a stuttering of  $\mathfrak{M}$ , by Lemma 2 for any  $v' \in W'$ ,  $((\langle, W'), V'[A'/p], v')$  is also a stuttering of  $((\langle, W), V[s(A')/p], s(v'))$  and by induction hypothesis:

$$((W', \langle), V'[A'/p], v') \models \psi(p) \text{ iff } ((\langle, W), V[s(A')/p], s(v')) \models \psi(p)$$

Hence  $\{v \mid ((W, \langle), V[s(A')/p], v) \models \psi(p)\} \subseteq s(A')$ . But  $\mathfrak{M} \models \mu x.\psi(x)$ . It follows that  $w \in s(A')$ , so  $s(w) \in A'$ , i.e.,  $w' \in A'$ .

Now for the right to left direction, suppose  $((W', \langle), V', w') \models \mu x.\psi(x)$ , i.e.,  $\forall A' \subseteq W'$ , if  $\{v \mid (W', \langle), V'[A'/p], v \models \psi(p)\} \subseteq A'$ , then  $w' \in A'$ . Consider  $A \subseteq W$  such that  $\{v \mid (W, \langle), V[A/p], v \models \psi(p)\} \subseteq A$ . We want to show that  $w \in A$ .  $\mathfrak{M}'$  being a stuttering of  $\mathfrak{M}$ , by Lemma 2 for any  $v \in W$ ,  $v' \in s^{-1}(v)$ ,  $((\langle, W'), V'[s^{-1}(A)/p], v')$  is also a stuttering of  $((\langle, W), V[A/p], v)$  and by induction hypothesis, for any  $v \in W$ ,  $v' \in s^{-1}(v)$ :

$$((W', \langle), V'[s^{-1}(A)/p], v') \models \psi(p) \text{ iff } ((W, \langle), V[A/p], v) \models \psi(p)$$

Hence  $\{v \mid ((W', \langle), V'[s^{-1}(A)/p], v) \models \psi(p)\} \subseteq s^{-1}(A)$ . But  $\mathfrak{M}' \models \mu x.\psi(x)$ . It follows that  $w' \in s^{-1}(A)$ , so  $s^{-1}(w') \subseteq A$ , i.e.,  $w \in A$ .  $\square$

**Corollary 1.** *Let  $K \subseteq \text{Str}_{\mathbf{T}_\omega}[\sigma]$  be stutter-invariant and let  $\phi \in \mu\text{TL}(\mathbf{U})[\sigma]$  be a sentence such that for each stutter-free  $\mathfrak{M} \in \text{Str}_{\mathbf{T}_\omega}[\sigma]$ ,  $\mathfrak{M} \models \phi$  if and only if  $\mathfrak{M} \in K$ . Then  $\phi$  defines  $K$ .*

We now show that (over  $\mathbf{T}_\omega$ )  $\mu\text{TL}(\mathbf{U})$  is the stutter-invariant fragment of  $\mu\text{TL}$ . The proof is a variant of [19], where Peled and Wilke show that stutter-invariant PLTL properties are expressible without X. We give it in detail, as the construction procedure below will be useful again later on in the paper.

**Lemma 3.** *Let  $\sigma$  be a modal vocabulary. For every  $\mu\text{TL}$  sentence  $\phi$  in vocabulary  $\sigma$ , there exists a  $\mu\text{TL}(\mathbf{U})$  sentence  $\phi^*$  in vocabulary  $\sigma$  that agrees with  $\phi$  on all stutter-free  $\sigma$ -structures over  $\mathbf{T}_\omega$ :*

$$\mathfrak{M} \models \phi \leftrightarrow \phi^* \text{ for all stutter free pointed structures } \mathfrak{M} \in \text{Str}_{\mathbf{T}_\omega}[\sigma]$$

*Proof.* Assume  $\sigma = \{p_0, \dots, p_{n-1}\}$ . The proof goes by induction on the structure of  $\phi$ . For convenience, we use expanded structures. The base case is clear:  $p^* = p$  for any propositional variable or letter  $p$ . Now as regards the induction step, we

can set  $(\neg\psi)^* = \neg\psi^*$ ,  $(\psi \wedge \xi)^* = \psi^* \wedge \xi^*$ ,  $(\psi \text{U} \xi)^* = \psi^* \text{U} \xi^*$ ,  $(\mu x.\psi)^* = \mu x.\psi^*$ . If  $\phi$  is of the form  $\text{X}\psi$ , we let  $B$  be the set of all possible valuations  $\sigma \rightarrow \{\perp, \top\}$ , and for each  $g \in B$ , we let  $\beta_g$  be the formula  $\alpha_0 \wedge \dots \wedge \alpha_{n-1}$  where  $\alpha_j = p_j$  if  $g(p_j) = \top$  and  $\alpha_j = \neg p_j$  if  $g(p_j) = \perp$ . Now observe that if  $g, g' \in B$  are such that  $g \neq g'$ , then

$$\mathfrak{M}, w \models \beta_g \wedge \text{X}\beta_{g'} \leftrightarrow \beta_g \text{U} \beta_{g'} \text{ for } \mathfrak{M} \in \text{Str}_{\mathbf{T}}[\sigma] \text{ stutter-free}$$

We have  $\mathfrak{M}, w \models \text{X}\psi$  if and only if every point in it satisfies the same set of proposition letters and  $\mathfrak{M}, w \models \psi$ , or the valuation function doesn't send the same set of proposition letters to  $w$  and to its immediate successor  $w'$  and  $\mathfrak{M}, w' \models \phi$ . Thus we can set:

$$(\text{X}\psi)^* = \bigvee_{g \in G} ((\text{G}\beta_g \wedge \psi^*) \vee \bigvee_{g \neq g'} (\beta_g \text{U} (\beta_{g'} \wedge \psi^*)))$$

**Theorem 2.** *Let  $\phi \in \mu\text{TL}[\sigma]$  be a sentence such that  $\text{Mod}_{\sigma}(\phi)$  is stutter-invariant. Then there exists  $\phi^* \in \mu\text{TL}(\text{U})[\sigma]$  such that  $\text{Mod}_{\sigma}(\phi) = \text{Mod}_{\sigma}(\phi^*)$ .*

*Proof.* Follows from Lemma 3 and Corollary 1.

Following [12], we now introduce a variant of the notion of projective class, that we call *harmonious projective class*, which preserves stutter-invariance. Before we define it, we first introduce the notion of a *harmonious expansion*. For any propositional signature  $\sigma$  and worlds  $w, w'$ , we write  $w \equiv_{\sigma} w'$  if  $w$  and  $w'$  satisfy the same propositions in  $\sigma$ .

**Definition 10 (Harmonious expansion).** Let  $\sigma \subseteq \tau$  be propositional signatures and  $\mathfrak{M} \in \text{Str}_{\mathbf{T}_w}[\sigma]$ . We say that  $\mathfrak{M}$  is a harmonious expansion of  $\mathfrak{M} \upharpoonright \sigma$  whenever  $\forall w, w' \in W$  such that  $w'$  is a direct successor of  $w$ ,  $w \equiv_{\sigma} w'$  implies  $w \equiv_{\tau} w'$ .

**Definition 11 (Harmonious projective class).** Let  $\sigma$  be a propositional signature and  $K \subseteq \text{Str}_{\mathbf{T}_w}[\sigma]$ . Then  $K$  is a *harmonious projective class* of a temporal language  $\mathcal{L}$  relative to  $\mathbf{T}_w$  whenever there is  $\phi \in \mathcal{L}[\tau]$  with  $\tau \supseteq \sigma$  such that for all  $\mathfrak{M} \in \text{Str}_{\mathbf{T}_w}[\sigma]$ :  $\mathfrak{M} \in K$  iff there is a harmonious  $\tau$ -expansion  $\mathfrak{M}^+$  of  $\mathfrak{M}$  such that  $\mathfrak{M}^+ \models \phi$ .

We will be using the following proposition in order to show Theorem 3. It refers to the notion of  $\omega$ -regular language, cf. [20]. We do not define this notion here as it is not central in this paper. The proof of the proposition in [12] uses a notion of stutter-invariant automata.

**Proposition 2 ([12]).** *On  $\mathbf{T}_w$ , harmonious projective classes of PLTL(U) define exactly the stutter-invariant  $\omega$ -regular languages.*

Now we are able to show the following theorem:

**Theorem 3.** *Let  $\sigma$  be a propositional signature. For any  $K \subseteq \text{Str}_{\mathbf{T}}[\sigma]$ , the following are equivalent:*

1.  $K$  is a harmonious projective class of  $\text{PLTL}(\mathbf{U})$  relative to  $\mathbf{T}_\omega$
2.  $K$  is definable by a  $\mu\text{TL}(\mathbf{U})$ -sentence  $\phi$  relative to  $\mathbf{T}_\omega$

*Proof.* Follows from Theorem 11 and Proposition 2, because by [12, 19],  $\text{PLTL}(\mathbf{U})$  is the stutter-invariant fragment of  $\text{PLTL}$  and by Theorem 2,  $\mu\text{TL}(\mathbf{U})$  is the stutter-invariant fragment of  $\mu\text{TL}$ .  $\square$

## 4 Temporal Languages with Craig Interpolation

In this section, we show that three of the temporal languages previously discussed have Craig interpolation.

**Definition 12 (Craig interpolation property).** Let  $\mathcal{L}$  be a temporal language and  $\mathbf{T}$  a frame class. Then  $\mathcal{L}$  has the Craig interpolation property over  $\mathbf{T}$  whenever the following holds. Let  $\phi \in \mathcal{L}[\sigma]$ ,  $\psi \in \mathcal{L}[\sigma']$ . Whenever  $\phi \models_{\mathcal{L}, \mathbf{T}} \psi$ , then there exists  $\theta \in \mathcal{L}[\sigma \cap \sigma']$  such that  $\phi \models_{\mathcal{L}, \mathbf{T}} \theta$  and  $\theta \models_{\mathcal{L}, \mathbf{T}} \psi$ .

They even satisfy a stronger form of interpolation, which is called uniform interpolation. Intuitively, if a temporal language has uniform interpolation, it means that the interpolant can be constructed so that it depends only on the signature of the antecedent and its intersection with the signature of the consequent.

**Definition 13 (Uniform Interpolation).** Let  $\mathcal{L}$  be a temporal language and  $\mathbf{T}$  a frame class.  $\mathcal{L}$  has the *uniform interpolation* property over  $\mathbf{T}$  if, for all signatures  $\sigma \subseteq \tau$  and for each formula  $\phi \in \mathcal{L}[\tau]$  there is a formula  $\theta \in \mathcal{L}[\sigma]$  such that  $\phi \models_{\mathcal{L}} \theta$  and for each formula  $\psi \in \mathcal{L}[\tau']$  with  $\tau \cap \tau' \subseteq \sigma$ , if  $\phi \models_{\mathcal{L}} \psi$  then  $\theta \models_{\mathcal{L}} \psi$ .

**Theorem 4.**  $\mu\text{TL}$  has uniform interpolation over  $\mathbf{T}_\omega$ .

*Proof.* MSO has uniform interpolation (for monadic predicates) on any class of structures (so in particular on  $\mathbf{T}_\omega$ ) because it has set quantifiers (see [8]). By [3, 18],  $\mu\text{TL}$  is expressively complete for MSO. Hence  $\mu\text{TL}$  uniform interpolants can always be obtained via translation into MSO.  $\square$

**Theorem 5.**  $\mu\text{TL}(\mathbf{U})$  has uniform interpolation over  $\mathbf{T}_\omega$ .

*Proof.* Let  $\sigma \subseteq \tau$  be modal signatures and let  $\phi \in \mu\text{TL}(\mathbf{U})[\tau]$ . By Theorem 4, there exists  $\theta \in \mu\text{TL}(\mathbf{U})[\sigma]$  such that  $\phi \models \theta$  and for each formula  $\psi \in \mu\text{TL}(\mathbf{U})[\tau']$  with  $\tau \cap \tau' \subseteq \sigma$ , if  $\phi \models \psi$ , then  $\theta \models \psi$ . Now let  $\theta^* \in \mu\text{TL}(\mathbf{U})$  be the formula that agrees with  $\theta$  on all stutter-free structures based on  $\mathbf{T}_\omega$  (by Lemma 3, such a formula exists). We want to show that  $\phi \models \theta^*$  and that for each formula  $\psi \in \mu\text{TL}(\mathbf{U})[\tau']$  with  $\tau \cap \tau' \subseteq \sigma$ , if  $\phi \models \psi$ , then  $\theta^* \models \psi$ . Let  $SMod(\phi)$  denote the set of stutter free structures in  $Mod(\phi)$ . As  $Mod(\phi) \subseteq Mod(\theta)$ ,  $SMod(\phi) \subseteq SMod(\theta)$ . Now by construction of  $\theta^*$  also  $SMod(\phi) \subseteq SMod(\theta^*)$ .  $Mod(\phi)$  and  $Mod(\theta^*)$  are both stutter-invariant classes. It follows from Corollary 1 that the closure under stuttering of  $SMod(\phi)$  is included in the closure under stuttering of  $SMod(\theta^*)$ , i.e.,  $Mod(\phi) \subseteq Mod(\theta^*)$ , i.e.,  $\phi \models \theta^*$ . The argument for  $\theta^* \models \psi$  is similar.  $\square$

**Theorem 6.**  $\text{PLTL}(\mathbf{X})$  has uniform interpolation over  $\mathbf{T}_\omega$ .

*Proof (Sketch).* We will show something much stronger, namely that every projective class of  $\text{PLTL}(\mathbf{X})$  is definable by a  $\text{PLTL}(\mathbf{X})$ -formula.

Let  $\phi \in \text{PLTL}(\mathbf{X})[\sigma \cup \{p\}]$ . We will show how to construct a formula  $\psi \in \text{PLTL}(\mathbf{X})[\sigma]$  that defines the class of  $\sigma$ -reducts of models of  $\phi$ . Let  $n$  be the maximal nesting depth of  $\mathbf{X}$ -operators in  $\phi$ . Intuitively,  $\phi$  can only talk about the first  $n$  world in the pointed structure (starting from the designated world). We can represent every valuation of  $p$  in these  $n$  worlds by a set  $S \subseteq \{0, \dots, n\}$ , where  $k \in S$  represents that  $p$  is true at the  $k$ -th world starting from the designated world. For each  $S \subseteq \{1, \dots, n\}$  we define  $\phi^S$  as follows: we replace each occurrence of  $p$  in  $\phi$  that is in the scope  $k$   $\mathbf{X}$ -operators ( $k \leq n$ ) by  $\top$  if  $k \in S$  and  $\perp$  otherwise. Then  $\phi$  and  $\phi^S$  are equivalent in all pointed structures in which the valuation of  $p$  is as described by  $S$ . This can be shown by a formula induction. Now, let  $\psi = \bigvee_{S \subseteq \{0, \dots, n\}} \phi^S$ . Then  $\psi$  holds in a pointed  $\sigma$ -structure  $\mathfrak{M}$  iff  $\mathfrak{M}$  has an expansion satisfying  $\phi$ .  $\square$

## 5 Interpolation Closure Results for Temporal Languages

In this section, we look at the fragments of PLTL that do not have Craig interpolation, and we address the question how much expressive power must be added in order to regain interpolation. We will phrase our main results in terms of the notion of interpolation closure, which we define by taking inspiration from abstract model theory (see [4]):

**Definition 14 (Interpolation Closure).** Let  $\mathbf{T}$  be a frame class.  $\mathcal{L}_2$  is the interpolation closure of  $\mathcal{L}_1$  over  $\mathbf{T}$  if  $\mathcal{L}_1 \subseteq_{\mathbf{T}} \mathcal{L}_2$ ,  $\mathcal{L}_2$  has interpolation over  $\mathbf{T}$ , and for every abstract temporal language  $\mathcal{L}_3$ , if  $\mathcal{L}_1 \subseteq \mathcal{L}_3$  and  $\mathcal{L}_3$  has Craig interpolation on  $\mathbf{T}$ , then  $\mathcal{L}_2 \subseteq_{\mathbf{T}} \mathcal{L}_3$ .

### 5.1 The Interpolation Closure of $\text{PLTL}(\mathbf{F}^<)$

A useful tool for proving interpolation closure results is the following lemma (see [4]):

**Definition 15 ( $\Delta$ -interpolation property).** Let  $\mathcal{L}$  be a temporal language and  $\mathbf{T}$  a frame class. Then  $\mathcal{L}$  has the  $\Delta$ -interpolation property over  $\mathbf{T}$  whenever the following holds: let  $\sigma$  be a propositional signature and  $K \subseteq \text{Str}_{\mathbf{T}}[\sigma]$ , if both  $K$  and  $\bar{K}$  are projective classes of  $\mathcal{L}$  relative to  $\mathbf{T}$ , there is a  $\mathcal{L}$ -formula  $\phi$  such that  $K = \text{Mod}_{\mathbf{T}}^{\sigma}(\phi)$ .

**Lemma 4.** Let  $\mathcal{L}$  be a temporal language with Craig interpolation on  $\mathbf{T}_\omega$ . Then  $\mathcal{L}$  has  $\Delta$ -interpolation over  $\mathbf{T}_\omega$ .

The proof of Lemma 4 is similar to the one given in [6] (we only need to remark that the substitution property assumed here of abstract temporal languages is stronger than implies the *renaming* property assumed in [6] of abstract modal languages).



Now we will show that  $\text{PLTL}(F^<, X)$  is contained in the interpolation closure of  $\text{PLTL}(F^<)$  over  $\mathbf{T}_\omega$ . As an intermediate step, we show that in every extension of  $\text{PLTL}(F^<)$  having Craig interpolation, the property  $Xp$  is “definable”. By this, we mean the following:

**Lemma 5.** *Let  $\mathcal{L}$  be an extension of  $\text{PLTL}(F^<)$  with Craig-interpolation over  $\mathbf{T}_\omega$ . Then there is  $\xi \in \mathcal{L}[\{p\}]$  such that  $\text{Mod}(\xi) = \text{Mod}(Xp)$ .*

*Proof.* Let  $q, r$  be new distinct propositional letters. Consider the two following projective classes of  $\text{PLTL}(F^<)$ :  $\text{Mod}(F^<(p \wedge q) \wedge \neg F^<F^<q) \upharpoonright \{p\}$  and  $\text{Mod}((F^<(\neg p \wedge r) \wedge \neg F^<F^<r) \vee G^<\perp) \upharpoonright \{p\}$ . As  $\text{PLTL}(F^<) \subseteq \mathcal{L}$ , these two classes are also projective classes of  $\mathcal{L}$  (by Lemma 4). They also complement each other, as a  $\{p\}$ -structure belongs to the first class exactly when the first node of this structure has a successor node where  $p$  holds and it belongs to the second class in all other cases. By  $\Delta$ -interpolation for  $\mathcal{L}$  on  $\mathbf{T}$ , it follows that the first class is definable in  $\mathcal{L}$  by means of some formula  $\xi$  in signature  $\{p\}$ , i.e., there is  $\xi \in \mathcal{L}[\{p\}]$  such that  $\text{Mod}(Xp) = \text{Mod}(\xi)$ .  $\square$

**Theorem 7.** *Every extension of  $\text{PLTL}(F^<)$  with Craig interpolation over  $\mathbf{T}_\omega$  is an extension of  $\text{PLTL}(F^<, X)$  over  $\mathbf{T}_\omega$ .*

*Proof.* Let  $\mathcal{L}$  be an extension of  $\text{PLTL}(F^<)$  with Craig interpolation over  $\mathbf{T}_\omega$  and  $\sigma$  a propositional signature. We show by induction on the complexity of  $\phi$  (number of Boolean and temporal operators in  $\phi$ ) that for all  $\phi \in \text{PLTL}(F^<, X)[\sigma]$ , there exists  $\phi' \in \mathcal{L}[\sigma]$  such that  $\text{Mod}(\phi) = \text{Mod}(\phi')$ . The base case is clear. The induction hypothesis says that for all  $\sigma$ , for all  $\phi \in \text{PLTL}(F^<, X)[\sigma]$  of complexity at most  $n$ , there exists  $\phi' \in \mathcal{L}[\sigma]$  such that  $\text{Mod}(\phi) = \text{Mod}(\phi')$ . Now let  $\phi$  be of complexity  $n + 1$ . If  $\phi := X\psi$ , by induction hypothesis there exists  $\psi' \in \mathcal{L}[\sigma]$  such that  $\text{Mod}(\psi) = \text{Mod}(\psi')$ . Pick any  $p \notin \sigma$ . By Lemma 5 and the expansion property we know:

1. There is  $\xi \in \mathcal{L}[\sigma \cup \{p\}]$  such that  $\text{Mod}(Xp) = \text{Mod}(\xi)$ .

We will define  $\phi'$  as  $\xi[p/\psi'] \in \mathcal{L}[\sigma]$  (by closure under uniform substitution of  $\mathcal{L}$ , such a formula exists). We need to show that  $\text{Mod}(X\psi) = \text{Mod}(\xi[p/\psi'])$ . From 1 we can derive as a particular case:

2. For any  $(\mathcal{T}, V, w) \in \text{Str}_{\mathbf{T}}[\sigma \cup \{p\}]$  where  $V(p) = \{w_i \mid (F, V, w_i) \models \psi'\}$ ,  $(\mathcal{T}, V, w) \models \xi$  iff there exists  $w' \in D$  such that  $w < w'$ , there is no  $w''$  such that  $w < w'' < w'$  and  $(\mathcal{T}, V, w') \models p$ .

Now by closure under uniform substitution of  $\mathcal{L}$ , 2 is equivalent to the following:

3. For any  $(\mathcal{T}, V, w) \in \text{Str}_{\mathbf{T}}[\sigma]$ ,  $(F, V, w) \models \xi[p/\psi']$  iff there exists  $w' \in D$  such that  $w < w'$ , there is no  $w''$  such that  $w < w'' < w'$  and  $(F, V, w') \models p[p/\psi']$ .

Finally,  $\psi'$  and  $p[p/\psi']$  holding exactly in the same models, we can replace  $p[p/\psi']$  by  $\psi'$  in the second member of the equivalence in 3. Hence  $\text{Mod}(X\psi) = \text{Mod}(\xi[p/\psi'])$ . We can use similar arguments for the operator  $F^<$  and for Boolean connectives.  $\square$

By putting Lemma 4 to use, we now improve Theorem 7 and identify the interpolation closure of  $\text{PLTL}(F^<)$ .

**Theorem 8.**  $\mu\text{TL}$  is the interpolation closure of  $\text{PLTL}(F^<, X)$  over  $\mathbf{T}_\omega$ .

*Proof.* Let  $\sigma$  be a propositional signature. Now let  $K \subseteq \text{Str}_{\mathbf{T}_\omega}[\sigma]$  be definable by a  $\mu\text{TL}$ -sentence  $\phi$  in signature  $\sigma$ . As  $\mu\text{TL}$  is closed under negation, there is a  $\mu\text{TL}$ -sentence  $\neg\phi$  in signature  $\sigma$ , which defines the complement of  $K$  over  $\text{Str}_{\mathbf{T}_\omega}[\sigma]$ . It follows by Theorem 1 that both  $K$  and its complement are projective classes of  $\text{PLTL}(F^<, X)$ . Now consider a temporal language  $\mathcal{L} \supseteq \text{PLTL}(F^<, X)$  with Craig interpolation over  $\mathbf{T}_\omega$ . By Lemma 1,  $K$  and its complement are also projective classes of  $\mathcal{L}$  and by Lemma 4, it follows that  $K$  is definable in  $\mathcal{L}$ .  $\square$

### 5.2 The Interpolation Closure of $\text{PLTL}(F)$

For the case of the stutter-invariant languages  $\text{PLTL}(F)$  and  $\text{PLTL}(U)$ , we need to refine the notion of  $\Delta$ -interpolation, by considering harmonious projective classes.

**Definition 16 (Harmonious  $\Delta$ -interpolation property).** Let  $\mathcal{L}$  be a temporal language. Then  $\mathcal{L}$  has the harmonious  $\Delta$ -interpolation property over  $\mathbf{T}_\omega$  whenever the following holds. Let  $K$  be a class of  $\mathcal{L}$ -structures based on  $\mathbf{T}_\omega$ . If both  $K$  and  $\bar{K}$  are harmonious projective classes of  $\mathcal{L}$  relative to  $\mathbf{T}_\omega$ , there is a  $\mathcal{L}$ -formula  $\phi$  such that  $K = \text{Mod}_{\mathbf{T}_\omega}(\phi)$ .

**Lemma 6.** If  $\mathcal{L}_1 \subseteq \mathcal{L}_2$ , then every harmonious projective class of  $\mathcal{L}_1$  is also a harmonious projective class of  $\mathcal{L}_2$ .

**Definition 17 (Harmonious temporal language).** A temporal language  $\mathcal{L}$  is *harmonious* for  $\mathbf{T}_\omega$  if the following holds. For every  $\sigma \subseteq \tau$  propositional signatures, there is a formula  $\phi \in \mathcal{L}[\tau]$  such that for every  $\mathfrak{M} \in \text{Str}_{\mathbf{T}_\omega}[\tau]$ ,  $\mathfrak{M} \models \phi$  if and only if  $\mathfrak{M}$  is an harmonious expansion of  $\mathfrak{M} \upharpoonright \sigma$ .

**Proposition 3.**  $\text{PLTL}(U)$  and its extensions are harmonious for  $\mathbf{T}_\omega$ .

*Proof.* Fix  $\sigma \subseteq \tau$  with  $|\sigma| = n$ ,  $|\tau \setminus \sigma| = m$ . We can represent any valuation over  $\sigma$  by a finite conjunction of atoms and negations of atoms. Let  $\{\sigma_i \mid i \in 2^n\}$  be the set of all such conjunctions. Also, for each  $\sigma_i$ , we define the corresponding set  $\{\tau_j^i \mid j \in 2^m\}$  as the set of conjunctions representing all possible ways of extending to  $\tau$  the valuation represented by  $\sigma_i$ . Now for every  $\mathfrak{M} \in \text{Str}_{\mathbf{T}}[\tau]$ ,

$$\mathfrak{M} \models \bigwedge_{i,j \in 2^n} (\sigma_i \cup \sigma_j \rightarrow \bigvee_{k,l \in 2^m} \tau_k^i \cup \tau_l^j)$$

if and only if  $\mathfrak{M}$  is an harmonious expansion of  $\mathfrak{M} \upharpoonright \sigma$ , i.e.,  $\text{PLTL}(U)$  is harmonious. It is immediate from definition 2 that every extension of a temporal language which is harmonious for  $\mathbf{T}_\omega$  is also harmonious for  $\mathbf{T}_\omega$ .  $\square$

**Lemma 7.** Let  $\mathcal{L}$  be a temporal language which has Craig interpolation and is harmonious for  $\mathbf{T}_\omega$ . Then  $\mathcal{L}$  has harmonious  $\Delta$ -interpolation over  $\mathbf{T}_\omega$ .

$\mathcal{L}$  being harmonious, we can use the formula  $\phi$  in Definition 17 and appeal for the proof of Lemma 7 to the same classical argument as for Lemma 4.

**Theorem 9.** *Every extension of PLTL(F) with Craig interpolation over  $\mathbf{T}_\omega$  is an extension of PLTL(U) over  $\mathbf{T}_\omega$ .*

*Proof.* The reasoning is similar as in the case of Lemma 7 and Theorem 7, but we consider  $Mod(pUq) = Mod(\mathbf{G}(Fr \rightarrow r) \wedge \mathbf{F}(q \wedge r) \wedge \mathbf{G}((r \wedge \neg q) \rightarrow p)) \upharpoonright \{p, q\}$  and  $Mod(\neg pUq) = Mod(\mathbf{F}q \rightarrow (\mathbf{F}(\neg p \wedge r) \wedge \mathbf{G}(Fr \rightarrow \neg q))) \upharpoonright \{p, q\}$ .  $\square$

**Theorem 10.**  *$\mu\text{TL}(\text{U})$  is the interpolation closure of PLTL(U) over  $\mathbf{T}_\omega$ .*

*Proof.* Let  $\sigma$  be a modal signature. Now let  $K \subseteq \text{Str}_{\mathbf{T}_\omega}[\sigma]$  be definable by a  $\mu\text{TL}(\text{U})$ -sentence  $\phi$  in signature  $\sigma$ . As  $\mu\text{TL}(\text{U})$  is closed under negation, there is a  $\mu\text{TL}(\text{U})$ -sentence  $\neg\phi$  in signature  $\sigma$ , which defines the complement  $\bar{K} \subseteq \text{Str}_{\mathbf{T}_\omega}[\sigma]$  of  $K$  over  $\text{Str}_{\mathbf{T}_\omega}[\sigma]$ . By Theorem 3, both  $K$  and  $\bar{K}$  are harmonious projective classes of PLTL(U). Now consider a temporal language  $\mathcal{L} \supseteq \text{PLTL}(\text{U})$  with Craig interpolation over  $\mathbf{T}$ . By Lemma 6,  $K$  and  $\bar{K}$  are also harmonious projective classes of  $\mathcal{L}$ . By Proposition 3,  $\mathcal{L}$  is harmonious and by Lemma 7, it follows that  $K$  is definable in  $\mathcal{L}$ , i.e.,  $\mathcal{L} \supseteq \mu\text{TL}(\text{U})$ .  $\square$

## 6 Finite Linear Orders

We restricted our attention to the frame class  $\mathbf{T}_\omega$ , but our results easily extend to finite linear orders. Let  $\mathbf{T}_{fin}$  be the class of frames  $(D, <)$  where  $D$  is a finite set and  $<$  is a strict linear order on  $D$ . All the definitions and results that we gave relative to  $\mathbf{T}_\omega$  also apply to  $\mathbf{T}_{fin}$ . An analogous of Theorem 1 for  $\mathbf{T}_{fin}$  can be obtained by considering automata on finite words. The proof of Proposition 2 can similarly be adapted by considering stutter-invariant automata on finite words. In the proof of Lemma 3, we can define  $(X\psi)^*$  as  $\bigvee_{g \neq g'} (\beta_g U (\beta_{g'} \wedge \psi^*))$  (i.e., we keep only the second disjoint, as no finite stutter free linear order exhibits two successor points satisfying the same set of proposition letters). The remaining of our arguments do not need any further adjustment.

## 7 Conclusions and Future Work

In this paper, we studied the temporal fragments of linear time  $\mu$ -calculus satisfying Craig interpolation, showing essentially that there are only three distinct such fragments:  $\mu\text{TL}$  itself,  $\mu\text{TL}(\text{U})$ , and  $\text{PLTL}(\text{X})$ . These results reconfirm the robustness of (linear time)  $\mu$ -calculus as compared to less expressive temporal logics. We are currently working on extending our results to other flows of time such as finite trees, infinite trees, and infinite linear orders other than the natural numbers (as in 5). There are some important differences in these settings. For example, it is known (see 2) that the branching time temporal logic with only Since and Until has Craig interpolation, while linear time fails to have this property. Also there is still no definitive consensus on the appropriate notion of stuttering for infinite branching time (see 13).

## References

- [1] Amir, E., McIlraith, S.A.: Partition-based logical reasoning for first-order and propositional theories. *Artif. Intell.* 162(1-2), 49–88 (2005)
- [2] Areces, C., de Rijke, M.: Interpolation and bisimulation in temporal logic. In: Guerra, R.J., de Queiroz, B., Finger, M. (eds.) *Proceedings of WoLLIC 1998*, pp. 15–21 (1998)
- [3] Arnold, A., Niwinski, D.: *Rudiments of  $\mu$ -Calculus*. Studies in Logic and Foundations of Mathematics, vol. 146. North-Holland, Amsterdam (2001)
- [4] Barwise, J., Feferman, S.: *Model-theoretic logics*. Springer, New York (1985)
- [5] Bruyère, V., Carton, O.: Automata on linear orderings. *J. Comput. System Sci.* 73(1), 1–24 (2007)
- [6] ten Cate, B.: *Model Theory for Extended Modal Languages*. PhD thesis, University of Amsterdam, ILLC Dissertation Series DS-2005-01 (2005)
- [7] Craig, W.: Three Uses of the Herbrand-Gentzen Theorem in Relating Model Theory and Proof Theory. *Journal of Symbolic Logic* 22(3), 269–285 (1957)
- [8] D’Agostino, G.: Interpolation in non-classical logics. *Synthese* 164(3), 421–435 (2008)
- [9] D’Agostino, G., Hollenberg, M.: Logical Questions Concerning the  $\mu$ -Calculus: Interpolation, Lyndon and Löb-Tarski. *Journal of Symbolic Logic* 65(1), 310–332 (2000)
- [10] Gerard, R., de Lavalette, R.: Interpolation in computing science: the semantics of modularization. *Synthese* 164(3), 437–450 (2008)
- [11] Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
- [12] Etessami, K.: Stutter-Invariant Languages,  $\omega$ -Automata, and Temporal Logic. In: Halbwachs, N., Peled, D.A. (eds.) *CAV 1999*. LNCS, vol. 1633, pp. 236–248. Springer, Heidelberg (1999)
- [13] Gross, R.: *Invariance under stuttering in branching-time temporal logic*. Master’s thesis, Israel Institute of Technology, Haifa (2008)
- [14] Kaivola, R.: *Using Automata to Characterise Fixed Point Temporal Logics*. PhD thesis, University of Edinburgh (1997)
- [15] Kamp, H.: *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, Los Angeles (1968)
- [16] Lamport, L.: What Good is Temporal Logic? In: Mason, R.E.A. (ed.) *Proceedings of the IFIP 9th World Computer Congress*, pp. 657–668. North-Holland/IFIP (1983)
- [17] Maksimova, L.: Temporal logics with “the next” operator do not have interpolation or the Beth property. *Siberian Mathematical Journal* 32(6), 989–993 (1991)
- [18] Niwinski, D.: Fixed Points vs. Infinite Generation. In: *Proceedings of LICS*, pp. 402–409 (1988)
- [19] Peled, D., Wilke, T.: Stutter-invariant temporal properties are expressible without the next-time operator. *Inf. Process. Lett.* 63(5), 243–246 (1997)
- [20] Thomas, W.: Languages, automata, and logic. In: *Handbook of formal Languages. Beyond Words*, vol. 3, pp. 389–455. Springer, New York (1997)

# On Model Checking Boolean BI

Heng Guo, Hanpin Wang, Zhongyuan Xu, and Yongzhi Cao

Key Laboratory of High Confidence Software Technologies, Ministry of Education,  
Institute of Software, School of Electronics Engineering and Computer Science,  
Peking University, Beijing, China  
{guoheng, whpxhy, xzy, caoyz}@pku.edu.cn

**Abstract.** The logic of bunched implications (BI), introduced by O’Hearn and Pym, is a substructural logic which freely combines additive and multiplicative implications. Boolean BI (BBI) denotes BI with classical interpretation of additives and its model is the commutative monoid. We show that when the monoid is finitely generated and propositions are recursively defined, or the monoid is infinitely generated and propositions are restricted to generator propositions, the model checking problem is undecidable. In the case of finitely related monoid and generator propositions, the model checking problem is EXPSPACE-complete.

## 1 Introduction

The logic of bunched implications (BI), introduced by O’Hearn and Pym [26], is a substructural logic which freely combines additive and multiplicative implications. The main purpose of BI is to reason about models which incorporate the notion of resource. Its best-known application in computer science is separation logic, which is a Hoare logic for reasoning about imperative program which can manipulate pointers [29]. Several other similar resource logics are developed, such as spatial logic [9,10], which is introduced independently, and context logic [6,7], which can be regarded as a generalization of BI and spatial logic.

Semantically, the models of BI vary from cartesian doubly closed categories, to partially ordered commutative monoids [27]. The interpretation of BI in the categorical models is necessarily intuitionistic. But the additive connectives can be interpreted classically in the monoid models, in which the partial order becomes an equivalence relation. This version of BI is called Boolean BI (BBI). Its expressivity is quite powerful [18], and has been shown to be a convenient way to characterize resource sensitive systems. Indeed, separation logic is interpreted on a partial monoid of heaps, which is a BBI model. Also the classical additives alongside multiplicative conjunction and implication could be found in spatial logic and context logic.

In this paper, we mainly discuss the model checking problem of BBI, *i.e.* given an element in a BBI model and a BI formula, decide whether the element satisfies the formula. It is shown that the provability of BI can be decided by Resource Tableauax (*cf.* [19]). However, this method cannot be applied to the model checking problem directly.

To give a decidable condition, a notion of boundable resource model is introduced in [2]. It is proved that the model checking problem in a boundable model is decidable.

However, this condition is given implicitly. To decide whether a model is boundable, one need to check whether the quotient of the monoid divided by some equivalence relations is finite. It is quite hard (and possibly undecidable) to verify this since there are infinitely many such relations. Compared with their work, in this paper, we show more explicit decidable conditions. To our best knowledge, there is not any other work concerning the model checking problem for BI.

In separation logic, all products are obtained from two heaps with disjoint domains (cf. [29]). But in general BBI model, the structure is less organized, since the generation relation can be defined arbitrarily. Hence, we expect that the model checking problem for BBI is quite complicated and would be decidable only under many restrictions.

First, we consider the propositions appeared in BI formulae. In a BBI model, a proposition variable is interpreted by the set composed of elements satisfying the formula. Obviously, when such a set is not recursive, we cannot check whether an element satisfies this proposition. However, even if propositions are recursively defined, we show that the model checking problem is undecidable for finitely generated free commutative monoid, somehow the simplest model, by reduction from *Hilbert 10th problem*.

Inspired by the fact that in separation logic atomic assertions are interpreted on one heap cell, we focus our attention on propositions that is satisfied by only one generator. But even with this restriction, the model checking problem is still undecidable in infinitely generated commutative monoid. The technique we use is to simulate *Minsky Machine* (cf. [25]), which is a classical and widely adopted method, like in the proof of the undecidability of full propositional linear logic [23] and the bisimulation relation between petri nets [20]. We should mention that although total monoid is a special case of partial monoid, the model checking problem for quantifier free assertion in separation logic, which is interpreted on an infinitely generated partial monoid, in contrast, is decidable [8].

To obtain decidable results, we put some additional restrictions on the model. We consider the case that the monoid is finitely generated. A special case of the model checking problem in this setting is equivalent to the word problem in monoids, which is shown to be EXPSPACE-complete in finitely generated commutative monoids (cf. [24]). This result sheds some light to the general problem and provides a complexity lower bound. With the help of some results from [21] and [16], we reduce the model checking problem to the problem of deciding whether two semi-linear sets overlap, which can be solved with the cost of at most exponential space. It follows that the in this case, the model checking problem is EXPSPACE-complete.

Furthermore, in the case of infinitely generated finitely related monoid, the model checking problem can be reduced to the finitely generated case. Indeed, every finitely generated monoid is finitely related (cf. [17]). Thus, for all finitely related monoid, the model checking problem is EXPSPACE-complete.

Model checking and validity problems for the spatial assertion language of separation logic are solved in [8]. Several decidable fragments are discovered and discussed [13]. In comparison, the model we considered is more general and its structure can be more chaotic, and the formula is propositional. Thus, both our decidability and undecidability results are essentially different from those of separation logic. It is easy to extend our complexity results to the case of partial monoid.

Interesting aspect of our undecidability proof is the explicit way of simulating Minsky Machine, which can be viewed as a sign of the strong expressivity of BBI and did not appear in the literature before. The technique used in our decidability proof reveals the relationship among BI formulae, rational sets in monoid and regular expressions. It suggests a way to extend BI and provide the possibility to apply classical algebraic results on the further analysis of BI or BBI.

In Section 2 we review some basic definitions and notations of BBI and semigroups. In Section 3 we show undecidability results, and Section 4 decidability and complexity results. Finally, in Section 5, some additional remarks are provided. For brevity, some of the proofs are omitted.

## 2 Preliminaries

We start with some basic definitions and denotations.

### 2.1 Boolean BI

In BI, there are additive connectives of classical propositional logic ( $\neg, \vee, \wedge, \rightarrow, \top, \perp$ ) and multiplicative connectives ( $*, \multimap, \top^*$ ).

**Definition 1 (BI formula).** *The set of BI formulae, denoted  $\mathcal{BI}$  and ranged over by  $\varphi, \varphi_1, \varphi_2$ , is defined by:*

$$\varphi = p \mid \top \mid \perp \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \top^* \mid \varphi_1 * \varphi_2 \mid \varphi_1 \multimap \varphi_2$$

in which  $p$  ranges over  $P$ , the set of atomic propositions.

**Definition 2 (BBI Model).** *A BBI model  $\mathcal{M}$  is a commutative monoid  $\{M, \varepsilon, \circ\}$  (denoted  $M$ , for brevity), in which  $\circ$  is the multiplication and  $\varepsilon$  its unit.*

Henceforth monoid will be used to denote commutative monoid, if not explicitly stated.

An environment mapping is needed to interpret proposition variables. The image of a proposition variable is the largest set in which every element satisfies it. By  $\mathcal{P}(M)$  we denote the power set of  $M$ .

**Definition 3 (Environment).** *An environment  $\rho_M$ , or  $\rho$  for short, is a function  $\rho : P \rightarrow \mathcal{P}(M)$ .*

**Definition 4 (Satisfaction Relation).** *The satisfaction relation for BBI is defined inductively on the structure of the formulae as follows:*

$$\begin{aligned} m \models p &\Leftrightarrow m \in \rho(p) \\ m \models \top &\Leftrightarrow \text{always} \\ m \models \neg\varphi &\Leftrightarrow m \not\models \varphi \\ m \models \varphi_1 \wedge \varphi_2 &\Leftrightarrow m \models \varphi_1 \text{ and } m \models \varphi_2 \\ m \models \top^* &\Leftrightarrow m = \varepsilon \\ m \models \varphi_1 * \varphi_2 &\Leftrightarrow \exists m_1, m_2. m = m_1 \circ m_2 \text{ s.t. } m_1 \models \varphi_1 \text{ and } m_2 \models \varphi_2 \\ m \models \varphi_1 \multimap \varphi_2 &\Leftrightarrow \forall m_1. m_1 \models \varphi_1 \text{ implies } m_1 \circ m \models \varphi_2 \end{aligned}$$

Since the additive connectives are interpreted classically, we treat  $\perp$ ,  $\vee$ , and  $\rightarrow$  as usual abbreviations. Define  $\varphi_1 * \exists \varphi_2 = \neg(\varphi_1 * \neg \varphi_2)$ . Then  $m \models \varphi_1 * \exists \varphi_2$  iff  $\exists m_1. m_1 \models \varphi_1$  and  $m_1 \circ m \models \varphi_2$ . It is an existence analogue of multiplicative implication.

Sometimes we use  $\models_M$  to emphasize the underlying model for the satisfaction relation.

We naturally extend the domain of environment function  $\rho$  from the set of propositions  $P$  to the set of BI formulae  $\mathcal{BT}$ ,  $\rho : \mathcal{BT} \rightarrow \mathcal{P}(M)$ ,  $\rho(\varphi) = \{m \mid m \in M \wedge m \models \varphi\}$ . Thus, the model checking problem of deciding whether  $m \models \varphi$ , is equivalent to the problem of deciding whether  $m \in \rho(\varphi)$ .

For  $M_1, M_2 \subseteq M$ , define:

$$\begin{aligned} M_1 \circ M_2 &= \{m \mid \exists m_1, m_2. m = m_1 \circ m_2 \wedge m_1 \in M_1 \wedge m_2 \in M_2\} \\ M_1 : M_2 &= \{m \mid \exists m_1. m_1 \in M_2 \wedge m \circ m_1 \models M_1\} \end{aligned}$$

Thus, we get:

$$\begin{aligned} \rho(\varphi_1 * \varphi_2) &= \rho(\varphi_1) \circ \rho(\varphi_2) \\ \rho(\varphi_1 * \exists \varphi_2) &= \rho(\varphi_2) : \rho(\varphi_1) \end{aligned}$$

Note that  $\varphi_1 * \varphi_2 = \neg(\varphi_1 * \exists \neg \varphi_2)$ . In the following, we will use  $* \exists$  when inducting on the structure of a formula.

*Remarks.* In the general semantics of BI, partial monoid, rather than total monoid, is more widely adopted, since it is complete and reflects some intrinsic properties of resources. Indeed, the heap model of separation logic is a partial monoid. However, there is one way to transform a partial model into a total model. Define a special element  $\pi$ , which does not satisfy any formulae, and let any undefined product of two elements, or the product of  $\pi$  and any other element equal  $\pi$ . Then we get a total monoid and only need to pay some attention for such  $\pi$  when handling the model checking problem. Hence, for the simplicity of analysis, we adopt the notion of total monoid in this paper.

## 2.2 Semigroup Presentation

Since BBI models are monoids, we need the way to describe a monoid. We assume the reader is familiar with some basic notions and results.

Let  $X$  be a set of generators or so-called alphabet and  $X^*$  denotes the *free monoid* generated by  $X$ . A relation  $C$  on  $X^*$  is a *congruence* if it is an equivalence relation and whenever  $(v, w) \in C$  then  $(v + u, w + u) \in C$ . For every relation  $R$ , it generates a congruence  $\equiv_R$ , which is the smallest congruence contains  $R$ .

If a semigroup  $M \cong X^* / \equiv_R$ , then the tuple  $(X; R)$  is called a presentation of  $M$ . For a little abuse of language, we write  $M = (X; R)$ .  $M$  is *finitely generated* (f.g.) if there exists a presentation  $(X; R)$  of  $M$  and  $X$  is finite, and is *finitely related* (f.r.) if finite  $R$  exists. By Redei's theorem (cf. [28, 14], also [17]), every f.g. commutative semigroup is f.r. . For a f.g. monoid  $M = (X; R)$ , every element  $m$  in  $M$  is a congruence class in  $X^*$ , denoted by  $[m]_R$ , or  $[m]$  for short.

It is easy to see that a f.g. free commutative monoid is isomorphic to  $\mathbb{N}^k$ , assuming the cardinality of the generator set is  $k$ .



### 3 Undecidability Results

In this section, we show two undecidability results. For a proposition  $p$ , if  $\rho(p)$  is a recursive set and the model is a f.g. free monoid, the satisfiability problem is not decidable. Hence so is the model checking problem. For a given BBI model  $M = (X; R)$ , if  $X$  and  $R$  is infinite, and for every  $p \in P$ ,  $\rho(p) = \{x\}$  for some  $x \in X$ , the model checking problem is also undecidable.

#### 3.1 Recursively Defined Propositions

Obviously, for a proposition  $p$ , if  $\rho(p)$  is not recursive, to check  $m \models p$  is not computable. However, even if  $\rho(p)$  is recursive, the model checking problem is still not computable. Indeed, there is a recursive set that we cannot decide whether it is empty. We will illustrate this using the result of *Hilbert 10th Problem* (H10).

**Proposition 1 (Negative Solution of H10).** *Given a polynomial of several variables  $P(k_1, \dots, k_m)$  with integer coefficients, it is undecidable whether there is a vector  $(k_1, \dots, k_m) \in \mathbb{N}^m$  that  $P(k_1, \dots, k_m) = 0$ .*

Thus, for any given polynomial  $P(k_1, \dots, k_m)$ , let  $X^* \cong \mathbb{N}^m$  and  $\rho(p) = \{x_1^{e_1} \dots x_m^{e_m} \mid P(e_1, \dots, e_m) = 0\}$ . Clearly  $\rho(p)$  is a recursive set since we can compute  $P(e_1, \dots, e_m)$  easily. But to model checking  $\varepsilon \models \top \text{--}^* p$  is equivalent to decide whether the equation  $P(k_1, \dots, k_m) = 0$  has solutions. Hence the model checking problem is undecidable.

Redei's theorem [17] tells that every f.g. monoid is f.r. . But given a recursive relation  $R$ , we cannot compute a finite relation  $R'$  that  $\equiv_R$  is the same as  $\equiv_{R'}$ . To see this, let  $R = \{(x_1^{e_1} \dots x_m^{e_m}, \varepsilon) \mid P(e_1, \dots, e_m) = 0\}$  and again H10 reduces to it. Thus, we cannot decide the structure of a f.g. monoid if the finite generation relation is not given explicitly. In the following, when a monoid  $G = (X; R)$  is finitely generated, we assume that  $R$  is finite.

#### 3.2 Infinitely Generated Monoid

F.g. free monoid is somehow the simplest monoid. It can be easily embedded into infinitely generated monoid and has an empty generation relation set, which is the major obstacle to the model checking problem. Since in this model to model checking BBI with recursively defined propositions is undecidable, later discussion will be restricted on a certain kind of propositions.

In most of the settings, the resource model is discrete and properties of interest can be decomposed into several atomic assertions based on a single piece of the resource. For example, in separation logic, every formula is constructed from atomic assertions interpreted on just one heap cell, like “ $x \mapsto -, -$ ”. Hence, we focus our attention on the proposition which holds only for one generator element. Given a monoid  $M = (X; R)$ , we call a proposition  $p$  “generator proposition”, if  $\rho(p) = \{x\}$ ,  $x \in X$ . It is an analogue of the assertion “ $x \mapsto -, -$ ” in separation logic. In the following, we will use  $p_x$  to denote the proposition which holds on  $x$ .

<sup>1</sup> In the original problem, the vector is required to be in  $\mathbb{Z}^m$ . Here we slightly modify the requirement and it is easy to show these two problems are equivalent.

Many propositions can be constructed via generator propositions with BI connectives. Especially, for a proposition  $p$ , if  $\rho(p)$  or  $M \setminus \rho(p)$  is finite, then it can be expressed through these propositions. The proposition defined in Section 3.1 cannot be expressed via them, since we cannot construct a formula to compute polynomials.

However, in an infinitely generated monoid, even if only generator propositions appear in the formula, the model checking problem is still undecidable. We show this by the reduction from the halting problem of Minsky machine [25]. This technique of encoding Minsky Machine is widely used to prove undecidability results, like the undecidability of full propositional linear logic [23] and bisimulation relation between petri nets [20].

**Definition 5 (Minsky Machine).** A Minsky machine  $C$  with nonnegative counters  $c_1, \dots, c_m$  is a program

$$1 : COMM_1; \dots; n : COMM_n$$

where  $COMM_n$  is a HALT-command and  $COMM_i$  ( $i \in I_{n-1}$ <sup>2</sup>) are commands of the following two types (assuming  $k, k_1, k_2 \in I_n, j \in I_m$ ):

1.  $c_j := c_j + 1; \text{ goto } k$
2. if  $c_j = 0$  then  $\text{ goto } k_1$  else  $(c_j := c_j - 1; \text{ goto } k_2)$

Note that type 2 command is indeed a branch command. We call “if  $c_j = 0$  then  $\text{ goto } k_1$ ” the zero test part and “ $c_j := c_j - 1; \text{ goto } k_2$ ” the decrease part.

Minsky machine is a deterministic computation model. During computation, current value of relating counter determines to take which branch of type-2 command. Every Minsky machine generates a corresponding sequence of executed command number, or so called a run. If the machine halts, the sequence will be finite, ended with  $n$ . Otherwise it will be infinite.

The status of a Minsky machine during the computation can be presented by a tuple  $\{k, c_1, \dots, c_m\}$ , in which  $k$  is the current command line, *i.e.* next command to be executed is  $COMM_k$ , and  $\{c_1, \dots, c_m\}$  expresses the status of counters. The initial state is  $\{1, c_1, \dots, c_m\}$  and  $\{c_1, \dots, c_m\}$  is considered as input. The halting problem of Minsky machine is to decide with empty input, whether the program halts with empty counters. It is known that even if a Minsky machine has only two counters, the halting problem is undecidable. In the following, to construct our reduction, we encode a two-counter Minsky machine in a countably infinitely generated monoid, and express the halting property by a satisfaction relation between an element in that monoid and a BBI formula.

Given Minsky machine  $C$  with two counters  $c_1, c_2$  and commands  $COMM_i$  ( $i \in I_n$ ). We will construct an infinitely generated monoid  $M_C = (X_C; R_C)$  to simulate the execution of  $C$ . Indeed, some congruence classes are corresponding to finite runs.

The generator set  $X_C$  is composed of four parts: set  $Q$  for the current command line,  $A_1$  and  $A_2$  for the current status of the two counters,  $S$  for the current position in a command sequence, and a special generator *halt*.

We let  $Q = \{q_i | i \in I_n\}$ . Here  $q_i$  represents that the next command is  $COMM_i$ . Let  $A_j = \{a_{j,i} | i \in \mathbb{N}\}$  ( $j \in I_2$ ),  $a_{j,i}$  represents that the current value of counter  $c_j$  is  $i$ . In our construction, the product  $q_i \circ a_{1,n} \circ a_{2,m}$  corresponds to the state tuple  $(i, n, m)$ .

<sup>2</sup>  $I_n$  denote the set  $\{1, 2, \dots, n\}$ .

The set  $S$  is a little more complex. Let  $I'_n = \{1', 2', \dots, n'\}$ . We use  $\lambda_k \in (I_n \cup I'_{n-1})^*$  to denote a command sequence of length  $k$  and  $\lambda_k[i]$  to denote the  $i$ th element in  $\lambda_k$ . If  $\lambda_k[i] \in I_{n-1}$ , then it represents the  $i$ th command in this sequence is type 1 or the decrease part of type 2, and if  $\lambda_k[i] \in I'_{n-1}$ , it represents zero test part of type 2.  $\lambda_k[i] = n$  means the  $i$ th command is the HALT-command. We denote the command sequence generated by the Minsky Machine  $C$  by  $\lambda_C$ . Every element in set  $S$  has two indices,  $i$  and  $\lambda_k$ , denoted by  $s_{i,\lambda_k}$ , which means the command sequence is  $\lambda_k$  and the current command is  $\lambda_k[i]$ . It is easy to see that the cardinality of  $X_C$  is countably infinite.

Then we define the generation relation  $R_C$ . Every equation in  $R_C$  corresponds to one step execution of  $C$ . For type-1 command  $COMM_m$  “ $c_j := c_j + 1$ ; goto  $r$ ” ( $j \in I_2$ ), if  $\lambda_k[i] = m$  ( $i \neq k$ ), then  $R_C$  contains the following equations:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,t} = s_{i+1,\lambda_k} \circ q_r \circ a_{j,t+1} (t \in \mathbb{N})$$

For type-2 command  $COMM_m$  “if  $c_j = 0$  then goto  $r_1$  else ( $c_j := c_j - 1$ ; goto  $r_2$ )” ( $j \in I_2$ ), if  $\lambda_k[i] = m$  ( $i \neq k$ ), then:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,t} = s_{i+1,\lambda_k} \circ q_{r_2} \circ a_{j,t-1} (t \in \mathbb{N}^+)$$

or  $\lambda_k[i] = m'$  ( $i \neq k$ ), then:

$$s_{i,\lambda_k} \circ q_m \circ a_{j,0} = s_{i+1,\lambda_k} \circ q_{r_1} \circ a_{j,0}$$

Finally, if  $\lambda_k[k] = n$ , then:

$$s_{k,\lambda_k} \circ q_n \circ a_{1,0} \circ a_{2,0} = s_{k,\lambda_k} \circ halt$$

*Example.* We have finished the construction of the infinitely generated monoid  $M_C = (X_C; R_C)$  corresponding to a Minsky machine  $C$ . Before proceeding to the reduction, let's give a simple example to illustrate how our construction goes. Let a Minsky machine  $C$  as following:

1.  $c_1 := c_1 + 1$ ; goto 3.
2.  $c_2 := c_2 + 1$ ; goto 2.
3. If  $c_1 = 0$  then goto 4 else ( $c_1 := c_1 - 1$ ; goto 2).
4. HALT.

Note that  $C$  never halts, and  $\lambda_C$  is the infinite sequence  $\{1, 3, 2, 2, 2, \dots\}$ .

In our construction,  $Q = \{q_i | i \in I_4\}$ . The generation relation  $R_C$  contains: ( $i \neq k$ )

$$\begin{aligned} s_{i,\lambda_k} \circ q_1 \circ a_{1,t} &= s_{i+1,\lambda_k} \circ q_3 \circ a_{1,t+1} \quad (\text{if } \lambda_k[i] = 1, t \in \mathbb{N}) \\ s_{i,\lambda_k} \circ q_2 \circ a_{2,t} &= s_{i+1,\lambda_k} \circ q_2 \circ a_{2,t+1} \quad (\text{if } \lambda_k[i] = 2, t \in \mathbb{N}) \\ s_{i,\lambda_k} \circ q_3 \circ a_{1,t} &= s_{i+1,\lambda_k} \circ q_2 \circ a_{1,t-1} \quad (\text{if } \lambda_k[i] = 3, t \in \mathbb{N}^+) \\ s_{i,\lambda_k} \circ q_3 \circ a_{1,0} &= s_{i+1,\lambda_k} \circ q_4 \circ a_{1,0} \quad (\text{if } \lambda_k[i] = 3') \\ s_{k,\lambda_k} \circ q_4 \circ a_{1,0} \circ a_{2,0} &= s_{k,\lambda_k} \circ halt \quad (\text{if } \lambda_k[k] = 4) \end{aligned}$$

For  $\lambda_3 = \{1', 3, 5\}$ , we can see that  $[s_{1,\lambda_3} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$  contains only one element, since the first command is not type 2. For  $\lambda_4 = \{1, 3', 2, 2\}$ ,  $s_{1,\lambda_4} \circ q_1 \circ a_{1,0} \circ a_{2,0} = s_{2,\lambda_4} \circ q_3 \circ a_{1,1} \circ a_{2,0}$ . The congruence class contains only these two elements, since the zero test in  $COMM_3$  fails. Generally, the elements contained in the congruence class  $[s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$  correspond to the longest common prefix of  $\lambda_k$  and  $\lambda_C$ .

**Lemma 1.** For  $\lambda_k$ , assume the length of the longest common prefix of  $\lambda_k$  and  $\lambda_C$  is  $k'$ . Every element in congruence class  $[s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$  has the form “ $s_{t,\lambda_k} \circ q_i \circ a_{1,j_1} \circ a_{2,j_2}$ ”, where  $t \leq k' + 1$  and the tuple  $(i, j_1, j_2)$  is the state after executing first  $t - 1$  elements in  $\lambda_C$ , or “ $s_{k,\lambda_k} \circ \text{halt}$ ”, when  $\lambda_k = \lambda_C$ .

*Proof.* It can be verified straightforwardly by induction on  $k'$ .  $\square$

Thus, the Minsky machine  $C$  halts if and only if there exists some  $\lambda_k$ , such that  $s_{k,\lambda_k} \circ \text{halt} \in [s_{1,\lambda_k} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ . All we left to do is to express this property through some satisfaction relation of BBI.

First define  $\phi_{as} = (\neg(\neg\top^* * \neg\top^*)) \wedge (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{\text{halt}})$ . For  $m \models \phi_{as}$ ,  $m \models \neg(\neg\top^* * \neg\top^*)$ . Thus  $m$  cannot be a product of any two non-unit elements in  $M_C$ , it follows that  $m \in X$ . But  $m \models (\bigwedge_i \neg p_{q_i}) \wedge (\neg p_{\text{halt}})$ , it implies that  $m \in S$  or  $m \in A_1 \cup A_2$ . The reverse obviously holds. So  $\rho(\phi_{as}) = S \cup A_1 \cup A_2$ .

We claim that the Minsky machine  $C$  halts if and only if  $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$ , where  $\varphi = \phi_{as} * \exists(p_{\text{halt}} * \phi_{as})$ . Clearly,  $\varphi$  is constructed via generator propositions.

If  $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$  holds, then there exists some  $s_1, s_2 \in S \cup A_1 \cup A_2$ , that  $s_2 \circ \text{halt} \in [s_1 \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ . If  $s_1 \in A_1 \cup A_2$ , then the congruence class  $[s_1 \circ q_1 \circ a_{1,0} \circ a_{2,0}]$  contains only one element since no generation relation can be applied, contradiction. Then  $s_1 \in S$ , without loss of generality we can assume that  $s_1 = s_{1,\lambda_k}$  for some  $\lambda_k$ , since we can discard the elements before  $s_1$  to get a new sequence. By Lemma [1](#),  $s_2 = s_{k,\lambda_k}$ , and hence  $C$  halts. On the other hand, if  $C$  halts and  $|\lambda_C| = k$ , then by Lemma [1](#),  $s_{k,\lambda_C} \circ \text{halt} \in [s_{1,\lambda_C} \circ q_1 \circ a_{1,0} \circ a_{2,0}]$ . Hence  $q_1 \circ a_{1,0} \circ a_{2,0} \models \varphi$  holds.

**Theorem 1.** *The model checking problem for countable infinitely generated monoid against BI formulae in which only generator propositions appear is not decidable.*

*Remarks.* Since total monoid is a special case of partial monoid, our undecidability result is instantly generalized to the case of partial monoid. In separation logic, the underlying model is a partially defined countably infinitely generated monoid. However, unlike our result, the model checking problem for quantifier free assertions of separation logic is decidable. This difference results from the organized structure of the model of separation logic, whereas an arbitrary monoid could be far more chaotic.

## 4 Decidability and Complexity Results

In this section we show that the model checking problem for finite related monoid against BI formulae, under the restriction of generator propositions, is decidable and EXPSpace-complete.

First we claim that for a infinitely generated finitely related monoid, the problem can be reduced to the f.g. case. In fact, there are finitely many generation relations in such a monoid. Thus only finitely many generators will appear in one congruence class. Then there will be only finitely many generators involved in a model checking problem under our assumptions. Every other generator can be treated as the same irrelevant generator.

Formally, given a monoid  $M = (X; R)$ , an element  $m$ , and a formula  $\varphi$ , where  $X$  is infinite,  $R$  is finite, and  $m \in M$ . Without loss of generality, we can assume

that generators appeared in  $m$  and  $R$  and generators whose corresponding propositions appeared in  $\varphi$  are first  $k$  generators. Let  $\delta$  be the homomorphism that maps every other generator to the  $k + 1$ th generator. Then by the induction on the structure of  $\varphi$ , the following lemma holds:

**Lemma 2.**  $m \models_M \varphi$  iff  $m \models_{\delta(M)} \varphi$ .

In the following, we will deal with the f.g. monoid. We prove that in this case, to check  $m \models \varphi$  is equivalent to check whether  $[m]_R$  and some set related to  $\varphi$  in  $X^*$  overlap. Indeed,  $[m]_R$  or every such set is computable semi-linear set and we can decide whether their intersection is empty. To show this, first we cite some notations and results about rational sets and semi-linear sets in [16].

**Definition 6 (Rational Sets).** Let  $M$  be a monoid (not necessarily be commutative). The class of rational subsets of  $M$  is the least class  $\mathcal{E}$  of subsets of  $M$  satisfying the following conditions:

1. The empty set is in  $\mathcal{E}$ ;
2. Each single element set is in  $\mathcal{E}$ ;
3. If  $X, Y \in \mathcal{E}$  then  $X \cup Y \in \mathcal{E}$ ;
4. If  $X, Y \in \mathcal{E}$  then  $X \circ Y \in \mathcal{E}$ ;
5. If  $X \in \mathcal{E}$  then  $X^* \in \mathcal{E}$ .

Here  $X^*$  denotes the submonoid of  $M$  generated by  $X$ . Note that a f.g. monoid  $M$  itself is a rational set.

**Definition 7 (Semi-linear Sets).** A subset  $X = \{a\} \circ B^*$  with  $a \in M$ ,  $B \subseteq M$ , and  $B$  finite, is called linear. A finite union of linear sets is called semi-linear.

Clearly every semi-linear set is completely determined by the series of  $a_i$  and  $B_i$ . Let  $A = \{a_1, \dots, a_n\}$ ,  $B = \{B_1, \dots, B_n\}$ . We call  $(A; B)$  the closed representation of a semi-linear set.

Then we tabulate several useful results from [16].

**Proposition 2.** For a f.g. commutative monoid  $M$ , A subset  $X \subseteq M$  is rational iff it is semi-linear.

**Proposition 3 (Th III, Cor III.1 Cor III.4 in [16]).** If  $X$  and  $Y$  are rational subsets of a commutative monoid  $M$ , then their intersection  $X \cap Y$ , difference  $Y \setminus X$  (hence  $\overline{X} = M \setminus X$ ) and  $Y : X$  are rational.

Recall that  $\rho(\varphi_1 * \varphi_2) = \rho(\varphi_1) \circ \rho(\varphi_2)$ ,  $\rho(\varphi_1 * \exists \varphi_2) = \rho(\varphi_2) : \rho(\varphi_1)$ . By induction, it follows that for all formulae  $\varphi$ ,  $\rho(\varphi)$  is semi-linear set.

Thus, if we can generate a closed representation of  $\rho(\varphi)$  and decide whether  $m$  belongs to it, we can check  $m \models \varphi$ . However, we are not aware of a constructive way to generate the closed representation of  $X \cap Y$ ,  $\overline{X}$ ,  $X \circ Y$ , and  $Y : X$  in the literature. We transform this problem to a corresponding problem in  $X^*$ , in order to avoid the complicated structure in  $M$ , which is caused by  $R$ .

Consider the canonical surjective morphism  $\alpha : X^* \mapsto M$ . It is easy to see that  $\alpha^{-1}(m) = [m]_R$  and  $m \in \rho(\varphi)$  is equivalent to  $[m]_R \subseteq \alpha^{-1}(\rho(\varphi))$ . If  $\exists x, x \in$

$[m]_R \cap \alpha^{-1}(\rho(\varphi))$ , then  $\alpha(x) \in \rho(\varphi)$ . Thus,  $[m]_R = [x]_R \subseteq \alpha^{-1}(\rho(\varphi))$ . It implies that  $m \in \rho(\varphi)$  is equivalent to  $[m]_R \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset$ .

Next we will show how to decide whether  $[m]_R \cap \alpha^{-1}(\rho(\varphi)) \neq \emptyset$ . Indeed,  $[m]_R$  is also a semi-linear set. In [21], an algorithm has been developed to compute the closed representation of a congruence class with the cost of at most exponential space.

**Proposition 4 (Th.10 in [21]).** *Let f.g. monoid  $M = (X; R)$ ,  $m \in M$ . There is an algorithm which generates a closed representation of  $[m]_R$  using at most space  $2^{c \cdot \text{size}(m, R)}$ , where  $c > 0$  is some constant independent of  $m$  and  $R$ .*

Thus, we can generate the representation of  $[m]_R$  for every  $m \in M$ . In order to compute the closed representation of  $\alpha^{-1}(\rho(\varphi))$ , we need to make induction on the structure of  $\varphi$ . It is easy to verify the following lemma.

**Lemma 3.** *For a f.g. monoid  $M = (X; R)$  and BI formulae  $\varphi$ ,  $\varphi_1$ , and  $\varphi_2$ , the following holds:*

- $\alpha^{-1}(\rho(p_x)) = [x]_R$
- $\alpha^{-1}(\rho(\top)) = X^*$
- $\alpha^{-1}(\rho(\neg\varphi)) = \overline{\alpha^{-1}(\rho(\varphi))}$
- $\alpha^{-1}(\rho(\varphi_1 \wedge \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \cap \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\top^*)) = [\varepsilon]_R$
- $\alpha^{-1}(\rho(\varphi_1 * \varphi_2)) = \alpha^{-1}(\rho(\varphi_1)) \circ \alpha^{-1}(\rho(\varphi_2))$
- $\alpha^{-1}(\rho(\varphi_1 * \exists \varphi_2)) = \alpha^{-1}(\rho(\varphi_2)) : \alpha^{-1}(\rho(\varphi_1))$

Since  $\alpha^{-1}(\rho(p_x)) = [x]_R$ , Proposition 4 also builds up the basis of our induction. To compute the closed representations of  $\overline{X}$ ,  $X \cap Y$ ,  $X \circ Y$ , and  $X : Y$ , we consider the case of  $\mathbb{N}^k$ , since for a generator set  $X$ ,  $|X| = k$ ,  $X^*$  and  $\mathbb{N}^k$  are isomorphic.

The case of generating the closed representation of  $\overline{X}$  is the most complicated. We need to compute the representation of a set in which every element is not larger than any element in a given set.

Formally, define a partial order  $\leq$  on  $\mathbb{N}^k$ . For two vectors  $v = \{v_1, \dots, v_k\}$ ,  $v' = \{v'_1, \dots, v'_k\}$  in  $\mathbb{N}^k$ ,  $v \leq v'$  iff  $\forall i, v_i \leq v'_i$ , and  $v < v'$  iff  $v \leq v'$  and  $\exists i, v_i \neq v'_i$ .

**Lemma 4.** *For a set of vectors  $B = \{b_1, \dots, b_n\}$  in  $\mathbb{N}^k$ , the set  $m(B) = \{a \mid a \in \mathbb{N}^k, \forall b_i \in B, a < b_i \text{ or } a \text{ and } b_i \text{ are not comparable.}\}$  is a semi-linear set, and there is an algorithm which generates a closed representation of it.*

*Proof (sketch).* Consider the  $k$ th component of one element in  $m(B)$ . It must be sandwiched between the counterparts of two divisions of the vectors. Then the first  $k - 1$  components of it should be smaller than or not comparable with the counterparts of the smaller division. These elements compose an instance of lower dimension. Thus, the closed representation can be generated inductively on the dimension  $k$ .  $\square$

**Lemma 5.** *For two semi-linear sets  $X, Y \subseteq \mathbb{N}^k$ , given their closed representation  $(A_X; B_X)$ ,  $(A_Y; B_Y)$ , there is an algorithm which generates a closed representation of  $\overline{X}$ ,  $X \cap Y$ ,  $X + Y$ , and  $Y - X$ .  $\square$*

<sup>3</sup> Note that the addition in  $\mathbb{N}^k$  corresponds to the multiplication in free monoid. Thus,  $X + Y$  and  $Y - X$  correspond  $X \circ Y$  and  $Y : X$ , respectively.

*Proof.* For two semi-linear sets  $X = \bigcup_i (a_i + B_i^*)$  and  $Y = \bigcup_j (a_j + B_j^*)$ , it is easy to see that

$$\begin{aligned} X + Y &= \bigcup_{i,j} ((a_i + B_i^*) + (a_j + B_j^*)) \\ X \cap Y &= \bigcup_{i,j} ((a_i + B_i^*) \cap (a_j + B_j^*)) \\ Y - X &= \bigcup_{i,j} ((a_j + B_j^*) - (a_i + B_i^*)) \\ \overline{X} &= \bigcap_i (a_i + B_i^*) \end{aligned}$$

Then we only have to deal with linear sets.

“ $X + Y$ ”: For two linear sets  $a + B^*$  and  $a' + B'^*$ , their summation is:

$$(a + a') + (B \cup B')^*$$

“ $X \cap Y$ ”: For two linear sets  $a + B^*, a' + B'^* \subseteq \mathbb{N}^k$ . Assume  $B = \{b_1, \dots, b_n\}$  and  $B' = \{b'_1, \dots, b'_{n'}\}$ , then every element in  $X \cap Y$  corresponds to two vectors  $x_i, x'_i \in \mathbb{N}^k$ , which satisfies the following system of linear diophantine equations:

$$\sum_{i=1}^n b_i x_i - \sum_{j=1}^{n'} b'_j x'_j = a' - a$$

Indeed, the solution of this system forms a semi-linear set, and there are many algorithms devoted to this problem, e.g. [15][22].

“ $Y - X$ ”: For two linear sets  $X = a + B^*$  and  $Y = a' + B'^*$ , assume  $B = \{b_1, \dots, b_n\}$  and  $B' = \{b'_1, \dots, b'_{n'}\}$ . We can see that

$$Y - X = \{(a' - a) + \sum_{i=1}^{n'} (t'_i b'_i) - \sum_{j=1}^n (t_j b_j) \mid t'_i, t_j \in \mathbb{N}\} \cap \mathbb{N}^k$$

It is similar to the “ $X \cap Y$ ” case. We can get the representation after solving the system of linear diophantine equations:

$$(a' - a) + \sum_{i=1}^{n'} (t'_i b'_i) - \sum_{j=1}^n (t_j b_j) = \sum_{i=1}^k x_i e_i$$

in which  $t'_i, t_i, x_i$  are variables.

“ $\overline{X}$ ”: Assume  $X = a + B^*$ . For  $x \in \mathbb{N}^k$ , if  $\exists b_i \leq x$ , then we can get the vector  $x - b_i$  still in  $\mathbb{N}^k$ . Repeat this operation and finally we will get some  $x' \in m(B)$ . By Lemma 4, a closed representation of  $m(B)$  is computable, denoted by  $\bigcup_j (a_j + B_j^*)$ . Thus,  $\mathbb{N}^k$  is decomposed as:

$$\mathbb{N}^k = \bigcup_j (a_j + B_j^* + B^*)$$

Assume we get  $a' \in m(B)$  from  $a, a = a' + \sum_{i=1}^n r_i b_i$ , and  $a' \in a_t + B_t^*$ . Such  $a'$  might not be unique but computable and the quantity is finite. Since  $X \subseteq (a_t + B_t^* + B^*)$ ,  $\overline{X}$  can be expressed as:

$$\overline{X} = \bigcup_{a \in a_t + B_t^* + B^*} ((a_t + B_t^* + B^*) \setminus X) \cup \bigcup_{j \neq t} (a_j + B_j^* + B^*)$$

$(a_t + B_t^*) \cap B^* = \emptyset$  follows that  $(a_t + B_t^* + B^*) \setminus X$  contains two part: elements that do not belong to  $a' + B^*$ , and elements that belong to  $a' + B^*$  but are smaller than  $a$ . The former can be expressed as  $((a_t + B_t^*) \setminus \{a'\}) + B^*$ , and the latter is finite, all semi-linear sets. It is easy to see that if there are more than one such  $a'$  belong to the same  $a_t + B_t^*$ , we only need to consider one of them. This concludes our argument.  $\square$

Mention that to decide whether two semi-linear sets overlap, we only need to compute their intersection, which is already solved. Thus, we have provided a way to decide whether  $[m]_R \subseteq \alpha^{-1}(\rho(\varphi))$ , which is equivalent to  $m \models \varphi$ . It follows that the model checking problem in this case is decidable.

As stated in Proposition 4, generating the closed representation of a congruence class or the set defining a proposition costs exponential space. Solving the system of linear diophantine system and other operations do not exceed the exponential space upper bound. Thus the overall space cost is at most exponential w.r.t. the length of  $\varphi$  and the sizes of  $m, R$ , and all propositions appeared in  $\varphi$ .

To get the complexity lower bound, we need to introduce the word problem of monoid. For a monoid, the word problem is to decide whether two words are in the same congruence class. In a f.g. commutative monoid  $M = (X; R)$ , for two words  $u$  and  $v$ , if  $u = \prod_{i=1}^n x_i^{r_i}$  and  $v = \prod_{i=1}^n x_i^{s_i}$  ( $r_i, s_i \in \mathbb{N}$ ), then the word problem is equal to check whether  $v \models \prod_{i=1}^n p_{x_i}^{r_i}$  or  $\varepsilon \models \prod_{i=1}^n p_{x_i}^{r_i} * \prod_{i=1}^n p_{x_i}^{s_i}$ . It is known that the word problem in a commutative monoid is EXPSPACE-complete (cf. [24]). Thus, the model checking problem is EXPSPACE-hard.

In summary, the model checking problem for f.g. monoids under our restriction of propositions is EXPSPACE-complete. Together with Lemma 2 and Redei's theorem [17], we conclude that:

**Theorem 2.** *The model checking problem for finitely related monoid against BI formulae in which only generator propositions appear is EXPSPACE-complete.*

*Remarks.* If we want to do model checking in a partial monoid, first define a special element  $\pi$  as stated before (Section 2). Then generate  $\rho(\varphi)$  normally, except that after computing every representation of the set specified by the subformula of  $\varphi$ , eliminate the component corresponding to  $\pi$ . Compute the congruence class  $[m]$  normally and it is easy to see it does not contain  $\pi$ . Thus the partial monoid can be model checked like total monoid.

## 5 Additional Remarks

In this section we provide additional remarks, along with some discussion of related works and future work.

### 5.1 Fragments and Complexity

It is natural to ask whether the complexity lower bound of EXPSPACE could be reduced if we only concern about some fragment of BI or for some special monoid. From our



reduction, as long as the multiplicative conjunction or implication is considered, the complexity cannot be lower.

For f.g. free monoid under our restriction of propositions, the model checking problem is PSPACE-complete. Indeed, the PSPACE-hardness follows from the result in [8], since their proof of PSPACE-hardness does not employ any essential property of the predicate or the partiality. For example, we can treat  $x \mapsto \text{nil}, \text{nil}$  as generator proposition  $p_x$ . The upper bound results from the fact that the cost of exponential space is caused by computing the congruence class, which is a singleton set in free monoid.

## 5.2 Automata Theory

Recall that Kleene theorem asserts that in a free commutative monoid, the class of rational set is equal to the class of set which can be recognized by finite automata. It is shown that in the case of finitely generated commutative monoid, Kleene Theorem holds in a monoid iff it is rational (cf. [30]). For a BI formula  $\varphi$ ,  $\rho(\varphi)$  is a rational set in f.g. monoid. Thus  $\rho(\varphi)$  is recognizable by finite automata in a rational monoid. If we extend BI with a new connective to characterize the set  $X^*$  in the monoid, then a set  $S$  is rational iff there is a  $\varphi$  that  $S = \rho(\varphi)$ . Hence the language generated by this kind of BI formula cannot be recognizable by finite automata in non-rational monoid. As a comparison, it is shown that all languages generated by context logic formulae are recognizable by finite automata (cf. [5]).

## 5.3 Model Checking Mobile Ambient

In [12,13,11], it is shown that if the calculus contains repetition or the logic contains guarantee, the model checking problem is not decidable. The model they treated is a free monoid if we omit the nesting of ambients. The guarantee connective is a counterpart of multiplicative implication in BBI. The repetition is just a counterpart of the connective discussed in Section 5.2 above. Introducing such a connective in BBI does not affect the decidability of model checking problem for rational monoids, and hence free monoids. Thus, the undecidability of their problem resulted from the nesting of ambients.

## 5.4 Model Checking BI and CBI

Our discussion is restricted in the domain of BBI. In the general monoid model of BI, the interpretation of a formula is intuitionistic, according to a partial order, which makes the structure of the model and the model checking problem more complicated. The result showed here is just a special case. Maybe some requirement like ascending chain condition is needed to obtain a decidability result.

Another line of future work is to solve the model checking problem for Classical BI (CBI) (cf. [4]), which is an extension of BBI. In CBI, the model is more organized, similar as an inverse monoid or a cancellative monoid. Thus, the decidable condition might be loosened and the complexity might be reduced. Furthermore, the model adopted in CBI is relational monoid. It is a generalization of partial and total monoid and was introduced in [18]. The model checking problem in this semantic setting needs further analysis.

## Acknowledgements

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No.2009CB320701, the National Natural Science Foundation of China under Grant No.60873061, and the National 863 Plans Projects of China under Grant No.2006AA01Z160.

## References

1. Berdine, J., Calcagno, C., O'Hearn, P.W.: A decidable fragment of separation logic. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 97–109. Springer, Heidelberg (2004)
2. Biri, N., Galmiche, D.: A separation logic for resource distribution. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 23–37. Springer, Heidelberg (2003)
3. Bozga, M., Iosif, R., Perarnau, S.: Quantitative separation logic and programs with lists. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 34–49. Springer, Heidelberg (2008)
4. Brotherston, J., Calcagno, C.: Classical bi: a logic for reasoning about dualising resources. In: Proc. 36th ACM Symp. Principles of Prog. Lang (POPL 1909), pp. 328–339. ACM Press, New York (2009)
5. Calcagno, C., Dinsdale-Young, T., Gardner, P.: Decidability of context logic (unpublished) (2008), <http://www.doc.ic.ac.uk/~ccris/ftp/decidCL.pdf>
6. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic and tree update. In: Proc. 32th ACM Symp. Principles of Prog. Lang (POPL 2005), pp. 271–282. ACM Press, New York (2005)
7. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic as modal logic: completeness and parametric inexpressivity. In: Proc. 34th ACM Symp. Principles of Prog. Lang (POPL 2007), pp. 123–134. ACM Press, New York (2007)
8. Calcagno, C., Yang, H., O'Hearn, P.W.: Computability and complexity results for a spatial assertion language for data structures. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 108–119. Springer, Heidelberg (2001)
9. Cardelli, L., Gordon, A.D.: Anytime, anywhere: Modal logics for mobile ambients. In: Proc. 27th ACM Symp. Principles of Prog. Lang (POPL 2000), pp. 365–377 (2000)
10. Cardelli, L., Gordon, A.D.: Mobile ambients. *Theor. Comput. Sci.* 240(1), 177–213 (2000)
11. Charatonik, W., Dal Zilio, S., Gordon, A.D., Mukhopadhyay, S., Talbot, J.-M.: The complexity of model checking mobile ambients. In: Honsell, F., Miculan, M. (eds.) FOSSACS 2001. LNCS, vol. 2030, pp. 152–167. Springer, Heidelberg (2001)
12. Charatonik, W., Dal-Zilio, S., Gordon, A.D., Mukhopadhyay, S., Talbot, J.-M.: Model checking mobile ambients. *Theor. Comput. Sci.* 308(1-3), 277–331 (2003)
13. Charatonik, W., Talbot, J.-M.: The decidability of model checking mobile ambients. In: Fribourg, L. (ed.) CSL 2001 and EACSL 2001. LNCS, vol. 2142, pp. 339–354. Springer, Heidelberg (2001)
14. Clifford, A.H., Preston, G.B.: *The Algebraic Theory of Semigroups*, vol. 2. The American Mathematical Society (1967)
15. Domenjoud, E., Lorraine, I.: Solving systems of linear diophantine equations: An algebraic approach. In: Tarlecki, A. (ed.) MFCS 1991. LNCS, vol. 520, pp. 141–150. Springer, Heidelberg (1991)
16. Eilenberg, S., Schutzenberger, M.-P.: Rational sets in commutative monoids. *J. Algebra* 13(2), 173–191 (1969)

17. Freyd, P.: Redei's finiteness theorem for commutative semigroups. *Proc. of the AMS* 19, 1003 (1968)
18. Galmiche, D., Larchey-Wendling, D.: Expressivity properties of boolean BI through relational models. In: Arun-Kumar, S., Garg, N. (eds.) *FSTTCS 2006*. LNCS, vol. 4337, pp. 357–368. Springer, Heidelberg (2006)
19. Galmiche, D., Méry, D., Pym, D.J.: Resource tableaux (Extended abstract). In: Bradfield, J.C. (ed.) *CSL 2002 and EACSL 2002*. LNCS, vol. 2471, pp. 183–199. Springer, Heidelberg (2002)
20. Jančar, P.: Undecidability of bisimilarity for petri nets and some related problems. *Theor. Comput. Sci.* 148(2), 281–301 (1995)
21. Koppenhagen, U., Mayr, E.W.: The complexity of the coverability, the containment, and the equivalence problems for commutative semigroups. In: Chlebus, B.S., Czaja, L. (eds.) *FCT 1997*. LNCS, vol. 1279, pp. 257–268. Springer, Heidelberg (1997)
22. Lankford, D.: Non-negative integer basis algorithms for linear equations with integer coefficients. *J. Automated Reasoning* 5(1), 25–35 (1989)
23. Lincoln, P., Mitchell, J.C., Scedrov, A., Shankar, N.: Decision problems for propositional linear logic. In: *Proc. 31st Symp. Found. of Comp. Sci (FOCS 1990)*, vol. II, pp. 662–671. IEEE Computer Society Press, Los Alamitos (1990)
24. Mayr, E., Meyer, A.: The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.* 46(3), 305–329 (1982)
25. Minsky, M.L.: *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs (1967)
26. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. *Bulletin of Symbolic Logic* 5(2), 215–244 (1999)
27. Pym, D.J., O'Hearn, P.W., Yang, H.: Possible worlds and resources: the semantics of bi. *Theor. Comput. Sci.* 315(1), 257–305 (2004)
28. Redei, L.: *The Theory of Finitely Generated Commutative Semigroups*. Oxford University Press, Oxford (1965)
29. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: *Proc. 17th IEEE Symp. Logic in Comp. Sci (LICS 2002)*, pp. 55–74. IEEE Computer Society, Los Alamitos (2002)
30. Rupert, C.P.: On commutative kleene monoids. *Semigroup Forum* 43(1), 163–177 (1991)

# Efficient Type-Checking for Amortised Heap-Space Analysis

Martin Hofmann and Dulma Rodriguez

Department of Computer Science, University of Munich  
Oettingenstr. 67, D-80538 München, Germany  
{martin.hofmann,dulma.rodriguez}@ifi.lmu.de

**Abstract.** The prediction of resource consumption in programs has gained interest in the last years. It is important for a number of areas, notably embedded systems and safety critical systems. Different approaches to achieve bounded resource consumption have been analysed. One of them, based on an amortised complexity analysis, has been studied by Hofmann and Jost in 2006 for a Java-like language.

In this paper we present an extension of this type system consisting of more general subtyping and sharing relations that allows us to type more examples. Moreover we describe efficient automated type-checking for a finite, annotated version of the system. We prove soundness and completeness of the type checking algorithm and show its efficiency.

**Keywords:** Type systems, Resource analysis, Semantics, OOP.

## 1 Introduction

The prediction of resource consumption in programs has gained interest in the last years. It is important for a number of areas, in particular embedded systems and mobile computing. A variety of approaches to resource analysis have been proposed based in particular on recurrence solving [AAG<sup>+</sup>07, Gro01], abstract interpretation [GL98, NCQR05], sized types [HP99], and amortised analysis [HJ03, HJ06, Cam08].

The amortised approach which the present paper belongs to is particularly useful in situations where heap-allocated data structures must be costed whose size is proportional to parts of the input. Typical examples are various sorting algorithms where trees, lists, or heaps appear as intermediate data structures. In such cases amortised analysis can infer very good bounds based on intuitive programmer annotations in the form of types and the solution of linear inequations.

In [HJ06] amortised analysis has been applied to a Java-like class-based object-oriented language without garbage collection, but with explicit deallocation similar to C's `free()`. The evaluation of such programs is carried out by maintaining a set of free memory units called freelist. When an object is created, a number of heap units required to store it is taken from the freelist if it contains enough units, otherwise the program execution is aborted. Finally, each deallocated heap unit is returned to the freelist.

The goal of the analysis is to predict a bound on the initial size that the freelist must have so that a given program may be executed without causing unsuccessful abortion due to insufficient memory. This has been achieved by combining amortized analysis [Tar85, Oka98] with type-based techniques in order to define potentials.

Essentially each object is ascribed an abstracted portion of the freelist, referred to as *potential*, which is just a number, denoting the size of freelist portion associated with the object. Any object creation must be paid for from the potential in scope. The initial potential thus represents an upper bound on the total heap consumption.

While type inference and automated type checking have already been developed for a functional language within the EmBounded Project ([HDF<sup>+</sup>05], [HBH<sup>+</sup>07]), most of the properties of the type system for the Java-like language (called Resource Aware JAva – RAJA) are still unknown.

This paper provides algorithmic typing rules for that system. We prove soundness and completeness of algorithmic typing with respect to the declarative typing from [HJ06]. This allows for automatic type checking under relatively mild annotations. In particular, we automatically construct types arising from sharing and conditionals which had to be provided manually beforehand. This enables a realistic implementation of the system which we also provide.

The notion of subtyping we use is slightly more flexible than the one from [HJ06] and thus allows more examples to be typed. Semantic soundness of the improved system is a direct extension of the soundness proof in [HJ06] and can be found in the following manuscript: [HJR].

*Contents.* Section 2 describes briefly the system RAJA and motivates it with some examples. In Section 3 we define the type-checking algorithm and show its soundness and completeness w.r.t. the declarative system. We then argue that typechecking can be performed efficiently, i.e. in small-degree polynomial time. Finally, in Sections 4 and 5, we discuss future and related work.

## 2 FJEU and RAJA

Our formal model of Java, FJEU, is an extension of Featherweight Java (FJ) [IPW99] with attribute update, conditional and explicit deallocation. It is thus similar to Classic Java [FKF98]. An FJEU program  $\mathcal{C}$  is a partial finite map from class names to class definitions, which we also refer to as *class table*. Each class table  $\mathcal{C}$  implies a subtyping relation  $<$ : among the class names in the standard way by inheritance. The syntax of FJEU is given in Fig. 1. The let-normal form of terms was merely chosen to eliminate boring redundancies from our proofs. In our implementation we transform nested expressions into let-normal form and infer a type for the let expressions by a simple preprocessing.

We will use a couple of shorthand notations: We write  $S(C)$  to denote the *super-class*  $D$  of a class  $C$ , provided that  $C$  has a super-class. We write  $A(C)$  to denote the ordered set of attributes of  $C$ , including inherited ones, i.e.  $A(C) := \{a_1, \dots, a_k\} \dot{\cup} A(D)$ . We write  $C.a_i$  to denote the class type of each attribute  $a_i$  of class  $C$ . Similarly we write  $M(C)$  to denote the set of all defined method names

```

c ::= class C [extends D] {A1; ...; Ak; M1 ... Mj}
A ::= C a
M ::= C0 m(C1 x1, ..., Cj xj){return e;}
e ::= x                               (Variable)
    | null                             (Constant)
    | new C                             (Construction)
    | free(x)                           (Destruction)
    | (C)x                               (Cast)
    | x.ai                             (Access)
    | x.ai<-x                           (Update)
    | x.m(x1, ..., xj)                 (Invocation)
    | if x instanceof C then e1 else e2 (Conditional)
    | let C x = e1 in e2               (Let)

```

**Fig. 1.** The syntax of FJEU

of  $C$ , including inherited ones. For a method  $m$  of class  $C$  we write  $M_{\text{body}}(C, m)$  to denote the term that comprises the *method body* of method  $m$  and  $C.m$  to denote the *method type* of  $m$  in class  $C$ . We base our statical resource analysis on the standard operational semantics that can be found in [HJR].

*Example 1 (Copy of singly-linked lists).* Suppose we have defined a class of singly-linked lists in an object-oriented style which harnesses dynamic dispatch to obtain the functionality of pattern-matching. Most programmers would use this style only for more complex tree-like data structures relying on “null” to model the empty list. We use it here in order to have a simple enough running example.

```

class List { List copy(){return null;} }
class Nil extends List { List copy() { return this; }}
class Cons extends List { int elem; List next;
  List copy(){ let List res = new Cons in
    let List res1 = res.elem <- this.elem in
    let List res2 = res1.next <- this.next.copy() in return res2;}}

```

## 2.1 The System RAJA

**Definition 1.** A RAJA program is an annotation of an FJEU class table  $\mathcal{C}$  in the form of a sextuple  $\mathcal{R} = (\mathcal{C}, \mathcal{V}, \langle \cdot \rangle, \text{A}^{\text{get}}(\cdot, \cdot), \text{A}^{\text{set}}(\cdot, \cdot), \text{M}(\cdot, \cdot))$  specified as follows:

$\mathcal{V}$  is a possibly infinite set of views. A RAJA class or refined type consists of a class  $C$  and a view  $r$  and is written  $C^r$ . We use the letters  $r, s, p, q$  to denote views. The meaning of views is given by the maps:

1.  $\langle \cdot \rangle$  assigns to each RAJA class  $C^r$  a number  $\langle \mathcal{C}^r \rangle \in \mathbb{D}$ , where  $\mathbb{D} = \mathbb{Q}^+ \cup \infty$ .
2.  $\text{A}^{\text{get}}(\cdot, \cdot)$  and  $\text{A}^{\text{set}}(\cdot, \cdot)$  assign to each RAJA class  $C^r$  and attribute  $a \in \text{A}(C)$  two views  $q = \text{A}^{\text{get}}(C^r, a)$  and  $s = \text{A}^{\text{set}}(C^r, a)$ .
3.  $\text{M}(\cdot, \cdot)$  assigns to each RAJA class  $C^r$  and method  $m \in \text{M}(C)$  having method type  $E_1, \dots, E_j \rightarrow E_0$  a  $j$ -ary polymorphic RAJA method type  $\text{M}(C^r, m)$ . A  $j$ -ary polymorphic RAJA method type is a (possibly empty or infinite) set of  $j$ -ary monomorphic RAJA method types. A  $j$ -ary monomorphic RAJA method

type consists of  $j+1$  views and two numbers  $p, q \in \mathbb{D}$ , written  $r_1, \dots, r_j \xrightarrow{p/q} r_0$ . We sometimes write  $E_1^{r_1}, \dots, E_j^{r_j} \xrightarrow{p/q} E_0^{r_0}$  to denote an FJEU method type combined with a corresponding monomorphic RAJA method type.

We introduce views and RAJA classes because we want to be able to assign objects of the same class different potentials. The number  $\langle\langle \cdot \rangle\rangle$  will be used to define the potential of a heap configuration under a given static RAJA typing. The exact definition is omitted here for lack of space and can be found in [HJR]. Essentially, the potential of a program state is the sum of the annotations of all its objects determined by their RAJA-type. Each access path (alias) to an object makes a separate contribution to that sum. In reasonable typings of circular data structures one arranges that all but finitely many paths make a nonzero contribution.

If  $D = C.a$  is the FJEU type of attribute  $a$  in  $C$  then the RAJA class  $D^{A^{sev}(C^r.a)}$  will be the type used when reading  $a$ , whereas the (intendedly stronger) type  $D^{A^{sev}(C^r.a)}$  must be used when updating  $a$ . The stronger typing is needed since an update will possibly affect several aliases.

If a method  $m$  has a RAJA method type  $E_1^{r_1}, \dots, E_j^{r_j} \xrightarrow{p/q} E_0^{r_0}$  then it may be called with arguments  $v_1 : E_1^{r_1}, \dots, v_j : E_j^{r_j}$ , whose associated potential will be consumed, as well as an additional potential of  $p$ . Upon successful completion the return value will be of type  $E_0^{r_0}$  hence carry an according potential. In addition to this a potential of another  $q$  units will be returned.

*Example 2 (RAJA annotation of copy of singly-linked lists).*

We aim at analysing the heap-space requirements of the program of Example 1. It is clear that the memory consumption of a call `l.copy()` will equal the length of the list `l`. To calculate this formally we use a view rich which assigns to `List` itself the potential 0, to `Nil` the potential 0 and to `Cons` the potential 1. Another view is needed to describe the result of `copy()` for otherwise we could repeatedly copy lists without paying for it. Thus, we introduce another view `poor` that assigns potential 0 to all classes. In the following we show the RAJA annotation of Example 1 in the syntax of our implementation.

```
class List { rich, poor : pot = 0;
  rich : List<poor>,0 copy() { return null; }
}
class Nil extends List { rich, poor : pot = 0;
  rich : List<poor>,0 copy() { return this; }
}
class Cons extends List { rich : pot = 1; poor : pot = 0;
  rich : List<rich,rich> next;
  poor : List<poor,poor> next;
  rich, poor: int elem;

  rich : List<poor>,0 copy() { let List res = new Cons in
    let List res1 = res.elem <- this.elem in
    let List res2 = res1.next <- this.next.copy() in return res2; }
}
```

The RAJA type of the method `copy` states that it is only defined in  $\text{List}^{\text{rich}}$ ,  $\text{Nil}^{\text{rich}}$  and  $\text{Cons}^{\text{rich}}$ , but not in, e.g.,  $\text{List}^{\text{poor}}$ . It will consume the potential of this and no additional potential. Upon successful completion the return value will be of type  $\text{List}^{\text{poor}}$  hence carry potential 0. In addition to this the method will return no more potential. Thus, the typing amounts to saying that the memory consumption of every call to `copy` is bounded by the potential of this, that in case of  $\text{Cons}^{\text{rich}}$  is equal 1 and in case of  $\text{Nil}^{\text{rich}}$  is equal 0. If a list of length  $n$  is to be copied, the method will be called  $n + 1$  times, and the potential consumed will be bounded by  $n$ . More examples can be found in the RAJA web page [\[raj\]](#).

**RAJA Subtyping Relation.** RAJA subtyping is an extension of FJEU subtyping ( $<$ ), which is based on inheritance. We provide here a new definition of subtyping w.r.t. [\[HJ06\]](#). There, a subtyping relation  $r \sqsubseteq s$  on views was defined, based on all classes of the class table. Then, subtyping of RAJA classes  $C^r <: D^s$  was defined as  $C <: D$  and  $r \sqsubseteq s$ . This made subtyping unnecessarily rigid. For example  $\text{Nil}^{\text{rich}} <: \text{Nil}^{\text{poor}}$  did not hold because  $\text{rich} \sqsubseteq \text{poor}$  did not hold due to the class `Cons`. The new subtyping relation is defined directly on refined types. However, the straightforward definition where  $C^r <: D^s$  only depends on  $C$  and  $D$  is unsound. It is necessary to analyse the subclasses of  $C$  and  $D$  as well because resource usage is determined by the dynamic type of the expressions.

**Definition 2 (Subtyping of RAJA types).** *We define a preorder  $<$ : on RAJA types  $C^r, D^s$  where  $C <: D$  in  $\mathcal{C}$  and  $r, s \in \mathcal{V}$ , as the largest relation ( $C^r <: D^s$ ) such that  $C^r <: D^s \iff$  for each  $E <: C, F <: D$  with  $E <: F$ :*

$$\langle\langle E^r \rangle\rangle \geq \langle\langle F^s \rangle\rangle \quad (2.1)$$

$$\forall a \in \mathbf{A}(F) . (F.a)^{\text{A}^{\text{set}}(E^r, a)} <: (F.a)^{\text{A}^{\text{set}}(F^s, a)} \quad (2.2)$$

$$\forall a \in \mathbf{A}(F) . (F.a)^{\text{A}^{\text{set}}(F^s, a)} <: (F.a)^{\text{A}^{\text{set}}(E^r, a)} \quad (2.3)$$

$$\forall m \in \mathbf{M}(F) . \forall \beta \in \mathbf{M}(F^s, m) . \exists \alpha \in \mathbf{M}(E^r, m) . (F.m)^\alpha <: (F.m)^\beta \quad (2.4)$$

where we extend  $<$ : to monomorphic RAJA method types as follows:

**Definition 3 (Subtyping of RAJA methods).** *If  $D.m = E_1, \dots, E_j \rightarrow E_0$ ,  $\alpha = r_1, \dots, r_j \xrightarrow{p/q} r_0$  and  $\beta = s_1, \dots, s_j \xrightarrow{t/u} s_0$  then  $(D.m)^\alpha <: (D.m)^\beta$  is defined as  $p \leq t$  and  $q \geq u$  and  $E_0^{s_0} <: E_0^{r_0}$  and  $E_i^{s_i} <: E_i^{r_i}$  for  $i = 1, \dots, j$ .*

**Sharing Relation.** The sharing relation  $\mathbb{Y}(\cdot | \cdot)$  is important for correctly using variables more than once. In a RAJA program, if a variable is to be used more than once, then the different occurrences must be given different types which are chosen such that the individual potentials assigned to each occurrence add up to the total potential available for that variable. For example if we have  $l : \text{List}^{\text{rich}}$  we can use the variable  $l$  with the types  $\text{List}^{s_1}$  and  $\text{List}^{s_2}$  if  $\mathbb{Y}(\text{List}^{\text{rich}} | \text{List}^{s_1}, \text{List}^{s_2})$  holds. In [\[HJ06\]](#) sharing was defined on views, i.e.  $\mathbb{Y}(r | s_1, \dots, s_n)$  which is less flexible and precludes several examples.



**Definition 4 (Sharing Relation).** We define the sharing relation between a single RAJA type  $C^r$  and a multiset of RAJA types  $D^{s_1}, \dots, D^{s_n}$  written  $\forall(C^r | D^{s_1}, \dots, D^{s_n})$  as the largest relation  $\forall$ , such that if  $\forall(C^r | D^{s_1}, \dots, D^{s_n})$  then for all  $E <: C$ ,  $F <: D$  with  $E <: F$ :

$$\diamond(E^r) \geq \sum_i \diamond(F^{s_i}) \quad (2.5)$$

$$\forall i. E^r <: F^{s_i} \quad (2.6)$$

$$\forall a \in \text{dom}(A(F)). \forall \left( (F.a)^{A^{\text{get}}(E^r, a)} \mid (F.a)^{A^{\text{get}}(F^{s_1}, a)}, \dots, (F.a)^{A^{\text{get}}(F^{s_n}, a)} \right) \quad (2.7)$$

We define sharing similarly to subtyping, so that the following can be proved: subtyping and sharing coincide when the multiset of RAJA types consists of only one element.

**Lemma 1.**  $C^r <: C^s \iff \forall(C^r | C^s)$

**Typing RAJA.** The RAJA-typing judgment is formally defined by the rules in Figure 2. The type system allows us to derive assertions of the form  $\Gamma \frac{n}{n'} e : C^r$  where  $e$  is an expression or program phrase,  $C$  is an FJEU class,  $r$  is a view (so  $C^r$  is a refined type).  $\Gamma$  maps variables occurring in  $e$  to refined types; we often write  $\Gamma_x$  instead of  $\Gamma(x)$ . Finally  $n, n'$  are nonnegative numbers. The meaning of such a judgment is as follows. If  $e$  terminates successfully in some environment  $\eta$  and heap  $\sigma$  with unbounded memory resources available then it will also terminate successfully with a bounded freelist of size at least  $n$  plus the potential ascribed to  $\eta, \sigma$  with respect to the typings in  $\Gamma$ . Furthermore, the freelist size upon termination will be at least  $n'$  plus the potential of the result with respect to the view  $r$ .

The typing rules extend the typing rules of FJEU. The most interesting ones are  $(\diamond\text{Share})$  and  $(\diamond\text{Waste})$ . First we notice that they are not syntax directed. Thus, they need to be eliminated when we come to implement the system in the next section.  $(\diamond\text{Waste})$  corresponds to the rule of subsumption of subtyping systems and weakens context, type, and effect. Herein,  $\Gamma <: \Theta$  means  $\forall x \in \Theta. \Gamma_x <: \Theta_x$ .

The purpose of the  $(\diamond\text{Share})$  rule is to ensure that a variable can be used twice without duplication of potential. Suppose we have the following expression:

$$\Gamma, l : \text{List}^{\text{rich}} \frac{n}{n'} \text{ let } nl = l.\text{copy}() \text{ in } l.\text{copy}() : \text{List}^{\text{poor}} \quad (2.8)$$

If we allow the second call to the copy method we would be creating objects without “paying” for it, which would be unsound. Since the method `copy` is only defined for the view `rich`, the only possibility of typing (2.8) would be that  $\forall(\text{List}^{\text{rich}} | \text{List}^{\text{rich}}, \text{List}^{\text{rich}})$  would hold, but it does not because  $\diamond(\text{Cons}^{\text{rich}}) < \diamond(\text{Cons}^{\text{rich}}) + \diamond(\text{Cons}^{\text{rich}})$ . Notice that the declarative type system as it is gives no procedure to find those intermediate views. To actually find them in order to implement the system is not trivial and will be discussed in the next section.

The judgment  $\vdash m : \alpha \text{ ok}$  means that  $\alpha$  is a valid RAJA type for a method  $m$  if the method body of  $m$  can be typed with the arguments, return type and effects as specified in  $\alpha$ . Programs, then, are well-typed if all method bodies admit the announced type and, moreover, view and potential annotations are compatible with subtyping. Formally,

$$\begin{array}{c}
\text{RAJA Typing} \quad \boxed{\Gamma \frac{n}{n'} e : C^r} \\
\\
\frac{}{\emptyset \vdash \frac{\diamond(C^r)+1}{0} \text{ new } C : C^r} (\diamond New) \quad \frac{}{x : C^r \vdash \frac{0}{\diamond(C^r)+1} \text{ free}(x) : E^s} (\diamond Free) \\
\\
\frac{C <: E}{x : E^r \vdash \frac{0}{0} (C)x : C^r} (\diamond Cast) \quad \frac{}{\emptyset \vdash \frac{0}{0} \text{ null} : C^r} (\diamond Null) \quad \frac{}{x : C^r \vdash \frac{0}{0} x : C^r} (\diamond Var) \\
\\
\frac{s = \text{A}^{\text{get}}(C^r, a) \quad D = C.a}{x : C^r \vdash \frac{0}{0} x.a : D^s} (\diamond Access) \quad \frac{\text{A}^{\text{set}}(C^r, a) = s \quad C.a = D}{x : C^r, y : D^s \vdash \frac{0}{0} x.a <- y : C^r} (\diamond Update) \\
\\
\frac{\Gamma_1 \frac{n}{n'} e_1 : D^s \quad \Gamma_2, x : D^s \frac{n'}{n''} e_2 : C^r}{\Gamma_1, \Gamma_2 \frac{n}{n''} \text{ let } C x = e_1 \text{ in } e_2 : C^r} (\diamond Let) \\
\\
\frac{(E_1^{q_1}, \dots, E_j^{q_j} \xrightarrow{n/n'} E_0^{q_0}) \in \text{M}(C^r, m)}{x : C^r, y_1 : E_1^{q_1}, \dots, y_j : E_j^{q_j} \frac{n}{n'} x.m(y_1, \dots, y_j) : E_0^{q_0}} (\diamond Invocation) \\
\\
\frac{x \in \Gamma \quad \Gamma \frac{n}{n'} e_1 : C^r \quad \Gamma \frac{n}{n'} e_2 : C^r}{\Gamma \frac{n}{n'} \text{ if } x \text{ instance of } E \text{ then } e_1 \text{ else } e_2 : C^r} (\diamond Conditional) \\
\\
\frac{\forall (D^s \mid D^{q_1}, \dots, D^{q_n}) \quad \Gamma, y_1 : D^{q_1}, \dots, y_n : D^{q_n} \frac{n}{n'} e : C^r}{\Gamma, x : D^s \frac{n}{n'} e[x/y_1, \dots, x/y_n] : C^r} (\diamond Share) \\
\\
\frac{n \geq u \quad n + u' \geq n' + u \quad \emptyset \vdash \frac{u}{u'} e : D^s \quad \Gamma <: \emptyset \quad D^s <: C^r}{\Gamma \frac{n}{n'} e : C^r} (\diamond Waste)
\end{array}$$

$$\text{RAJA Method Typing} \quad \boxed{\vdash m : \alpha \text{ ok}}$$

$$\frac{m \in \text{M}(C) \quad \alpha = E_1^{r_1}, \dots, E_j^{r_j} \xrightarrow{n/n'} E_0^{r_0} \in \text{M}(C^r, m) \quad \forall (C^r \mid C^q, C^s)}{\frac{\text{this} : C^q, x_1 : E_1^{r_1}, \dots, x_j : E_j^{r_j} \frac{n + \diamond(C^s)}{n'} \text{ M}_{\text{body}}(C, m) : E_0^{r_0}}{\vdash m : \alpha \text{ ok}} (\diamond MBody)}$$

Fig. 2. Typing RAJA

**Definition 5 (Well-typed RAJA-program).** A RAJA-program

$\mathcal{R} = (\mathcal{C}, \mathcal{V}, \diamond(\cdot), \text{A}^{\text{get}}(\cdot, \cdot), \text{A}^{\text{set}}(\cdot, \cdot), \text{M}(\cdot, \cdot))$  is well-typed if for all  $C \in \mathcal{C}$  and  $r \in \mathcal{V}$  the following conditions are satisfied:

1.  $\text{S}(C) = D \Rightarrow C^r <: D^r$
2.  $\forall a \in \text{A}(C) . (C.a)^{\text{A}^{\text{set}}(C^r, a)} <: (C.a)^{\text{A}^{\text{get}}(C^r, a)}$
3.  $\forall m \in \text{M}(C) . \forall \alpha \in \text{M}(C^r, m) . \vdash m : \alpha \text{ ok}$

## 2.2 Algorithmic Views and Complete RAJA Programs

In this section we define algorithmic views which provide least upper and greatest lower bounds for subtyping restricted to refinements of a fixed FJEU type and also a formal addition operation on these refinements allowing us to infer the necessary type of a variable from the types of its (multiple) occurrences. Finally, they include operations to construct the intermediate views in method typings.

Recall the copy method of Example 2. We need one item of potential in order to create a  $\text{Cons}^{\text{poor}}$  object. We said before that this object creation will be payed with the potential of `this`, but how exactly? In order to use the potential of the variable `this` of RAJA-class  $\text{Cons}^{\text{rich}}$ , we put it in the context with a modified type, for example,  $\text{Cons}^{\text{rich}-1}$ , which is a view defined just like `rich` but with potential 0 everywhere. Moreover we find another view with potential 1, which we call  $1(\text{rich})$ , such that  $\forall (\text{Cons}^{\text{rich}} \mid \text{Cons}^{\text{rich}-1}, \text{Cons}^{1(\text{rich})})$  holds. Then we can derive:  $\text{this} : \text{Cons}^{\text{rich}-1} \vdash_0^1 \text{let List res} = \text{new Cons}^{\text{poor}} \text{ in } \dots \text{ in return res}$ ;

The declarative rule gives no information about how to find the views  $q$  and  $s$ . In order to find them algorithmically, we will introduce special algorithmic views like  $\text{rich} - 1$  and  $1(\text{rich})$ .

**Definition 6 (Algorithmic views).** Let  $\mathcal{R}$  be a RAJA-program. We extend the given set of views  $\mathcal{V}$  by algorithmic views

$$\delta, \gamma ::= s_1 \vee s_2 \mid s_1 \wedge s_2 \mid s_1 + s_2 \mid s - n \mid n(s) \quad s, s_1, s_2 \in \mathcal{V}, n \in \mathbb{D}$$

by extending the given maps  $\diamond(\cdot)$ ,  $\text{A}^{\text{get}}(\cdot, \cdot)$ ,  $\text{A}^{\text{set}}(\cdot, \cdot)$ ,  $\text{M}(\cdot, \cdot)$  according to Fig. 3.

**Definition 7 (Complete RAJA-program).** A RAJA-program

$\mathcal{R} = (\mathcal{C}, \mathcal{V}, \diamond(\cdot), \text{A}^{\text{get}}(\cdot, \cdot), \text{A}^{\text{set}}(\cdot, \cdot), \text{M}(\cdot, \cdot))$  is complete if the following conditions are satisfied. Let  $*$   $\in \{\wedge, \vee, +\}$ .

1.  $s_1 * s_2 \in \mathcal{V}$ , for all  $s_1, s_2 \in \mathcal{V}$ .
2.  $s - n, n(s) \in \mathcal{V}$ , for all  $s \in \mathcal{V}, n \in \mathbb{D}$ .
3. The annotation table of  $\mathcal{R}$  satisfies the equations from Def. 6.

Given a RAJA program  $\mathcal{R}$  we can complete it with algorithmic views.  $C^{s_1 \vee s_2}$  is the least upper bound of  $C^{s_1}$  and  $C^{s_2}$  and  $C^{s_1 \wedge s_2}$  is the greatest lower bound of  $C^{s_1}$  and  $C^{s_2}$ .  $C^{s_1 + s_2}$  is defined such that  $\forall (C^s \mid C^{s_1}, C^{s_2})$  is equivalent to  $C^s <: C^{s_1 + s_2}$ . This way we can deal only with subtyping instead of sharing, which is simpler and more intuitive. Finally, the views  $n(s)$  are neutral views of potential  $n$  and set-views like  $s$ . They are intended to be used together with the views  $s - n$ , which are nothing but the view  $s$ , with  $n$  units of potential stripped-off. This way, we get  $\forall (C^s \mid C^{s-n}, C^{n(s)})$ . These algorithmic views are useful for implementing  $\vdash m : \alpha \text{ ok}$ . If we need to use  $n$  units of potential of the type  $C^s$  of `this` in the method body of a given method, we give `this` the type  $C^{s-n}$  and use the potential of  $C^{n(s)}$  in the method.

Of course, we are free to use the algorithmic views from the beginning and in particular in the provided class and method typings. They may be seen as a

Let  $C \in \mathcal{C}$ ,  $a \in \mathbf{A}(C)$  and  $m \in \mathbf{M}(C)$ . We set:

$\diamond(C^{s_1 \wedge s_2})$	$= \max(\diamond(C^{s_1}), \diamond(C^{s_2}))$	$\mathbf{A}^{\text{get}}(C^{s_1 \wedge s_2}; a)$	$= \mathbf{A}^{\text{get}}(C^{s_1}; a) \wedge \mathbf{A}^{\text{get}}(C^{s_2}; a)$
$\diamond(C^{s_1 \vee s_2})$	$= \min(\diamond(C^{s_1}), \diamond(C^{s_2}))$	$\mathbf{A}^{\text{get}}(C^{s_1 \vee s_2}; a)$	$= \mathbf{A}^{\text{get}}(C^{s_1}; a) \vee \mathbf{A}^{\text{get}}(C^{s_2}; a)$
$\diamond(C^{s_1 + s_2})$	$= \diamond(C^{s_1}) + \diamond(C^{s_2})$	$\mathbf{A}^{\text{get}}(C^{s_1 + s_2}; a)$	$= \mathbf{A}^{\text{get}}(C^{s_1}; a) + \mathbf{A}^{\text{get}}(C^{s_2}; a)$
$\diamond(C^{n(s)})$	$= n$	$\mathbf{A}^{\text{get}}(C^{n(s)}; a)$	$= 0(s)$
$\diamond(C^{s \dot{-} n})$	$= \begin{cases} \diamond(C^s) - n & \diamond(C^s) \geq n \\ 0 & \text{otherwise} \end{cases}$	$\mathbf{A}^{\text{get}}(C^{s \dot{-} n}; a)$	$= \mathbf{A}^{\text{get}}(C^s; a)$
$\mathbf{M}(C^{s_1 \wedge s_2}; m)$	$= \mathbf{M}(C^{s_1}; m) \cap \mathbf{M}(C^{s_2}; m)$	$\mathbf{A}^{\text{set}}(C^{s_1 \wedge s_2}; a)$	$= \mathbf{A}^{\text{set}}(C^{s_1}; a) \vee \mathbf{A}^{\text{set}}(C^{s_2}; a)$
$\mathbf{M}(C^{s_1 \vee s_2}; m)$	$= \mathbf{M}(C^{s_1}; m) \cup \mathbf{M}(C^{s_2}; m)$	$\mathbf{A}^{\text{set}}(C^{s_1 \vee s_2}; a)$	$= \mathbf{A}^{\text{set}}(C^{s_1}; a) \wedge \mathbf{A}^{\text{set}}(C^{s_2}; a)$
$\mathbf{M}(C^{s_1 + s_2}; m)$	$= \mathbf{M}(C^{s_1}; m) \cap \mathbf{M}(C^{s_2}; m)$	$\mathbf{A}^{\text{set}}(C^{s_1 + s_2}; a)$	$= \mathbf{A}^{\text{set}}(C^{s_1}; a) \vee \mathbf{A}^{\text{set}}(C^{s_2}; a)$
$\mathbf{M}(C^{n(s)}; m)$	$= \emptyset$	$\mathbf{A}^{\text{set}}(C^{n(s)}; a)$	$= \mathbf{A}^{\text{set}}(C^s; a)$
$\mathbf{M}(C^{s \dot{-} n}; m)$	$= \mathbf{M}(C^s; m)$	$\mathbf{A}^{\text{set}}(C^{s \dot{-} n}; a)$	$= \mathbf{A}^{\text{set}}(C^s; a)$

**Fig. 3.** Definition of  $\diamond(\cdot)$ ,  $\mathbf{A}^{\text{get}}(\cdot, \cdot)$ ,  $\mathbf{A}^{\text{set}}(\cdot, \cdot)$ ,  $\mathbf{M}(\cdot, \cdot)$  of algorithmic views

shorthand for a longer table which includes them explicitly. We stress, though, that efficient type checking for *incomplete* programs is not possible with the techniques from this paper.

We do not consider typechecking of incomplete programs to be of any practical relevance. The following lemma summarizes the desirable order- and proof-theoretic properties of algorithmic views:

**Lemma 2.** *Let  $C, D \in \mathcal{C}$  and  $s, s_1, s_2, \dots, s_n, q_1, q_2, \dots, q_n \in \mathcal{V}$ .*

1.  $C^{s_1 \vee s_2}$  is the least upper bound of  $C^{s_1}$  and  $C^{s_2}$ .
2.  $C^{s_1 \wedge s_2}$  is the greatest lower bound of  $C^{s_i}$ .
3.  $\forall (C^s \mid C^{s_1}, \dots, C^{s_n}) \iff C^s <: C^{s_1 + \dots + s_n}$ .
4. If  $C^s <: C^{s_1 + \dots + s_n}$  and  $C^{s_i} <: C^{q_i}$  for all  $i$ , then  $C^s <: C^{q_1 + \dots + q_n}$ .
5.  $C^{s+0(s)} = C^s$ .
6. If  $n \leq \diamond(C^s)$  then  $\forall (C^s \mid C^{s \dot{-} n}, C^{n(s)})$ . Moreover,  $\forall (C^s \mid C^{s_1}, C^{s_2})$  and  $\diamond(C^{s_2}) \geq n$  imply  $C^{s \dot{-} n} <: C^{s_1}$ .

Algorithmic typechecking now faces one more obstacle. Officially, one method can have infinitely many RAJA types. This does not compromise semantic type soundness, but must of course be restricted to finitely many to enable algorithmic type checking.

Moreover, the rule ( $\diamond$ Invocation) chooses non-deterministically one monomorphic RAJA method type according to the given method call. In order for algorithmic typing to be efficient (not NP-complete) we need to make sure that there is an optimal such choice in any situation.

**Definition 8.** *If  $\alpha = r_1, \dots, r_j \xrightarrow{p/p'} r_0$  and  $\beta = s_1, \dots, s_j \xrightarrow{n/n'} s_0$  then  $\alpha \sqsubseteq \beta$  iff  $p \leq n$  and  $p - p' \leq n - n'$ .*

**Definition 9.** A RAJA-program is algorithmic if it is finite, complete, and for all  $C, r, m$  the set  $M(C^r, m)$  is totally ordered by the ordering in Def. 8.

From now on we assume that all RAJA-programs are algorithmic without explicit notice.

### 3 Algorithmic Typing of RAJA Programs

In this section we present an algorithm for typechecking RAJA programs. Algorithmic type-checking must consist of syntax directed rules, thus, the rules ( $\diamond Share$ ) and ( $\diamond Waste$ ) must be integrated in other rules. Instead of using ( $\diamond Waste$ ), we integrate subtyping in the rules.

The purpose of the ( $\diamond Share$ ) rule is to ensure that a variable can be used more than once without unsound incrementation of potential. The main challenge for implementing it is that it contains no information about how to find the views  $q_1$  to  $q_n$  for the different occurrences. The current implementation does not include inference of these views. Instead, every variable occurrence has been annotated with the corresponding view, which can be an algorithmic view. The task of the type checker is then to check the correctness of the given sharing, or, more exactly, since, as we saw in last section, using algorithmic views a sharing task can be reduced into a subtyping task, the algorithm checks only subtyping. The inference of these intermediate views remains under investigation.

The computed resource annotations in rules ( $\vdash Let$ ) and ( $\vdash Cond.$ ) are a bit intricate. Ultimately, they are justified by soundness and completeness. Rule ( $\vdash Let$ ) may be easier to understand if broken down into the two cases  $m \geq n'$  and  $m < n'$ . In the latter case the output of the first computation suffices to satisfy the second one. In the former case extra input potential must be provided for the second computation. In rule ( $\vdash Cond.$ ) we must cater for both computations, hence the max and the min. The adaptations  $u - n$  and  $u - m$  cater for the case where, say,  $m \geq n$  units were provided due to the max, yet the first branch of the conditional was taken hence only  $n$  units “used” and vice versa.

In the rule ( $\vdash Inv.$ ) we choose the minimal RAJA monomorphic type that satisfies the subtyping conditions. Since the algorithmic system considers only finite programs and the set of RAJA monomorphic types is totally ordered according to  $\sqsubseteq$ , every nonempty subset of  $M(G^r, m)$  has a minimal element.

We define the judgment  $\Delta^\Psi \vdash_{n'}^{n} e^\circ \Leftarrow C^\gamma$  inductively by the rules in Figure 4, where  $\Delta$ ,  $e^\circ$  and  $C^\gamma$  are inputs and  $\Psi$ ,  $n$  and  $n'$  are outputs.  $\Delta$  is an FJEU context, i.e. a map from variable names to FJEU types.  $\Psi$  is a map from variable names to algorithmic views.  $C^\gamma$  is an algorithmic RAJA type, which is an FJEU class refined with an algorithmic view and  $e^\circ$  is an annotated FJEU expression.

The notation  $\Delta^\Psi$  means that for every variable  $x \in \Delta$ , if  $\Delta_x = C$  and  $\Psi_x = \delta$  then  $\Delta_x^\Psi = C^\delta$ . We also use the notation  $\Delta^{\Psi_1 + \Psi_2}$  for meaning that if  $\Delta_x^{\Psi_1} = C^{\delta_1}$  and  $\Delta_x^{\Psi_2} = C^{\delta_2}$  then  $\Delta_x^{\Psi_1 + \Psi_2} = C^{\delta_1 + \delta_2}$ . The meaning of  $\Delta^{\Psi_1 \wedge \Psi_2}$  is similar. We write  $x : C^r, + y : D^s$  for the following two cases. The usual case is  $x \neq y$  and then it means nothing but  $x : C^r, y : D^s$ . On the other hand, if  $x = y$ , then  $C = D$  too, and the notation means  $x : C^{r+s}$ . We write  $\Delta^{\Psi_0}$  for meaning  $\Delta_x^{\Psi_0} = C^{0(s)}$  where

$\Delta_x = C$  and  $s$  is one of the view annotations of  $x$  or any view if  $x$  is not used in the program. The idea is to return neutral views for variables that are not used in the given expression. Finally, let  $e^\circ$  denote an annotated RAJA expression. In summary, we define the partial function  $\text{typecheck}(\Delta, e^\circ, C^\gamma)$  (Fig. 4).

Next, we define the algorithmic judgment  $\vdash_a m : \alpha \text{ ok}$  based on algorithmic typing. (Fig. 4). The typechecking algorithm returns a greater context than the declared one. This has to be checked. Moreover, it calculates the space consumption  $u$  of the method body. If  $u \leq n$  then  $n$  items are enough and we do not need any potential from this. Otherwise, we calculate how many items of potential we need from this, i.e.  $p = u - n$ , and we of course have to check whether the potential of this is at least  $p$ . Finally, the amount of freelist units  $u'$  released by the expression should be at least  $n' + u - (n + p)$ .

In the following we show that the algorithmic typing system we just defined is correct w.r.t. the declarative typing system of RAJA. If  $\Gamma$  is a RAJA context, we write  $|\Gamma|$  for meaning its underlying FJEU context.

**Lemma 3 (Soundness of algorithmic RAJA typing)**

If  $\Delta^\Psi \vdash_{n'}^u e^\circ \Leftarrow C^\gamma$  then  $\Delta^\Psi \vdash_{n'}^u e : C^\gamma$ .

*Proof* By induction on algorithmic typing derivations, using the ( $\diamond\text{Waste}$ ) rule and Lemma 2.

**Lemma 4 (Soundness of algorithmic RAJA method typing).** *Given a RAJA type  $C^r$ , a method  $m \in \mathbf{M}(C)$  and a RAJA method type  $\alpha \in \mathbf{M}(C^r, m)$ , if  $\vdash_a m : \alpha \text{ ok}$  then  $\vdash m : \alpha \text{ ok}$ .*

*Proof.* Follows by Lemma 3.

The completeness proof is a bit more complicated than the soundness proof. The reason for this is that we have eliminated the rules ( $\diamond\text{Share}$ ) and ( $\diamond\text{Waste}$ ) and we have to show that typing derivations that use these rules are still admissible in the algorithmic system. The following lemma states the admissibility of sharing in the algorithmic system.

**Lemma 5 (Share).** *Let  $\Delta^\Psi, y_1 : D^{\delta_1}, \dots, y_n : D^{\delta_n} \vdash_{n'}^u e^\circ \Leftarrow C^\gamma$ . Then  $\Delta^\Psi, x : D^\delta \vdash_{n'}^u e[x/y_1, \dots, x/y_n]^\circ \Leftarrow C^\gamma$  where either  $\delta = \delta_1 + \dots + \delta_n$  or  $\delta = \delta_1 \wedge \dots \wedge \delta_n$ .*

*Proof.* By induction on algorithmic typing derivations.

**Lemma 6 (Waste).** *Let  $\Delta^\Psi \vdash_{u'}^u e^\circ \Leftarrow D^\gamma$ ,  $D^\gamma < C^\delta$  and  $\Delta < A$  then  $\Delta^\Psi \vdash_{w'}^u e^\circ \Leftarrow C^\delta$  for some  $w \leq u$  and  $w' \geq u' + w - u$ .*

*Proof.* By induction on algorithmic typing derivations.

**Lemma 7 (Completeness of algorithmic RAJA typing).**

If  $\Gamma \vdash_{n'}^u e : C^r$  then there is an annotated version  $e^\circ$  of the expression  $e$  with  $|\Gamma|^\Psi \vdash_{u'}^u e^\circ \Leftarrow C^r$  for some  $u \leq n$  and  $u' \geq n' + u - n$  so that  $\Gamma < : |\Gamma|^\Psi$ .

*Proof.* By induction on typing derivations, using Lemma 5 and 6.

$$\begin{array}{c}
\text{Algorithmic RAJA Typing} \quad \boxed{\Delta^\Psi \frac{n}{n'} e^\circ \Leftarrow C^r} \\
\\
\frac{\Delta^{\Psi_\emptyset} \mid \frac{\chi(D^\gamma)+1}{0}}{D^\gamma <: C^\gamma} (\vdash \text{New}) \quad \frac{\Delta^{\Psi_\emptyset}, x: E^q \mid \frac{0}{0}}{E^q <: C^\gamma} (\vdash \text{Var}) \\
\\
\frac{\Delta^{\Psi_\emptyset}, x: C^q \mid \frac{0}{\chi(C^q)+1}}{\text{free}(x^q) \Leftarrow E^\gamma} (\vdash \text{Free}) \\
\\
\frac{D <: E \text{ (or } E <: D) \quad D^q <: C^\gamma}{\Delta^{\Psi_\emptyset}, x: E^q \mid \frac{0}{0} (D)x^q \Leftarrow C^\gamma} (\vdash \text{Cast}) \quad \frac{}{\Delta^{\Psi_\emptyset} \mid \frac{0}{0} \text{ null} \Leftarrow C^\gamma} (\vdash \text{Null}) \\
\\
\frac{A^{\text{get}}(C^r, a) = q \quad C.a = E \quad E^q <: D^\gamma}{\Delta^{\Psi_\emptyset}, x: C^r \mid \frac{0}{0} x^r.a \Leftarrow D^\gamma} (\vdash \text{Access}) \\
\\
\frac{A^{\text{set}}(E^q, a) = s \quad E.a = D \quad F^p <: D^s \quad E^q <: C^\gamma}{\Delta^{\Psi_\emptyset}, x: E^q, + y: F^p \mid \frac{0}{0} x^q.a \leftarrow y^p \Leftarrow C^\gamma} (\vdash \text{Update}) \\
\\
\frac{\Delta^{\Psi'} \frac{n}{n'} e_1^\circ \Leftarrow D^{\gamma_1} \quad \Delta^{\Psi''}, x: D^{\gamma_1} \frac{m}{m'} e_2^\circ \Leftarrow C^{\gamma_2}}{\Delta^{\Psi'+\Psi''} \mid \frac{\max(n, n+m-n')}{\max(m', m'+n'-m)}} \text{let } D x = e_1^\circ \text{ in } e_2^\circ \Leftarrow C^{\gamma_2} (\vdash \text{Let}) \\
\\
\frac{x \in \Delta \quad \Delta^{\Psi'} \frac{n}{n'} e_1^\circ \Leftarrow C^\gamma \quad \Delta^{\Psi''} \frac{m}{m'} e_2^\circ \Leftarrow C^\gamma \quad u = \max(m, n)}{\Delta^{\Psi' \wedge \Psi''} \mid \frac{u}{\min(n'+u-n, m'+u-m)}} \text{if } x \text{ instanceof } E \text{ then } e_1^\circ \text{ else } e_2^\circ \Leftarrow C^\gamma (\vdash \text{Cond.}) \\
\\
\frac{p/p' = \arg \min\{(E_1^{q_1}, \dots, E_j^{q_j} \xrightarrow{p/p'} E_0^{q_0}) \in M(G^r, m) \mid \forall i. F_i^{t_i} <: E_i^{q_i}, E_0^{q_0} <: C^\gamma\}}{\Delta^{\Psi_\emptyset}, x: G^r, + y_1: F_1^{t_1}, + \dots, + y_j: F_j^{t_j} \mid \frac{p}{p'} x^r.m(y_1^{t_1}, \dots, y_j^{t_j}) \Leftarrow C^\gamma} (\vdash \text{Inv.})
\end{array}$$

Typecheck function

$$\text{typecheck}(\Delta, e^\circ, C^\gamma) = \begin{cases} (\Psi, n, n') & \text{if } \Delta^\Psi \frac{n}{n'} e^\circ \Leftarrow C^\gamma \\ \text{fail} & \text{otherwise} \end{cases}$$

$$\text{Algorithmic RAJA Method Typing} \quad \boxed{\vdash_a m : \alpha \text{ ok}}$$

$$\begin{array}{c}
\alpha = E_1^{r_1}, \dots, E_j^{r_j} \frac{n/n'}{E_0^{r_0}} \in M(C^r, m) \\
\text{this: } C^\beta, x_1: E_1^{\beta_1}, \dots, x_j: E_j^{\beta_j} \mid \frac{u}{u'} M_{\text{body}}(C, m)^\circ \Leftarrow E_0^{r_0} \\
\frac{E_i^{r_i} <: E_i^{\beta_i} \quad p = u - n \quad u' \geq n' + u - (n + p) \quad \chi(C^r) \geq p \quad C^{r-p} <: C^\beta}{\vdash_a m : \alpha \text{ ok}}
\end{array}$$

Fig. 4. Algorithmic RAJA Typing

**Lemma 8 (Completeness of algorithmic RAJA method typing).** *Given a RAJA type  $C^r$ , a method  $m \in M(C)$  and a RAJA method type  $\alpha \in M(C^r, m)$ , if  $\vdash m : \alpha$  ok then  $\vdash_a m : \alpha$  ok.*

*Proof.* Follows by Lemma 7.

**Lemma 9 (Efficiency of algorithmic RAJA typing).**

$\Delta^\Psi \frac{n}{n} e^\circ \Leftarrow C^r$  is decidable in polynomial time.

*Proof (sketch).* The syntax-directed backwards application of the algorithmic typing rules produces a linear number of subtyping and sharing constraints. Furthermore, the algorithmic view expressions occurring in these constraints are themselves of linear size. It then suffices to restrict attention to the views that occur as subexpressions of the ones appearing in the constraints. Their number is therefore polynomial in the size of the program. A complete table of the subtyping and sharing judgments for this relevant subset can then be computed iteratively in polynomial time. In practice, a goal-directed implementation performs even better.

**Lemma 10.** *Given a RAJA class  $C$ , a view  $r$ , a method  $m \in M(C)$  and a RAJA method type  $\alpha \in M(C^r, m)$ ,  $\vdash m : \alpha$  ok is decidable.*

*Proof.* Follows by Lemmas 4, 8 and 9.

**Theorem 1 (Decidability of RAJA typing).** *Given a RAJA-Program  $\mathcal{R}$ , its well-typedness is decidable.*

*Proof.* Follows by Lemma 10.

## 4 Related Work

Since [HJ06] several authors have made contributions towards costing heap consumption of object-oriented programs. [MP07] uses methods from abstract interpretation and term rewriting (quasi interpretations) to estimate the size of data structures and thus indirectly heap consumption. The approach is promising, but aliasing does not seem to have been taken into account properly and not many examples are given. The interpretation of methods must be provided manually.

COSTA [AAG<sup>+</sup>07] is similar in that it assigns cost functions to methods and program parts. These refer directly to heap consumption and are given as solutions of automatically constructed recurrence systems. The main contribution of COSTA is an improved solver for these recurrences. COSTA is not as general as RAJA which, however, is not fully automatic.

Another promising fully automatic system is [GMC09] which works by instrumenting code with resource-counting, integer-valued “ghost”-variables and using modern tools from static analysis for estimating their range of values. The examples given are stunning, but do not involve dynamically allocated data



structures. With further progress with automatic analysis of arithmetic relationships between integer variables systems like SPEED may eventually render type-based analyses obsolete. More likely, however, is a combination of the two.

Finally, Java(X) [DTW07] is a type system quite similar to RAJA and developed independently which has, however, a different purpose, namely ensuring the correct usage of resources like files etc. according to a specified protocol. The paper [DTW07] does not present algorithmic type checking, let alone automatic type inference; it is likely that the algorithmic system presented here could be adapted to Java(X).

## 5 Conclusions

We have provided a type checking algorithm for RAJA programs and proved its correctness and efficiency in the sense of polynomial-time computability. In order to do this, we introduced algorithmic views which render the subtyping lattice more well behaved and could also be a useful addition to the declarative system which is exposed to the programmer. In this way, we were able to get rid of most type annotations in method bodies although we still have to indicate the types of multiple occurrences of a variable, i.e., how the potential belonging to the variable is to be split among the different occurrences.

The algorithmic typechecking and the implementation allow us to investigate larger examples which might prompt further extensions to RAJA. In particular, we would like to investigate the typability of the Iterator pattern and more challengingly patterns involving callbacks like Observer. From a pragmatic viewpoint, polymorphic quantification over views could be a useful extension, too.

Of course, full-blown type inference is also on our agenda, thus potentially rendering RAJA into a push-button analysis.

*Acknowledgment.* We acknowledge support by the EU integrated project MOBIUS IST 15905. We thank Andreas Abel, Lennart Beringer and Steffen Jost for valuable comments.

## References

- [AAG<sup>+</sup>07] Albert, E., Arenas, P., Genaim, S., Puebla, G., Zanardini, D.: COSTA: Design and implementation of a cost and termination analyzer for java bytecode. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 113–132. Springer, Heidelberg (2008)
- [Cam08] Campbell, B.: Type-based amortized stack memory prediction. PhD thesis, University of Edinburgh (2008)
- [DTW07] Degen, M., Thiemann, P., Wehr, S.: Tracking linear and affine resources with JAVA(X). In: Ernst, E. (ed.) ECOOP 2007. LNCS, vol. 4609, pp. 550–574. Springer, Heidelberg (2007)
- [FKF98] Flatt, M., Krishnamurthi, S., Felleisen, M.: Classes and mixins. In: The 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1998), New York, January 1998, pp. 171–183. Association for Computing Machinery (1998)

- [GL98] Gómez, G., Liu, Y.A.: Automatic accurate time-bound analysis for high-level languages. In: Müller, F., Bestavros, A. (eds.) LCTES 1998. LNCS, vol. 1474, p. 31. Springer, Heidelberg (1998)
- [GMC09] Gulwani, S., Mehra, K.K., Chilimbi, T.M.: SPEED: precise and efficient static estimation of program computational complexity. In: Shao, Z., Pierce, B.C. (eds.) POPL, pp. 127–139. ACM Press, New York (2009)
- [Gro01] Grobauer, B.: Topics in Semantics-based Program Manipulation. PhD thesis, BRICS Aarhus (2001)
- [HBH<sup>+</sup>07] Herrmann, C.A., Bonenfant, A., Hammond, K., Jost, S., Loidl, H.-W., Pointon, R.: Automatic amortised worst-case execution time analysis. In: 7th Int'l Workshop on Worst-Case Execution Time (WCET) Analysis, Proceedings, pp. 13–18 (2007)
- [HDF<sup>+</sup>05] Hammond, K., Dyckhoff, R., Ferdinand, C., Heckmann, R., Hofmann, M., Jost, S., Loidl, H.-W., Michaelson, G., Pointon, R.F., Scaife, N., Srot, J., Wallace, A.: The embounded project (project start paper). In: van Eekelen, M.C.J.D. (ed.) Trends in Functional Programming. Trends in Functional Programming, vol. 6, pp. 195–210. Intellect (2005)
- [HJ03] Hofmann, M., Jost, S.: Static prediction of heap space usage for first-order functional programs. In: POPL: 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (2003)
- [HJ06] Hofmann, M.O., Jost, S.: Type-based amortised heap-space analysis. In: Sestoft, P. (ed.) ESOP 2006. LNCS, vol. 3924, pp. 22–37. Springer, Heidelberg (2006)
- [HJR] Hofmann, M., Jost, S., Rodriguez, D.: Type-based amortised heap space analysis (complete soundness proof),  
<http://raja.tcs.ifi.lmu.de/download/files/rajaSoundProof.pdf>
- [HP99] Hughes, J., Pareto, L.: Recursion and dynamic data-structures in bounded space, June 21 (1999)
- [IPW99] Igarashi, A., Pierce, B., Wadler, P.: Featherweight Java: A minimal core calculus for Java and GJ. In: Meissner, L. (ed.) Proceedings of the 1999 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages & Applications (OOPSLA 1999), N.Y., vol. 34(10), pp. 132–146 (1999)
- [MP07] Marion, J.-Y., Péchoux, R.: Resource control of object-oriented programs. CoRR, abs/0706.2293, informal publication (2007)
- [NCQR05] Nguyen, H.H., Chin, W.N., Qin, S., Rinard, M.C.: Memory usage inference for object-oriented programs (January 2005)
- [Oka98] Okasaki, C.: Purely Functional Data Structures. Cambridge University Press, Cambridge (1998)
- [raj] <http://raja.tcs.ifi.lmu.de>
- [Tar85] Tarjan, R.E.: Amortized computational complexity. SIAM Journal on Algebraic and Discrete Methods 6(2), 306–318 (1985)

# Deciding the Inductive Validity of $\forall\exists^*$ Queries

Matthias Horbach and Christoph Weidenbach

Max Planck Institute for Informatics, Saarbrücken, Germany

{horbach,weidenbach}@mpi-inf.mpg.de

**Abstract.** We present a new saturation-based decidability result for inductive validity. Let  $\Sigma$  be a finite signature in which all function symbols are at most unary and let  $N$  be a satisfiable Horn clause set without equality in which all positive literals are linear. If  $N \cup \{A_1, \dots, A_n \rightarrow\}$  belongs to a class that can be finitely saturated by ordered resolution modulo variants, then it is decidable whether a sentence of the form  $\forall x.\exists \vec{y}.A_1 \wedge \dots \wedge A_n$  is valid in the minimal model of  $N$ .

## 1 Introduction

We consider the problem of deciding whether a formula  $\phi$  holds in the minimal model of a given satisfiable Horn clause set  $N$  over a signature  $\Sigma$ , or equivalently whether  $N \cup \{\neg\phi\}$  does not have a Herbrand model over  $\Sigma$ . If so, we write this relation as  $N \models_{Ind} \phi$ . If all signature symbols are at most unary and all positive literals in  $N$  are linear, even first-order unsatisfiability is still undecidable: Consider a Post correspondence problem over the alphabet  $\{a, b\}$  with given word pairs  $(u_i, v_i)$ . We model words by monadic terms over the unary function symbols  $a, b$  with empty word 0. Then the PCP has a solution iff the following Horn clause set is unsatisfiable:

$$\begin{array}{ll} \rightarrow \text{PCP}(0, 0) & \text{PCP}(a(x), a(x)) \rightarrow \\ \text{PCP}(x, y) \rightarrow \text{PCP}(u_i(x), v_i(y)) & \text{PCP}(b(x), b(x)) \rightarrow \end{array}$$

Equivalently, it has a solution iff  $\{\rightarrow \text{PCP}(0, 0), \text{PCP}(x, y) \rightarrow \text{PCP}(u_i(x), v_i(y))\} \models_{Ind} \exists x.\text{PCP}(a(x), a(x)) \vee \text{PCP}(b(x), b(x))$ . In this paper, we identify a range of classes of clause sets and of query formulas for which validity in the minimal model is decidable. The main result is as follows:

Let  $N$  be a satisfiable set of Horn clauses without equality over a finite signature  $\Sigma$  and let  $\{A_1, \dots, A_n\}$  be a set of atoms over  $\Sigma$ , where

- (1) all function symbols in  $\Sigma$  are at most unary,
- (2) all positive literals in  $N$  are linear, i.e. every variable occurs at most once, and
- (3)  $N \cup \{A_1, \dots, A_n \rightarrow\}$  belongs to a class that can be finitely saturated by ordered resolution (with variant subsumption and tautology deletion).

Then the problem

$$N \models_{Ind} \forall x.\exists y_1, \dots, y_m.(A_1 \wedge \dots \wedge A_n)$$

is decidable, where  $x, y_1, \dots, y_m$  are the variables in  $A_1, \dots, A_n$ . There are no restrictions on purely negative clauses as well as no restrictions on the structure of the terms appearing in negative literals. Instead of proving  $N \models_{Ind} \forall x. \exists y_1, \dots, y_m. (A_1 \wedge \dots \wedge A_n)$ , we refute  $N \models_{Ind} \exists x. \forall y_1, \dots, y_m. \neg(A_1 \wedge \dots \wedge A_n)$ .

The result also holds for an arbitrary positive quantifier-free query formula  $\phi$  in disjunctive normal form  $\bigvee_i \bigwedge_j A_{i,j}$ , i.e. the problem  $N \models_{Ind} \forall\exists^* \phi$  is decidable if  $N$  can be finitely saturated together with the clauses  $A_{i,1}, \dots, A_{i,n_i} \rightarrow$ .

The first-order unsatisfiability problem for Horn classes satisfying conditions (1)–(2) is still undecidable, as the above encoding of the PCP shows. Therefore, the basis of our decidability result is finite first-order saturation (3). The side conditions (1)–(2) as well as the restriction to variant subsumption and tautology deletion for the saturation process are needed for our current proof. The latter is not an essential restriction, since most decidability results based on saturation show termination by restricting the depth of the occurring terms and the number of variables in each clause, which corresponds exactly to a saturation modulo variants and tautologies (cf. e.g. [9]).

The proof of our result is constructive. We demonstrate it on the example

$$N_G = \left\{ \begin{array}{l} \rightarrow G(s(s(0)), s(0)) , \\ G(x, y) \rightarrow G(s(x), s(y)) \quad , \\ G(s(x), s(y)) \rightarrow G(x, y) \quad \quad \quad \end{array} \right\}$$

with query  $\forall x. \exists y. G(y, x)$ . In the minimal model of  $N_G$ , the relation  $G$  is the “one greater” relation on the naturals. The clause set  $N_G$  satisfies conditions (1)–(2) and can be finitely saturated by ordered resolution, generating one additional clause  $\rightarrow G(s(0), 0)$ . It can also be finitely saturated after adding  $G(y, x) \rightarrow$ .

In [11], we have presented a superposition-based calculus that is complete in the limit for unsatisfiability of queries of the form  $\exists^*\forall^*(A_1, \dots, A_n \rightarrow)$  with respect to minimal models of Horn clause sets. The basic idea of the calculus is to treat the existentially quantified variables via an additional constraint. The result is a constrained query clause of the form  $\vec{v} \approx \vec{x} \parallel A_1, \dots, A_n \rightarrow$  where the existential variables  $\vec{v}$  are replaced in the constrained clause by fresh universally quantified variables  $\vec{x}$ . The generalized ordered resolution rule of the calculus takes care of the compatibility of constraints (Section 3). For our example, we obtain the constrained query clause  $v \approx x \parallel G(y, x) \rightarrow$ .

In general, the algorithm of [11] does not terminate, i.e. it does not decide (and not even semi-decide) whether a given query holds in a minimal model. This even holds on a set  $N$  and a constrained query clause that satisfy conditions (1)–(3). For the above example, we can generate infinitely many non-redundant constrained clauses of the form  $v \approx x\sigma^n \parallel G(y, x) \rightarrow$  and  $v \approx x\sigma^n\tau \parallel \square$  for  $\sigma^0 = \{x \mapsto x\}$ ,  $\sigma^{n+1} = \{x \mapsto s(x\sigma^n)\}$  and  $\tau = \{x \mapsto 0\}$ . The contribution of this paper is to generalize our previously developed constraint language [11] to substitution expressions for the existentially quantified variables (Section 2). For example a constraint  $v \approx x\sigma^*$  represents all possible constraints of the form  $v \approx x\sigma^n$ . Together with conditions (1)–(3), this enables the termination of the query saturation process (Proposition 8). The finite saturation result with

variant subsumption and tautology deletion is essential for the proof. For the above example, we obtain the two additional constrained clauses  $v \approx x\sigma^* \parallel G(y, x) \rightarrow$  and  $v \approx x\sigma^* \tau \parallel \square$  for  $\sigma = \{x \mapsto s(x)\}$ ,  $\tau = \{x \mapsto 0\}$ .

What remains to be shown is that the substitutions in the constraints of all derived constrained empty clauses are covering, i.e. represent all possible instantiations for the variables  $\vec{x}$ : If this is the case, then the clause set does not have a Herbrand model over the given signature. The conjunction of all regular substitution expressions for the constrained empty clauses can be transformed into a monadic Horn clause set containing only linear clauses whose head literal contains all variables of the clause (Section 4). The initial substitution expressions are covering iff a certain predicate  $P$  introduced in the translation is the total relation in the minimal model of the generated Horn clause set. For our example, this translation results in the following Horn clauses:

$$\begin{array}{ll} \rightarrow P_1(0) & P_1(x) \rightarrow P(x) \\ P(x) \rightarrow P_2(s(x)) & P_2(x) \rightarrow P(x) \end{array}$$

Deciding totality of  $P$  for such monadic clause sets is usually difficult. However, several results are known about the decidability of emptiness. Applying predicate completion [7], we can generate a Horn clause set for the complement of  $P$ , named  $\check{P}$ , for any Horn clause set generated from a substitution expression, such that  $P$  is total iff  $\check{P}$  is empty in the respective minimal models (Section 4.1). The clause set for  $\check{P}$  does not contain function symbols in negative literals anymore. Moreover, because of the restriction of the signature to unary function symbols, the translation causes the clause set to contain monadic predicates only. These properties enable the final decidability of the emptiness of  $\check{P}$  by ordered resolution (Theorem 17). The complement  $\check{P}$  of  $P$  for our example is defined by the clauses

$$\begin{array}{ll} \rightarrow \check{P}_1(s(x)) & \rightarrow \check{P}_2(0) \\ \check{P}_1(x), \check{P}_2(x) \rightarrow \check{P}(x) & \check{P}(x) \rightarrow \check{P}_2(s(x)) \end{array}$$

that belong to a class where emptiness is known to be decidable by ordered resolution [16, 15]. For the above clause set, the theory of the relation  $\check{P}$  is empty in the minimal model, hence  $N \models_{Ind} \forall x. \exists y. G(y, x)$  holds. Note that such clause sets can in general not be represented by tree automata (even with constraints).

To the best of our knowledge, our result is the first decidability result for inductive validity based on a finite first-order saturation concept for Horn clauses. Related approaches to inductive reasoning based on superposition include the works of Ganzinger and Stuber [10] and Comon and Nieuwenhuis [7]. Both are also applicable to equational clauses, but did not lead to new decidability results. Other work in the area of automated inductive theorem proving includes the test set calculi of Bouhoula et al. [2, 3] and approaches via term rewrite systems by Caferra and Zabel [4] and Kapur [13, 8]. All these approaches consider only purely universal queries and do not admit quantifier alternations.

Due to space restrictions, some proofs have been excluded from this article. An extended version is available as a technical report [12].

## 2 Preliminaries

We build on the notions of [11, 17] and shortly recall here the most important concepts as well as the specific extensions needed for the new calculus.

**Terms and Clauses.** Let  $\Sigma = (\mathcal{P}, \mathcal{F})$  be a *signature* consisting of a set  $\mathcal{P}$  of predicate symbols of fixed arity and a set  $\mathcal{F}$  of function symbols of fixed arity, and let  $X \cup V$  be an infinite set of variables such that  $X$ ,  $V$ , and  $\mathcal{F}$  are disjoint and  $V$  is finite. Elements of  $X$  are called *universal variables* and denoted as  $x, y, z$ , and elements of  $V$  are called *existential variables* and denoted as  $v$ .

We denote by  $\mathcal{T}(\mathcal{F}, X)$  the set of all *terms* over  $\mathcal{F}$  and  $X$  and by  $\mathcal{T}(\mathcal{F})$  the set of all *ground terms* over  $\mathcal{F}$ . Throughout this article, we require that  $\mathcal{T}(\mathcal{F})$  is infinite. To improve readability, term tuples  $(t_1, \dots, t_n)$  will often be denoted by  $\vec{t}$ . The variables occurring in a term  $t$  or a term tuple  $\vec{t}$  are denoted by  $\text{vars}(t)$  or  $\text{vars}(\vec{t})$ , respectively.

An *atom* is an expression of the form  $P(t_1, \dots, t_n)$ , where  $P \in \mathcal{P}$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n \in \mathcal{T}(\mathcal{F}, X)$  are terms over the universal variables. A *clause* is a pair of multisets of atoms, written  $\Gamma \rightarrow \Delta$ , interpreted as the conjunction of all atoms in the *antecedent*  $\Gamma$  implying the disjunction of all atoms in the *succedent*  $\Delta$ . A clause is *Horn* if  $\Delta$  contains at most one atom. The *empty clause* is denoted by  $\square$ . A Horn clause is *universally reductive* if all variables of the antecedent appear also in the succedent.

**Constrained Clauses.** A (*basic*) *substitution*  $\sigma$  is a map from a finite set  $X' \subseteq X$  of universal variables to  $\mathcal{T}(\mathcal{F}, X)$ . The application of  $\sigma$  to a term  $t$  or a term tuple  $\vec{t}$  is denoted by  $t\sigma$  or  $\vec{t}\sigma$ , respectively. The substitution  $\sigma$  is *linear* if no variable occurs twice in the term set  $\{x\sigma \mid x \in X'\}$ . The *most general unifier* of two terms  $s, t$  is denoted by  $\text{mgu}(s, t)$ .

*Substitution expressions* are build over substitutions and constructors  $\circ$  (composition),  $|$  (disjunction), and  $*$  (loop) of arity 2, 2 and 1, respectively. Substitution expressions are denoted as  $\bar{\sigma}, \bar{\tau}$ . The symbols  $\circ$  and  $|$  are written in infix notation, and  $*$  is written in postfix notation. We will often write  $\bar{\sigma} \circ \bar{\tau}$  as  $\bar{\sigma}\bar{\tau}$ .

The *domain*  $\text{dom}(\bar{\sigma})$  and the *variable range*  $\text{VRan}(\bar{\sigma})$  of a substitution expression are defined as follows: For a substitution  $\sigma : \{x_1, \dots, x_n\} \rightarrow \mathcal{T}(\mathcal{F}, X)$ , we define  $\text{dom}(\sigma) = \{x_1, \dots, x_n\}$  and  $\text{VRan}(\sigma) = \text{vars}(x_1\sigma, \dots, x_n\sigma)$ . For complex expressions, we have

$$\begin{aligned} \text{dom}(\bar{\sigma} \circ \bar{\tau}) &= \text{dom}(\bar{\sigma}) & \text{VRan}(\bar{\sigma} \circ \bar{\tau}) &= \text{VRan}(\bar{\tau}) \\ \text{dom}(\bar{\sigma}_1 | \bar{\sigma}_2) &= \text{dom}(\bar{\sigma}_1) \cup \text{dom}(\bar{\sigma}_2) & \text{VRan}(\bar{\sigma}_1 | \bar{\sigma}_2) &= \text{VRan}(\bar{\sigma}_1) \cap \text{VRan}(\bar{\sigma}_2) \\ \text{dom}(\bar{\sigma}^*) &= \text{dom}(\bar{\sigma}) & \text{VRan}(\bar{\sigma}^*) &= \text{dom}(\bar{\sigma}) \end{aligned}$$

A *constrained clause*  $v_1 \approx x_1 \bar{\sigma}, \dots, v_n \approx x_n \bar{\sigma} \parallel C$ , also written  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$ , consists of a sequence of equations  $v_1 \approx x_1 \bar{\sigma}, \dots, v_n \approx x_n \bar{\sigma}$  called the *constraint* and a clause  $C$ , such that  $\{v_1, \dots, v_n\} = V$ ,  $x_1, \dots, x_n \in X$  are universal variables and  $\bar{\sigma}$  is a substitution expression with domain  $\{x_1, \dots, x_n\}$ . We abbreviate  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  as  $\vec{v} \approx \vec{x} \parallel C$  if  $\bar{\sigma}$  is the identity substitution on  $\{x_1, \dots, x_n\}$ .

**Orderings.** Any ordering  $\prec$  on atoms can be extended to clauses in the following way. We consider clauses as multisets of occurrences of atoms. The occurrence of an atom  $A$  in the antecedent is identified with the multiset  $\{A, A\}$ ; the occurrence of an atom  $A$  in the succedent is identified with the multiset  $\{A\}$ . Now we lift  $\prec$  to atom occurrences as its multiset extension, and to clauses as the multiset extension of this ordering on atom occurrences.

An occurrence of an atom  $A$  in a clause  $C$  is *maximal* if there is no occurrence of an atom in  $C$  that is strictly greater with respect to  $\prec$  than the occurrence of  $A$ . It is *strictly maximal* if there is no other occurrence of an atom in  $C$  that is greater than or equal to the occurrence of  $A$  with respect to  $\prec$ . Throughout this paper, we will assume a reduction ordering  $\prec$  that is total on ground atoms.

**Denotations and Models.** We define the *denotation*  $\llbracket \bar{\sigma} \rrbracket$  of a substitution expression  $\bar{\sigma}$  inductively as follows:

$$\begin{aligned} \llbracket \sigma \rrbracket &= \{\sigma\} \\ \llbracket \bar{\sigma}\bar{\tau} \rrbracket &= \{\sigma\tau \mid \sigma \in \llbracket \bar{\sigma} \rrbracket, \tau \in \llbracket \bar{\tau} \rrbracket\} \\ \llbracket \bar{\sigma}_1 | \bar{\sigma}_2 \rrbracket &= \llbracket \bar{\sigma}_1 \rrbracket \cup \llbracket \bar{\sigma}_2 \rrbracket \\ \llbracket \bar{\sigma}^* \rrbracket &= \bigcup_{n \geq 0} \llbracket \bar{\sigma}^n \rrbracket \end{aligned}$$

Here  $\bar{\sigma}^0$  denotes the substitution  $\{x \mapsto x \mid x \in \text{dom } \bar{\sigma}\}$ , and  $\bar{\sigma}^{n+1} = \bar{\sigma} \circ \bar{\sigma}^n$ .

Because of the associativity of substitution composition and of set union,  $\llbracket (\bar{\sigma}_1 \bar{\sigma}_2) \bar{\sigma}_3 \rrbracket = \llbracket \bar{\sigma}_1 (\bar{\sigma}_2 \bar{\sigma}_3) \rrbracket$  and  $\llbracket (\bar{\sigma}_1 | \bar{\sigma}_2) | \bar{\sigma}_3 \rrbracket = \llbracket \bar{\sigma}_1 | (\bar{\sigma}_2 | \bar{\sigma}_3) \rrbracket$ , i.e.  $\circ$  and  $|$  are associative. Hence we will identify  $(\bar{\sigma}_1 \bar{\sigma}_2) \bar{\sigma}_3$  and  $\bar{\sigma}_1 (\bar{\sigma}_2 \bar{\sigma}_3)$ , writing both as  $\bar{\sigma}_1 \bar{\sigma}_2 \bar{\sigma}_3$  (and analogously for  $|$ ).

Moreover, we define  $t\llbracket \bar{\sigma} \rrbracket = \{\bar{t}\sigma \mid \sigma \in \llbracket \bar{\sigma} \rrbracket\}$ . A substitution expression  $\bar{\sigma}$  with domain  $\{x_1, \dots, x_n\}$  is *covering for a set*  $T \subseteq \mathcal{T}(\mathcal{F}, X)^n$  if all ground instances of elements of  $T$  are instances of an element of  $(x_1, \dots, x_n)\llbracket \bar{\sigma} \rrbracket$ . If  $\bar{\sigma}$  is covering for  $\mathcal{T}(\mathcal{F})^n$ , we say that  $\bar{\sigma}$  is *covering*.

For a constrained clause  $\bar{v} \approx \bar{x}\bar{\sigma} \parallel C$ , let  $\llbracket \bar{v} \approx \bar{x}\bar{\sigma} \parallel C \rrbracket$  be the (potentially infinite) formula set  $\llbracket \bar{v} \approx \bar{x}\bar{\sigma} \parallel C \rrbracket = \{\forall \bar{y}. \bar{v} \approx \bar{x}\sigma \rightarrow C \mid \sigma \in \llbracket \bar{\sigma} \rrbracket\}$ , where the universal quantifier ranges over the variables of  $\bar{x}\sigma$  and  $C$ . For a set  $N$  of constrained clauses, let  $\llbracket N \rrbracket = \bigcup_{\bar{v} \approx \bar{x}\bar{\sigma} \parallel C \in N} \llbracket \bar{v} \approx \bar{x}\bar{\sigma} \parallel C \rrbracket$ . An interpretation  $\mathcal{I}$  is said to *model*  $N$ , written  $\mathcal{I} \models N$ , if and only if the formula  $\exists \bar{v}. \bigwedge_{\phi \in \llbracket N \rrbracket} \phi$  is valid in  $\mathcal{I}$ . In this case,  $\mathcal{I}$  is called a *model* of  $N$ . A constrained clause set is *satisfiable* if it has a model.

If  $M$  and  $N$  are two constrained clause sets, we write  $N \models M$  if each model of  $N$  is also a model of  $M$ . If  $N$  is satisfiable and Horn, we write  $N \models_{\text{Ind}} M$  if the minimal model of  $N$  models  $M$ .

Two constrained clauses  $\bar{v} \approx \bar{x}\bar{\sigma} \parallel C$  and  $\bar{v} \approx \bar{x}\bar{\sigma}' \parallel C'$  are *variants* if there is a variable renaming  $\pi : \text{VRan}(\bar{\sigma}) \cup \text{vars}(C) \rightarrow \text{VRan}(\bar{\sigma}') \cup \text{vars}(C')$  such that  $\pi$  maps the variables of  $\text{VRan}(\bar{\sigma})$  to  $\text{VRan}(\bar{\sigma}')$ ,  $C\pi = C'$ , and  $\llbracket \bar{\sigma}\pi \rrbracket = \llbracket \bar{\sigma}' \rrbracket$ . If both  $C$  and  $C'$  are unconstrained, this reduces to the usual notion of variants. Note that the denotations of variants agree up to renaming of universally quantified variables. If  $\bar{\sigma}$  is a variable renaming and  $C$  does not contain any variables of  $\bar{v} \approx \bar{x}\bar{\sigma}$ , then we abbreviate the constrained clause as  $\parallel C$ . We call a constrained clause  $\parallel C$  *unconstrained* and identify it with its clausal part  $C$ .

**Inferences and Redundancy.** An *inference rule* is a relation on constrained clauses. Its elements are called *inferences* and written as

$$\frac{\alpha_1 \parallel C_1 \dots \alpha_k \parallel C_k}{\alpha \parallel C} .$$

The constrained clauses  $\alpha_1 \parallel C_1, \dots, \alpha_k \parallel C_k$  are called the *premises* and  $\alpha \parallel C$  the *conclusion* of the inference. An *inference calculus* is a set of inference rules.

A constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  is *redundant* with respect to a constrained clause set  $N$  if  $C$  is a tautology or if there is a variant  $\vec{v} \approx \vec{x} \bar{\tau} \parallel C$  of a constrained clause in  $N$  such that  $\llbracket \bar{\sigma} \rrbracket \subseteq \llbracket \bar{\tau} \rrbracket$ . An inference is called *redundant* with respect to  $N$  if its conclusion is redundant wrt.  $N$  or if a premise  $C$  is redundant wrt.  $N \setminus \{C\}$ . A constrained clause set  $N$  is *saturated* (wrt. a given inference calculus) if each inference with premises in  $N$  is redundant wrt.  $N$ .

A *derivation* is a finite or infinite sequence  $N_0, N_1, \dots$  such that for each  $i$ , there is an inference with premises in  $N_i$  and conclusion  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  that is not redundant wrt.  $N_i$ , such that  $N_{i+1} = N_i \cup \{\vec{v} \approx \vec{x} \bar{\sigma} \parallel C\}$ .

### 3 A Calculus for Constrained Clauses

In [11], we introduced a superposition based calculus that is sound and complete for unsatisfiability of queries  $\exists^* \forall^*(A_1, \dots, A_n \rightarrow)$  with respect to minimal Horn clause models. The basic idea is to express the query as a constrained clause where the constraint part allows a special treatment of the existential variables appearing in a derivation.

We call a set of constrained Horn clauses of the form  $\parallel \Gamma \rightarrow A$  or  $\vec{v} \approx \vec{x} \parallel \Gamma \rightarrow$  an *existential query problem*. Each Horn clause as well as the negation of each query of the form  $\forall \vec{x}. \exists \vec{y}. A_1 \wedge \dots \wedge A_n$  corresponds naturally to an existential query problem. In the example presented in the introduction, where  $V = \{v\}$ , the unconstrained theory clause  $G(x, y) \rightarrow G(s(x), s(y))$  corresponds to the constrained clause  $v \approx x \parallel G(y_1, y_2) \rightarrow G(s(y_1), s(y_2))$ , and the negation of the query  $\forall x. \exists y. G(y, x)$  corresponds to the constrained clause  $v \approx x \parallel G(y, x) \rightarrow$ .

In our current non-equational Horn setting, where all clauses containing a positive literal will be unconstrained, the original calculus boils down to the following single inference rule, which was originally not defined for substitution expressions but for substitutions only:

**Ordered Resolution:**

$$\frac{\Gamma_1 \rightarrow A_1 \quad \vec{v} \approx \vec{x} \bar{\sigma} \parallel \Gamma_2, A_2 \rightarrow \Delta_2}{\vec{v} \approx \vec{x} \bar{\sigma} \tau' \parallel \Gamma_1 \tau, \Gamma_2 \tau \rightarrow \Delta_2 \tau}$$

where (1)  $\tau$  is the most general unifier of  $A_1$  and  $A_2$ , (2)  $\tau' : \text{VRan}(\bar{\sigma}) \rightarrow \mathcal{T}(\mathcal{F}, X)$  maps  $y$  to  $y\tau$  if  $y \in \text{dom}(\bar{\sigma})$  and to  $y$  otherwise, and (3)  $A_1\tau$  is strictly maximal in  $(\Gamma_1 \rightarrow A_1)\tau$  and  $A_2\tau$  is maximal in  $(\Gamma_2, A_2 \rightarrow \Delta_2)\tau$ , where  $\Delta_2$  is either empty or contains a single atom. Note that the rightmost premise may also be unconstrained.



Ordered resolution can as usual be restricted by means of a literal selection function. We now extend the calculus to an inference system consisting of ordered resolution and a *melting* rule. To define melting, we need the notion of an ancestor. In an ordered resolution inference, the *ancestors* of the conclusion are the rightmost premise and all of its ancestors.

**Melting:**

$$\frac{\vec{v} \approx \vec{x} \bar{\sigma} \parallel C \quad \vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau}' \parallel C'}{\vec{v} \approx \vec{x} \bar{\sigma}'' \parallel C}$$

if (1)  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  is an ancestor of  $\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau}' \parallel C'$ , and (2)  $\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau}' \parallel C'$  is a variant of  $\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau} \parallel C$ , and either (3.i)  $\bar{\sigma}$  is of the form  $\bar{\sigma} = \bar{\sigma}_1 \bar{\sigma}_2^*$  and  $\bar{\sigma}'' = \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^*$ , or (3.ii)  $\bar{\sigma}$  is not of this form and  $\bar{\sigma}'' = \bar{\sigma} \bar{\tau}^*$ .

The premise  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  is called the *base clause* for the melting, and the conclusion the *melted clause*. The ancestors of the conclusion are defined as the ancestors of the base clause.

### 3.1 Soundness and Completeness

The melting rule introduces constrained clauses that do not directly follow from the premises. For example, the clauses  $v \approx x \parallel G(y, x) \rightarrow$  and  $v \approx x \sigma \parallel G(y, x) \rightarrow$  by themselves do not imply  $v \approx x \sigma^* \parallel G(y, x) \rightarrow$ . To establish the soundness of our calculus, the main objective will be to prove that all elements of the conclusion's denotation in a melting inference step are really consequences of the premises. Here it is essential that the leftmost premise of each ordered resolution inference is unconstrained.

**Lemma 1 (Soundness of Ordered Resolution).** *The ordered resolution rule is sound.*

*Proof.* For an inference

$$\frac{\Gamma_1 \rightarrow A_1 \quad \vec{v} \approx \vec{x} \bar{\sigma} \parallel \Gamma_2, A_2 \rightarrow \Delta_2}{\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau}' \parallel \Gamma_1 \tau, \Gamma_2 \tau \rightarrow \Delta_2 \tau}$$

to be sound, it suffices that each inference

$$\frac{\Gamma_1 \rightarrow A_1 \quad \vec{v} \approx \vec{x} \bar{\sigma} \parallel \Gamma_2, A_2 \rightarrow \Delta_2}{\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau}' \parallel \Gamma_1 \tau, \Gamma_2 \tau \rightarrow \Delta_2 \tau}$$

for  $\sigma \in \llbracket \bar{\sigma} \rrbracket$  is sound. This is the case because of the soundness of the base calculus rule from [11].

Note that the leftmost premise of each ordered resolution inference is unconstrained, which means that such an inference with rightmost premise  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  and conclusion  $\vec{v} \approx \vec{x} \bar{\sigma} \bar{\tau} \parallel D$  can also be made with any other constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma}' \parallel C$  with the same clausal part but a different constraint, then resulting in  $\vec{v} \approx \vec{x} \bar{\sigma}' \bar{\tau} \parallel D$ . Moreover, if the former inference is sound, so is the latter.

If a derivation starts from an existential query problem, then all stars appearing in constraints during the derivation come from a melting step.

**Lemma 2 (Soundness of Melting).** *For derivations starting from an existential query problem, the melting rule is sound.*

*Proof.* Consider a derivation step from a clause set  $N$  to  $N'$  where a melting inference

$$\frac{\vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \parallel C \quad \vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \bar{\tau}' \parallel C'}{\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^* \parallel C}$$

is performed. We will show that, for each integer  $n \geq 0$ ,  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^n \parallel C$  is implied by the constrained clauses in  $N$ .

Since the derivation started from constrained clauses whose constraints do not contain any stars, the constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \parallel C$  must have been derived from a constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \parallel C$  to account for the star around  $\bar{\sigma}_2$ .

So the case  $n = 0$  is trivial. If  $n > 0$ , assume that the constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \parallel C$  is implied. Moreover, we may inductively assume that all previous steps in the derivation are sound. Starting from  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \parallel C$ , we could do the same set of inference steps needed to derive  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \parallel C$  from  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \parallel C$  to derive the constrained clause  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \bar{\sigma}_2^* \parallel C$ . This directly implies  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \bar{\sigma}_2 \parallel C$ .

Moreover, we could do the same set of inference steps needed to derive  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \bar{\tau} \parallel C$  from  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \bar{\sigma}_2^* \parallel C$  to derive  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \bar{\tau} \parallel C$ .

Thus, we know that both  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \bar{\sigma}_2 \parallel C$  and  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} \bar{\tau} \parallel C$  are implied constrained clauses, and hence also  $\vec{v} \approx \vec{x} \bar{\sigma}_1 (\bar{\sigma}_2 | \bar{\tau})^{n-1} (\bar{\sigma}_2 | \bar{\tau}) \parallel C$  is implied, which is what we wanted to prove.

The argument for a melting of type (3.ii) is similar.

Concerning completeness, we can make use of the following proposition, which is an instance of [11, theorem 1]:

**Proposition 3 (Completeness).** *Let  $N$  be a finite existential query problem, let  $N^*$  be a finite saturation of  $N$  with respect to ordered resolution, and let  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \parallel \square, \dots, \vec{v} \approx \vec{x} \bar{\sigma}_m \parallel \square$  be the constrained clauses in  $N^*$  with empty clausal part. Then  $N$  has a Herbrand model iff  $\bar{\sigma}_1 | \dots | \bar{\sigma}_m$  is not covering.*

As the ordered resolution rule alone is already complete, the same holds for the combination of ordered resolution and melting.

Hence, provided saturation terminates, we can express the initial problem whether  $N \models_{\text{Ind}} \forall \vec{x}. \forall \vec{y}. A_1 \wedge \dots \wedge A_n$  in terms of a coverage problem:

**Corollary 4.** *Let  $N$  be a satisfiable set of unconstrained Horn clauses,  $N^*$  a finite saturation of  $N \cup \{\vec{v} \approx \vec{x} \parallel A_1, \dots, A_n \rightarrow\}$ , and let  $\vec{v} \approx \vec{x} \bar{\sigma}_1 \parallel \square, \dots, \vec{v} \approx \vec{x} \bar{\sigma}_m \parallel \square$  be the set of constrained clauses in  $N^*$  with empty clausal part. The following are equivalent:*

- (1)  $N \models_{\text{Ind}} \forall \vec{x}. \exists \vec{y}. A_1 \wedge \dots \wedge A_n$
- (2)  $\bar{\sigma}_1 | \dots | \bar{\sigma}_m$  is not covering.

### 3.2 Termination

We now show that, if a clause set  $N \cup \{A_1, \dots, A_n \rightarrow\}$  can be finitely saturated, then our calculus finitely saturates  $N \cup \{\vec{v} \approx \vec{x} \parallel A_1, \dots, A_n \rightarrow\}$ . Termination also ensures that the calculus is *fair*, i.e. that every possible inference between derived constrained clauses will finally be redundant.

A derivation strategy that ensures termination is as follows:<sup>1</sup>

- (1) Perform ordered resolution according to the strategy that saturates  $N \cup \{A_1, \dots, A_n \rightarrow\}$ .
- (2) Perform melting inferences eagerly.
- (3) Directly after each melting inference step redo, starting from the melted clause, all previous (non-redundant) resolution inferences that have the base clause as an ancestor and all (non-redundant) meltings that are possible with the newly derived clauses. This is called an *elementary update*. Afterwards continue recursively for the repeated meltings. This whole procedure is called an *update*.

During an update, clauses in the former derivation become redundant. This means that they can and will be ignored for the rest of the derivation and are effectively *replaced* by their more general counterparts. We will use this manner of speaking in the following propositions.

**Lemma 5.** *If clauses in a derivation are arranged in a graph defined by the direct ancestor relation, this graph is a forest (a set of trees).*

*Proof.* Initially, each clause forms its own tree, and no inference can connect existing trees or introduce loops.

**Lemma 6 (Termination of Updates).** *The update following a melting inference step terminates.*

*Proof.* We consider only the case of a melting of type (3.i) as follows:

$$\frac{\vec{v} \approx \vec{x} \bar{\alpha} \bar{\sigma}^* \parallel C \quad \vec{v} \approx \vec{x} \bar{\alpha} \bar{\sigma}^* \bar{\tau}' \parallel C'}{\vec{v} \approx \vec{x} \bar{\alpha} (\bar{\sigma} | \bar{\tau})^* \parallel C}$$

The proof for a melting of type (3.ii) works similarly. Each elementary update is terminating, since there are only finitely many inferences to repeat. We show that the number of elementary updates in an update is finite and proceed by induction over the depth of the base clause in the ancestor-based forest.

Let

$$\frac{\vec{v} \approx \vec{x} \bar{\beta} \parallel E \quad \vec{v} \approx \vec{x} \bar{\beta} \bar{\rho}' \parallel E'}{\vec{v} \approx \vec{x} \bar{\beta}_1 (\bar{\beta}_2 | \bar{\rho})^* \parallel E}$$

be a melting inference that is redundant before the current elementary update. There are several possible cases, depending on whether and where one of the premises of the initial melting appears as an ancestor of  $\vec{v} \approx \vec{x} \bar{\beta} \bar{\rho}' \parallel E'$ :

<sup>1</sup> A more straightforward alternative is presented in [12].

- $\vec{v}\approx\vec{x}\bar{\alpha}\bar{\sigma}^* \parallel C$  is not an ancestor of  $\vec{v}\approx\vec{x}\bar{\beta}\bar{\rho} \parallel E$ . Then this melting is not affected by the elementary update.
- $E = C$ . Then also  $\bar{\beta} = \bar{\alpha}\bar{\sigma}^*$ , and the first elementary update leads to a melting candidate

$$\frac{\vec{v}\approx\vec{x}\bar{\alpha}(\bar{\sigma}|\bar{\tau})^* \parallel C \quad \vec{v}\approx\vec{x}\bar{\alpha}(\bar{\sigma}|\bar{\tau})^*\bar{\rho} \parallel C}{\vec{v}\approx\vec{x}\bar{\beta}(\bar{\sigma}|\bar{\tau})^* \parallel C}$$

Since the original melting was redundant before,  $\llbracket\bar{\rho}\rrbracket \subseteq \llbracket\bar{\sigma}\rrbracket$ . So the new melting candidate is also redundant and this branch of the update stops here.

- $\vec{v}\approx\vec{x}\bar{\alpha}\bar{\sigma}^* \parallel C$  is an ancestor of  $\vec{v}\approx\vec{x}\bar{\beta} \parallel E$ , but not vice versa. Then  $\bar{\beta} = \bar{\alpha}\bar{\sigma}^*\bar{\pi}$  and the elementary update leads to a melting candidate

$$\frac{\vec{v}\approx\vec{x}\bar{\alpha}(\bar{\sigma}|\bar{\tau})^*\bar{\pi} \parallel E \quad \vec{v}\approx\vec{x}\bar{\alpha}(\bar{\sigma}|\bar{\tau})^*\bar{\pi}\bar{\rho} \parallel E}{\vec{v}\approx\vec{x}\bar{\alpha}(\bar{\sigma}|\bar{\tau})^*\bar{\pi}_1(\bar{\pi}_2|\bar{\rho})^* \parallel E}$$

where  $\bar{\pi} = \bar{\pi}_1\bar{\pi}_2^*$ . Since the original melting was redundant before,  $\llbracket\bar{\rho}\rrbracket \subseteq \llbracket\bar{\pi}_2\rrbracket$ . So the new melting candidate is also redundant and this branch of the update stops before the melting.

- $\vec{v}\approx\vec{x}\bar{\beta} \parallel E$  is an ancestor of  $\vec{v}\approx\vec{x}\bar{\alpha}\bar{\sigma}^* \parallel C$ . Then this melting has a base that lies strictly above the base of the originally inspected melting in the ancestor-based clause forest. Because of Lemma 5, we may inductively assume that the update initiated by the melting of  $\vec{v}\approx\vec{x}\bar{\beta} \parallel E$  terminates.

**Lemma 7 (Uniqueness of Melting Steps).** *In a derivation, each constrained clause (including its replacements) is the non-base clause for at most one melting that occurs outside an update.*

*Proof.* Consider the first melting step with a given base clause. Directly after the melting has been performed, it becomes redundant. If one of the premises is later in the derivation replaced during an update and the replacing premise allows a new non-redundant melting, then this melting is executed during the same update.

**Theorem 8 (Termination).** *Let  $N \cup \{A_1, \dots, A_n \rightarrow\}$  be a finite set of Horn clauses that can be finitely saturated by ordered resolution. Then our calculus finitely saturates the clause set  $N \cup \{\vec{v}\approx\vec{x} \parallel A_1, \dots, A_n \rightarrow\}$ .*

*Proof.* Let  $N^*$  be a saturated constrained clause set derived from the set  $N \cup \{\vec{v}\approx\vec{x} \parallel A_1, \dots, A_n \rightarrow\}$ . Because of the finite saturation of  $N \cup \{A_1, \dots, A_n \rightarrow\}$ , only a finite number of clausal parts appears in  $N^*$ .

To show that  $N^*$  is finite, consider again the ancestor-based forest for  $N^*$ , where all “replaced” nodes (cf. the definition of updates) have been erased. We call this forest  $G$ .

We will first show that  $G$  is finite. To do so, we will show that it has only finitely many roots, is of finite depth, and is finitely branching.

Only the constrained clauses in  $N$  (or replacements thereof) can appear as roots. We know that the set  $\{C \mid (\vec{v}\approx\vec{x}\bar{\sigma} \parallel C) \in N^*\}$  (modulo renaming of variables) is finite. Because of our strategy of applying the melting rule whenever

possible, no two constrained clauses  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  and  $\vec{v} \approx \vec{x} \bar{\sigma}' \parallel C'$  in a common branch of  $G$  can be melted, so every branch in  $G$  contains at most a finite number of nodes for each element of this set (one for each subset of  $\text{vars}(C)$ , since these correspond to the possibilities for  $\text{VRan}(\bar{\sigma}) \cap \text{vars}(C)$  and hence to constrained  $C$  clauses in a branch that cannot be melted), i.e. the graph is of finite depth.

Finally, each node in  $G$  has a finite arity: As only finitely many ordered resolution inferences into the constrained clause at this point have been possible (namely from some of the finitely many unconstrained clauses only), plus at most one melting (cf. Lemma 7), the arity is finite.

So  $G$  is finite. It remains to show that only finitely many “replaced” constrained clauses have been excluded from  $G$ . We know from lemma 7 that there has been at most one melting step per node in  $G$  outside of an update. By lemma 6 each of these caused only a finite number of replacements (i.e. deletions for  $G$ ). Hence  $N^*$  is finite.

## 4 Substitution Expressions as Clause Sets

Quantor elimination 6 allows to decide whether a finite disjunction  $\sigma_1 \mid \dots \mid \sigma_n$  of *basic* substitutions is covering. To do so, the problem is reduced to an emptiness problem that is trivially decidable.

For substitution expressions, we follow a related approach. We will transform a substitution expression  $\bar{\sigma}$  into a set  $N_{\bar{\sigma}}$  of Horn clauses, such that  $\bar{\sigma}$  is covering iff a predicate  $P_{\bar{\sigma}}$  arising in the transformation is the total relation in the minimal model of  $N_{\bar{\sigma}}$ , i.e. iff  $N_{\bar{\sigma}} \models_{\text{Ind}} \forall \vec{x}. P_{\bar{\sigma}}(\vec{x})$ . Then a completion procedure allows us to generate a Horn clause set  $\tilde{N}_{\bar{\sigma}}$  for the complement of  $P_{\bar{\sigma}}$ , named  $\tilde{P}_{\bar{\sigma}}$ , such that  $P_{\bar{\sigma}}$  is total in the minimal model of  $N_{\bar{\sigma}}$  iff  $\tilde{P}_{\bar{\sigma}}$  is empty in the minimal model of  $\tilde{N}_{\bar{\sigma}}$ , and such that this emptiness can be decided by ordered resolution.

For the rest of this chapter, we require that any two substitutions appearing in a substitution expression are named differently, i.e. we never write  $\{x \mapsto x\} \circ \{x \mapsto x\}$  as  $\sigma \circ \sigma$ , but possibly as  $\sigma_1 \circ \sigma_2$ .

**Definition 9 (Substitutions and Predicates).** *Given a substitution expression  $\bar{\sigma}$ , we assign a predicate  $P_{\bar{\tau}}$  to every substitution expression  $\bar{\tau}$  that is a subexpression of  $\bar{\sigma}$ . If  $\bar{\tau}$  is a substitution or disjunction or loop, we let  $P_{\bar{\tau}}$  be a fresh predicate of arity  $|\text{dom}(\bar{\tau})|$ . We set  $P_{\bar{\tau}} = P_{\bar{\tau}_1}$  if  $\bar{\tau} = \bar{\tau}_1 \bar{\tau}_2$  is a composition.*

**Definition 10 (Substitutions and Clauses).** *We translate substitution expressions  $\bar{\sigma}$  to clause sets  $N_{\bar{\sigma}}^0$  (which is just an intermediate representation) and  $N_{\bar{\sigma}}$  as follows. Let  $P_{\text{glue}}$  be a fresh predicate of arity 0. This predicate will be used as a means to glue together the sets corresponding to different substitution expressions. We assume an ordering on the domain  $\text{dom}(\bar{\sigma}) = \{x_1, \dots, x_n\}$  of any given substitution and write  $\vec{x} = \text{dom}(\bar{\sigma})$  if  $x_1 < \dots < x_n$ . Expressions of the form  $R[B/A]$  denote textual replacement of every occurrence of atom  $A$  in the clause set  $R$  by the atom  $B$ .*

$$\begin{aligned}
N_\sigma^0 &= \{P_{glue} \rightarrow P_\sigma(\vec{x}\sigma)\} \text{ where } \vec{x} = \text{dom}(\sigma) \\
N_{\bar{\sigma}\bar{\tau}}^0 &= N_\tau^0 \cup N_{\bar{\sigma}}^0[P_{\bar{\tau}}(\text{dom}(\bar{\tau}))]/P_{glue}] \\
N_{\bar{\sigma}^*}^0 &= \{P_{glue} \rightarrow P_{\bar{\sigma}^*}(\text{dom}(\bar{\sigma}))\} \cup N_{\bar{\sigma}}^0[P_{\bar{\sigma}^*}(\text{dom}(\bar{\sigma}))]/P_{glue}] \\
&\quad \cup \{P_{\bar{\sigma}}(\text{dom}(\bar{\sigma})) \rightarrow P_{\bar{\sigma}^*}(\text{dom}(\bar{\sigma}))\} \\
N_{\bar{\sigma}_1|\bar{\sigma}_2}^0 &= N_{\bar{\sigma}_1}^0 \cup \{P_{\bar{\sigma}_1}(\vec{x}_1) \rightarrow P_{\bar{\sigma}_1|\bar{\sigma}_2}(\vec{y})\} \cup N_{\bar{\sigma}_2}^0 \cup \{P_{\bar{\sigma}_2}(\vec{x}_2) \rightarrow P_{\bar{\sigma}_1|\bar{\sigma}_2}(\vec{y})\} \\
&\quad \text{where } \vec{x}_i = \text{dom}(\bar{\sigma}_i) \text{ and } \vec{y} = \text{dom}(\bar{\sigma}_1|\bar{\sigma}_2)
\end{aligned}$$

The set  $N_{\bar{\sigma}}$  arises from  $N_{\bar{\sigma}}^0$  by deletion of all occurrences of  $P_{glue}$ .

*Example 11.* We consider the two substitutions  $\sigma = \{x \mapsto s(x)\}$  and  $\tau = \{x \mapsto 0\}$ . Then  $N_\sigma^0 = \{P_{glue} \rightarrow P_\sigma(s(x))\}$  and  $N_\tau^0 = \{P_{glue} \rightarrow P_\sigma(s(x))\}$ .

The set  $N_{\sigma^*}$  consists of the clauses  $\rightarrow P_{\sigma^*}(x)$ ,  $P_{\sigma^*}(x) \rightarrow P_\sigma(s(x))$ , and  $P_\sigma(x) \rightarrow P_{\sigma^*}(x)$  and  $N_{\sigma^*\tau} = \{\rightarrow P_\tau(0)$ ,  $P_\tau(x) \rightarrow P_{\sigma^*\tau}(x)$ ,  $P_{\sigma^*\tau}(x) \rightarrow P_\sigma(s(x))$ ,  $P_\sigma(x) \rightarrow P_{\sigma^*\tau}(x)\}$ . (Note that  $P_{\sigma^*\tau}$  equals  $P_{\sigma^*}$ .)  $N_{\sigma^*\tau}$  builds terms bottom-up: The first clause creates the constant 0, the second clause enters the  $\sigma$  loop, and the last two clauses allow to repeatedly wrap applications of  $s(\ )$  around the term.

We will now show that each clause set  $N_{\bar{\sigma}}$  describes exactly the (instances of the) term tuples generated by the respective substitution expression  $\bar{\sigma}$ , provided that this substitution stems from a derivation.

**Definition 12.** Let  $\vec{v} \approx \vec{x}\bar{\sigma}_1, \dots, \vec{v} \approx \vec{x}\bar{\sigma}_n$  be finitely many constraints appearing in a derivation starting from an existential query problem. Then  $\bar{\sigma}_1 | \dots | \bar{\sigma}_n$  is called a derivation substitution.

The clauses in  $N_{\bar{\sigma}}^0$  and  $N_{\bar{\sigma}}$  are particularly simple. On the one hand, all terms appearing on the left hand side are variables, on the other hand, the clauses are universally reductive, i.e. all these variables also occur in the head, a property that is necessary for the applicability of the completion procedure we will use later on:

**Proposition 13 (Universal Reductiveness of  $N_{\bar{\sigma}}$ ).** Let  $\bar{\sigma}$  be a derivation substitution. Then  $N_{\bar{\sigma}}$  is universally reductive.

**Proposition 14 (Equivalence of Substitution Expressions and Clauses).** Let  $\bar{\sigma}$  be a derivation substitution. Then  $\vec{t} \in \vec{x}[\bar{\sigma}]$  iff  $P_{\bar{\sigma}}(\vec{t})$  can be derived from  $N_{\bar{\sigma}}$  by resolution.

Since resolution is first-order complete, we can conclude that the terms entailed by  $N_{\bar{\sigma}}$  are exactly those covered by  $\bar{\sigma}$ :

**Corollary 15.** Let  $\bar{\sigma}$  be a derivation substitution. Then  $N_{\bar{\sigma}} \models P_{\bar{\sigma}}(\vec{t})$  iff  $\bar{\sigma}$  is covering for  $\{\vec{t}\}$ , i.e. the set  $\{\vec{t} \mid N_{\bar{\sigma}} \models P_{\bar{\sigma}}(\vec{t})\}$  is the maximal set for which  $\bar{\sigma}$  is covering.

## 4.1 Predicate Completion

Now that we know how to transform  $\bar{\sigma}$  into an equivalent set  $N_{\bar{\sigma}}$  of Horn clauses, we will concentrate on how to decide whether  $P_{\bar{\sigma}}$  is the total relation in the minimal model of  $N_{\bar{\sigma}}$ .

Comon and Nieuwenhuis [7] introduced a predicate completion procedure based on Clark's completion [5] and quantifier elimination [6]. Given a universally reductive Horn clause set  $R$  over a finite signature  $\Sigma$ , this procedure computes a set  $R'$  of (possibly equational) clauses. Let  $\check{N}$  arise from  $R'$  by replacing each atom  $P(\vec{t})$  by  $\neg\check{P}(\vec{t})$ . Then  $R \models_{\text{Ind}} P(\vec{x})$  iff  $\check{N} \models_{\text{Ind}} \neg\check{P}(\vec{x})$ . For general non-equational clause sets  $R$ , the clauses in  $\check{N}$  are of the form  $\Gamma \rightarrow \check{P}(\vec{t}) \wedge E$ , where  $E$  is a conjunction of syntactic equations. E.g.,  $\{Q(s(x)) \rightarrow P(x, x, z)\}$  is transformed to  $\{\check{Q}(s(x)) \rightarrow \check{P}(x, y, z), x \simeq y\}$ . In the case of derivation substitutions, however,  $\check{N}$  does not contain equations and falls into a class for which emptiness is decidable by ordered resolution (Theorem [17]).

**Proposition 16 (Shape of  $\check{N}_{\bar{\sigma}}$ ).** *Let  $\bar{\sigma}$  be a derivation substitution such that all substitutions appearing in  $\bar{\sigma}$  are linear. For the predicates in  $N_{\bar{\sigma}}$ , the algorithm by Comon and Nieuwenhuis computes clauses of the following types:*

- (1)  $\rightarrow \check{P}(\vec{t})$
- (2)  $\check{P}_1(\vec{x}) \rightarrow \check{P}(\vec{t})$
- (3)  $\check{P}_1(\vec{x}), \check{P}_2(\vec{x}) \rightarrow \check{P}(\vec{x})$

*The positive literal of each computed clause is linear.*

That all negative literals contain only variables is inherited from  $N_{\bar{\sigma}}$ . That no equations appear is due to the linearity of all positive literals in  $N_{\bar{\sigma}}$ .

In the example presented in the appendix, all predicates are monadic. When there is more than one existential variable or when the signature contains function symbols of arity at least two, also predicates of higher arity appear in  $\check{N}$ .

**Theorem 17 (Decidability of Coverage).** *Let  $\bar{\sigma}$  be a derivation substitution over a finite signature such that all basic substitutions in  $\bar{\sigma}$  have a unary domain and are linear. It is decidable whether  $\bar{\sigma}$  is covering.*

*Proof.* We translate  $\bar{\sigma}$  into a clause set  $N_{\bar{\sigma}}$ . All predicates in  $N_{\bar{\sigma}}$  are unary. The resulting clause set  $\check{N}_{\bar{\sigma}}$  defining the completion of all appearing predicates again contains only clauses of the form  $\check{P}_1(x), \dots, \check{P}_n(x) \rightarrow \check{P}(t)$  (Proposition [16]). Weidenbach [16] showed that such a clause set is equivalent to a so-called *sort theory*, a clause set in which additionally all clauses are shallow. For sort theories, emptiness is decidable by ordered resolution [16, 15]. Emptiness of  $\check{P}_{\bar{\sigma}}$  in turn is equivalent to the coverage of the substitution  $\bar{\sigma}$  (completion and Corollary [15]).

## 5 Decidability of Inductive Validity

As a combination of our complete and terminating ordered resolution calculus for constrained clauses and the completion-based treatment of the substitution expressions that can appear during saturation, we obtain the following decidability result:

**Theorem 18.** *Let  $N \cup \{A_1, \dots, A_n \rightarrow\}$  be a set of Horn clauses without equality over a finite signature  $\Sigma$ , where*

- (1) *all function symbols in  $\Sigma$  are at most unary,*
- (2) *all positive literals in  $N$  are linear, and*
- (3)  *$N \cup \{A_1, \dots, A_n \rightarrow\}$  can be finitely saturated by ordered resolution,*

*Let  $x, y_1, \dots, y_m$  be the variables in  $A_1, \dots, A_n$ . Then it is decidable whether  $N \models_{Ind} \forall x. \exists y_1, \dots, y_m. (A_1 \wedge \dots \wedge A_n)$ .*

*Proof.* By Corollary 4 and Theorem 8 the clause set  $N \cup \{v \approx x \parallel A_1, \dots, A_n \rightarrow\}$  can be finitely saturated by ordered resolution with melting, such that the deduced constrained clauses  $v \approx x \bar{\sigma}_1 \parallel \square, \dots, v \approx x \bar{\sigma}_k \parallel \square$  with empty clausal part in the saturated set correspond to a substitution expression  $\bar{\sigma} = \bar{\sigma}_1 | \dots | \bar{\sigma}_k$  that is covering iff  $N \models_{Ind} \forall x. \exists y_1, \dots, y_m. (A_1 \wedge \dots \wedge A_n)$ .

Since the domain  $\text{dom}(\bar{\sigma}) = \{x\}$  of  $\bar{\sigma}$  contains only one element and  $\Sigma$  contains only unary function symbols, the domain of all basic substitutions appearing in  $\bar{\sigma}$  has cardinality 1. These substitutions are also linear because all positive literals in  $N$  are linear and hence the most general unifiers appearing in each resolution step are linear. Hence coverage of  $\bar{\sigma}$  is decidable by Theorem 17.

## 6 Conclusion

We have shown that the problem  $N \models_{Ind} \forall\exists^*(A_1 \wedge \dots \wedge A_n)$  is decidable over a finite signature consisting of constants and unary function symbols if all positive literals in  $N$  are linear and  $N \cup \{A_1, \dots, A_n \rightarrow\}$  belongs to a class that can be finitely saturated by first-order ordered resolution (with variant subsumption and tautology deletion). Our proof is constructive and based on an ordered resolution calculus for constrained clauses  $\vec{v} \approx \vec{x} \bar{\sigma} \parallel C$  and predicate completion.

Among the related work on automated inductive theorem proving, the approach most closely related to ours is the one by Comon and Nieuwenhuis 7. Given a universally reductive clause set  $N$  and a query  $\forall^*C$ , they use Clark's completion to compute a so-called  $I$ -axiomatization  $M$  and check the first-order satisfiability of  $N \cup M \cup \{C\}$ . This method is complete but not terminating. In fact, since  $M$  is a clause set over the original predicates  $P$  (and not over  $\bar{P}$ ), it is usually not Horn nor does the saturation of  $N \cup M \cup \{C\}$  terminate.

Another general method based on saturation is the one by Ganzinger and Stuber 10. Given a universally reductive clause set  $N$  and a query  $\forall^*C$ , they basically saturate  $N \cup \{C\}$ . Even if  $N \cup \{C\}$  saturates finitely, this results in a non-complete procedure. They also present a way to guarantee completeness, at the cost that the resulting algorithm almost never terminates.

A subtle, but important difference between both these methods and ours is that we add the negated query to  $N$  for saturation (while they add the query positively), which makes all derived clauses also hold in the minimal model.

Another intensely studied approach is via test sets 13, 2, 3. Test set methods are complete for several classes of equational clauses or rewrite systems and



universal queries. Termination results for these approaches usually require strong properties like a terminating rewrite system on constructor terms.

Approaches in the tradition of Caferra and Zabel [4] or Kapur [13, 8] also provide decision procedures. However, they also consider only term rewrite systems. Related publications by Peltier [14] require that  $N$  has a unique Herbrand model (not only a unique minimal one), which leads back to I-Axiomatizations.

In summary, our approach is the first to yield a both terminating and complete algorithm to decide the inductive validity of queries that contain a  $\forall\exists^*$  quantifier alternation.

Extensions of the approach might include a relaxation of its side conditions. Although both the superposition calculus of [11] and the completion procedure of [7] are also applicable to clauses containing equality literals, it is not obvious how to extend this treatment of equality also to clauses containing substitution expressions. The main problem here is that term rewriting cannot easily be extended to the rewriting of substitution expressions.

However, since both our resolution calculus and the completion procedure work equally well on  $\forall^*\exists^*$  queries, on clauses over an arbitrary signature, and on clauses containing non-linear positive atoms, the reduction to an emptiness problem is also possible in these extended settings. The resulting set  $\check{R}$  may contain both non-monic predicates and equational atoms. It is a natural next step to explore under which conditions these extensions lead to predicates  $\check{P}$  that are nevertheless defined in such a way that emptiness remains decidable.

**Acknowledgements.** We thank our reviewers for their detailed and valuable comments. Matthias Horbach and Christoph Weidenbach are supported by the German Transregional Collaborative Research Center SFB/TR 14 AVACS.

## References

- [1] Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation* 4(3), 217–247 (1994)
- [2] Bouhoula, A.: Automated theorem proving by test set induction. *Journal of Symbolic Computation* 23(1), 47–77 (1997)
- [3] Bouhoula, A., Jouannaud, J.-P.: Automata-driven automated induction. In: *Information and Computation*, Warsaw, Poland, pp. 14–25 (1997) (press)
- [4] Caferra, R., Zabel, N.: A method for simultaneous search for refutations and models by equational constraint solving. *J. Symb. Comp.* 13(6), 613–642 (1992)
- [5] Clark, K.L.: Negation as failure. In: *Logic and Data Bases*, pp. 293–322 (1977)
- [6] Comon, H., Lescanne, P.: Equational problems and disunification. *Journal of Symbolic Computation* 7(3-4), 371–425 (1989)
- [7] Comon, H., Nieuwenhuis, R.: Induction = I-axiomatization + first-order consistency. *Information and Computation* 159(1/2), 151–186 (2000)
- [8] Falke, S., Kapur, D.: Inductive decidability using implicit induction. In: Hermann, M., Voronkov, A. (eds.) *LPAR 2006. LNCS (LNAI)*, vol. 4246, pp. 45–59. Springer, Heidelberg (2006)
- [9] Ganzinger, H., Nivelle, H.D.: A superposition decision procedure for the guarded fragment with equality. In: *Proc. 14th IEEE Symposium on Logic in Computer Science*, pp. 295–305. IEEE Computer Society Press, Los Alamitos (1999)

- [10] Ganzinger, H., Stuber, J.: Inductive theorem proving by consistency for first-order clauses. In: Rusinowitch, M., Remy, J.-L. (eds.) CTRS 1992. LNCS, vol. 656, pp. 226–241. Springer, Heidelberg (1993)
- [11] Horbach, M., Weidenbach, C.: Superposition for fixed domains. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 293–307. Springer, Heidelberg (2008)
- [12] Horbach, M., Weidenbach, C.: Deciding the inductive validity of  $\forall\exists^*$  queries. Research Report MPI-I-2009-RG1-001, Max-Planck Institute for Informatics, Saarbrücken, Germany (May 2009)
- [13] Kapur, D., Narendran, P., Zhang, H.: Automating inductionless induction using test sets. *Journal of Symbolic Computation* 11(1/2), 81–111 (1991)
- [14] Peltier, N.: Model building with ordered resolution: extracting models from saturated clause sets. *Journal of Symbolic Computation* 36(1-2), 5–48 (2003)
- [15] Seidl, H., Verma, K.N.: Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 79–94. Springer, Heidelberg (2005)
- [16] Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE 1999. LNCS (LNAI), vol. 1632, pp. 314–328. Springer, Heidelberg (1999)
- [17] Weidenbach, C.: Combining superposition, sorts and splitting. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, ch. 27, vol. 2, pp. 1965–2012. Elsevier, Amsterdam (2001)

# On the Parameterised Intractability of Monadic Second-Order Logic

Stephan Kreutzer\*

Oxford University Computing Laboratory  
kreutzer@comlab.ox.ac.uk

**Abstract.** One of Courcelle’s celebrated results states that if  $\mathcal{C}$  is a class of graphs of bounded tree-width, then model-checking for monadic second order logic ( $\text{MSO}_2$ ) is fixed-parameter tractable (fpt) on  $\mathcal{C}$  by linear time parameterised algorithms. An immediate question is whether this is best possible or whether the result can be extended to classes of unbounded tree-width.

In this paper we show that in terms of tree-width, the theorem can not be extended much further. More specifically, we show that if  $\mathcal{C}$  is a class of graphs which is closed under colourings and satisfies certain constructibility conditions such that the tree-width of  $\mathcal{C}$  is not bounded by  $\log^{16} n$  then  $\text{MSO}_2$ -model checking is not fpt unless SAT can be solved in sub-exponential time. If the tree-width of  $\mathcal{C}$  is not poly-log. bounded, then  $\text{MSO}_2$ -model checking is not fpt unless all problems in the polynomial-time hierarchy can be solved in sub-exponential time.

## 1 Introduction

In 1990, Courcelle proved a fundamental result stating that every property of graphs definable in *monadic second-order logic with edge set quantification* ( $\text{MSO}_2$ ) can be decided in linear time on any class  $\mathcal{C}$  of graphs of bounded tree-width. Courcelle’s theorem has important consequences both in logic and in algorithm theory. In the design of efficient algorithms on graphs, it can often be used as a simple way of establishing that a property can be solved in linear time on graph classes of bounded tree-width. Besides being of interest for specific algorithmic problems, results such as Courcelle’s and similar *algorithmic meta-theorems* lead to a better understanding how far certain algorithmic techniques, dynamic programming and decomposition in the case of  $\text{MSO}_2$ , range and establish general upper bounds for the parameterised complexity of a wide range of problems. See [9,10] for recent surveys on algorithmic meta-theorems.

From a logical perspective, Courcelle’s theorem establishes a sufficient condition for tractability of  $\text{MSO}_2$  formula evaluation on classes of graphs or structures: whatever the class  $\mathcal{C}$  may look like, if it has bounded tree-width, then  $\text{MSO}_2$ -model checking is tractable on  $\mathcal{C}$ . An obvious question to ask is how tight Courcelle’s theorem is, i.e. whether it can be extended to classes of unbounded tree-width and if so, how large the tree-width of graphs in the class can be in general. Given the considerable interest in Courcelle’s theorem, it is somewhat surprising that not much is known about such limits

---

\* Research supported by DFG grant KR 2898/1-3. Part of this work was done while the author participated at the workshop “Graph Minors” at Banff Intern. Research Station, October 2008.

for MSO<sub>2</sub> model checking. Recently, the question has informally been raised in the community and has led, e.g., to a conjecture by Grohe [9, Conjecture 8.3] that MSO-model checking is not fixed-parameter tractable on any class  $\mathcal{C}$  of graphs which is closed under taking subgraphs and whose tree-width is not poly-logarithmically bounded, i.e. there are no constants  $c, d$  such that  $\text{tw}(G) \leq d \cdot \log^c |G|$  for all  $G \in \mathcal{C}$ . But to the best of my knowledge, the question has so far not been studied systematically. This is the main motivation for the work reported in this paper.

It follows from the NP-completeness of 3-colourability on planar graphs [8] that MSO-model checking is not fixed-parameter tractable on the class of planar graphs (unless  $P = \text{NP}$ ). Furthermore, it is a simple consequence of the excluded grid theorem that on minor- or topological-minor closed classes of graphs of unbounded tree-width, MSO-model checking is not fpt unless  $P = \text{NP}$  (see Section 2). In this paper we establish a strong intractability result by showing that in terms of tree-width, Courcelle’s theorem can not be extended much further to classes of unbounded tree-width. Throughout the paper, we will work with coloured graphs, i.e. we will fix a set  $\Gamma$  of edge and vertex colours. A class  $\mathcal{C}$  of graphs is said to be closed under  $\Gamma$ -colourings if whenever  $G \in \mathcal{C}$  and  $G'$  is obtained from  $G$  by recolouring, i.e. the underlying undirected graphs are isomorphic, then  $G' \in \mathcal{C}$ . We will mostly consider classes of graphs closed under colourings. An alternative characterisation is to consider relational structures over a signature  $\sigma$  with at most binary relation symbols. We can then fix a class  $\mathcal{C}'$  of graphs and consider the class of all finite  $\sigma$ -structures whose Gaifman-graphs are in  $\mathcal{C}$ . However, in this paper we prefer to work with coloured graphs rather than Gaifman-graphs of structures. Given a class  $\mathcal{C}$  of graphs, we write  $\text{MC}(\text{MSO}_2, \mathcal{C})$  for the model-checking problem for MSO<sub>2</sub> on  $\mathcal{C}$  (see Section 2 for details).

**Definition 1.1.** The tree-width of a class  $\mathcal{C}$  of graphs is *strongly unbounded* by a function  $f : \mathbb{N} \rightarrow \mathbb{N}$  if there is a polynomial  $p(x)$  such that for all  $n \geq 1$  there is a graph  $G \in \mathcal{C}$  of tree-width between  $n$  and  $p(n)$  whose tree-width is not bounded by  $f(|G|)$  and 2) given  $n$ ,  $G_n$  can be constructed in time  $2^{(|n|_u)^\varepsilon}$ , for some  $\varepsilon < 1$ , where  $|n|_u$  denotes the unary encoding of  $n$ . The tree-width of  $\mathcal{C}$  is *strongly unbounded poly-logarithmically* if it is strongly unbounded by  $\log^c n$ , for all  $c$ .

Essentially, *strongly* means that a) there are not too big gaps between the tree-width of graphs witnessing that the tree-width of  $\mathcal{C}$  is not bounded by  $f(n)$  and b) we can compute such witnesses efficiently. We will see below why this condition is needed. The following is the main result of the paper. Let  $\Gamma$  be a set of colours with at least one edge and two vertex colours.

**Theorem 1.2.** *Let  $\mathcal{C}$  be a constructible class of  $\Gamma$ -coloured graphs closed under colourings.*

1. *If the tree-width of  $\mathcal{C}$  is strongly unbounded poly-logarithmically then  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is not in XP, and hence not fpt, unless all problems in NP (in fact, all problems in the polynomial-time hierarchy) can be solved in sub-exponential time.*
2. *If the tree-width of  $\mathcal{C}$  is strongly unbounded by  $\log^{16} n$  then  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is not in XP unless SAT can be solved in sub-exponential time.*

See Section 2 for a definition of FPT and XP. We refer to Definition 3.6 for a precise definition of constructible classes but will explain the concept informally below. Let

us give some applications of the theorem. For  $c > 0$  let  $\mathcal{C}_c$  be the class of all graphs  $G$  of tree-width at most  $\log^c |G|$ . This class is constructible and hence its closure under colourings has intractable  $\text{MSO}_2$  model-checking, if  $c > 16$ . Similarly, the class of planar graphs of tree-width at most  $\log^c n$  is constructible. Finally, all (topological) minor-closed classes of unbounded tree-width are constructible and rich. All these examples show that Courcelle's theorem can not be extended to classes of graphs with only poly-logarithmic or a  $\log^c n$  bound on the tree-width, for  $c > 16$ .

**High level description of the proof.** Let us give an intuitive account of the proof of the previous theorem. Clearly, with today's methods we cannot hope to prove that  $\text{MSO}_2$ -model-checking is fixed-parameter intractable for a class of graphs without relating it to assumptions in complexity theory. Consequently, we prove that  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is fixed-parameter intractable for a class  $\mathcal{C}$  by reducing an NP-complete problem  $P$  to  $\text{MC}(\text{MSO}_2, \mathcal{C})$  such that if there is an fpt-algorithm for  $\text{MC}(\text{MSO}_2, \mathcal{C})$ , then  $P$  can be solved in sub-exponential time  $2^{o(n)}$ . More precisely, for each language  $P \in \text{NP}$  we construct a formula  $\varphi_P$  and then, given a word  $w$ , we construct a graph  $G_w \in \mathcal{C}$  such that  $G_w \models \varphi_P$  if, and only if,  $w \in P$ . We will see that the number of vertices of  $G_w$  can be bounded by  $2^{|w|^{\frac{1}{y}}}$ , for some  $y > 1$ , so that if there was an algorithm for  $\text{MC}(\text{MSO}_2, \mathcal{C})$  with running time  $f(|\varphi|) \cdot |G|^c$  then this would imply that  $w \in P$  could be decided in time  $\mathcal{O}(2^{c|w|^{\frac{1}{y}}}) = 2^{o(|w|)}$ . Here, Condition 1) of Definition 1.1 ensures that  $\mathcal{C}$  contains a graph  $G_w$  the word  $w$  can be reduced to and Condition 2) ensures that we can compute it in time sub-exponential in the length of  $|w|$ . For this reduction to work, we need some intermediate steps.

It is well-known that  $\text{MSO}_2$ -model checking is fixed-parameter intractable on the class of coloured grids (see Figure 1), which can be seen as follows: suppose  $P$  can be solved by a non-deterministic Turing-machine  $M$  in time  $n^c$ , where  $n$  is the length of the input. Given a word  $w$  of length  $n$ , we choose a  $(n^c \times n^c)$ -grid  $G_w$  and label its top-most row by  $w$  from left to right. From the Turing-machine  $M$  deciding  $P$  we can compute an  $\text{MSO}_2$ -formula  $\varphi_M$  depending only on  $M$  such that  $G_w \models \varphi_M$  if, and only if,  $w$  is accepted by  $M$  and hence  $w \in P$ . Essentially, the  $\text{MSO}$  formula uses the grid to guess the computation table of a successful run of  $M$  on  $w$ . Hence, an fpt-algorithm for  $\text{MSO}$ -model checking on grids yields a polynomial time algorithm for  $P$ .

Clearly, if we are just given a class of graphs of tree-width not bounded poly-logarithmically, then there is no guarantee that it contains any grids. But we will show that we can define grids in graphs of this class by  $\text{MSO}_2$ -formulas. Adapting a recent proof by Reed and Wood, we first show that if  $G$  is a graph of large tree-width then it contains a large structure which we call *coloured pseudo-wall*. Pseudo-walls do not actually occur as minors or subgraphs of  $G$  but as topological minors of certain intersection graphs formed by sets of disjoint paths in  $G$ . However, it turns out that this is enough to define coloured grids in coloured pseudo-walls by  $\text{MSO}_2$ -formulas. It follows that if the tree-width of  $G$  is not bounded by  $k$  then we can define an  $(l \times l)$ -grid in  $G$  in  $\text{MSO}_2$ , where  $l$  is roughly  $\sqrt[10]{k}$  (see Theorem 3.5 for details), and this grid can be coloured. We call a class *constructible* if we can construct these pseudo-walls in graphs  $G \in \mathcal{C}$  in polynomial time and it is such classes with which we work in this paper (see Definition 3.6 for details).

The important aspect here is that the size of the grids we define is polynomially related to the tree-width of the graph, in contrast to the grids guaranteed by the excluded grid theorem (see Theorem 3.1), where the tree-width is exponentially larger than the grids we are guaranteed to find. Hence, using pseudo-walls, if the tree-width of a graph is  $\log^c n$  then we can define  $(l \times l)$ -grids in  $G$  for  $l \cong \log^{\frac{c}{10}} n$  and this is enough for the reduction sketched above to work.

Obtaining sub-exponential time algorithms for problems such as TSP or SAT is an important open problem in the algorithms community and the common assumption is that no such algorithm exists. This has led to the *exponential-time hypothesis* (ETH) which says that there is no such sub-exponential time algorithm for SAT, a hypothesis widely believed in the community.

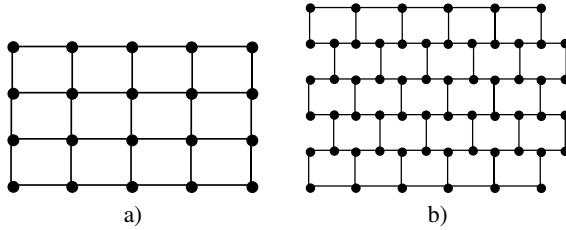
Let us briefly comment on the restrictions imposed on the classes  $\mathcal{C}$  we study here. While every graph of large enough tree-width contains large pseudo-walls, we do not yet know if we can always compute these structures in polynomial time (and hence we impose the additional restriction to constructible classes). It is conceivable that large pseudo-walls can indeed be computed in polynomial time. This would essentially show that all classes are constructible and effectively remove this condition from our main result. We pose this question as an open problem.

As mentioned above, Grohe conjectured that MSO-model checking is not fpt on any class of graphs closed under subgraphs and whose tree-width is not poly-logarithmically bounded. The statement of the conjecture and the main result of this paper are incomparable as I require closure under colourings whereas Grohe does not. On the other hand, the conjecture requires closure under subgraphs which I do not. Note that while tree-width is preserved by taking subgraphs, logarithmic tree-width is not, i.e. a graph whose tree-width is bounded by  $\log n$  may contain a subgraph of order  $m$  whose tree-width is not bounded by  $\log m$ . Closure under subgraphs therefore does rule out natural examples of graph classes. For instance, the class of all graphs of tree-width at most  $\log n$  is not closed under subgraphs. On the other hand, Grohe's conjecture does not require colours or constructibility conditions and refers to classes of plain graphs.

Note that it is important for our results that we work with  $\text{MSO}_2$  and allow quantification over sets of edges as well as over sets of vertices. If we only consider vertex set quantification, i.e. deal with  $\text{MSO}_1$ , then the theorem is false, as for instance,  $\text{MSO}_1$ -model checking is fpt on the class of cliques.

Following Courcelle's theorem, a series of algorithmic meta-theorems for first-order logic on planar graphs [7], (locally) minor-free graphs [6,2] and various other classes have been obtained. Again, no deep lower bounds, i.e. intractability conditions, are known (see [10] for some simple bounds and [9,10] for recent surveys of the topic). The aim of this paper is to initiate a thorough study of sufficient conditions for intractability in terms of structural properties of input instances.

**Organisation.** We recall monadic second-order logic and what we need about its parameterised complexity in Section 2. The main result is then proved as follows. To show that  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is hard on classes  $\mathcal{C}$  of tree-width not poly-logarithmically bounded, we first use a result by Reed and Wood [12] to show that any graph of large enough tree-width contains a structure that is grid-like enough for our purposes. This is proved in Section 3. While these structures do not exist as minors in the graphs, they turn out



**Fig. 1.** a) A  $(4 \times 5)$ -grid and b) an elementary wall of order 5

to be  $\text{MSO}_2$ -definable, which is shown in Section 4. To use the  $\text{MSO}_2$ -definability for our result, we introduce a new kind of interpretations between structures, called  $\text{MSO}_2 - \text{MSO}_2$ -transductions (see Section 5). Finally, in Section 6, we combine all this to show the main result of the paper. Due to space restrictions, some proofs have been removed from this abstract. See <http://arxiv.org/abs/0904.1302> for a full version.

## 2 Complexity of Monadic Second-Order Logic

We first need some notation and a few concepts from graph theory. We refer to [3] for background on graphs. For  $k \geq 1$ , we define  $[k] := \{1 \dots, k\}$ . All graphs in this paper are finite and undirected. We write  $V(G)$  for the set of vertices and  $E(G)$  for the set of edges in a graph  $G$ . A graph  $H$  is a *sub-division* of  $G$  (a *1-subdivision*) if  $H$  is obtained from  $G$  by replacing edges in  $G$  by paths of arbitrary length (of length 2, resp.).  $H$  is a *topological minor* of  $G$  if a subgraph  $G' \subseteq G$  is isomorphic to a sub-division of  $H$ .

An *elementary wall*  $W$  is a graph as in Figure 1b). The cycles of minimal length in  $W$  are called *bricks*. A *wall* is a subdivision of an elementary wall. The *height* of a wall is the number of rows of bricks and its *width* the number of columns of bricks. An  $l \times k$ -wall is a wall of height  $l$  and width  $k$  and a wall of *order*  $l$  is an  $l \times l$ -wall. Finally, the *nails* of a wall are the vertices of degree 3 in it together with the 4 corners. Hence, in an elementary wall all vertices are nails whereas in a general wall only the vertices of the underlying elementary wall are nails.

For the purpose of this paper, it might be easier to think of  $k \times k$ -grids instead of  $k \times k$ -walls and everything would go through with grids also. The important property of walls is that their maximum degree is 3. And if a graph  $H$  of degree  $\leq 3$  is a minor of  $G$ , then  $H$  is also a topological minor of  $G$  (see [3, Prop. 1.7.2]). Hence, a sub-division of  $H$  actually occurs as subgraph of  $G$ . Defining topological minors in  $\text{MSO}_2$  is much easier than defining minors as we do not need contraction. We will therefore work with walls instead of grids in this paper.

I assume familiarity with basic notions of mathematical logic (see e.g. [4]). In this paper we will only consider signatures  $\sigma := \{E, B_1, \dots, B_s, C_1, \dots, C_t\}$  of coloured graphs, where  $E$  denotes the edge relation,  $B_i$  the colours of edges and  $C_i$  the colours of vertices. We allow multiple colours per edge or vertex. We denote  $\sigma$ -structures by Roman letters  $A, G, H, \dots$ . If  $R \in \sigma$  is a relation symbol and  $A$  a  $\sigma$ -structure, we write  $R(A)$  for the interpretation of  $R$  in  $A$ .

The class of formulas of *monadic second-order logic with edge set quantification* over a signature  $\sigma$ , denoted  $\text{MSO}_2[\sigma]$ , is defined by the rules for first-order logic with the following additional rules: if  $X$  is a second-order variable either ranging over sets of vertices or over sets of edges and  $\varphi \in \text{MSO}_2[\sigma \dot{\cup} \{X\}]$ , then  $\exists X \varphi \in \text{MSO}_2[\sigma]$  and  $\forall X \varphi \in \text{MSO}_2[\sigma]$  with the obvious semantics where, e.g., a formula  $\exists F \varphi$ ,  $F$  being a variable over sets of edges, is true in a structure  $A$  if there is a subset  $F \subseteq E(A)$  such that  $(A, F) \models \varphi$ . If  $\varphi(x)$  is a formula with a free variable  $x$  and  $A$  is a structure, we write  $\varphi(A)$  for the set  $\{a : A \models \varphi[a]\}$ . See [11] for more on MSO.

In [15], Vardi proved that model checking for  $\text{MSO}_2$  is PSPACE-complete on the class of all graphs. The complexity of model-checking problems can elegantly be studied in the framework of *parameterised complexity* (see [5] for background on parameterised complexity). If  $\mathcal{C}$  is a class of  $\sigma$ -structures, we define the *parameterised model-checking problem*  $\text{MC}(\text{MSO}_2, \mathcal{C})$  for  $\text{MSO}_2$  on  $\mathcal{C}$  as the problem to decide, given  $G \in \mathcal{C}$  and  $\varphi \in \text{MSO}_2[\sigma]$ , if  $G \models \varphi$ . The *parameter* is  $|\varphi|$ . The problem is *fixed-parameter tractable* (fpt), or in the class FPT, if there is a computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  and  $k \in \mathbb{N}$ , such that for all  $G \in \mathcal{C}$  and  $\varphi \in \text{MSO}_2[\sigma]$ ,  $G \models \varphi$  can be decided in time  $f(|\varphi|) \cdot |G|^k$ . The problem is in the class XP, if it can be decided in time  $|G|^{f(|\varphi|)}$ . FPT in the parameterised world corresponds to polynomial-time in the classical framework as the class of problems that can be solved efficiently. XP can be seen as the parameterised exponential-time and is obviously a much larger class of problems than FPT.

Tree-width is a global connectivity measure of graphs that was introduced by Robertson and Seymour in their graph minor series. We refer the reader to [3] for a definition of tree-width. Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be a function and  $\mathcal{C}$  be a class of graphs. The tree-width of  $\mathcal{C}$  is *bounded by  $f$* , if  $\text{tw}(G) \leq f(|G|)$  for all  $G \in \mathcal{C}$ .  $\mathcal{C}$  has *bounded tree-width* if its tree-width is bounded by a constant. Many natural classes of graphs, for instance series-parallel graphs, are found to have bounded tree-width. The following lemma, whose proof is standard, will be needed below.

**Lemma 2.1.** *Let  $M$  be a non-deterministic Turing-machine. There is a formula  $\varphi_M \in \text{MSO}_2$  such that for all words  $w \in \Sigma^*$ , if  $G$  is a  $k \times k$ -wall whose top-most row is coloured by  $w$  from the left, then  $G \models \varphi_M$  if, and only if,  $M$  accepts  $w$  in at most  $k$  steps. Furthermore, the formula  $\varphi_M$  can be constructed effectively from  $M$ . The same holds if  $M$  is an alternating Turing-machine with a bounded number of alternations, as they are used to define the polynomial-time hierarchy.*

Lemma 2.1 together with Theorem 3.1 shows that if  $\mathcal{C}$  is closed under (topological) minors and has unbounded tree-width, then  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is not fpt unless  $P = \text{NP}$ .

### 3 Pseudo-walls in Graphs

One of the fundamental results of Robertson and Seymour’s theory of graph minors is the excluded grid theorem [14], saying that there is a computable function  $f : \mathbb{N} \rightarrow \mathbb{N}$  such that every graph of tree-width at least  $f(k)$  contains a  $k \times k$ -grid as a minor. The best explicit bound known for the function  $f$  is given by the following theorem.

**Theorem 3.1** (Robertson, Seymour, Thomas [13]). *Every graph of tree-width at least  $20^{2 \cdot k^5}$  contains a  $k \times k$  grid as a minor.*



Robertson et al. [13] also proved that there are graphs of tree-width proportional to  $k^2 \log k$  that do not contain  $G_{k \times k}$  as a minor. So far this is the best lower bound known for the function  $f$  above. In particular it is open whether  $f(k)$  above can be bounded by a polynomial. In [12] Reed and Wood consider a different type of obstructions to small tree-width, called *grid-like* minors. A grid-like minor of order  $l$  in a graph  $G$  is a set  $\mathcal{P}$  of paths in  $G$  such that the intersection graph  $\mathcal{I}(\mathcal{P})$  contains a  $K_l$ -minor, where  $K_l$  denotes the complete graph on  $l$  vertices. Here, the *intersection graph* of a set  $\mathcal{P}$  of paths is the graph with vertex set  $\mathcal{P}$  and an edge between two paths  $P, Q \in \mathcal{P}$  if  $P \cap Q \neq \emptyset$ . If  $\mathcal{P}, \mathcal{Q}$  are sets of paths in  $G$ , we write  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$  for  $\mathcal{I}(\mathcal{P} \dot{\cup} \mathcal{Q})$ , the intersection graph of their disjoint union.

**Theorem 3.2** (Reed, Wood [12]). *Every graph of tree-width at least  $ck^4 \sqrt{\log k}$  contains a grid-like minor of order  $k$ , for some constant  $c$ . Conversely, every graph that contains a grid-like minor of order  $l$  has tree-width at least  $\lceil \frac{l}{2} \rceil - 1$ .*

While I do not yet know how to use this result directly, we can use its proof to find the structures in  $G$  we need.

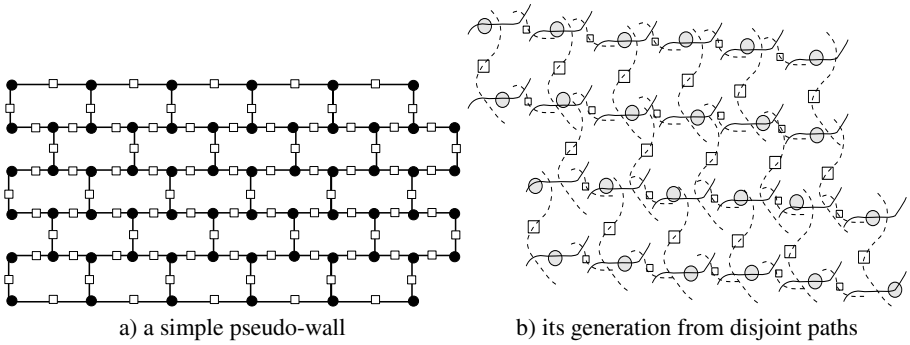
**Definition 3.3.** A *pseudo-wall* of order  $l$  in  $G$  is a pair  $(\mathcal{P}, \mathcal{Q})$  of sets of disjoint paths in  $G$  such that  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$  is a wall of order  $l$ .

We will see below that every graph of large enough tree-width contains a large pseudo-wall and that these can be defined in  $\text{MSO}_2$ . Essentially, to show that  $\text{MSO}_2$  model-checking is fixed-parameter intractable on a class  $\mathcal{C}$  of large enough tree-width, we will use pseudo-walls in a similar way as walls are used in Lemma 2.1. In particular, we want to label the top-most row of the pseudo-wall by a word  $w$  over a finite alphabet. However, pseudo-walls do not occur as subgraphs of the graphs  $G$ , which makes labelling them somewhat more difficult. Instead, we have to colour the graph  $G$  so that this colouring induces the labelling of the pseudo-wall it contains. The main difficulty is that the colouring of  $G$  must induce a unique labelling of the pseudo-wall and that both the pseudo-wall as well as its labelling can be defined inside  $G$  by  $\text{MSO}_2$ -formulas. Unfortunately, this makes the definition of a coloured pseudo-wall technically somewhat more complicated. Let  $\Sigma$  be a set of colours and let  $B$  be an additional colour for edges and  $R$  an additional colour for vertices. Let  $\Gamma := \{B, R\} \dot{\cup} \Sigma$ .

A  $\Sigma$ -coloured pseudo-wall of order  $l$  in a  $\Gamma$ -coloured graph  $G$  is a triple  $(\mathcal{P}, \mathcal{Q}, A)$  such that one of the following holds:

**Simple pseudo-walls.**  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$  is a 1-subdivision of an elementary wall  $W$  of order  $l$  such that the vertices of  $W$  (which we called *nails* above) are exactly the paths in  $\mathcal{P}$ . See Figure 2 for an illustration. Figure 2 a) shows the pseudo-wall, where the solid black circles are the vertices from  $\mathcal{P}$  and squares denote the vertices from  $\mathcal{Q}$ . Figure 2 b) shows how (a part of) this pseudo-wall corresponds to paths in  $G$ , where dashed lines represent paths in  $\mathcal{Q}$  and solid lines paths in  $\mathcal{P}$ . Note, though, that in general the paths could intersect in much more complicated ways than displayed and that paths can intersect more than 3 other paths although walls have maximal degree 3.

Let  $\mathcal{P} := \{P_1, \dots, P_k\}$  be such that  $P_1 \dots P_l$  form the nails of the top-most row of  $W$  in order from left to right. Recall that each  $P_i$  is a path in  $G$ . Then  $A$  is the path in  $G$  obtained from the “concatenation”  $P_1 \cdot P_2 \cdots P_k$ , i.e.  $V(A) := \bigcup_{1 \leq i \leq k} V(P_i)$



**Fig. 2.** A simple pseudo-wall and the paths  $\mathcal{P}$  (solid) and  $\mathcal{Q}$  (dashed) generating it

and  $E(A)$  consists of  $\bigcup_{1 \leq i \leq k} E(P_i)$  together with additional edges connecting one endpoint of  $P_i$  to an endpoint of  $P_{i+1}$ , for  $1 \leq i < k$ , so that this results in a path.

Furthermore, the edges in  $E(A)$  are coloured by colour  $B \in \Gamma$ . The two endpoints of each  $P_i$  are coloured by  $R$  and the vertices in the paths  $P_1, \dots, P_l$  carry colours from  $\Sigma$  so that all vertices in a path  $P_i$  are coloured by the same colour from  $\Sigma$ .

This colouring of  $G$  induces a labelling of the wall of order  $l$  where the nails  $v_1, \dots, v_l$  in the top-most row are labelled so that  $v_i$  is labelled by the colour  $C_i \in \Sigma$  of the path  $P_i$ . If  $w := C_1 \dots C_l$  is the sequence of colours on  $P$  we say that  $(\mathcal{P}, \mathcal{Q}, A)$  encodes the word  $w \in \Sigma^*$ .

The motivation behind simple pseudo-walls is as follows. If we find this structure in a graph  $G$  then the path  $A$  tells us what the top-most row of the wall is and it also gives us an order on the vertices of the top-most row. Colouring  $A$  by  $B$  will enable us to define this coloured pseudo-wall in  $\text{MSO}_2$ . If we want to encode a word  $w := w_1, \dots, w_l \in \Sigma^*$  in the wall then we can simply label the paths  $P_1, \dots, P_l$  in  $G$  which form the top-most row of the wall by  $w_1, \dots, w_l$  and this induces the correct labelling of the wall  $\mathcal{I}(\mathcal{Q}, \mathcal{P})$ .

**Complex pseudo-walls.** Complex walls are structures as illustrated in Figure 3. Essentially, they consist of a subdivision  $W'$  of a wall  $W$  in  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ . To define the colouring of the wall, there will be additional paths in  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$  connecting some of the vertices of the top-most row of  $W'$  to the path  $A$  so that the order is preserved, i.e. the paths do not “cross”. We can then colour the path  $A$  and thereby induce a colouring of the top-most row.

Formally, for complex coloured pseudo-walls,  $A$  is a path in  $G$  such that each  $U \in \mathcal{P}$  has exactly one endpoint in  $A$  and no path in  $\mathcal{Q}$  has an endpoint in  $A$ . Furthermore, there are subsets  $\mathcal{P}' \subseteq \mathcal{P}$  and  $\mathcal{Q}' \subseteq \mathcal{Q}$  such that  $\mathcal{I}' := \mathcal{I}(\mathcal{P}', \mathcal{Q}')$  is a wall of order  $l$ .

Let  $T \subseteq \mathcal{I}'$  be the top-most row of the wall and let  $x_1 \dots x_k$  be the vertices of  $T$  in order from left to right. Let  $I := \{i_1, \dots, i_l\}$  be the index set such that  $x_{i_1}$  is the top-left corner,  $x_{i_l}$  is the top-right corner and  $(x_{i_j})_{1 < j < l}$  lists the vertices in  $T$  of degree 3 in order from left to right. For  $1 < s < t \leq l$  let  $T(s, t]$  be the segment of  $T$  between  $x_{i_s}$  and  $x_{i_t}$  including the latter but not the former. We define  $T[0, 1]$  to be the segment containing the vertices  $x_{i_1}, \dots, x_{i_2}$ . Now, the sets  $\mathcal{P}_r := \mathcal{P} \setminus \mathcal{P}'$  and  $\mathcal{Q}_r := \mathcal{Q} \setminus \mathcal{Q}'$  induce disjoint paths  $P_1 \dots P_l$  in  $\mathcal{I} := \mathcal{I}(\mathcal{P}, \mathcal{Q})$  (i.e. each  $P_i$  consists of a set of paths

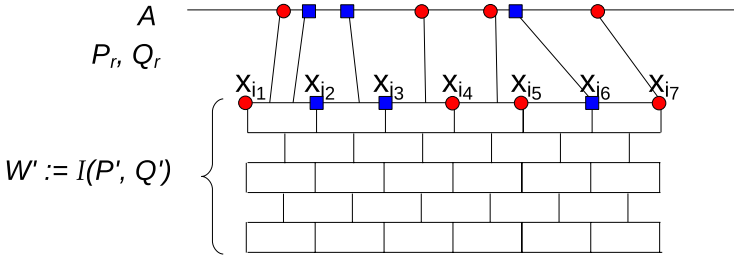


Fig. 3. A complex pseudo-wall

in  $G$ ) such that 1) one endpoint of each path  $P_i$  in  $\mathcal{I}$  is incident to a vertex  $x_i$  of the top-most row of the wall so that each  $T(s, t]$ , for  $1 < s < t \leq l$ , contains exactly one  $x_i$  and  $T[0, 1]$  contains 2 and 2) for the other endpoint  $u_i$  of  $P_i$  in  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$  (which is a path in  $G$ ) we have  $\{v_i\} = u_i \cap A$ , where  $v_i \in V(G)$ , and 3)  $v_1, \dots, v_l$  occur in this order on  $A$ .

Now suppose  $v_1 \dots v_l$  are coloured by  $C_1 \dots C_l$  respectively. Then this colouring induces the labelling of  $\mathcal{I}(\mathcal{P}', \mathcal{Q}')$  where  $x_{i_s}$  gets colour  $C_s$ ,  $1 \leq s \leq l$ . We say that  $(\mathcal{P}, \mathcal{Q}, A)$  encodes the word  $w := C_1 \dots C_l$ .

A crucial feature of pseudo-walls in coloured graphs is that they are unique in the sense that if  $G$  is a graph coloured by  $\{B, R\} \dot{\cup} \Sigma$ , then every pseudo-wall  $(\mathcal{P}, \mathcal{Q}, A)$  in  $G$  encodes the same word  $w$  (there may be no coloured pseudo-wall in  $G$ ).

This is obvious for simple pseudo-walls, as the path  $A$  is uniquely determined by its colouring ( $B$ -edges and the  $P_i$ 's separated by  $R$ -vertices) and this uniquely determines the colouring of the wall and hence the encoded word. For complex walls, the path  $A$  is again determined by its colouring and this fixes the order of the colours occurring on  $A$  and hence on the wall. Here we use that the paths connecting  $A$  to the wall preserve the order.

**Definition 3.4.** We say that a graph  $G$  encodes  $w \in \Sigma^*$  if it contains a  $\Sigma$ -coloured pseudo-wall encoding  $w$ . We say that  $G$  encodes  $w$  with power  $k$ , for some  $k \geq 1$ , if  $G$  contains a  $\Sigma$ -coloured pseudo-wall of order  $|w|^k$  encoding  $w$ .

The proof of the next theorem is essentially the proof of Theorem 3.2 with some modifications to get coloured pseudo-walls instead of grid-like minors.

**Theorem 3.5.** *There is a constant  $c$  such that if  $G$  is a graph of tree-width at least  $c \cdot m^8 \cdot \sqrt{\log(m^2)}$ , then  $G$  contains a  $\Sigma$ -coloured pseudo-wall of order  $m$ .*

We can now give a formal definition of *constructible* classes of graphs.

**Definition 3.6.** Let  $\mathcal{C}$  be a class of graphs closed under  $\Gamma$ -colourings.  $\mathcal{C}$  is called *constructible* if in every graph  $G \in \mathcal{C}$  of tree-width at least  $c \cdot m^8 \cdot \sqrt{\log(m^2)}$ , where  $c$  is from Theorem 3.5, we can compute in polynomial time a coloured pseudo-wall of order  $m$ .

It is conceivable that the large pseudo-walls whose existence we proved above can always be computed in polynomial time. This would imply that all classes of graphs are constructible. We leave this for future research.

Theorem 3.5 shows that in any graph of sufficiently large tree-width we find a large pseudo-wall. We will show below that this is enough to define large walls in graphs of large tree-width by means of  $\text{MSO}_2$ -formulas. It follows from Theorem 3.5 above that if  $\mathcal{C}$  is a class of graphs of unbounded tree-width which is closed under colouring then for each  $w \in \Sigma^*$ ,  $\mathcal{C}$  contains a graph encoding  $w$ . In fact, for each  $c \geq 1$ ,  $\mathcal{C}$  contains a graph encoding  $w$  with power  $c$ . The following lemma summarises what we will need about colourings in the following sections.

**Lemma 3.7.** *Let  $\mathcal{C}$  be a class of graphs closed under  $\Gamma$ -colourings and let  $w \in \Sigma^*$  be a word of length  $m$ . If there is a graph  $G \in \mathcal{C}$  of tree-width  $c \cdot (m^k)^8 \cdot \sqrt{\log(m^k)^2}$ , where  $c$  is the constant from Theorem 3.5 whose tree-width is not bounded by  $\log^{8k} |G|$  then there is a graph  $G \in \mathcal{C}$  encoding  $w$  with power  $k$  such that  $|G| < 2^{c' \cdot m^{\frac{1}{y}}}$ , for some constants  $y > 1$  and  $c' := c(k)$  depending on  $k$  but not on  $w$ .*

## 4 Defining Coloured Pseudo-walls in Graphs of Large Tree-Width

In this section we aim at defining  $\Sigma$ -coloured pseudo-walls in graphs of large enough tree-width in  $\text{MSO}_2$ . Fix a set  $\Sigma$  of colours and let  $\Gamma := \Sigma \dot{\cup} \{B, R\}$  be as defined in Section 3. Let  $G$  be a  $\Gamma$ -coloured graph and  $\mathcal{P}, \mathcal{Q}, A \subseteq E(G)$  be sets of edges. For  $(\mathcal{P}, \mathcal{Q}, A)$  to be a  $\Sigma$ -coloured pseudo-wall in  $G$ , we first need to say that  $\mathcal{P}$  and  $\mathcal{Q}$  are sets of pairwise disjoint paths in  $G$ . Note that  $\mathcal{P}$  induces a set of pairwise disjoint paths if, and only if, *i*) every vertex  $v \in G$  is incident to at most two edges in  $\mathcal{P}$  and *ii*) the subgraph of  $G$  induced by the edges in  $\mathcal{P}$  is acyclic. This can easily be defined in  $\text{MSO}_2$  and we will see the formulas below for the more complicated case of paths and acyclicity in  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ . Furthermore, we have to say that the edges of  $A$ , and only those, are coloured by  $B$ . Now, we have to distinguish between simple and complex coloured pseudo-walls. This can easily be done in  $\text{MSO}_2$  as in the first case  $\bigcup_{P \in \mathcal{P}} P \subseteq A$  (at least in the pseudo-walls generated in the previous section, in general pseudo-walls this is only true for the paths in the top-most row, but that could equally be used to distinguish the two types of walls) whereas this fails in the second. We will present the case for simple pseudo-walls explicitly. The other case follows using the same ideas.

We first need a few auxiliary formulas. To ease the presentation we assume that no path  $P$  occurs in both  $\mathcal{P}$  and  $\mathcal{Q}$ . This is guaranteed by the pseudo-walls generated in Section 3 but we could also easily modify the formulas below to avoid this assumption (see also Section 5).

In what follows we will use  $\text{MSO}_2$ -formulas, interpreted in  $G$ , to speak about the intersection graph  $\mathcal{I} := \mathcal{I}(\mathcal{P}, \mathcal{Q})$ . To increase readability of formulas we agree on the following convention: lower case letters are used for first-order variables, variables  $P, Q, \dots$  range over sets of edges and variables  $E, F, H$  range over sets of vertices. It may seem bizarre to use  $F, H$  for a set of vertices. The reason will become clear below as we will be using variables  $E$  for sets of vertices in  $G$  which represent sets of edges in  $\mathcal{I}$ . As a final piece of notation, we write “ $P \in \mathcal{P}$ ” to say that  $P$  is a component of

$\mathcal{P}$ , i.e. one of the paths in  $\mathcal{P}$ , and analogously for  $Q \in \mathcal{Q}$ . Furthermore, we will write  $x \in V(P)$  for the formula  $\exists y Pxy$  to say that  $x$  is adjacent to an edge in  $P$ .

Recall that for two paths  $P \in \mathcal{P}$  and  $Q \in \mathcal{Q}$  there is an edge  $\{P, Q\} \in E(\mathcal{I})$  if  $P \cap Q \neq \emptyset$  in  $G$ . As  $\mathcal{P}$  and  $\mathcal{Q}$  are sets of disjoint paths, there are no three distinct paths in  $\mathcal{P} \cup \mathcal{Q}$  intersecting in a single vertex. Hence, we can represent edges  $\{P, Q\} \in E(\mathcal{I})$  by a vertex  $v \in V(P \cap Q)$ . However, in  $\text{MSO}_2$  we cannot pick a single vertex from  $V(P \cap Q)$  and therefore will represent the edge  $\{P, Q\}$  by the set  $V(P \cap Q)$ . Let  $\varphi_E(x) := \exists P \in \mathcal{P} \exists Q \in \mathcal{Q} x \in V(P \cap Q)$ ,  $\text{inc}(x, P) := x \in V(P)$  and  $x \sim y := \exists P \in \mathcal{P} \exists Q \in \mathcal{Q} x, y \in V(P \cap Q)$  be  $\text{MSO}_2$ -formulas, where we will usually write  $\sim(x, y)$  in infix notation.  $\sim$  defines an equivalence relation on the set of vertices satisfying  $\varphi_E(x)$  and we can represent edges in  $\mathcal{I}$  by equivalence classes of  $\sim$  in  $G$ . Hence,  $\mathcal{I}$  is isomorphic to the graph  $\mathfrak{J} := (V, E, \sigma)$  with vertex set  $V := \mathcal{P} \cup \mathcal{Q}$  and edge set  $E := \{[x]_{\sim} : x \in \varphi_E(G)\}$ , where a vertex  $P \in V$  is incident with an edge  $e \in E$  if there is a vertex  $v \in e \cap P$  (and hence  $e \subseteq P$ ).  $\mathfrak{J}$  is  $\text{MSO}_2$ -definable in  $G$ , by the formulas  $\varphi_E$ ,  $\text{inc}$  and  $\sim$  with parameters  $\mathcal{P}, \mathcal{Q}$  and we can represent variables over elements of  $\mathfrak{J}$  by variables ranging over sets of edges in  $G$  by enforcing that these are interpreted by a path from either  $\mathcal{P}$  or  $\mathcal{Q}$ . Variables  $X$  over sets of elements of  $\mathfrak{J}$  can be represented in  $G$  by pairs  $X_P, X_Q$  of variables ranging over sets of edges so that a set  $X \subseteq V(\mathfrak{J})$  is represented by the pair of sets  $X_P := X \cap \mathcal{P}$  and  $X_Q := X \cap \mathcal{Q}$ . Finally, sets  $F \subseteq E(\mathfrak{J})$  of edges can be represented by sets  $F' \subseteq \varphi_E(G)$  closed under  $\sim$  so that if  $\{P, Q\} \in F$  then  $V(P \cap Q) \subseteq F'$ . Using this idea we can then say about  $\mathfrak{J}$ , and hence about  $\mathcal{I}$ , that  $\mathfrak{J}$  is a wall as follows: 1) There are two sets  $\mathcal{H}, \mathcal{V} \subseteq E(\mathcal{I})$  of edges, each of which induces a set of pairwise vertex disjoint paths in  $\mathcal{I}$  (which we will think of as horizontal and vertical paths in a wall). 2) For all  $P \in \mathcal{H}$  and  $Q \in \mathcal{V}$ ,  $P \cap Q$  is connected and  $V(P \cap Q) \cap V(H) = \emptyset$  for all  $H \in (\mathcal{V} \cup \mathcal{H}) \setminus \{P, Q\}$ . 3) There is a path  $L \in \mathcal{V}$  such that the intersection of  $L$  with each  $Q \in \mathcal{H}$  contains an endpoint of  $Q$  (we think of  $L$  as the left-most vertical path in the wall). Once we have  $L$ , we can give the horizontal paths  $P \in \mathcal{H}$  a direction, where we say that  $p \in V(P)$  is to the left of  $p' \in V(P)$ , if the subpath of  $P$  containing  $p'$  and a vertex in  $L$  also contains  $p$ . 4) There is a path  $T \in \mathcal{H}$  such that the intersection of  $T$  with each  $P \in \mathcal{V}$  contains an endpoint of  $P$  ( $T$  is the top-most horizontal path in the wall). We can now use  $T$  to give the vertical paths  $P \in \mathcal{V}$  a direction and say that  $p \in V(P)$  is above  $p' \in V(P)$ , if the subpath of  $P$  containing  $p'$  and a vertex in  $T$  also contains  $p$ . 5) For each path  $P \in \mathcal{V}$  except  $L$  there is a path  $P' \in \mathcal{V}$  (the path immediately to the left of  $P$ ) such that for all  $Q \in \mathcal{H}$ : if  $p \in V(P \cap Q)$  and  $p' \in V(P' \cap Q)$  are vertices in the intersection of  $Q$  and  $P, P'$  resp., then  $p'$  is to the left of  $p$  in  $Q$  and there is no  $S \in \mathcal{H}$  such that any  $s \in V(S \cap Q)$  lies in the subpath of  $Q$  between  $p$  and  $p'$ . The analogue condition for horizontal paths.

To demonstrate the idea of the  $\text{MSO}_2$ -formalisation we give precise formulas for the set  $\mathcal{H}$  in Condition 1. It will be clear that the other conditions can be formalised analogously. We have to say that there is a set  $\mathcal{H} \subseteq E(\mathfrak{J})$  of edges inducing a set of pairwise disjoint paths in  $\mathfrak{J}$ . To define this in  $G$ , we first need a formula  $\text{Path}(P, Q, H)$  saying that there is a path from  $P \in \mathcal{P} \cup \mathcal{Q}$  to  $Q \in \mathcal{P} \cup \mathcal{Q}$  using only edges from  $H$ , where  $H$  is a subset of  $\varphi_E(G)$ , closed under  $\sim$ , representing edges in  $\mathfrak{J}$ . The usual way of expressing that two vertices  $x, y$  in a graph are connected within a set  $H$  of

edges is to say that all sets  $U$  of vertices which contain  $x$  and are closed under the edge relation  $H$  also contain  $y$ . In our case, sets of vertices of  $\mathcal{J}$  are represented by pairs of sets  $U_P \subseteq \mathcal{P}$  and  $U_Q \subseteq \mathcal{Q}$  consisting of connected components of  $\mathcal{P}$  and  $\mathcal{Q}$ . Hence, the idea above is expressed by the formula  $\text{Path}(P, Q, H)$  defined as

$$\forall U_P \subseteq \mathcal{P} \forall U_Q \subseteq \mathcal{Q} \left( (P \in U_P \cup U_Q \wedge \forall X, Y \in \mathcal{P} \cup \mathcal{Q} [X \in U_P \cup U_Q \wedge \exists e (e \in H \wedge \text{inc}(e, X) \wedge \text{inc}(e, Y)) \rightarrow Y \in U_P \cup U_Q]) \rightarrow Q \in U_P \cup U_Q \right)$$

where we write  $X \in U_P \cup U_Q$  to say that  $X$  is a component either of  $U_P$  or  $U_Q$  and  $U_P \subseteq \mathcal{P}$  to say that  $U_P$  is a set of components of  $\mathcal{P}$ . Now, we can say that  $\mathcal{H}$  induces a set of pairwise disjoint paths as follows. We first say that every vertex in  $\mathcal{H}$  has degree at most 2:  $\forall P \in \mathcal{P} \cup \mathcal{Q} (\exists \leq 2 f \in \mathcal{H} \text{ inc}(f, P))$ , where  $\exists \leq 2 f \dots$  is an abbreviation for: there are at most 2 edges  $f$  such that  $\dots$ . To say that  $\mathcal{H}$  induces an acyclic graph we say that for all  $P \in \mathcal{P} \cup \mathcal{Q}$ , if  $P$  is incident to an edge  $e := \{P, Q\} \in \mathcal{H}$  then there is no path from  $P$  to  $Q$  in  $\mathcal{H} - e$ . The latter can be expressed using the formula  $\text{Path}$  above. This concludes the formalisation of Condition 1).

Clearly, if  $\mathcal{V}$  and  $\mathcal{H}$  satisfy the conditions above, then they generate a wall in  $\mathcal{J}$  and conversely, the disjoint horizontal and vertical paths of a wall satisfy the conditions. Hence,  $\mathcal{J}$  is a wall if such  $\mathcal{V}$  and  $\mathcal{H}$  exist containing all vertices and edges of  $\mathcal{J}$ . Formalising all this gives us a formula which says of  $\mathcal{P}, \mathcal{Q}$  that the pair  $(\mathcal{P}, \mathcal{Q})$  is a pseudo-wall. Note that so far we have not used the additional path  $A$ . Hence, if we are not interested in coloured pseudo-walls but simply in pseudo-walls we can use this formula.

We now proceed to define coloured walls and the induced colouring of  $\mathcal{I}(\mathcal{P}, \mathcal{Q})$ . From the formalisation above we now have sets  $\mathcal{H}, \mathcal{V}$  containing the horizontal and vertical paths of the wall as well as two paths  $L, T$  giving the top-most row and left-most column. The left-most row gives us an ordering on the top-most row and all we have to do is to define the colours of the vertices on the top-most row from the additional path  $A$ , which is easily done. Hence, we can write formulas  $\varphi_C(P)$ , for  $C \in \Sigma$  which are true for the vertices in the wall coloured by  $C$ . Complex pseudo-walls can be defined analogously. Taken together, we have a formula  $\varphi_U(\mathcal{P}, \mathcal{Q}, A)$  which says that  $(\mathcal{P}, \mathcal{Q}, A)$  is a coloured pseudo-wall. Here, the sets  $\mathcal{P}$  and  $\mathcal{Q}$  define the vertices of the pseudo-wall whereas  $A$  is an additional parameter used in the formulas. It will be convenient to take the sets  $T, L$  defining the top- and left-most row and column as parameters also rather than defining them. Hence, we have a formula  $\varphi_U(\mathcal{P}, \mathcal{Q}, A, L, T)$  which says that  $(\mathcal{P}, \mathcal{Q}, A)$  is a  $\Sigma$ -coloured pseudo-wall with left-most column  $L$  and top-most row  $T$ , formulas  $\varphi_E(x, \mathcal{P}, \mathcal{Q}, A, L, T)$ ,  $\text{inc}(x, \mathcal{P}, \mathcal{Q}, A, L, T)$  and  $\sim(x, y, \mathcal{P}, \mathcal{Q}, A, L, T)$  defining the edge relation of the pseudo-wall and formulas  $\varphi_B(x, \mathcal{P}, \mathcal{Q}, A, L, T)$  and  $\varphi_C(P, \mathcal{P}, \mathcal{Q}, A, L, T)$ , where  $C \in \Sigma \dot{\cup} \{R\}$ , defining the colours.

All formulas together define, in graphs of large enough tree-width coloured properly, a large wall whose top-most row is labelled by a word over  $\Sigma$ . Hence, if  $\mathcal{C}$  is a class of graphs of unbounded tree-width, closed under colourings, we can define arbitrarily large coloured walls in  $\mathcal{C}$ . We know already that (presumably)  $\text{MSO}_2$ -model checking is not fixed-parameter tractable on the class of coloured walls. To prove the main result of this paper we need a way to translate  $\text{MSO}_2$ -formulas  $\varphi$  over walls to  $\text{MSO}_2$ -formulas  $\varphi^*$  over the graphs in which we define the walls. We could do this in an ad-hoc way

and modify the formulas  $\varphi_U \dots$  for each given formula  $\varphi$ . We find it more convenient, though, to treat these modifications uniformly within the framework of interpretations. In the next section we therefore introduce a new form of interpretations which simplifies dealing with the intersection graphs we have to define and which might also be useful elsewhere.

## 5 MSO<sub>2</sub>–MSO<sub>2</sub>-Transductions

In this section we introduce a class of interpretations, called MSO<sub>2</sub>–MSO<sub>2</sub>-*transductions*, between classes of graphs which allow us to define one class  $\mathcal{B}$  of graphs inside another class  $\mathcal{C}$  so that we can translate MSO<sub>2</sub>-formulas over  $\mathcal{B}$  to MSO<sub>2</sub>-formulas saying the same over the graphs in  $\mathcal{C}$ . Unlike first-order interpretations, MSO<sub>2</sub>–MSO<sub>2</sub>-transductions associate with every structure a class of structures and in this sense resemble MSO-transductions as, e.g., studied by Courcelle. Let  $\sigma := \{E, B_1, \dots, B_t, C_1, \dots, C_s\}$  be a signature of coloured graphs as defined in Section 2. Let  $\tau$  be a signature.

**Definition 5.1** (MSO<sub>2</sub>–MSO<sub>2</sub>-transduction). Let  $\overline{U} := U_1, \dots, U_k$  and  $\overline{X} := X_1, \dots, X_l$  be tuples of binary relation symbols. An MSO<sub>2</sub>–MSO<sub>2</sub>-transduction of  $\sigma$  in  $\tau$  with parameters  $\overline{U}, \overline{X}$  is a tuple  $\Theta := \left( \varphi_U(U_1, \dots, U_k, \overline{X}), \left( \varphi_E^{i,j}(x), \text{inc}_E^{i,j}(x, P, Q), \sim^{i,j}(x) \right)_{1 \leq i < j \leq k}, \left( \varphi_F^{i,j}(x) \right)_{1 \leq i < j \leq k, F \in \{B_1, \dots, B_t\}}, \left( \varphi_C(P)^i \right)_{C \in \sigma, 1 \leq i \leq k} \right)$ , where  $P, Q$  are unary second-order variables and  $x$  is a first-order variable, such that for all  $\tau$ -structures  $A$  and sets  $\overline{U}, \overline{X} \subseteq E(A)$  with  $(A, \overline{U}, \overline{X}) \models \varphi_U$ :

- $\sim^{i,j}$  defines an equivalence relation on  $\varphi_E^{i,j}(A)$
- for all  $x \in V(A)$  and  $1 \leq i < j \leq k$ , if  $(A, \overline{U}, \overline{X}) \models \varphi_E^{i,j}(x)$  then there are exactly two sets  $P_i \subseteq U_i$  and  $P_j \subseteq U_j$  such that  $(A, \overline{U}, \overline{X}) \models \text{inc}_E^{i,j}(x, P_i, P_j)$  and if  $(A, \overline{U}, \overline{X}) \models x \sim^{i,j} y$  then  $(A, \overline{U}, \overline{X}) \models \text{inc}_E^{i,j}(y, P_i, P_j)$
- for all  $F \in \{B_1, \dots, B_t\}$ ,  $\varphi_F(A) \subseteq \varphi_E(A)$ .

We abbreviate MSO<sub>2</sub>–MSO<sub>2</sub>-transductions of  $\sigma$  in  $\tau$  as  $\sigma$ - $\tau$ -transductions. Let  $\Theta$  be a  $\sigma$ - $\tau$ -transduction. To every  $\tau$ -structure  $A$ ,  $\Theta$  associates a class  $\Theta(A)$  of  $\sigma$ -structures defined as follows. If  $U_1, \dots, U_k, X_1, \dots, X_l \subseteq E(A)$  are sets of edges such that  $(A, \overline{U}, \overline{X}) \models \varphi_U$ , then we define the structure  $B := \Theta(A, \overline{U}, \overline{X})$  as follows:

- $V(B) := \dot{\bigcup}_{1 \leq i \leq k} V_i$  where  $V_i := \{V \subseteq U_i : V \text{ is a connected component of } U_i\}$
- $E(B) := \dot{\bigcup}_{1 \leq i < j \leq k} E^{i,j}$  where  $E^{i,j} := \{[v]_{\sim^{i,j}} : v \in \varphi_E^{i,j}(A)\}$  and the  $E^{i,j}$  are taken to be disjoint.
- an edge  $e \in E^{i,j}$  is incident to vertices  $P \in V_i$  and  $Q \in V_j$  if  $A \models \text{inc}_E^{i,j}(e, P, Q)$  for some  $Q \in V_j$  and likewise for  $P \in V_j$ .
- an edge  $e \in E^{i,j}$ , for  $1 \leq i < j \leq k$ , is coloured by  $F$ , where  $F \in \sigma$  is binary, if  $A \models \varphi_F^{i,j}(e)$ .
- a vertex  $P \in V_i$  is coloured by  $C \in \sigma$ , where  $C$  is unary, if  $(A, \overline{U}, \overline{X}) \models \varphi_C^i(P)$ .

Hence, with every structure  $A$  and satisfying assignment  $U_1, \dots, U_k, \overline{X}$  of  $\varphi_U$  the transduction  $\Theta$  associates structures whose universes consist of the connected components

of the  $U_i$ . For classes  $\mathcal{A}$  of  $\tau$ -structures we define  $\Theta(\mathcal{A}) := \{B : B \in \Theta(A) \text{ for some } A \in \mathcal{A}\}$ . The definition of the edge relation may seem to be overly complicated, as we define the edges and their incidence by different formulas and furthermore do it separately for each pair  $i, j$ . The reason is that we want to use  $\text{MSO}_2$ -formulas over the structures  $\Theta(A)$  and hence have to be able to quantify over sets of edges in  $B \in \Theta(A)$ . As  $\text{MSO}_2$  does not allow quantification over arbitrary binary relations, we have to encode edges by individual elements of  $A$  and then use sets over vertices to encode sets of edges.

As all interpretations,  $\text{MSO}_2$ – $\text{MSO}_2$ -transductions define a way of transforming one class of structures into another and on the other hand, provide a translation of  $\text{MSO}_2$ -formulas  $\varphi$  over  $\sigma$ -structures into  $\text{MSO}_2$ -formulas  $\varphi^*$  over  $\tau$ -structures so that if  $\varphi$  is a formula with free variables  $F_1, \dots, F_l, X_1, \dots, X_s, y_1, \dots, y_r$ , where the  $F_i$ 's are binary, the  $X_i$ 's are unary and the  $y_i$ 's are individual variables, then  $\varphi^*$  is a formula with free variables  $(F_i)_1^*, \dots, (F_i)_k^*, (X_i)_1^*, \dots, (X_i)_k^*$ , and  $(Y_i)_1^*, \dots, (Y_i)_k^*$ , where the  $(F_i)_j^*$ 's are binary and all other unary. In addition, the parameters  $\overline{U}, \overline{X}$  of the transduction occur free in  $\varphi^*$ . We refer to the full version for details.

**Lemma 5.2** (interpretation lemma). *Let  $A$  be a  $\tau$ -structure and  $\overline{U}, \overline{X} \subseteq E(A)$  be such that  $(A, \overline{U}, \overline{X}) \models \varphi_U$ . Let  $B := \Theta(A, \overline{U}, \overline{X})$ . For all  $\varphi \in \text{MSO}_2[\sigma]$ ,  $(A, \overline{U}, \overline{X}) \models \Theta(\varphi)$  if, and only if,  $B \models \varphi$ .*

**Corollary 5.3.** *Let  $\varphi \in \text{MSO}_2[\sigma]$  and  $\psi := \exists \overline{U} \exists \overline{X} \varphi^* \in \text{MSO}_2[\tau]$ . For all  $\tau$ -structures  $A$ ,  $A \models \psi$  iff there is a  $B \in \Theta(A)$  such that  $B \models \varphi$ .*

## 6 Putting It All Together

In this section we prove Theorem 1.2. Let  $\Sigma := \{C_1, \dots, C_l\}$ , with  $l \geq 2$ , be a set of colours and  $\Gamma := \Sigma \cup \{B, R\}$ , where  $B$  is a binary and  $R$  a unary relation symbol. Let  $\mathcal{C}$  be a constructible class of  $\Gamma$ -coloured graphs closed under colourings such that the tree-width of  $\mathcal{C}$  is strongly unbounded by  $\log^{8k} n$ , for some  $k \geq 1$ . We first observe that the formulas  $\varphi_U(\mathcal{P}, \mathcal{Q}, A, L, T)$ ,  $\varphi_E, \text{inc}, \sim, \varphi_B, \varphi_C$  as constructed in Section 4 can be used to define an  $\text{MSO}_2$ – $\text{MSO}_2$ -transduction  $\Theta$  such that  $\Theta(\mathcal{C})$  is the class of coloured walls in graphs  $G \in \mathcal{C}$ . Here, we take  $\overline{U} := \mathcal{P}, \mathcal{Q}$  as the parameters defining the vertex set of the resulting graphs and  $\overline{X} := A, L, T$  as additional parameters used in the transduction.

By Lemma 3.7,  $\Theta(\mathcal{C})$  contains for each  $w \in \Sigma^*$  a wall encoding  $w$  with power  $k$ , i.e. there is a  $(|w|^k \times |w|^k)$ -wall in  $\Theta(\mathcal{C})$  whose top-most row is labelled by  $w$  from the left. In particular, as SAT can be solved in time quadratic in the size of the input by a non-deterministic Turing-machine, if  $k \geq 2$  then for each CNF formula  $w$  of length  $m$ ,  $\Theta(\mathcal{C})$  contains a wall of size  $m^2 \times m^2$  labelled by  $w$ .

Now take a formula  $\varphi_{\text{CNF}}$  which, on a wall  $W$  encoding  $w$ , checks whether  $w$  correctly encodes a CNF-formula and whether the order of  $W$  is at least  $|w|^2$ . This can be done by simulating a non-deterministic Turing machine doing this test. Let  $\psi_{\text{CNF}} := \exists \mathcal{P} \mathcal{Q} \text{ALT}(\varphi_U \wedge \Theta(\varphi_{\text{CNF}}))$  and let  $\mathcal{C}_{\text{CNF}} := \{A \in \mathcal{C} : A \models \psi_{\text{CNF}}\} \subseteq \mathcal{C}$ . By the interpretation Lemma 5.2,  $\mathcal{C}_{\text{CNF}}$  contains for each CNF-formula  $w$  a graph  $G \in \mathcal{C}$  encoding  $w$  with power 2 and conversely each graph  $G \in \mathcal{C}_{\text{CNF}}$  encodes a CNF-formula with power 2.



Now, let  $\varphi$  be the  $\text{MSO}_2$ -sentence from Section 2 which, by simulating an appropriate Turing-machine, is true in a wall of order  $|w|^2$  encoding a CNF-formula  $w$  if, and only if,  $w$  is satisfiable and let  $\vartheta := \Theta(\varphi)$ . It follows that if the tree-width of  $\mathcal{C}$  is strongly unbounded by  $\log^{16} n$ , then model-checking  $\vartheta := \Theta(\varphi)$  in  $\mathcal{C}_{\text{CNF}}$  is equivalent to solving SAT. If in addition  $\mathcal{C}$  is constructible then this allows us to formally define a subexponential time reduction from SAT to  $\mathcal{C}$  as follows. Given a CNF-formula  $w$ , we construct a graph  $G \in \mathcal{C}$  such that  $G$  encodes  $w$  with power 2 and  $|G| < 2^{c \cdot |w|^{\frac{1}{y}}}$ , for some  $y > 1$  and  $c > 0$ . By definition of constructibility and strongly unboundedness, such a graph  $G$  exists in  $\mathcal{C}$  and can be constructed in time  $|G|^r$ , for some fixed  $r > 0$ , and hence in time  $< (2^{|w|^{\frac{1}{y}}})^r = 2^{r \cdot |w|^{\frac{1}{y}}}$ . Now suppose  $\text{MC}(\text{MSO}_2, \mathcal{C})$  was in XP, i.e. for some computable function  $f$ , given a graph  $G \in \mathcal{C}$  and  $\varphi \in \text{MSO}_2$ ,  $G \models \varphi$  could be decided in time  $|G|^{f(|\varphi|)}$ . Hence, we could decide if  $G \models \vartheta$ , where  $\vartheta$  is the formula defined above, in time  $|G|^{f(|\vartheta|)} < 2^{f(|\vartheta|) \cdot |w|^{\frac{1}{y}}}$ . Taken together, we could decide if  $w$  is satisfiable in time  $< 2^{(r+f(|\vartheta|)) \cdot |w|^{\frac{1}{y}}}$ , for fixed  $r, y > 1$  and a fixed formula  $\vartheta$ . Hence, SAT would be decidable in sub-exponential time.

The same argument shows that if  $\mathcal{C}$  is a rich and constructible class of  $\Gamma$ -coloured graphs closed under colourings whose tree-width is effectively not bounded by  $\log^{8 \cdot k} n$  and  $\mathcal{L}$  is a problem that can be decided by a non-deterministic Turing-machine in time  $n^k$ , then  $\text{MC}(\text{MSO}_2, \mathcal{C})$  is not in XP unless  $\mathcal{L}$  can be solved in sub-exponential time. This implies Theorem 1.2. The extension to the polynomial time hierarchy follows as we can simulate alternating Turing-machines with bounded number of alternations in  $\text{MSO}_2$  in the same way as non-deterministic Turing-machines.

**Acknowledgements.** I would like to thank Mark Weyer for pointing out that the result proved here readily extends to problems in the polynomial time hierarchy.

## References

1. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. 2, pp. 194–242. Elsevier, Amsterdam (1990)
2. Dawar, A., Grohe, M., Kreutzer, S.: Locally excluding a minor. In: Logic in Computer Science (LICS), pp. 270–279 (2007)
3. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)
4. Ebbinghaus, H.-D., Flum, J., Thomas, W.: Mathematical Logic. Springer, Heidelberg (1994)
5. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
6. Flum, J., Grohe, M.: Fixed-parameter tractability, definability, and model checking. SIAM Journal on Computing 31, 113–145 (2001)
7. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. Journal of the ACM 48, 1148–1206 (2001)
8. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete problems. In: ACM Symposium on Theory of Computing (STOC), pp. 47–63 (1974)
9. Grohe, M.: Logic, graphs, and algorithms. In: Wilke, T., Flum, J., Grädel, E. (eds.) Logic and Automata – History and Perspectives. Amsterdam University Press (2007)
10. Kreutzer, S.: Algorithmic meta-theorems (to appear), <http://arxiv.org/abs/0902.3616>

11. Libkin, L.: *Elements of Finite Model Theory*. Springer, Heidelberg (2004)
12. Reed, B., Wood, D.: Polynomial treewidth forces a large grid-like minor. arXiv:0809.0724v3 [math.CO] (2008) (unpublished)
13. Robertson, N., Seymour, P., Thomas, R.: Quickly excluding a planar graph. *Journal of Combinatorial Theory, Series B* (1994)
14. Robertson, N., Seymour, P.D.: Graph minors V. Excluding a planar graph. *Journal of Combinatorial Theory, Series B* 41(1), 92–114 (1986)
15. Vardi, M.: On the complexity of relational query languages. In: *Proc. of the 14th Symposium on Theory of Computing (STOC)*, pp. 137–146 (1982)

# Automatic Structures of Bounded Degree Revisited

Dietrich Kuske and Markus Lohrey\*

Universität Leipzig, Institut für Informatik, Germany

**Abstract.** It is shown that the first-order theory of an automatic structure, whose Gaifman graph has bounded degree, is decidable in doubly exponential space (for injective automatic presentations, this holds even uniformly). Presenting an automatic structure of bounded degree whose theory is hard for  $2\text{EXPSPACE}$ , we also prove this result to be optimal. These findings close the gap left open in [14].

## 1 Introduction

The idea of an automatic structure goes back to Büchi and Elgot who used finite automata to decide, e.g., Presburger arithmetic [5]. Automaton decidable theories [9] and automatic groups [6] are similar concepts. A systematic study was initiated by Khoussainov and Nerode [10] who also coined the name “*automatic structure*”. In essence, a structure is automatic if the elements of the universe can be represented as strings from a regular language (an element can be represented by several strings) and every relation of the structure can be recognized by a finite automaton with several heads that proceed synchronously. Automatic structures received increasing interest over the last years [3, 11, 12, 15, 1]. One of the main motivations for investigating automatic structures is that their (first-order) theories can be decided uniformly (i.e., the input is an automatic presentation and a first-order sentence). But even the theory of a specific automatic structure might be far from efficient: There exist automatic structures with a nonelementary theory. This motivates the search for subclasses of automatic structures with elementary theory. The first such class was identified by the second author in [14] who showed that the theory of every automatic structure of *bounded degree* can be decided in triply exponential alternating time with linearly many alternations. A structure has bounded degree, if in its Gaifman graph, the number of neighbors of a node is bounded by some fixed constant. The paper [14] also presents a specific automatic structure of bounded degree whose theory is hard for doubly exponential alternating time with linearly many alternations. Hence, an exponential gap between the upper and lower bound remained. An upper bound of 4-fold exponential alternating time with linearly many alternations was shown for *tree automatic structures* (which are defined analogously to automatic structures using tree automata) of bounded degree.

---

\* The second author is supported by the DFG research project GELO.

Our paper [12] proves a triply exponential space bound for the theory of an injective  $\omega$ -automatic structure (that is defined via Büchi-automata) of bounded degree; this result was recently applied to one-dimensional cellular automata [7]. Here, injectivity means that every element of the structure is represented by a *unique*  $\omega$ -word from the underlying regular language.

In this paper, we achieve three goals: (i) We close the complexity gaps from [14] for automatic structures of bounded degree. (ii) We investigate, for the first time, the complexity of the *uniform* theory (where the automatic presentation is part of the input) of automatic structures of bounded degree. (iii) We refine our complexity analysis using the growth function of a structure. This function measures the size of a sphere in the Gaifman graph depending on the radius of the sphere. The growth function of a structure of bounded degree can be at most exponential.

Our main results are the following:

(a) The uniform theory for injective automatic presentations is 2EXPSPACE-complete. The lower bound already holds in the non-uniform setting, i.e. there exists an automatic structure of bounded degree with a 2EXPSPACE-complete theory.

(b) For every automatic structure of bounded degree, where the growth function is polynomially bounded, the theory is in EXPSPACE, and there exists an example with an EXPSPACE-complete theory.

In addition, the full version [13] of this extended abstract also contains analogous results for tree-automatic structures that had to be left out for space restrictions:

(c) The uniform theory for injective tree automatic presentations belongs to 4EXPTIME; the non-uniform one to 3EXPTIME for arbitrary tree automatic structures, and to 2EXPTIME if the growth function is polynomial. Our bounds for the non-uniform problem are sharp, i.e., there are tree automatic structures of bounded degree (and polynomial growth) with a 3EXPTIME-complete (2EXPTIME-complete, resp.) first-order theory.

We conclude this paper with some results on the complexity of first-order fragments with fixed quantifier alternation depth one or two on automatic structures of bounded degree. For a full version of this paper see [13].

## 2 Preliminaries

Let  $\Gamma$  be a finite alphabet and  $w \in \Gamma^*$  be a finite word over  $\Gamma$ . The length of  $w$  is denoted by  $|w|$ . We also write  $\Gamma^n = \{w \in \Gamma^* \mid n = |w|\}$ .

Let  $\exp(0, x) = x$  and  $\exp(n + 1, x) = 2^{\exp(n, x)}$  for  $x \in \mathbb{N}$ . We assume basic knowledge in complexity theory. For  $k \geq 1$ , we denote with  $k$ EXPSPACE (resp.  $k$ EXPTIME) the class of all problems that can be accepted in space (resp. time)  $\exp(k, n^{O(1)})$  on a deterministic Turing machine. For 1EXPSPACE we write just EXPSPACE, EXPTIME is to be understood similarly. A problem is called *elementary* if it belongs to  $k$ EXPTIME for some  $k \in \mathbb{N}$ .

Recall that emptiness and inclusion of the languages of finite nondeterministic automata are complete for NL (nondeterministic logspace) and PSPACE (polynomial space), resp..

### 2.1 Structures and First-Order Logic

A *signature* is a finite set  $\mathcal{S}$  of relational symbols, where every symbol  $r \in \mathcal{S}$  has some fixed arity  $m_r$ . The notion of an  $\mathcal{S}$ -structure (or model) is defined as usual in logic. We only consider relational structures. Sometimes, we will also use constants, but in our context, a constant  $c$  can be replaced by the unary relation  $\{c\}$ . Let us fix an  $\mathcal{S}$ -structure  $\mathcal{A} = (A, (r^{\mathcal{A}})_{r \in \mathcal{S}})$ , where  $r^{\mathcal{A}} \subseteq A^{m_r}$ . To simplify notation, we will write  $a \in \mathcal{A}$  for  $a \in A$ . For  $B \subseteq A$  we define the restriction  $\mathcal{A} \upharpoonright B = (B, (r^{\mathcal{A}} \cap B^{m_r})_{r \in \mathcal{S}})$ . Given further constants  $a_1, \dots, a_k \in \mathcal{A}$ , we write  $(\mathcal{A}, a_1, \dots, a_k)$  for the structure  $(A, (r^{\mathcal{A}})_{r \in \mathcal{S}}, a_1, \dots, a_k)$ . In the rest of the paper, we will always identify a symbol  $r \in \mathcal{S}$  with its interpretation  $r^{\mathcal{A}}$ . A *congruence* on the structure  $\mathcal{A} = (A, (r)_{r \in \mathcal{S}})$  is an equivalence relation  $\equiv$  on  $A$  such that for every  $r \in \mathcal{S}$  and all  $a_1, b_1, \dots, a_{m_r}, b_{m_r} \in A$  we have: If  $(a_1, \dots, a_{m_r}) \in r$  and  $a_1 \equiv b_1, \dots, a_{m_r} \equiv b_{m_r}$ , then also  $(b_1, \dots, b_{m_r}) \in r$ . As usual, the equivalence class of  $a \in A$  w.r.t.  $\equiv$  is denoted by  $[a]_{\equiv}$  or just  $[a]$  and  $A/\equiv$  denotes the set of all equivalence classes. We define the *quotient structure*  $\mathcal{A}/\equiv = (A/\equiv, (r/\equiv)_{r \in \mathcal{S}})$ , where  $r/\equiv = \{([a_1], \dots, [a_{m_r}]) \mid (a_1, \dots, a_{m_r}) \in r\}$ .

The *Gaifman-graph*  $G(\mathcal{A})$  of the  $\mathcal{S}$ -structure  $\mathcal{A}$  is the symmetric graph on the universe  $A$  of  $\mathcal{A}$ , which contains an edge between  $a$  and  $b$  if and only if there exists a tuple  $(a_1, \dots, a_{m_r}) \in r$  in some of the relations  $r \in \mathcal{S}$  such that  $a$  and  $b$  both belong to  $\{a_1, \dots, a_{m_r}\}$ . With  $d_{\mathcal{A}}(a, b)$ , where  $a, b \in \mathcal{A}$ , we denote the distance between  $a$  and  $b$  in  $G(\mathcal{A})$ , i.e., it is the length of a shortest path connecting  $a$  and  $b$  in  $G(\mathcal{A})$ . For  $a \in \mathcal{A}$  and  $d \geq 0$  we denote with  $S_{\mathcal{A}}(d, a) = \{b \in A \mid d_{\mathcal{A}}(a, b) \leq d\}$  the  $d$ -sphere around  $a$ . If  $\mathcal{A}$  is clear from the context, then we will omit the subscript  $\mathcal{A}$ . We say that the structure  $\mathcal{A}$  is *locally finite* if its Gaifman graph  $G(\mathcal{A})$  is locally finite (i.e., every node has finitely many neighbors). Similarly, the structure  $\mathcal{A}$  has *bounded degree*, if  $G(\mathcal{A})$  has bounded degree, i.e., there exists a constant  $\delta$  such that every  $a \in A$  is adjacent to at most  $\delta$  many other nodes in  $G(\mathcal{A})$ ; the minimal such  $\delta$  is called the *degree of  $\mathcal{A}$* . For a structure  $\mathcal{A}$  of bounded degree we define its *growth function*  $g_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$  as  $g_{\mathcal{A}}(n) = \max\{|S_{\mathcal{A}}(n, a)| \mid a \in \mathcal{A}\}$ . Note that if the function  $g_{\mathcal{A}}$  is not bounded then  $g_{\mathcal{A}}(n) \geq n$  for all  $n \geq 1$ . For us, it is more convenient to not have a bounded function describing the growth. Therefore, we define the *normalized growth function*  $g'_{\mathcal{A}}$  by  $g'_{\mathcal{A}}(n) = \max\{n, g_{\mathcal{A}}(n)\}$ . Note that  $g_{\mathcal{A}}$  and  $g'_{\mathcal{A}}$  are different only in the case that all connected components of  $\mathcal{A}$  contain at most  $m$  elements (for some fixed  $m$ ). Clearly,  $g'_{\mathcal{A}}(n)$  can grow at most exponentially if  $\mathcal{A}$  has bounded degree. We say that  $\mathcal{A}$  has *exponential growth* if  $g'_{\mathcal{A}}(n) \in 2^{\Omega(n)}$ . If  $g'_{\mathcal{A}}(n) \in n^{O(1)}$ , then  $\mathcal{A}$  has *polynomial growth*.

We consider (first-order) formulas with equality over the signature  $\mathcal{S}$ . The *quantifier depth* of a formula  $\varphi$  is the maximal nesting of quantifiers in  $\varphi$ . A formula without free variables is called *closed*. The *theory* of  $\mathcal{A}$ , denoted by  $\text{Th}(\mathcal{A})$ , is the set of all closed formulas  $\varphi$  with  $\mathcal{A} \models \varphi$ .

## 2.2 Structures from Automata

**Automatic Structures.** Next we introduce automatic structures, more details can be found in [10,3]. Let us fix  $n \in \mathbb{N}$  and a finite alphabet  $\Gamma$ . Let  $\$ \notin \Gamma$  be an additional padding symbol. For words  $w_1, w_2, \dots, w_n \in \Gamma^*$  we define the *convolution*  $w_1 \otimes w_2 \otimes \dots \otimes w_n$ , which is a word over the alphabet  $(\Gamma \cup \{\$\})^n$ , as follows: Let  $w_i = a_{i,1}a_{i,2} \dots a_{i,k_i}$  with  $a_{i,j} \in \Gamma$  and  $k = \max\{k_1, \dots, k_n\}$ . For  $k_i < j \leq k$  define  $a_{i,j} = \$$ . Then  $w_1 \otimes \dots \otimes w_n = (a_{1,1}, \dots, a_{n,1}) \dots (a_{1,k}, \dots, a_{n,k})$ . Thus, for instance  $aba \otimes bbabb = (a, b)(b, b)(a, a)(\$, b)(\$, b)$ . An  $n$ -ary relation  $R \subseteq (\Gamma^*)^n$  is called *automatic* if the language  $\{w_1 \otimes \dots \otimes w_n \mid (w_1, \dots, w_n) \in R\}$  is a regular language.

An  $m$ -dimensional (synchronous) automaton over  $\Gamma$  is just a finite automaton  $A = (Q, \Delta, q_0, F)$  over  $(\Gamma \cup \{\$\})^m$  such that  $L(A) \subseteq \{w_1 \otimes \dots \otimes w_m \mid w_1, \dots, w_m \in \Gamma^*\}$ . Such an automaton defines an  $m$ -ary relation

$$R(A) = \{(w_1, \dots, w_m) \mid w_1 \otimes \dots \otimes w_m \in L(A)\} .$$

We define the size  $|A|$  of  $A$  as  $\max\{1, |\Delta|\} \cdot m$ . Reasonably assuming that every state is the target state of some transition and that every letter from  $\Gamma$  appears in some transition (implying  $|\Gamma|, |Q| \leq |\Delta|$ ), the size  $|A|$  bounds the number of bits needed to store  $A$  (up to some polynomial).

An *automatic presentation* is a tuple  $P = (\Gamma, \mathcal{S}, A_0, A_=(, (A_r)_{r \in \mathcal{S}})$ , where: (i)  $\Gamma$  is a finite alphabet, (ii)  $\mathcal{S}$  is the signature of  $P$  (as before  $m_r$  is the arity of the symbol  $r \in \mathcal{S}$ ), (iii)  $A_0$  is an automaton over the alphabet  $\Gamma$ , (iv) for every  $r \in \mathcal{S}$ ,  $A_r$  is an  $m_r$ -dimensional automaton over  $\Gamma$  with  $R(A_r) \subseteq L(A_0)^{m_r}$ , and (v)  $A_=($  is a 2-dimensional automaton over  $\Gamma$  such that  $R(A_=($  is a congruence on the structure  $(L(A_0), (R(A_r))_{r \in \mathcal{S}})$ . This presentation  $P$  is *injective* if  $R(A_=($  is the identity relation on  $L(A_0)$ . The structure presented by  $P$  is the quotient  $\mathcal{A}(P) = (L(A_0), (R(A_r))_{r \in \mathcal{S}}) / R(A_=($ . A structure  $\mathcal{A}$  is *automatic* if there exists an automatic presentation  $P$  such that  $\mathcal{A} \simeq \mathcal{A}(P)$ . We will write  $[u]$  for the element  $[u]_{R(A_=($  ( $u \in L(A_0)$ ) of the structure  $\mathcal{A}(P)$ . The presentation  $P$  has *bounded degree* if the structure  $\mathcal{A}(P)$  has bounded degree. The size of the presentation  $P = (\Gamma, \mathcal{S}, A_0, A_=(, (A_r)_{r \in \mathcal{S}})$  is  $|P| = |A_0| + |A_=(| + \sum_{r \in \mathcal{S}} |A_r|$ . Note that  $|\mathcal{S}| \leq |P|$  and  $m_r \leq |P|$  for all  $r \in \mathcal{S}$ .

Typical examples of automatic structures are  $(\mathbb{N}, +)$  and  $(\mathbb{Q}, \leq)$ . Examples of automatic structures of bounded degree are transition graphs of Turing machines and Cayley-graphs of automatic groups [6] as well as the queue structure [16] (the set of finite words together with functions prefixing and suffixing a word by a fixed letter). There are automatic structures  $\mathcal{A}$  of bounded degree with growth  $g_{\mathcal{A}}(n) \in 2^{\Theta(\sqrt{n})}$  [13].

We will consider the following classes of automatic presentations [1]:

- SA: the class of all automatic presentations.
- SAB: the class of all automatic presentations of bounded degree.
- iSAb: the class of all injective automatic presentations of bounded degree.

---

<sup>1</sup> The letter S in the below classes refers to “string”, the full paper [13] also contains classes starting with T that refers to “tree”.

**The Model Checking Problem.** For the above classes of automatic presentations, we will be interested in the following decision problems.

**Definition 2.1.** *Let  $\mathcal{C}$  be a class of automatic presentations. Then the model checking problem  $\text{MC}(\mathcal{C})$  for  $\mathcal{C}$  denotes the set of all pairs  $(P, \varphi)$  where  $P \in \mathcal{C}$ , and  $\varphi$  is a closed formula over the signature of  $P$  such that  $\mathcal{A}(P) \models \varphi$ .*

If  $\mathcal{C} = \{P\}$  is a singleton, then the model checking problem  $\text{MC}(\mathcal{C})$  for  $\mathcal{C}$  can be identified with the theory of the structure  $\mathcal{A}(P)$ . An algorithm deciding the model checking problem for a class  $\mathcal{C}$  decides the theories of each element of  $\mathcal{C}$  uniformly.

The following two results are the main motivations for investigating automatic structures.

**Proposition 2.2 (cf. [10]).** *There is an algorithm that computes, from an automatic presentation  $P = (\Gamma, \mathcal{S}, A_0, A_=(, (A_r)_{r \in \mathcal{S}})$  and a formula  $\varphi(x_1, \dots, x_m)$ , an  $m$ -dimensional automaton  $A$  over  $\Gamma$  with  $R(A) = \{(u_1, \dots, u_m) \in L(A_0)^m \mid \mathcal{A}(P) \models \varphi([u_1], \dots, [u_m])\}$ .*

The automaton is constructed by induction on the structure of the formula  $\varphi$ : disjunction corresponds to the disjoint union of automata, existential quantification to projection, and negation to complementation. The following result is a direct consequence.

**Theorem 2.3 (cf. [10]).** *The model checking problem  $\text{MC}(\text{SA})$  for all automatic presentations is decidable. In particular, the theory  $\text{Th}(\mathcal{A})$  of every automatic structure  $\mathcal{A}$  is decidable.*

Strictly speaking, [10] devises algorithms that, given an automatic presentation and a closed formula, decide whether the formula holds in the presented structure. But a priori, it is not clear whether it is decidable, whether a given tuple  $(\Gamma, \mathcal{S}, A_0, A_=(, (A_r)_{r \in \mathcal{S}})$  is an automatic presentation. Prop. 2.5(a) below shows that SA is indeed decidable in polynomial space, which then completes the proof of this theorem.

Thm. 2.3 holds even if we add quantifiers for “there are infinitely many  $x$  such that  $\varphi(x)$ ” [23] and “the number of elements satisfying  $\varphi(x)$  is divisible by  $k$ ” (for  $k \in \mathbb{N}$ ) [11]. This implies in particular that it is decidable whether an automatic presentation describes a locally finite structure. But the decidability of the theory is far from efficient, since there are automatic structures with a nonelementary first-order theory [3]. An example for this is the infinite binary tree with the prefix relation, see e.g. [4, Example 8.3]. A locally finite example can be obtained by taking the disjoint union of all finite binary-labeled linear orders, see e.g. [4].

**Preliminary Complexity Results.** It will be convenient to work with injective automatic presentations. The following lemma says that this is no restriction, if we allow an exponential jump in complexity.

**Lemma 2.4** ([10, Cor. 4.3]). *From  $P \in \text{SA}$  we can compute in time  $2^{O(|P|)}$  an injective automatic presentation  $P' \in \text{iSA}$  with  $\mathcal{A}(P) \simeq \mathcal{A}(P')$ .*

Next, we give complexity bounds for the class of all automatic structures as well as for those of bounded degree.

**Proposition 2.5.** *(a) The class SA is PSPACE-complete and (b) the class SAb belongs to EXPTIME.*

*Proof.* Statement (a) is shown in the full version [13]. For (b) we can assume by (a) that the input indeed belongs to SA (which can be checked in polynomial space and therefore in exponential time). In exponential time, the automatic presentation can then be transformed into an equivalent injective one  $P \in \text{iSA}$  of exponential size. Using simple automata constructions, we can compute a 2-dimensional automaton  $A$  for the edge relation of the Gaifman-graph of  $\mathcal{A}(P)$  (in fact,  $A$  can be computed in time polynomial in  $|P|$ ). Since  $P$  is injective (i.e. every equivalence class  $[u]$  is the singleton  $\{u\}$ ),  $\mathcal{A}(P)$  is of bounded degree iff  $A$  (seen as a transducer) is finite-valued. But this is decidable in time polynomial in  $|P|$  [17]. Since  $P$  is exponential in the input, this completes the proof.  $\square$

In contrast to this decidability results, it is undecidable, whether a given automatic structure of bounded degree has polynomial growth, see the complete version [13].

Finally, since we deal with structures of bounded degree, it will be important to estimate the degree of such a structure given its presentation. Such estimates are provided by the following result.

**Proposition 2.6.** *The following holds:*

- (a) *If  $P \in \text{iSAb}$ , then the degree of  $\mathcal{A}(P)$  is bounded by  $\exp(1, |P|^{O(1)})$ .*
- (b) *If  $P \in \text{SAb}$ , then the degree of  $\mathcal{A}(P)$  is bounded by  $\exp(2, |P|^{O(1)})$ .*

*Proof.* For statement (a) let  $P \in \text{iSAb}$ . From  $P$  we can construct a 2-dimensional automaton  $A$  of size  $|P|^{O(1)}$  that accepts the edge relation of the Gaifman graph of  $\mathcal{A}(P)$ . Then the degree of  $\mathcal{A}(P)$  equals the maximal out-degree of the relation  $R(A)$ . For string transducers, this number is exponential in the size of  $A$ , i.e., it is in  $\exp(1, |P|^{O(1)})$  [17].

For  $P \in \text{SAb}$ , the bound  $\exp(2, |P|^{O(1)})$  follows immediately from Lemma 2.4 and (a).  $\square$

The bound in Prop. 2.6 for  $P \in \text{iSAb}$  is sharp, see the complete version [13] for an example.

### 3 Upper Bounds

It is the aim of this section to give an algorithm that decides the theory of an automatic structure of bounded degree. The algorithm from Thm. 2.3 (that in particular solves this problem) is based on Prop. 2.2, i.e., the inductive construction of an automaton accepting all satisfying assignments. Differently, we base our algorithm on Gaifman’s Thm. 3.1, i.e., on the combinatorics of spheres. We therefore start with some model theory.



### 3.1 Model-Theoretic Background

For a structure  $\mathcal{A}$ ,  $\bar{a} = (a_1, \dots, a_k) \in \mathcal{A}^k$  and  $d \geq k \geq 0$ , we denote with  $\mathcal{A}[d, \bar{a}]$  the induced substructure  $\mathcal{A} \upharpoonright \bigcup_{i=1}^k S(7^{d-i}, a_i)$ . The following locality principle of Gaifman implies that super-exponential distances cannot be handled in first-order logic:

**Theorem 3.1** ([S]). *Let  $\mathcal{A}$  be a structure,  $\bar{a}, \bar{b} \in \mathcal{A}^k$  and  $d \geq 0$  such that  $(\mathcal{A}[d+k, \bar{a}], \bar{a}) \simeq (\mathcal{A}[d+k, \bar{b}], \bar{b})$  (i.e. there is an isomorphism between the two induced substructures  $\mathcal{A}[d+k, \bar{a}]$  and  $\mathcal{A}[d+k, \bar{b}]$  that maps the  $i^{\text{th}}$  component of  $\bar{a}$  to the  $i^{\text{th}}$  component of  $\bar{b}$  for all  $1 \leq i \leq k$ ). Then, for every formula  $\varphi(x_1, \dots, x_k)$  of quantifier depth at most  $d$ , we have:  $\mathcal{A} \models \varphi(\bar{a}) \iff \mathcal{A} \models \varphi(\bar{b})$ .*

Let  $\mathcal{S}$  be a signature and let  $k, d \in \mathbb{N}$  with  $0 \leq k \leq d$ . A *potential  $(d, k)$ -sphere* is a tuple  $(\mathcal{B}, \bar{b})$  such that  $\mathcal{B}$  is an  $\mathcal{S}$ -structure,  $\bar{b} \in \mathcal{B}^k$ , and  $\mathcal{B} = \mathcal{B}[d, \bar{b}]$ . There is only one potential  $(d, 0)$ -sphere namely the empty sphere  $\emptyset$ . For our later applications,  $\mathcal{B}$  will be always a finite structure, but in this subsection finiteness is not needed. The potential  $(d, k)$ -sphere  $(\mathcal{B}, \bar{b})$  is *realized in the structure  $\mathcal{A}$*  if there exists  $\bar{a} \in \mathcal{A}^k$  such that  $(\mathcal{A}[d, \bar{a}], \bar{a}) \simeq (\mathcal{B}, \bar{b})$ .

Let  $\sigma = (\mathcal{B}, \bar{b})$  be a potential  $(d, k)$ -sphere and let  $\sigma' = (\mathcal{C}, \bar{c}, c)$  be a potential  $(d, k+1)$ -sphere ( $k+1 \leq d$ ,  $\bar{c} \in \mathcal{C}^k$ ,  $c \in \mathcal{C}$ ). Then  $\sigma'$  *extends*  $\sigma$  (abbreviated  $\sigma \preceq \sigma'$ ) if  $\sigma \simeq (\mathcal{C}[d, \bar{c}], \bar{c})$ . The following definition is the basis for our decision procedure.

**Definition 3.2.** *Let  $\mathcal{A}$  be an  $\mathcal{S}$ -structure,  $\psi(y_1, \dots, y_k)$  a formula of quantifier depth at most  $d$ , and let  $\sigma = (\mathcal{B}, \bar{b})$  be a potential  $(d+k, k)$ -sphere. The Boolean value  $\psi_\sigma \in \{0, 1\}$  is defined inductively as follows:*

- If  $\psi(y_1, \dots, y_k)$  is an atomic formula, then

$$\psi_\sigma = 1 \iff \mathcal{B} \models \psi(\bar{b}). \tag{1}$$

- $(\neg\theta)_\sigma = 1 - \theta_\sigma$  and  $(\alpha \vee \beta)_\sigma = \max\{\alpha_\sigma, \beta_\sigma\}$
- If  $\psi(y_1, \dots, y_k) = \exists y_{k+1} \theta(y_1, \dots, y_k, y_{k+1})$  then

$$\psi_\sigma = \max\{\theta_{\sigma'} \mid \sigma' \text{ is a potential } (d+k, k+1)\text{-sphere with } \sigma \preceq \sigma' \text{ that is realized in } \mathcal{A}\}. \tag{2}$$

The following result ensures for every closed formula  $\psi$  that  $\psi_\emptyset = 1$  if and only if  $\mathcal{A} \models \psi$ . Hence the above definition can possibly be used to decide validity of the formula  $\varphi$  in the structure  $\mathcal{A}$ .

**Proposition 3.3.** *Let  $\mathcal{S}$  be a signature,  $\mathcal{A}$  an  $\mathcal{S}$ -structure,  $\bar{a} \in \mathcal{A}^k$ ,  $\psi(y_1, \dots, y_k)$  a formula of quantifier depth at most  $d$ , and  $\sigma = (\mathcal{B}, \bar{b})$  a potential  $(d+k, k)$ -sphere with*

$$(\mathcal{A}[d+k, \bar{a}], \bar{a}) \simeq \sigma. \tag{3}$$

*Then  $\mathcal{A} \models \psi(\bar{a}) \iff \psi_\sigma = 1$ .*

*Proof.* We prove the lemma by induction on the structure of  $\psi$ . First assume that  $\psi$  is atomic, i.e.  $d = 0$ . We have

$$\psi_\sigma = 1 \stackrel{(1)}{\iff} \mathcal{B} \models \psi(\bar{b}) \stackrel{(3)}{\iff} \mathcal{A}[0 + k, \bar{a}] \models \psi(\bar{a}) \iff \mathcal{A} \models \psi(\bar{a}),$$

where the last equivalence holds since  $\psi$  is atomic. The cases  $\psi = \neg\theta$  and  $\psi = \alpha \vee \beta$  are straightforward.

We finally consider the case  $\psi(y_1, \dots, y_k) = \exists y_{k+1} \theta(y_1, \dots, y_k, y_{k+1})$ . First assume that  $\psi_\sigma = 1$ . By (2), some potential  $(d + k, k + 1)$ -sphere  $\sigma'$  is realized in  $\mathcal{A}$  with  $\sigma \preceq \sigma'$  and  $\theta_{\sigma'} = 1$ . Since  $\sigma'$  is realized, there exist  $\bar{a}' \in \mathcal{A}^k, a' \in \mathcal{A}$  with

$$(\mathcal{A}[d + k, \bar{a}', a'], \bar{a}', a') \simeq (\mathcal{B}', \bar{b}, b) = \sigma'. \tag{4}$$

By induction, we have  $\mathcal{A} \models \theta(\bar{a}', a')$  and therefore  $\mathcal{A} \models \psi(\bar{a}')$ . From (4),  $\sigma \preceq \sigma'$ , and (3), we also obtain

$$(\mathcal{A}[d + k, \bar{a}', \bar{a}']) \simeq (\mathcal{A}[d + k, \bar{a}], \bar{a})$$

and therefore by Gaifman's Thm. 3.1  $\mathcal{A} \models \psi(\bar{a})$ .

Conversely, let  $a \in \mathcal{A}$  with  $\mathcal{A} \models \theta(\bar{a}, a)$ . Let  $\sigma' = (\mathcal{B}', \bar{b}, b)$  be the unique (up to isomorphism) potential  $(d + k, k + 1)$ -sphere such that

$$(\mathcal{A}[d + k, \bar{a}, a], \bar{a}, a) \simeq (\mathcal{B}', \bar{b}, b). \tag{5}$$

Then (3) implies  $\sigma \preceq \sigma'$ . Moreover, by (5),  $\sigma'$  is realized in  $\mathcal{A}$ , and  $\mathcal{A} \models \theta(\bar{a}, a)$  implies by induction  $\theta_{\sigma'} = 1$ . Hence, by (2), we get  $\psi_\sigma = 1$  which finishes the proof.  $\square$

### 3.2 The Decision Procedure

Now suppose we want to decide whether the closed formula  $\varphi$  holds in an automatic structure  $\mathcal{A}$  of *bounded degree*. By Prop. 3.3 it suffices to compute the Boolean value  $\varphi_\emptyset$ . This computation will follow the inductive definition of  $\varphi_\sigma$  from Def. 3.2. Since every  $(d, k)$ -sphere that is realized in  $\mathcal{A}$  is finite, we only have to deal with finite spheres. The crucial part of our algorithm is to determine whether a finite potential  $(d, k)$ -sphere is realized in  $\mathcal{A}$ . In the following, for a finite potential  $(d, k)$ -sphere  $\sigma = (\mathcal{B}, b_1, \dots, b_k)$ , we denote with  $|\sigma|$  the number of elements and with  $\delta(\sigma)$  the degree of the finite structure  $\mathcal{B}$ .

For a class of automatic presentations  $\mathcal{C}$  the *realizability problem*  $\text{REAL}(\mathcal{C})$  for  $\mathcal{C}$  denotes the set of all pairs  $(P, \sigma)$  where  $P \in \mathcal{C}$  and  $\sigma$  is a *finite* potential  $(d, k)$ -sphere over the signature of  $P$  for some  $0 \leq k \leq d$  such that  $\sigma$  can be realized in  $\mathcal{A}(P)$ . In the complexity estimates in the following lemma,  $\sigma$  is the input potential  $(d, k)$ -sphere and  $P$  is the input automatic presentation.

**Lemma 3.4.**  $\text{REAL}(\text{iSA})$  can be solved in space  $|\sigma|^{O(|P|)} \cdot 2^{O(\delta(\sigma))}$ .

*Proof.* Let  $P = (\Gamma, \mathcal{S}, A_0, A_=, (A_r)_{r \in \mathcal{S}}) \in \text{iSA}$ . Let  $\sigma = (\mathcal{B}, b_1, \dots, b_k)$  and let  $c_1, \dots, c_{|\sigma|}$  be a list of all elements of  $\mathcal{B}$ ; every  $b_i$  occurs in this list. Let  $E_{\mathcal{A}(P)}$  (resp.  $E_{\mathcal{B}}$ ) be the edge relation of the Gaifman graph  $G(\mathcal{A}(P))$  (resp.  $G(\mathcal{B})$ ). Then  $\sigma$  is realized in  $\mathcal{A}(P)$  iff there are  $u_1, \dots, u_{|\sigma|} \in \Gamma^*$  with

- (a)  $u_i \in L(A_0)$  for all  $1 \leq i \leq |\sigma|$ ,
- (b)  $u_i \neq u_j$  for all  $1 \leq i < j \leq |\sigma|$ ,
- (c) For all  $r \in \mathcal{S}$ :  $(u_{i_1}, \dots, u_{i_{m_r}}) \in R(A_r)$  if and only if  $(c_{i_1}, \dots, c_{i_{m_r}}) \in r^{\mathcal{B}}$ , and
- (d) there is no  $v \in L(A_0)$  such that, for some  $1 \leq j \leq |\sigma|$  and  $1 \leq i \leq k$  with  $d(c_j, b_i) < 2^{d-i}$ , we have:  $(u_j, v) \in E_{\mathcal{A}(P)}$  and  $v \notin \{u_p \mid (c_j, c_p) \in E_{\mathcal{B}}\}$ .

Then (a-c) express that the mapping  $c_i \mapsto u_i$  is well-defined and an embedding of  $\mathcal{B}$  into  $\mathcal{A}(P)$ . In (d),  $(u_j, v) \in E_{\mathcal{A}(P)}$  implies that  $v$  belongs to  $\bigcup_{1 \leq i \leq k} S(2^{d-i}, u_i)$ . Hence (d) expresses that all elements of  $\bigcup_{1 \leq i \leq k} S(2^{d-i}, u_i)$  belong to the image of this embedding.

Using standard automata constructions for Boolean operations and projection, we can construct a  $|\sigma|$ -dimensional automaton  $A$  over the alphabet  $\Gamma$  that checks (a-d). A detailed size estimate shows that  $A$  has at most  $\exp(1, |\sigma|^{O(|P|)} \cdot 2^{O(\delta(\sigma))})$  many states. Hence checking emptiness of its language (and therefore realizability of  $\sigma$  in  $\mathcal{A}(P)$ ) can be done in space logarithmic to the number of states, i.e., in space  $|\sigma|^{O(|P|)} \cdot 2^{O(\delta(\sigma))}$  which proves the statement.  $\square$

In the following, for an automatic presentation  $P$  of bounded degree,  $g'_P$  denotes the normalized growth function  $g'_{\mathcal{A}(P)}$  of the structure  $\mathcal{A}(P)$ . In the complexity estimates in the following theorem,  $\varphi$  is the input sentence and  $P$  is the input automatic presentation.

**Theorem 3.5.** *MC(iSAb) can be solved in space  $g'_P(2^{|\varphi|})^{O(|P|)} \exp(2, |P|^{O(1)})$ .*

*Proof.* It suffices by Prop. 3.3 to compute the Boolean value  $\varphi_\emptyset$ . Recall the inductive definition of  $\varphi_\sigma$  from Def. 3.2 that we now translated into an algorithm for computing  $\varphi_\emptyset$ . Such an algorithm has to handle potential  $(d, k)$ -spheres for  $1 \leq k \leq d \leq |\varphi|$  ( $d$  is the quantifier rank of  $\varphi$ ) that are realized in  $\mathcal{A}(P)$ . The number of nodes of a potential  $(d, k)$ -sphere realized in  $\mathcal{A}(P)$  is bounded by  $k \cdot g'_P(2^d) \leq g'_P(2^{|\varphi|})^{O(1)}$  since  $k < 2^{|\varphi|} \leq g'_P(2^{|\varphi|})$ . The number of relations of  $\mathcal{A}(P)$  as well as each arity is bounded by  $|P|$ . Hence, any potential  $(d, k)$ -sphere can be stored in space  $|P| \cdot g'_P(2^{|\varphi|})^{O(|P|)} = g'_P(2^{|\varphi|})^{O(|P|)}$ .

The set of  $(d, k)$ -spheres with  $0 \leq k \leq d$  (ordered by the extension relation  $\preceq$ ) forms a tree of depth  $d+1$ . The algorithm visits the nodes of this tree in a depth-first manner and descends when unraveling an existential quantifier. Hence, we have to store  $d + 1 \leq |\varphi|$  many spheres, for which space  $|\varphi| \cdot g'_P(2^{|\varphi|})^{O(|P|)} = g'_P(2^{|\varphi|})^{O(|P|)}$  is sufficient.

Moreover, during the unraveling of a quantifier, the algorithm has to check realizability of a potential  $(d, k)$ -sphere for  $1 \leq k \leq d$ . Any such sphere has at most  $g'_P(2^{|\varphi|})^{O(1)}$  many elements and the degree  $\delta$  of  $\mathcal{A}$  is bounded by  $\exp(1, |P|^{O(1)})$  by Prop. 2.6(a). Hence, by Lemma 3.4, realizability can be checked in space  $g'_P(2^{|\varphi|})^{O(|P|)} \cdot \exp(2, |P|^{O(1)})$ .

At the end, we have to check whether a tuple  $\bar{b}$  satisfies an atomic formula  $\psi(\bar{y})$ , which is trivial. Thus, the totally needed space is at most  $g'_P(2^{|\varphi|})^{O(|P|)} \cdot \exp(2, |P|^{O(1)})$ .  $\square$

We derive a number of consequences on the combined and expression complexity of automatic structures of bounded degree. The first one concerns the combined complexity and is a direct consequence of Thm. 3.5:

**Corollary 3.6.** *The following holds:*

- (a)  $\text{MC}(\text{iSAb})$  is in  $2\text{EXPSPACE}$ .
- (b)  $\text{MC}(\text{SAb})$  is in  $3\text{EXPSPACE}$ .

*Proof.* Statement (a) follows from Thm. 3.5 and the fact that (i)  $g'_{\mathcal{A}}(2^{|\varphi|}) \leq \delta^{2^{|\varphi|}}$  if  $\delta$  is the degree of  $\mathcal{A}(P)$  and (ii) Prop. 2.6(a), which allows to bound  $\delta$  by  $2^{|P|^{O(1)}}$ . Statement (b) follows from (a) and Lemma 2.4, which allows to make an automatic presentation injective with an exponential blow up.  $\square$

Next we concentrate on the expression complexity, i.e., we fix the structure.

**Corollary 3.7.** *If  $\mathcal{A}$  is an automatic structure of bounded degree, then  $\text{Th}(\mathcal{A})$  belongs to  $2\text{EXPSPACE}$ . If in addition,  $\mathcal{A}$  has also polynomial growth, then  $\text{Th}(\mathcal{A})$  belongs to  $\text{EXPSPACE}$ .*

*Proof.* Since  $\mathcal{A}$  is automatic, it has a fixed injective automatic presentation  $P$ , i.e.,  $|P|$  is a fixed constant. Hence, the first statement follows immediately from Thm. 3.5. If  $\mathcal{A}$  has in addition polynomial growth, then, again, the claim follows immediately from Thm. 3.5 since  $g'_{\mathcal{A}}(2^{|\varphi|})^{O(|P|)} = 2^{O(|\varphi|)}$ .  $\square$

## 4 Lower Bounds

In this section, we will prove that the upper bounds for the expression complexities (Cor. 3.7) are sharp. This will imply that the upper bounds for the combined complexity for injective automatic presentations from Thm. 3.5 is sharp as well.

For a binary relation  $r$  and  $m \in \mathbb{N}$  we denote with  $r^m$  the  $m$ -fold composition of  $r$ . The following lemma is folklore.

**Lemma 4.1.** *Let the signature  $\mathcal{S}$  contain a binary symbol  $r$ . From a given number  $m$  (encoded unary), we can construct in linear time a formula  $\varphi_m(x, y)$  such that for every  $\mathcal{S}$ -structure  $\mathcal{A}$  and all elements  $a, b \in \mathcal{A}$  we have:  $(a, b) \in r^{2^m}$  if and only if  $\mathcal{A} \models \varphi_m(a, b)$ .*

For a bit string  $u = a_1 \cdots a_m$  ( $a_i \in \{0, 1\}$ ) let  $\text{val}(u) = \sum_{i=0}^{m-1} a_{i+1}2^i$  be the integer value represented by  $u$ . Vice versa, for  $0 \leq i < 2^m$  let  $\text{bin}_m(i) \in \{0, 1\}^m$  be the unique string with  $\text{val}(\text{bin}_m(i)) = i$ .

**Theorem 4.2.** *There exists a fixed automatic structure  $\mathcal{A}$  of bounded degree such that  $\text{Th}(\mathcal{A})$  is  $2\text{EXPSPACE}$ -hard.*

*Proof.* Let  $M$  be a fixed Turing machine with a space bound of  $\exp(2, n)$  such that  $M$  accepts a  $2\text{EXPSPACE}$ -complete language; such a machine exists by standard arguments. Let  $\Gamma$  be the tape alphabet,  $\Sigma \subseteq \Gamma$  be the input alphabet, and  $Q$  be the set of states. The initial (resp. accepting) state is  $q_0 \in Q$  (resp.  $q_f \in Q$ ), the blank symbol is  $\square \in \Gamma \setminus \Sigma$ . Let  $\Omega = Q \cup \Gamma$ . A configuration of  $M$  is described by a string from  $\Gamma^*Q\Gamma^+ \subseteq \Omega^+$  (later, symbols of configurations will be preceded with additional counters). For two configurations  $u$  and  $v$  with

$|u| = |v|$  we write  $u \vdash_M v$  if  $u$  can evolve with a single  $M$ -transition into  $v$ . Note that there exists a relation  $\alpha_M \subseteq \Omega^3 \times \Omega$  such that for all configurations  $u = a_1 \cdots a_m$  and  $v = b_1 \cdots b_m$  ( $a_i, b_i \in \Omega$ ) we have  $u \vdash_M v$  if and only if

$$\forall i \in \{2, \dots, m-1\} : (a_{i-1}a_i a_{i+1}, b_i) \in \alpha_M. \quad (6)$$

Let  $\Delta = \{0, 1, \#\} \cup \Omega$ , and let  $\pi : \Delta \rightarrow \Omega \cup \{\#\}$  be the projection morphism with  $\pi(a) = a$  for  $a \in \Omega \cup \{\#\}$  and  $\pi(0) = \pi(1) = \varepsilon$ . For  $m \in \mathbb{N}$ , a string  $x \in \Delta^*$  is an *accepting  $2^m$ -computation* if  $x$  can be factorized as  $x = x_1 \# x_2 \# \cdots x_n \#$  for some  $n \geq 1$  such that:

- For every  $1 \leq i \leq n$  there exist  $a_{i,0}, \dots, a_{i,2^m-1} \in \Omega$  such that  $x_i = \prod_{j=0}^{2^m-1} \text{bin}_m(j) a_{i,j}$ .
- For every  $1 \leq i \leq n$ ,  $\pi(x_i) \in \Gamma^* Q \Gamma^+$ .
- $\pi(x_1) \in q_0 \Sigma^* \square^*$  and  $\pi(x_n) \in \Gamma^* q_f \Gamma^+$
- For every  $1 \leq i < n$ ,  $\pi(x_i) \vdash_M \pi(x_{i+1})$ .

From  $M$  we now construct a fixed automatic structure  $\mathcal{A}$  of bounded degree. We start with the following regular language  $U_0$ :

$$U_0 = \pi^{-1}((\Gamma^* Q \Gamma^+ \#)^*) \cap \quad (7)$$

$$(0^+ \Omega (\{0, 1\}^+ \Omega)^* 1^+ \Omega \#)^+ \cap \quad (8)$$

$$0^+ q_0 (\{0, 1\}^+ \Sigma)^* (\{0, 1\}^+ \square)^* \# \Delta^* \cap \quad (9)$$

$$\Delta^* q_f (\Delta \setminus \{\#\})^* \# \quad (10)$$

A string  $x \in U_0$  is a candidate for an accepting  $2^m$ -computation of  $M$ . With [\(7\)](#) we describe the basic structure of such a computation, it consists of a list of configurations separated by  $\#$ . Moreover, every symbol in a configuration is preceded by a bit string, which represents a *counter*. By [\(8\)](#) every counter is non-empty, the first symbol in a configuration is preceded by a counter from  $0^+$ , the last symbol is preceded by a counter from  $1^+$ . Moreover, by [\(9\)](#), the first configuration is an initial configuration, whereas by [\(10\)](#), the last configuration is accepting (i.e. the state is  $q_f$ ).

For the further considerations, let us fix some  $x \in U_0$ . Hence, we can write  $x$  as  $x = x_1 \# x_2 \# \cdots x_n \#$  such that:

- For every  $1 \leq i \leq n$ , there exist  $m_i \geq 1$ ,  $a_{i,0}, \dots, a_{i,m_i} \in \Omega$  and counters  $u_{i,0}, \dots, u_{i,m_i} \in \{0, 1\}^+$  such that  $x_i = \prod_{j=0}^{m_i} u_{i,j} a_{i,j}$ .
- For every  $1 \leq i \leq n$ ,  $u_{i,0} \in 0^+$ ,  $u_{i,m_i} \in 1^+$ , and  $\pi(x_i) \in \Gamma^* Q \Gamma^+$ .
- $\pi(x_1) \in q_0 \Sigma^* \square^*$  and  $\pi(x_n) \in \Gamma^* q_f \Gamma^+$

We next want to construct, from  $m \in \mathbb{N}$ , a small formula expressing that  $x$  is an accepting  $2^m$ -computation. To achieve this, we add some structure around strings from  $U_0$ . Then the formula we are seeking has to ensure two facts:

- (a) The counters behave correctly, i.e. for all  $1 \leq i \leq n$  and  $0 \leq j \leq m_i$ , we have  $|u_{i,j}| = m$  and if  $j < m_i$ , then  $\text{val}(u_{i,j+1}) = \text{val}(u_{i,j}) + 1$ . Note that this enforces  $m_i = 2^m - 1$  for all  $1 \leq i \leq n$ .

(b) For two successive configurations, the second one is the successor configuration of the first one, i.e.,  $\pi(x_i) \vdash_M \pi(x_{i+1})$  for all  $1 \leq i < n$ .

In order to achieve (a), we introduce the following three binary automatic relations:

$$\begin{aligned} \sigma_0 &= \{(0v\#, v0\#) \mid v \in (\Delta \setminus \{\#\})^*\}^+ \\ \sigma_\Omega &= \left( \{(au, ua) \mid u \in \{0, 1\}^+, a \in \Omega\}^+(\#, \#) \right)^+ \\ \delta &= \left( \{(ua, va) \mid a \in \Omega, u, v \in \{0, 1\}^+, |u| = |v|, \right. \\ &\quad \left. \text{val}(v) = \text{val}(u) + 1 \pmod{2^{|u|}}\}^+(\#, \#) \right)^+ \end{aligned}$$

Hence,  $\sigma_0$  cyclically rotates every configuration to the left for one symbol, provided the first symbol is 0, whereas  $\sigma_\Omega$  shifts all  $\Omega$ -symbols one step to the right in every configuration. The relation  $\delta$  increments every counter modulo  $2^{\text{length of the counter}}$ . The crucial fact is the following:

**Fact 1.** For every  $m \in \mathbb{N}$ , the following two properties are equivalent (recall that  $x \in U_0$ ):

- $\exists y_1, y_2 \in \Delta^* : \delta(x, y_2), \sigma_0^m(x, y_1), \sigma_\Omega(y_1, y_2)$ .
- For all  $1 \leq i \leq n$  and  $0 \leq j \leq m_i$ , we have  $|u_{i,j}| = m$  and if  $j < m_i$ , then  $\text{val}(u_{i,j+1}) = \text{val}(u_{i,j}) + 1$ .

Assume now that  $x \in U_0$  satisfies one (and hence both) of the two properties from Fact 1 for some  $m$ . It follows that  $m_i = 2^m - 1$  for all  $1 \leq i \leq n$  and

$$x = x_1\#x_2\#\dots\#x_n\#, \text{ where } x_i = \prod_{j=0}^{2^m-1} \text{bin}_m(j)a_{i,j} \text{ for every } 1 \leq i \leq n. \quad (11)$$

In order to establish (b) we need additional structure. The idea is, for every counter value  $0 \leq j < 2^m$ , to have a word  $y_j$  that coincides with  $x$ , but has all the occurrences of  $\text{bin}_m(j)$  marked. Then an automaton can check that successive occurrences of the counter  $\text{bin}_m(j)$  obey the transition condition of the Turing machine. There are two problems with this approach: first, in order to relate  $x$  and  $y_j$ , we would need a binary relation of degree  $2^m$  (for arbitrary  $m$ ) and, secondly, an automaton cannot mark all the occurrences of  $\text{bin}_m(j)$  at once (for some  $j$ ). In order to solve these problems, we introduce a binary relation  $\mu$ , which for every  $x \in U_0$  as in (11) generates a binary tree of depth  $m$  with root  $x$ ; this will be the only relation in our automatic structure that causes exponential growth. This relation will mark in  $x$  every occurrence of an arbitrary counter. For this, we need two copies  $\bar{0}$  and  $\underline{0}$  of 0 as well as two copies  $\bar{1}$  and  $\underline{1}$  of 1. For  $b \in \{0, 1\}$ , we define the mapping  $f_b : \{\underline{0}, \bar{0}, \underline{1}, \bar{1}\}^* \{0, 1\}^+ \rightarrow \{\underline{0}, \bar{0}, \underline{1}, \bar{1}\}^+ \{0, 1\}^*$  as follows (where  $u \in \{\underline{0}, \bar{0}, \underline{1}, \bar{1}\}^*$ ,  $c \in \{0, 1\}$ , and  $v \in \{0, 1\}^*$ ):

$$f_b(ucv) = \begin{cases} u\underline{c}v & \text{if } b \neq c \\ u\bar{c}v & \text{if } b = c \end{cases}$$

We extend  $f_b$  to  $((\{\underline{0}, \overline{0}, \underline{1}, \overline{1}\}^* \{0, 1\}^+ \Omega)^+ \#)^*$  as follows: For  $w = w_1 a_1 \cdots w_\ell a_\ell$  with  $w_i \in \{\underline{0}, \overline{0}, \underline{1}, \overline{1}\}^* \{0, 1\}^+$  and  $a_i \in \Omega \cup \Omega \#$  let  $f_b(w) = f_b(w_1) a_1 \cdots f_b(w_\ell) a_\ell$ . Since  $f_b$  can be computed with a synchronized transducer, the relation  $\mu = f_0 \cup f_1$  (here  $f_b$  is viewed as a binary relation) is automatic.

Let  $x \in U_0$  as in (11), let the word  $y$  be obtained from  $x$  by overlining or underlining each bit in  $x$ , and let  $u \in \{0, 1\}^m$  be some counter. We say the counter  $u$  is marked in  $y$  if every occurrence of the counter  $u$  is marked by overlining each bit, whereas all other counters contain at least one underlined bit.

**Fact 2.** Let  $x \in U_0$  be as in (11).

- For every counter  $u \in \{0, 1\}^m$ , there is a unique  $y$  such that  $(x, y) \in \mu^m$  and  $u$  is marked in  $y$ .
- If  $(x, y) \in \mu^m$ , then there exists a unique counter  $u \in \{0, 1\}^m$  such that  $u$  is marked in  $y$ .

Now, we can achieve our final goal, namely checking whether two successive configurations in  $x \in U_0$  represent a transition of the machine  $M$ . Let the counter  $u \in \{0, 1\}^m$  be marked in  $y$ . We describe a finite automaton  $A_1$  that checks on the string  $y$ , whether at position  $\text{val}(u)$  successive configurations in  $x$  are “locally consistent”. The automaton  $A_1$  searches for the first marked counter in  $y$ . Then it stores the next three symbols  $a_1, a_2, a_3$  from  $\Omega$  (if the separator  $\#$  is seen before, then only one or two symbols may be stored), walks right until it finds the next marked counter, reads the next three symbols  $b_1, b_2, b_3$  from  $\Omega$ , and checks whether  $(a_1 a_2 a_3, b_2) \in \alpha_M$ , where  $\alpha_M$  is from (6). If this is not the case, then  $A_1$  will reject, otherwise it will store  $b_1 b_2 b_3$  and repeat the procedure described above. Let  $U_1 = L(A_1)$ . Together with Fact 1 and 2, the behavior of  $A_1$  implies that for all  $x \in U_0$  and all  $m \in \mathbb{N}$ ,  $x$  represents an accepting  $2^m$ -computation of  $M$  iff  $x$  satisfies the formula

$$\Phi(x) = \exists y_1, y_2 (\delta(x, y_2) \wedge \sigma_0^m(x, y_1) \wedge \sigma_\Omega(y_1, y_2)) \wedge \forall y (\mu^m(x, y) \rightarrow y \in U_1).$$

Let us now fix some input  $w = a_1 a_2 \cdots a_n \in \Sigma^*$  with  $|w| = n$ , and let  $a_{n+1} = \square$  and  $m = 2^n$ . Thus,  $w$  is accepted by  $M$  if and only if there exists an accepting  $2^m$ -computation  $x$  such that in the first configuration of  $x$ , the tape content is of the form  $w \square^+$ . It remains to add some structure that allows us to express the latter by a formula. But this is straightforward: Let  $\triangleright$  be a new symbol and let

$$\Pi = \Delta \cup \{\underline{0}, \overline{0}, \underline{1}, \overline{1}, \triangleright\};$$

this is our final alphabet. Define the binary automatic relations  $\iota_{01}$  and  $\iota_a$  ( $a \in \Omega$ ) as follows:

$$\begin{aligned} \iota_{01} &= \{(u \triangleright av, ua \triangleright v) \mid a \in \{0, 1\}, u, v \in \Delta^*\} \cup \{(0v, 0 \triangleright v) \mid v \in \Delta^*\} \\ \iota_a &= \{(u \triangleright av, ua \triangleright v) \mid u, v \in \Delta^*\}. \end{aligned}$$

Then, the first configuration of  $x$  has a tape from  $w \square^+$  if and only if  $x$  satisfies the formula

$$\Psi(x) = \exists y_0, z_0, \dots, y_{n+1}, z_{n+1} (\iota_{01}^m(x, y_0) \wedge \iota_{q_0}(y_0, z_0) \wedge \bigwedge_{i=1}^{n+1} \iota_{0,1}^m(z_{i-1}, y_i) \wedge \iota_{a_i}(y_i, z_i)).$$

Then,  $\mathcal{A} = (\Pi^*, \sigma_0, \sigma_\Omega, \delta, \mu, \iota_{01}, (\iota_a)_{a \in \Omega}, U_0, U_1)$  is an automatic structure of bounded degree such that  $M$  accepts  $w$  iff the formula  $\exists x \in U_0 (\Phi(x) \wedge \Psi(x))$  holds in  $\mathcal{A}$ . Lemma 4.1 allows to compute in time  $O(\log(m)) = O(n)$  an equivalent formula over the signature of  $\mathcal{A}$ . This concludes the proof.  $\square$

The proof of the next result is in fact a simplification of the proof of Thm. 4.2 since we do not need counters. In particular, the  $\mu$ -relation in the proof of Thm. 4.2 which was responsible for exponential growth, is not needed:

**Theorem 4.3.** *There exists a fixed automatic structure  $\mathcal{A}$  of bounded degree and polynomial growth (in fact linear growth) such that  $\text{Th}(\mathcal{A})$  is EXPSPACE-hard.*

## 5 Bounded Quantifier Alternation Depth and Open Problems

In this section we state some facts about first-order fragments of fixed quantifier alternation depth. These results can be deduced by reusing the construction from Section 4.

For  $n \geq 0$ , a  $\Sigma_n$ -formula is a formula in prenex normal form, where the quantifier prefix consists of  $n$  alternating blocks and the first block is a block of existential quantifiers. The  $\Sigma_n$ -theory of a structure  $\mathcal{A}$  is the set of all  $\Sigma_n$ -formulas in  $\text{Th}(\mathcal{A})$ . For a class  $\mathcal{C}$  of automatic presentations, the  $\Sigma_n$ -model checking problem  $\Sigma_n\text{-MC}(\mathcal{C})$  of  $\mathcal{C}$  denotes the set of all pairs  $(P, \varphi)$  where  $P \in \mathcal{C}$ , and  $\varphi$  belongs to the  $\Sigma_n$ -theory of  $\mathcal{A}(P)$ . The following result can be found in [3]:

**Theorem 5.1 (cf. [3]).** *The problem  $\Sigma_1\text{-MC}(\text{SA})$  is in PSPACE. Moreover, there is a fixed automatic structure with a PSPACE-complete  $\Sigma_1$ -theory.*

From our construction in the proof of Thm. 4.3, we can slightly sharpen the lower bound in this theorem:

**Theorem 5.2.** *There exists a fixed automatic structure of bounded degree and polynomial growth (in fact linear growth) with a PSPACE-complete  $\Sigma_1$ -theory.*

Let us now move on to  $\Sigma_2$ -formulas and structures of arbitrary growth:

**Theorem 5.3.**  *$\Sigma_2\text{-MC}(\text{SA})$  is in EXPSPACE. Moreover, there exists a fixed automatic structure of bounded degree with an EXPSPACE-complete  $\Sigma_2$ -theory.*



For  $n \geq 3$ , the precise complexity of the  $\Sigma_n$ -theory of an automatic structure of bounded degree remains open. From our results, it follows that the complexity is somewhere between EXPSPACE and 2EXPSPACE.

*Conjecture 5.4.* For  $n \geq 3$ ,  $\Sigma_n$ -MC(Sab) is in EXPSPACE.

A possible attack to this conjecture would follow the line of argument in the proof of Thm. 3.5 and would therefore be based on Gaifman's theorem. To make this work, the exponential bound in Gaifman's theorem would have to be reduced which leads to the following conjecture:

*Conjecture 5.5.* Let  $\mathcal{A}$  be a structure,  $\bar{a}, \bar{b} \in \mathcal{A}^k$ , and  $d, n \geq 0$  such that the spheres of radius  $d \cdot 2^n$  around  $\bar{a}$  and  $\bar{b}$  are isomorphic. Then, for every  $\Sigma_n$ -formula  $\varphi(x_1, \dots, x_k)$  of quantifier depth at most  $d$ , we have:  $\mathcal{A} \models \varphi(\bar{a})$  iff  $\mathcal{A} \models \varphi(\bar{b})$ .

## References

1. Bárány, V., Kaiser, L., Rubin, S.: Cardinality and counting quantifiers on omega-automatic structures. In: STACS 2008, pp. 385–396. IFIB Schloss Dagstuhl (2008)
2. Blumensath, A.: Automatic structures. Technical report, RWTH Aachen (1999)
3. Blumensath, A., Grädel, E.: Automatic Structures. In: LICS 2000, pp. 51–62. IEEE Computer Society Press, Los Alamitos (2000)
4. Compton, K., Henson, C.: A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic* 48, 1–79 (1990)
5. Elgot, C.: Decision problems of finite automata design and related arithmetics. *Trans. Am. Math. Soc.* 98, 21–51 (1961)
6. Epstein, D., Cannon, J., Holt, D., Levy, S., Paterson, M., Thurston, W.: *Word Processing In Groups*. Jones and Bartlett Publishers, Boston (1992)
7. Finkel, O.: On decidability properties of one-dimensional cellular automata. arXiv.org (2008), <http://arxiv.org/abs/0903.4615>
8. Gaifman, H.: On local and nonlocal properties. In: *Logic Colloquium 1981*, pp. 105–135. North-Holland, Amsterdam (1982)
9. Hodgson, B.: On direct products of automaton decidable theories. *Theoret. Comput. Sci.* 19, 331–335 (1982)
10. Khoussainov, B., Nerode, A.: Automatic presentations of structures. In: Leivant, D. (ed.) *LCC 1994*. LNCS, vol. 960, pp. 367–392. Springer, Heidelberg (1995)
11. Khoussainov, B., Rubin, S., Stephan, F.: Definability and regularity in automatic structures. In: Diekert, V., Habib, M. (eds.) *STACS 2004*. LNCS, vol. 2996, pp. 440–451. Springer, Heidelberg (2004)
12. Kuske, D., Lohrey, M.: First-order and counting theories of  $\omega$ -automatic structures. *J. Symbolic Logic* 73, 129–150 (2008)
13. Kuske, D., Lohrey, M.: Automatic structures of bounded degree revisited. arXiv.org (2008), <http://arxiv.org/abs/0810.4998>
14. Lohrey, M.: Automatic structures of bounded degree. In: Vardi, M., Voronkov, A. (eds.) *LPAR 2003*. LNCS, vol. 2850, pp. 344–358. Springer, Heidelberg (2003)
15. Rubin, S.: Automata presenting structures: A survey of the finite string case. *Bull. Symbolic Logic* 14, 169–209 (2008)
16. Rybina, T., Voronkov, A.: Upper bounds for a theory of queues. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) *ICALP 2003*. LNCS, vol. 2719, pp. 714–724. Springer, Heidelberg (2003)
17. Weber, A.: On the valuedness of finite transducers. *Acta Inform.* 27, 749–780 (1990)

# Nondeterminism and Observable Sequentiality

James Laird

Department of Computer Science, University of Bath

**Abstract.** We give operational, intensional and extensional characterizations of a class of higher-order functionals which may be computed sequentially but nondeterministically.

Sequential algorithms on concrete data structures have been shown to correspond to (deterministic) “observably sequential functionals”, which can be computed in observably sequential PCF (SPCF), and in fact, in an affine version of SPCF in which there are no nested or recursively defined functions.

In this work, we extend these results to a setting with nondeterminism. The main new step is to define notions of concrete data structure in which the sets of cells, values and events are ordered. The nondeterministic states over an ordered CDS form a *biorder* in (essentially) the sense of Berry, and we show that co-stable and continuous functions, and stable and continuous functions on these biorders each correspond to states on a function-space concrete data structure (non-deterministic sequential algorithms), proving Cartesian closure for the corresponding categories.

We use these results to define a category of “convex sequential algorithms” which combine both stable and co-stable states, and use these give a model of SPCF extended with non-deterministic choice, for which we prove universality at finite types, and thus full abstraction.

## 1 Introduction

Capturing non-deterministic behaviour in higher-order, sequential functional languages presents a challenge for denotational semantics. Domain theoretic approaches to describing functions accurately up to total correctness are liable to encounter problems in reflecting sequentiality, as in the deterministic case. On the other hand, describing the branching behaviour of nondeterministic functional programs in intensional representations such as game semantics has also proved (perhaps surprisingly) difficult. In particular, combining the notion of innocence with non-determinism is problematic [6].

In this paper, we shall develop an account based on the notions of *observably sequential functional* and sequential algorithms. (Deterministic) observably sequential functionals were introduced by Cartwright and Felleisen [3], as a model for SPCF — a functional language in which different evaluation strategies for functions are observable, due to the presence of errors and simple control jumps. The original notion of observably sequential functional combines extensional and intensional aspects, but a purely intensional characterization was given in terms

of sequential algorithms on sequential data structures in [4], whilst a purely extensional characterization has been given in terms of “bistable biorders” [12]. In fact, an apparently much weaker version of SPCF, in which functions cannot share variables with their arguments is still sufficient to define the computable observably sequential functionals.

The object of this paper is to develop an analogous tripartite description of non-deterministic observably sequential functions, giving purely extensional (biorder) and purely intensional characterizations, both of which yield fully abstract models of SPCF with nondeterministic choice, and of its affinely typed restriction. The extensional aspect of our account is provided by biorders essentially related to Berry’s original bidomains [1]. This builds on previous work by the author describing fully abstract models of sequential languages such as the lazy  $\lambda$ -calculus using these domains [11]. However, although these models technically carry all observable information about program behaviour, they do not do so in a transparent way. Sequential algorithms on concrete data structures (or, more generally, games and strategies), by contrast, offer an intuitively appealing model of computation, in particular, by exposing the nature of interaction between function and argument. Thus the main contribution of this paper is to develop a suitable notion of nondeterministic sequential algorithm making this possible, and exhibiting the correspondence with stable and continuous functions. This requires an ordering on cells and values (i.e. game positions), to reflect the fact that (for example) any program which may diverge in response to a given argument, may also diverge in response to an argument with a wider range of behaviours. Since it proves to be impossible to fully “sequentialize” the states of ordered concrete data structures interpreting terms in the model, so that there is a unique sequence of moves to each position, our semantics may be seen as employing an intrinsically *positional* form of games [8], in an essential way.

Our fully abstract model of non-deterministic SPCF may be compared with the game semantics of an analogous prototypical non-deterministic functional language with state (erratic Idealized Algol) described by Harmer and McCusker [7]. The latter is given by representing strategies as pairs of their sets of divergent and non-divergent traces. Denotations in our may-and-must testing semantics are, similarly, “convex sequential algorithms” given by a pair of sequential algorithms consisting of a may interpretation and a must-interpretation.

## 2 Nondeterministic Observably Sequential PCF

Observably sequential PCF is Scott’s PCF extended with a set of “error” terms and a simple control operator capturing statically bound exceptions. We form “erratic SPCF”, or ESPCF by adding nondeterministic binary choice to SPCF (with a single error,  $\top$ ). (A possible interpretation is that  $\top$  represents deadlock, distinct from livelock/divergence.)

Types are given by the following grammar:

$$T ::= \text{bool} \mid \text{nat} \mid S \times T \mid S \rightarrow T$$

To the simply-typed  $\lambda$ -calculus with products over these types, we add the following typed constants:

**Arithmetic** — Numerals, Booleans, successor, predecessor, zero-testing.

**Error** —  $\top : T$ .

**Conditional** —  $\text{If} : \text{bool} \times (B \times B) \rightarrow B$ ,

**Recursion** — Fixpoint combinators  $\mathbf{Y}_T : (T \rightarrow T) \rightarrow T$  at each type  $T$ .

**Control** —  $\text{strict?} : (T \rightarrow B) \rightarrow \text{bool}$ , which returns **tt** if its argument is strict, **ff** if it is non-strict (and both if it is a choice between strict and non-strict functions).

**Choice** —  $+ : B \times B \rightarrow B$  (will be written infix).

The operational semantics for programs — closed terms of basic type — is based on *evaluation contexts*, which are given by the grammar:

$E[\_] ::= [\_] | E[\_]M | \text{If } \langle E[\_], M \rangle M | \text{strict? } E[\_] | \text{strict? } \lambda k. E[\_] | \pi_i E[\_] | \text{op } E[\_]$  where **op** is drawn from the arithmetic operations. We write  $E_k[\_]$  for an evaluation context which does not capture the variable  $k$ . The basic, non-arithmetic reduction rules are as follows:

$$\begin{array}{ll}
 E[(\lambda x.M) N] & \longrightarrow E[M[N/x]] \\
 E[\pi_i \langle M, N \rangle] & \longrightarrow E[\pi_i M] \quad i \in \{1, 2\} \\
 E[\text{strict? } \lambda k. E_k[k]] & \longrightarrow E[\text{tt}] \\
 E[\text{strict? } \lambda k.v] & \longrightarrow E[\text{ff}] \\
 E[\text{If } \langle \text{tt}, M \rangle] & \longrightarrow E[\pi_1 M] \\
 E[\text{If } \langle \text{ff}, M \rangle] & \longrightarrow E[\pi_2 M] \\
 E[\mathbf{Y} M] & \longrightarrow E[M(Y M)] \\
 E[M_1 + M_2] & \longrightarrow E[M_i] \quad i \in \{1, 2\}
 \end{array}$$

For a program  $M$  (closed term of ground type) we write  $M \Downarrow$  if there exists a terminating reduction of  $M$  (i.e. if  $M : \text{nat}$ , then  $M \longrightarrow^* C$  where  $C$  is a numeral, or  $E[\top]$ ), and  $M \Uparrow$  if there exists a non-terminating reduction of  $M$  — i.e. a  $\omega$ -chain  $\{M_i \mid i \in \omega\}$  such that  $M_i \longrightarrow M_{i+1}$  for all  $i$ .

We derive a notion of observational approximation for each test, and for both combined: given terms  $M, N : T$ ,

- $M \lesssim^\top N$  if for all compatible program contexts  $C[\_]$ ,  $C[M] \Downarrow$  implies  $C[N] \Downarrow$ .
- $M \lesssim_\perp N$  if for all compatible program contexts  $C[\_]$ ,  $C[N] \Uparrow$  implies  $C[M] \Uparrow$ .
- $M \lesssim_\perp^\top N$  if  $M \lesssim^\top N$  and  $M \lesssim_\perp N$ .  $\simeq_\perp^\top = \lesssim_\perp^\top \cap (\lesssim_\perp^\top)^c$ .

The first two relations represent a form of partial correctness — one with respect to error, and one with respect to divergence. The combination successfully captures a form of total correctness — e.g.  $M \simeq_\perp^\top \mathbf{n}$  if and only if  $M$  may reduce only to **n**, and cannot diverge or produce an error.

We give an example showing the limits of observable sequentiality in a non-deterministic functional setting, and illustrating one difficulty in giving a fully abstract semantics for it. Given  $M, N : \text{bool}$  let  $M; N = \text{If } \langle M, \langle N, N \rangle \rangle$ . Then the terms  $\lambda xy.x; y$ ,  $\lambda xy.y; x$ ,  $\lambda xy.y; x$ ,  $\lambda xy.y; x; y$  of type  $\text{bool} \rightarrow \text{bool} \rightarrow \text{bool}$  are pairwise (may or must) observationally distinguishable.

However the terms  $\lambda xy.x; y + y; x$  and  $\lambda xy.x; y; x + y; x; y$  are may-and-must equivalent — both terms nondeterministically choose an order in which to test both arguments, and then return either the left or right one. To distinguish them it would be necessary to determine the order in which the arguments were tested *after* the final value has been returned. This suggests that a strongly sequential representation of program-argument interaction as *sequences* of moves is not what is required.

### 3 Ordered Concrete Data Structures

Concrete data structures [2] consist of sets of *cells*, *values*, *events* (which are pairs of cells and values), and an *enabling relation* between set of events and values. The idea is that each step of a sequential computation is represented as an event (the filling of a cell with a value), which may be dependent on a previous events having occurred (as specified by the enabling relation). Deterministic programs are interpreted as states, which specify a unique value for filling enabled cells. Thus they are represented as sets of events which satisfy two conditions: *consistency* — every cell must be filled with a unique value — and *safety* — for every filled cell there is a finite chain of enablings of filled cells within the state, back to an “initial cell”.

In order to model nondeterministic computation we must clearly drop the consistency condition. Instead, we place an ordering on cells and values, and require upwards closure under this ordering. To model the “observably sequential” character of the computation, we include a distinct “failure element”,  $\bullet$ , with which cells may be filled. This has a different meaning depending on the kind of failure being tested for: filling a cell with  $\bullet$  in the stable (total correctness) model represents divergence, whereas failing to fill an enabled cell represents failure due to error. Conversely, in the co-stable (partial correctness) model,  $\bullet$  represents error and an unfilled cell represents divergence.

**Definition 1.** A (filiform) ordered concrete data structure (ocds)  $A$  is a tuple  $(C_A, V_A, \vdash_A, E_A)$  where  $C_A, V_A$  are partial orders not containing the distinguished element  $\bullet$ ,  $E_A \subseteq C_A \times V_A$  is a set of events and  $\vdash_A \subseteq (E_A \cup \{*\}) \times C_A$  is an enabling relation such that  $(c, v) \vdash c'$  implies  $c \leq c'$ . We shall further assume that every ocds is bounded (every cell dominates finitely many cells and every value is dominated by finitely many values).

An interesting subclass of ocds is obtained by imposing the further condition that if  $(c_1, v) \vdash c'$  and  $(c_2, v) \in E_A$  then  $(c_2, v) \vdash c'$ , so that  $\vdash$  is equivalent to a relation between values and cells. Then we may think of cells and values as positions in a game in which Player moves are events, and Opponent moves are enablings. These “graph game” ocds form full Cartesian closed subcategories of our categories of ocds and stable or co-stable maps.

We write  $E_\bullet(A)$  for the partial order  $E_A \cup (C_A \times \{\bullet\})$ , with  $(c, \bullet) \leq (c', v)$  if  $c \leq c'$ .

So, for example, for any set  $X$  we have a “powerdomain” ocds  $\tilde{X}$  with a single cell which may be filled by any element of  $X$ :  $\tilde{X} = (\{c\}, X, \{c\} \times X, (\bullet, c))$ .

**Definition 2.** A proof  $P$  of an event  $e$  is a finite sequence of events  $\langle (c_i, v_i) \mid i \leq n \rangle$  such that  $* \vdash c_0$ ,  $e_n = e$  and  $e_\alpha \vdash c_{\alpha+1}$  for  $i \leq n$ .

We write  $x \vdash^* e$  if there is a proof of  $e$ , all of the elements of which are in  $x$  (we write  $x \vdash^* c$  if  $x \vdash^* (c, \bullet)$ ).

Note that, given a proof  $\langle (c_i, v_i) \mid i \leq n \rangle$  of a cell  $d$ ,  $c_i < d$  for all  $i \leq n$ , so in a bounded ocds, every cell has at most finitely many proofs.

A state of an ocds  $A$  is an upper set  $x \subseteq E_\bullet(A)$  satisfying:

**Safety.** If  $e \in x$  then there exists  $e' \leq e$  such that  $x \vdash^* e'$ .

A state is *finite*-branching if every cell which contains infinitely many values also contains  $\bullet$ .

We write  $D(M)$  for the set of states of the ocds  $M$ , and  $D_{fin}(M)$  for the finitely branching states.

So, for any set  $X$ , the states of  $\tilde{X}$  are the full set of events (up-closure of  $(c, \bullet)$ ) and for any  $Y \subseteq X$ , the state  $\{(c, v) \mid v \in y\}$ . The finitely branching states are the full state, and those for which the set of values filling  $c$  is finite.

We write  $F(x)$  for the set of filled cells of the state  $x$  over  $M$  — i.e.  $F(x) = \{c \in C_m \mid \exists a.(c, a) \in x\}$ . We write  $A(x)$  for the set of *accessible* cells (enabled but unfilled) from  $x$ :  $\{c \in C_M \mid x \vdash^* c \wedge c \notin F(x)\}$ . For a set of values  $V$ , we write  $x + (c, V)$  for the state  $\bigcup \{x + (c, v) \mid v \in V\}$ . We say that a state  $x \in D(A)$  is *total* if  $x \subseteq E_A$ .

We now describe how to construct *biorders* from the sets of all states, and sets of finite states of an ordered CDS. A (meet) biorder [15] is a set  $D$  with partial orders  $\sqsubseteq, \leq_s$  such that:

- $(D, \sqsubseteq)$  (the extensional order) is a meet semi-lattice.
- $(D, \leq_s)$  (the stable order) is included in  $(D, \sqsubseteq)$ , the meet operator is monotone with respect to  $\leq_s$ , and its unit is the  $\leq_s$ -least element in  $(D, \leq_s)$ .

A join biorder is the dual of a meet biorder — in this case we refer to the dual of the stable order  $\leq_s$ , as the co-stable order  $\leq_c$ .

To generate a biorder from the set of (all, or just finitely branching) states of an ocds, we take the extensional order to be reverse inclusion (meet biorder) or inclusion (join biorder), and define the stable coherence relation (i.e. the relation of being bounded above in the stable order, or below in the co-stable order) on states as follows:  $x \uparrow y$  if

$$\begin{aligned} (c, v) \in x &\text{ implies } (c, v) \in y \text{ or } (c, \bullet) \in x \\ (c, v) \in y &\text{ implies } (c, v) \in x \text{ or } (c, \bullet) \in y, \end{aligned}$$

Since every cell in a state which is filled with  $\bullet$  is also filled with any acceptable value, two coherent states may differ only according to which cells are filled with  $\bullet$ .

In  $\tilde{X}$ , the full state (up-closure of  $(c, \bullet)$ ) is stably coherent with all other states; every other pair of states is incoherent.

We may define the stable order by  $x \leq_s y$  if  $y \subseteq x$  and  $x \uparrow y$  — i.e. if  $(c, v) \in x$  then  $(c, v) \in y$  or  $(c, \bullet) \in x$ . Clearly, this is a well-defined order.

**Lemma 1.**  $x \uparrow y$  if and only if  $x$  and  $y$  are bounded above in  $\leq_s$ .

*Proof.* If  $x \uparrow y$ , then the set  $z$  of events in  $e \in x \cap y$  such that there exists an event  $e' \leq e$  with a proof in  $x \cap y$  is clearly a well-defined state. It remains to show that  $x \leq_s z$ . So suppose  $(c, v) \in x$ . Then there exists  $(c', v') \in x$  with a proof  $\{e_1, \dots, e_n\}$  in  $x$ . If  $\{e_1, \dots, e_n\} \subseteq y$ , then  $(c, v) \in z$  as required. Otherwise, there exists  $i$  with  $(c_i, v_i) \notin y$  and hence  $(c_i, \bullet) \in x$ . So  $(c, \bullet) \in x$  as required.

$(D_{fin}(M), \supseteq)$  is evidently a semi-lattice with respect to the union operation, with unit  $E_\bullet(M)$ . Thus to show that  $(D_{fin}(M), \supseteq, \leq_s)$  is a meet biorder, it suffices to observe that if  $x \uparrow y$  then  $x \cup y \uparrow x$ . (Suppose  $(c, v) \in x \cup y$ , then  $(c, v) \in x$  (as required) or else  $(c, v) \in y$ , and since  $y \uparrow x$ , we have either  $(c, \bullet) \in y \subseteq x \cup y$  or else  $(c, v) \in x$ .)

Similarly, if the co-stable order is defined  $x \leq_c y$  if  $x \subseteq y$  and  $x \uparrow y$ , then  $(D(M), \subseteq, \leq_c)$  is a join biorder.

Requiring the extensional order to be complete leads to the notions of meet and join bicpo. If  $X, Y$  are directed sets, we define  $X \leq_s Y$  (resp.  $X \leq_c Y$ ) if for all  $(x, y) \in X \times Y$ , there exists  $(x', y') \in X \times Y$  with  $x \sqsubseteq x', y \sqsubseteq y'$  and  $x' \leq_s y'$ .

- A *meet bicpo* is a meet biorder in which the extensional order is a dcpo, the meet operation is continuous, and for any directed sets  $X, Y$ ,  $X \leq_s Y$  implies  $\bigsqcup X \leq_s \bigsqcup Y$ . (So the stable order is also complete.)
- A *join bicpo* is a join biorder in which the extensional order is a dcpo, and for any directed sets  $X, Y$ ,  $X \leq_c Y$  implies  $\bigsqcup X \leq_c \bigsqcup Y$ .

(Note that these are not dual.)

**Proposition 1.** For any ordered cds  $M$ ,  $(D(M), \subseteq, \leq_c)$  is a join bicpo and  $(D_{fin}(M), \supseteq, \leq_s)$  is a meet bicpo.

*Proof.* It remains to show that the intersection operation on directed sets of states is well defined. Let  $S \subseteq D_{fin}(M)$  be a  $\supseteq$ -directed set of states. Then  $\bigcap S$  is a state (and thus a  $\supseteq$ -supremum):

**Safety** If  $e \in \bigcap S$  is  $\leq$ -minimal then there is a proof of  $e$  in each state  $x \in S$ .

But there are only finitely many proofs of  $e$ , and thus at least one proof occurs in every state in  $S$ , and thus in  $\bigcap S$ .

**Finite Branching** If  $\{v \mid (c, v) \in \bigcap S\}$  is infinite, then  $\{v \mid (c, v) \in x\}$  is infinite in each  $x \in S$  and so  $(c, \bullet) \in \bigcap S$  as required.

If  $X \leq_s Y$  then  $\bigcap X \leq_s \bigcap Y$ : suppose, for example, that  $(c, v) \in \bigcap X$ , but  $(c, v) \notin \bigcap Y$  — i.e. there exists  $y \in Y$  with  $(c, v) \not\sqsubseteq y$ . Then for all  $x \in X$ ,  $(c, v) \in X$ , and there exists  $y' \in Y$  with  $x \leq_s y'$  and  $y' \subseteq y$ , and so  $(c, v) \not\sqsubseteq y$  and hence  $(c, \bullet) \in x$ , and so  $(c, \bullet) \in \bigcap X$  as required.

**Lemma 2.**  $(D_{fin}(A), \leq_s)$  is algebraic.

*Proof.* An element of  $(D_{fin}(A), \leq_s)$  is (stably) compact if it contains finitely many filled cells which are not filled with  $\bullet$ . For any state  $x$ , we write  $\mathcal{K}(x)$  for the set

of compact stable approximants to the state  $x$ . For any element  $x$ , the set  $\mathcal{K}(x)$  is directed — if  $a, b \in \mathcal{K}(x)$  then their meet (as defined above) is in  $\mathcal{K}(x)$ . Any proof  $P$  of an event in  $x$  corresponds to a compact approximant  $y_p = P \cup \{(c, \bullet) \mid c \in F(x)\} \leq_s x$  and the intersection of all such approximants is  $x$ .

### 3.1 Stable Functions and Sequentiality

A  $\sqsubseteq$ -continuous function  $f : D \rightarrow E$  between algebraic meet bicpos is *stable* — in the sense of having a *trace* — if and only if it is conditionally multiplicative — i.e. for any  $x, y \in D$  which are stably coherent,  $f(x \sqcap y) = f(x) \sqcap f(y)$ . Similarly, a continuous function between join bicpos is *co-stable* if it is conditionally additive (i.e. preserves joins of coherent pairs). Note that this implies that for any set  $X$  bounded below in the co-stable order,  $f(\bigsqcup X) = \bigsqcup \{f(x) \mid x \in X\}$ .

So we may define categories  $\mathcal{B}$  — of meet bicpos and continuous, stable functions, and  $\mathcal{BC}$  — of complete biorders and continuous co-stable functions. These are both Cartesian closed  $\square$ , having products defined pointwise, with internal homs being sets of continuous and stable/co-stable functions, with the standard definition of extensional and stable orderings, dualized to give the co-stable order — i.e.

$$f \leq_c g \text{ if for all } x, y \in D, x \leq_c y \implies f(x) \leq_c g(y) \text{ and } g(y) = f(y) \sqcup g(x).$$

By Proposition  $\square$ , we may define two categories in which objects are ordered concrete data structures —  $\mathcal{OC}$ , in which morphisms from  $A$  to  $B$  are  $\sqsubseteq$ -continuous and conditionally additive functions from  $D(A)$  to  $D(B)$  — and  $\mathcal{OC}_{fin}$ , in which morphisms from  $A$  to  $B$  are  $\supseteq$ -continuous conditionally multiplicative functions from  $D_{fin}(A)$  to  $D_{fin}(B)$ . Both have all small products, with the product of two ordered ocDs being  $(C_1, V_1, E_1, \vdash_1) \times (C_2, V_2, E_2, \vdash_2) = ((C_1 + C_2), V_1 \cup V_2, \{(c.i, v) \mid (c, v) \in E_i, i \in \{1, 2\}\} \cup V_1 \cup V_2, \{(c.2, v) \mid (c, v) \in E_2\}, \{(c.i, v), c.i) \mid (c, v), c \in E_i, i \in I\}$ .

There are evident fully faithful functors from  $\mathcal{OC}$  into  $\mathcal{BC}$ , and from  $\mathcal{OC}_{fin}$  into  $\mathcal{B}$  which preserve products (up to isomorphism). Hence to establish Cartesian closure of  $\mathcal{OC}$  and  $\mathcal{OC}_{fin}$ , it suffices to define an exponent ocDs  $A \Rightarrow B$  for each  $A, B$ , such that  $D(A \Rightarrow B) \cong [D(A), D(B)]$  in  $\mathcal{BC}$  and  $D_{fin}(A \Rightarrow B) \cong [D_{fin}(A), D_{fin}(B)]$ . This is the key result of the paper, since it establishes that every stable and continuous function between sets of (finite/infinite branching) states of an ocDs is computed by a unique sequential algorithm.

Sequential data structures were introduced in order to describe sequential functions  $\mathcal{Q}$ . Essentially, a function between (orders generated from) sequential data structures is Kahn-Plotkin sequential if any argument (state)  $x$ , and cell  $c$  which is accessible in  $f(x)$ , and filled in  $f(y)$  can be associated with a cell, accessible from  $x$ , which must be filled in any state  $y$  such that  $c$  is filled in  $f(y)$ . In this original setting, divergence is represented implicitly, by not filling an enabled cell, whereas, in effect, we require “sequentiality with respect to the token  $\bullet$ ” (which may represent either error, or divergence). Thus we may translate the original definition of Kahn-Plotkin sequentiality to the current setting by (essentially) replacing the role of “accessible cell” with that of “cell



filled with  $\bullet$ ”, and “filled cell” with “enabled cell not filled with  $\bullet$ ”. (We also replace the inclusion relation on states (which, in the original setting, is the stable order) with coherence.

**Definition 3.** A function  $f : D(A) \rightarrow D(B)$  (or from  $D_{fin}(A)$  to  $D_{fin}(B)$ ) is  $\bullet$ -sequential if whenever  $(c, \bullet) \in f(x)$  then either of the following conditions hold:

- For all  $y$  with  $x \uparrow y$ ,  $(c, \bullet) \in f(y)$
- There exists  $(c', \bullet) \in x$  such that  $\forall z \uparrow x.(c', \bullet) \in f(z)$  implies  $(c, \bullet) \in f(z)$ .

Given a state  $x$  and an event  $(c, a)$  with  $x \vdash^* c$ , we write  $x + (c, a)$  for the state  $x \cup \{e \mid (c, a) \leq e\}$ .

**Proposition 2.** Any conditionally additive  $\subseteq$ -continuous function from  $D(A)$  to  $D(B)$  is  $\bullet$ -sequential.

*Proof.* Suppose  $f : D(A) \rightarrow D(B)$  is conditionally additive and  $\subseteq$ -continuous. For any  $x \in D(A)$  such that  $(c, \bullet) \in f(x)$  but  $(c, \bullet) \notin f(x \cap E_A)$ , the set  $Y = \{(x \cap E_A) + (c', \bullet) \mid c' \in A(x \cap E_A)\}$  is pairwise coherent with  $\bigcup Y = x$ . So  $(c, \bullet) \in f(x) = \bigcup \{f(y) \mid y \in Y\}$  — i.e. there exists  $c'$  with  $f(c, \bullet) \in ((x \cap E_A) + (c', \bullet))$  as required.

**Proposition 3.** Any conditionally multiplicative  $\supseteq$ -continuous function from  $D_{fin}(A)$  to  $D_{fin}(B)$  is  $\bullet$ -sequential.

*Proof.* Suppose  $f : D_{fin}(A) \rightarrow D_{fin}(B)$  is conditionally multiplicative and  $\supseteq$ -continuous. For any  $x \in D_{fin}(A)$  such that  $(c, \bullet) \in f(x)$  but  $(c, \bullet) \notin f(x \cap E_A)$ , we show that there exists  $z$  such that  $x \cap E_A \subseteq z \subseteq x$  and the set of cells filled with  $\bullet$  in  $z$  is finite, from which  $\bullet$ -sequentiality follows as for Prop. □

Let  $S$  be the set of states  $y$  such that  $x \cap E_A \subseteq y \subseteq x$ , and the set of cells filled (with  $\bullet$ ) in  $x$  but not in  $y$  is finite. Then  $S$  is  $\supseteq$ -directed, with least upper bound  $x \cap E_A$ , so by continuity, there exists  $y \in S$  with  $(c, \bullet) \notin f(y)$ . Let  $z = (x \cap E_A) \cup (x - y)$ , so that  $y \cup z = x$ , and hence  $(c, \bullet) \in f(z)$  by conditional multiplicativity (and the set of cells filled with  $\bullet$  in  $z$  is finite).

### 3.2 Nondeterministic Sequential Algorithms

We will now show that  $\mathcal{OC}$  and  $\mathcal{OC}_{fin}$  are Cartesian closed by defining an exponent  $\text{ocds}$ , and showing that  $D(A \Rightarrow B) \cong [D(A), D(B)]$  and  $D_{fin}(A \Rightarrow B) \cong [D_{fin}(A), D_{fin}(B)]$ .

- The *cells* of  $A \Rightarrow B$  are pairs of a total, finite state of  $A$  and a cell of  $B$ :  
 $C_{A \Rightarrow B} = (D(A) \cap \mathcal{P}_{fin}(E_A)) \times C_B$  — with  $(x, c) \leq (x', c')$  if  $x \subseteq x'$  and  $c \leq c'$ .
- A *value* of  $A \Rightarrow B$  is either a cell from  $A$  or a value from  $B$  — the order being determined pointwise from that of  $V_B$  and the dual of  $C_A$ :  
 $V_{A \Rightarrow B} = C_A^c \uplus V_B$

- A cell  $(x, c)$  of  $A \Rightarrow B$  may be *filled* with a cell accessible from  $x$  in  $A$  or a value filling  $c$  in  $B$ :  
 $E(A \Rightarrow B) = \{((x, c), c') \mid x \vdash_A^* c'\} \cup \{((x, c), v) \mid (c, v) \in E_B\}$
- A cell  $(x, c)$  is *initial* if  $x$  is the empty state and  $c$  is initial:  
 $* \vdash (x, c)$  if  $x = \{\}$  and  $* \vdash c$ .  
 A cell  $(x, c)$  filled with an  $A$ -cell  $c'$  enables another cell by filling  $c'$  in  $A$ :  
 $((x, c'), c) \vdash (y, c'')$  if  $c' = c''$  and  $\exists V \subseteq V_A. y = x + (c, V)$   
 A cell  $(x, c)$  filled with a  $B$ -value  $v$  enables another cell as in  $B$ :  
 $((x, c), v) \vdash (y, c')$  if  $x = y$  and  $(c, v) \vdash_B c'$ .

As a simple example, consider the exponent ocdds  $\tilde{\mathcal{O}}^X \Rightarrow \tilde{\mathcal{O}}$ . There is a single total state of  $\tilde{\mathcal{O}}^X$  — the empty state  $\{\}$ . Thus there is a single cell in  $\tilde{\mathcal{O}}^X \Rightarrow \tilde{\mathcal{O}}$  —  $(\{\}, c)$ , where  $c$  is the initial cell in  $\tilde{\mathcal{O}}$ . The values that can fill this cell are initial cells in  $\tilde{\mathcal{O}}^X$  — i.e.  $\{c.i \mid i \in X\}$ . Thus this ocdds is equivalent (i.e. there are isomorphisms of cells and values preserving the event and enabling relations) to  $\tilde{X}$ . More particularly, note that if  $A$  is any non-empty ocdds, there is a morphism *strict?* from  $A \Rightarrow \tilde{X}$  into  $\{\mathbf{tt}, \mathbf{ff}\}$  such that  $(c, \mathbf{tt}) \in \text{strict?}(x)$  iff there exists an (initial) cell  $c'$  in  $A$  with  $((\{\}, c), c') \in x$  and  $(c, \mathbf{ff}) \in \text{strict?}(x)$  if there exists a value  $v \in X$  with  $((\{\}, c), v) \in x$ . This will be the denotation of the *strict?* operator.

We first show how to obtain a continuous, co-stable function from a (possibly infinitely branching) sequential algorithm. Given a sequential algorithm  $\sigma \in D(A \Rightarrow B)$ , define:  $\text{fun}(\sigma)(x) =$

$$\{(c, a) \in E_\bullet(B) \mid \exists x' \subseteq_{fin} x. ((x', c), a) \in \sigma \vee \exists c'. (c', \bullet) \in x \wedge ((x', c), c') \in \sigma\}$$

To show that this is well-defined, we first observe that for any  $x$ ,  $\text{fun}(\sigma)(x)$  is a state — upwards closure is a consequence of upwards closure for  $\sigma$ , whilst if  $(c, a) \in \text{fun}(\sigma)(x)$ , then there exists  $((y, c), b) \in \sigma$  with  $y \subseteq x$ , and a proof of  $(y, c)$  in  $\sigma$  which therefore restricts to a proof of  $c$  in  $f(y)$ .

**Lemma 3.** *fun(σ) is a stable and continuous function.*

*Proof.*  $\text{fun}(\sigma)$  is clearly continuous with respect to inclusion. To show that it preserves coherence, suppose  $x \uparrow y$ . We show that if  $(c, v) \in \text{fun}(\sigma)(x \cup y)$  then  $(c, v) \in \text{fun}(\sigma)(x)$  or  $(c, \bullet) \in \text{fun}(\sigma)(y)$ .

Suppose  $(c, v) \in \text{fun}(\sigma)(x \cup y)$  but  $(c, v) \notin \text{fun}(\sigma)(x)$ . Then there exists  $w \subseteq x \cup y$  with  $((w, c), v) \in \sigma$  but  $w \not\subseteq x$ . Suppose  $P$  is a proof of  $((w, c), v)$  in  $\sigma$  and let  $((w', c'), a')$  be the least element of  $P$  such that  $w' \not\subseteq x$ . Then there is an immediately preceding event  $((w'', c''), c'')$  such that  $w' = w'' + (c'', V)$  for some set of values  $V$ , including a value  $u$  such that  $(c'', u) \notin x$ . Because  $w' \subseteq x \cup y$ ,  $(c'', u) \in y$  and so  $(c'', \bullet) \in y$  by coherence of  $x$  and  $y$ , and hence  $(c', \bullet) \in \text{fun}(\sigma)(y)$  and hence  $(c, \bullet) \in \text{fun}(\sigma)(y)$  as required.

Similarly, if  $(c, v) \in f(x \cup y)$  then either  $(c, v) \in f(x)$  or  $(c, v) \geq (c, \bullet) \in f(y)$ , and so  $f(x \cup y) \subseteq f(x) \cup f(y)$ .

**Lemma 4.** *If  $\sigma \uparrow \tau$  then  $\text{fun}(\sigma) \uparrow \text{fun}(\tau)$ .*

*Proof.* It is straightforward to show that  $\text{fun}(\sigma)(x) \leq_s \text{fun}(\tau)(x)$  for all  $x$ , so suppose  $x \uparrow y$  and  $(c, a) \in \text{fun}(\sigma)(x)$  — we need to show that  $(c, a) \in f_\tau(x)$  or  $(c, a) \in \text{fun}(\sigma)(y)$ . By Lemma 3 if  $(c, a) \notin \text{fun}(\sigma)(y)$ , then  $(c, \bullet) \in \text{fun}(\sigma)(x)$ , and so we may assume that  $a = \bullet$ .

So there exists an event  $((z, c), a) \in \sigma$  with  $z \subseteq x$  and either  $a = \bullet$  or else there exists  $z + (c', \bullet) \subseteq x$  such that  $((z, c), c') \in \sigma$ . But this latter case reduces to the first one, since either  $((z, c), c') \in \tau$  (and so  $(c, \bullet) \in \text{fun}(\tau)(x)$  and we are done), or else  $((z, c), \bullet) \in \sigma$ .

Assuming  $(c, \bullet) \notin \text{fun}(\sigma)(y)$ , let  $P$  be a proof of  $((z, c), \bullet)$  in  $\sigma$ , and let  $((z', c'), e')$  be the least element of  $P$  such that  $z' \not\subseteq y$ . Then there must be an immediately preceding event in  $P$  of the form  $((z'', c'), c'')$ , where  $z' = z'' + (c'', V)$  for some  $V$ , and hence  $z'' + (c'', \bullet) \subseteq x$  by coherence. If  $((z'', c'), c'') \in \tau$  then  $(c, \bullet) \in \text{fun}(\tau)(x)$ . Otherwise  $((z'', c'), \bullet) \in \sigma$  and so  $(c', \bullet) \in \text{fun}(\sigma)(y)$ .

For any sequential algorithms  $\sigma, \tau \in D(A \Rightarrow B)$ ,  $\text{fun}(\sigma \cup \tau) = \text{fun}(\sigma) \sqcup \text{fun}(\tau)$  by construction. Thus we have defined a continuous and conditionally additive map from  $D(A \Rightarrow B)$  to  $[D(A), D(B)]$

We now show that that function  $\text{fun}$  is an isomorphism by defining an inverse. Given a conditionally additive function  $f : D(A) \rightarrow D(B)$ , we define  $\text{strat}(f) \in D(A \Rightarrow B)$  as follows:

$$\{((x, c), a) \in E_\bullet(A \Rightarrow B) \mid (c, a) \in f(x) \vee (c' \in C_A \wedge (c, \bullet) \in f(x + (c', \bullet)))\}$$

**Lemma 5.** *For any conditionally additive continuous function  $f : D(A) \rightarrow D(B)$ ,  $\text{strat}(f)$  is a well-defined sequential algorithm.*

*Proof.* (Sketch.) It is straightforward to show that this is an up-closed set of events. So it remains to show that every event  $((x, c), a)$  in  $\text{strat}(f)$  has a proof in  $\text{strat}(f)$ . We construct such a proof using the sequentiality property for stable and continuous functions.

Let  $P = \langle (d_i, v_i) \mid i < l \rangle$  be a proof of  $c$  in  $f(x)$ . For each integer  $j$ , we define an event  $e_j = ((x_j, c_j), a_j) \in \text{strat}(f)$  such that  $\vdash e_0$ , and for all  $j$ :

- $x_j \subseteq x$  and  $c_j \leq c$
- Either  $e_j = ((x, c), a)$  or  $e_j \vdash e_{j+1}$ .
- $c_j = d_i$  for some  $i \leq j$ , and if  $a_j \in V(B)$ , then  $a_j = v_i$  for some  $i \leq j$ .

Starting with  $x_0 = \{\}$  and  $c_0 = d_0$ , we form  $x_{j+1}$  by adding all events in  $x$  which are fillings of  $c_j$  to  $x_j$ , and  $c_{j+1}$  to be the next reachable cell in the proof  $P$ . If this is filled in  $f(x_{j+1})$ , by a value  $v$  such that  $(c_{j+1}, v)$  s in  $P$ , then we may set  $a_{j+1} = v$ , otherwise, by the sequentiality property for  $f$  we may find a cell  $c' \in A(x_{j+1}) \cap F(x)$  such that  $(c_j, \bullet) \in f(x_{j+1} + (c', \bullet))$  and so we may set  $a_{j+1} = c'$ . By compactness of  $(x, c)$ , there exists  $n$  with  $x_n = x$  and  $c_n = c$ , and so  $\langle e_j \mid j \leq n \rangle$  is a proof of  $(x, c)$  in  $\text{strat}(f)$

We may also verify that  $\text{strat}(\_)$  preserves the inclusion and coherence relations, so that continuity and co-stability are consequences of the following.

**Theorem 1.** *The maps  $\text{strat}$  and  $\text{fun}$  are inverse isomorphisms.*

**Corollary 1.** *The category  $\mathcal{OC}$  of ordered concrete data structures and co-stable  $\sqsubseteq$ -continuous functions is Cartesian closed.*

The above results may be adapted to give an isomorphism between  $[D_{fin}(A), D_{fin}(B)]$  and  $D_{fin}(A \Rightarrow B)$  in  $\mathcal{B}$  for any  $A, B$ . We first observe that the operation  $\mathbf{strat}$  produces finite-branching states from  $\sqsupseteq$ -continuous functions.

**Proposition 4.** *If  $f : D_{fin}(A) \rightarrow D_{fin}(B)$  is stable continuous then  $\mathbf{strat}(f)$  is finite-branching.*

*Proof.* Suppose the set of  $a$  such that  $\{a \in V(A \Rightarrow B) \mid ((x, c), a) \in \mathbf{strat}(f)\}$  is infinite. If there are infinitely many  $v$  such that  $((x, c), v) \in \mathbf{strat}(f)$ , then  $((x, c), \bullet) \in \mathbf{strat}(f)$  because  $f(x)$  is finite-branching by definition.

So suppose there are infinitely many cells  $c'$  such that  $((x, c), c') \in \mathbf{strat}(f)$ . Let  $S$  be the set of states  $y$  such that  $y \leq_s x$  and  $A(x) - F(y)$  is finite. Then for any  $y \in S$  there exist infinitely many cells  $c' \in A(x)$  such that  $(c', \bullet) \in y$  and  $(c, \bullet) \in f(x + (c', \bullet))$ . Hence  $(c' \perp) \in f(y)$  for all  $y \in S$ , and  $S$  is  $\sqsupseteq$ -directed with supremum  $x$ , and thus  $(c, \bullet) \in f(x)$  by continuity, and so  $((x, c), \bullet) \in \mathbf{strat}(f)$  as required.

However, the function  $\mathbf{fun}(\sigma)$  is not  $\sqsupseteq$ -continuous in general, even for finite-branching (or even, deterministic)  $\sigma$ . Take, for example, the strategy from  $\widehat{1}^\omega$  to  $\widehat{\mathcal{O}}$  which queries each cell in  $\widehat{1}^\omega$  in turn — i.e.  $\sigma = \{(x_i, c'), c.i \mid i \in \omega\}$ , where  $x_i = \{(c.j, *) \mid j \leq i\}$ . Let  $y_i = x_i \cup \{(c.j, \perp) \mid j > i\}$  — then  $(c', \perp) \in \mathbf{fun}(\sigma)(y_i)$  for all  $i$ , but  $\mathbf{fun}(\sigma)(\bigcap_{i \in \omega} y_i) = \{\}$ .

So to extract a stable continuous function from a finite-branching sequential algorithm, we need to take account of the possibility of “livelock” in the interaction between function and argument. We may do this using the algebraic structure of domains of finite branching strategies.

**Proposition 5.**  *$\mathbf{strat} : [D_{fin}(A), D_{fin}(B)] \rightarrow D_{fin}(A \Rightarrow B)$  is an isomorphism.*

*Proof.* Define  $\mathbf{fun}_c(\sigma) : D_{fin}(A) \Rightarrow D_{fin}(B)$  by  $\mathbf{fun}_c(\sigma)(x) = \bigsqcup \{\mathbf{fun}(\sigma)(y) \mid y \in \mathcal{K}(x)\}$ . This is a continuous function by definition, and is stable thanks to the stability of the least upper bound operator.

For any  $f$ ,  $\mathbf{fun}_c(\mathbf{strat}(f))(y) = \mathbf{fun}(\mathbf{strat}(f))(y) = f(y)$  for compact states  $y$ , and thus  $\mathbf{fun}_c \cdot \mathbf{strat} = I$ , and since every finite and total state is compact, we also have  $\mathbf{strat}(\mathbf{fun}(\sigma)) = \sigma$  for all sequential algorithms  $\sigma$ .

We may also give a more direct characterization of the  $\sqsupseteq$ -continuous function corresponding to a finite branching sequential algorithm, by explicitly describing the livelocks which lead to divergence (in similar style to [7]). By including these,  $\sqsupseteq$ -continuity of the  $\mathbf{fun}$  operation may be recovered.

For any finite branching sequential algorithm  $\sigma : A \Rightarrow B$ , a livelock of  $\sigma$  at  $c$  is a total state  $x \in D_{fin}(A)$  such that there exists an  $\omega$ -chain of events  $\langle (x_i, c), c_i \mid i \in \omega \rangle$  in  $\sigma$  such that  $((x_i, c), c_i) \vdash (x_{i+1}, c)$  for each  $i$ , and  $\bigcup_{i \in \omega} x_i = x$ . Let  $\sigma^\perp$  be the set of tuples  $((x, c), \bullet) \in (D_{fin}(A) \times C_B) \times \{\bullet\}$  such that  $x$  is a livelock of  $\sigma$  at  $c$ . Using the fact that from each chain of approximants  $\langle x_i \mid i \in \omega \rangle$  to  $x$  such that  $(c, \bullet) \in f(x_i)$  for each  $i$  we may derive a livelock for  $x$  at  $c$ , and vice-versa, we prove:

**Proposition 6.**  $\text{fun}_c(\sigma) = \text{fun}(\sigma \cup \sigma^\ddagger)$ .

### 3.3 Convex Sequential Algorithms

As we shall show, the two categories of sequential algorithms give a full characterization of the behaviour of each ESPCF program with respect to may and must testing — hence we may construct a fully abstract model in the category of ocds and pairs consisting of a co-stable function between the join bicpos of states and a stable function between the meet bicpos of finite-branching states.

However, it remains to identify which such pairs may arise as the denotation of a term — there is no program with the denotation  $\{(c, \mathbf{tt})\}$  under may-testing interpretation and the denotation  $\{(c, \mathbf{ff})\}$  under must testing interpretation, for example. To give a may-and-must testing model with a (finite) definability property, we introduce the notion of *convex* sequential algorithm.

**Definition 4.** A convex state of the ocds  $A$  is a pair  $(\bar{x}, \underline{x}) \in D(A) \times D_{fin}(A)$  such that  $\underline{x} \uparrow \bar{x}$  (coherence) and  $A(\underline{x} \cup \bar{x}) = \emptyset$ , (repleteness). Informally, we may explain these conditions: the components of a convex state can only differ with respect to deadlock/livelock behaviour, and at any enabled cell, must either fill it with a value, or fail due to explicit livelock or deadlock.

Let  $D_{cx}(A)$  be the set of convex states of  $A$ .

Convex sequential algorithms from  $A$  to  $B$  are convex states of  $A \Rightarrow B$ . We may compare these to the nondeterministic strategies defined in [7], which are represented as pairs of sets of non-divergent traces, and of divergent (livelocked or deadlocked) traces, respectively. A convex sequential algorithm is represented as a pair of a set of active or deadlocked positions, and a set of active or livelocked positions.

Convex sequential algorithms are composed pointwise: given  $(\bar{\sigma}, \underline{\sigma}) \in D_{cx}(A \Rightarrow B)$  and  $(\bar{\tau}, \underline{\tau}) \in D_{cx}(B \Rightarrow C)$ ,  $\sigma; \tau = ((\text{strat}(\text{fun}(\bar{\tau}) \cdot \text{fun}(\bar{\sigma})), (\text{strat}(\text{fun}_c(\bar{\tau}) \cdot \text{fun}_c(\bar{\sigma})))$ . To establish that that this is a convex sequential algorithm, we observe that composition of sequential algorithms respects the coherence relation (as for Lemma 4), and:

**Lemma 6.** A pair of sequential algorithms  $(\underline{\sigma}, \bar{\sigma}) \in D(A \Rightarrow B) \times D_{fin}(A \Rightarrow B)$  is replete if and only if for all replete pairs of states  $(\bar{x}, \underline{x}) \in D(A) \times D_{fin}(A)$ ,  $(\text{fun}(\bar{\sigma})(\bar{x}), \text{fun}_c(\underline{\sigma})(\underline{x}))$  is replete.

*Proof.* If  $(\underline{\sigma}, \bar{\sigma})$  is replete, then for any enabled cell  $c$  in  $(\text{fun}(\bar{\sigma})(\bar{x}), \text{fun}(\underline{\sigma})(\underline{x}))$ , we may compute a chain of substates  $y_i \subseteq \bar{x} \cup \underline{x}$  cells  $((y_i, c), y_{i+1}) \in \bar{\sigma} \cup \underline{\sigma}$  such that either  $((y_n, c), v) \in \bar{\sigma} \cup \underline{\sigma}$ , where  $v$  fills  $c$ , for some  $n$ , or else  $y_1, \dots, y_n, \dots$  is a livelock in  $\underline{x}$  at  $c$ .

Conversely, for any pair  $(\sigma, \sigma') \in D(A \Rightarrow B) \times D_{fin}(A \Rightarrow B)$ , failure repleteness entails the existence of a specific  $x$  such that  $\text{fun}(\sigma)(x)$  and  $\text{fun}(\sigma')(x)$  are not replete.

So we may form a category  $\mathcal{OS}_{cx}$  in which morphisms from  $A$  to  $B$  are convex states of  $A \Rightarrow B$ , with composition defined pointwise. Cartesian closed structure derives from that of  $\mathcal{OC}, \mathcal{OC}_{fin}$ .  $\mathcal{OS}_{cx}$  is also cpo-enriched: we may show that for any chain  $X$  of states in  $(D_{cx}(A), \subseteq \times \supseteq)$ ,  $(\bigcup \pi_1 X, \bigcap \pi_2 X)$  is a convex state.

## 4 Denotational Semantics of Erratic SPCF

The denotational semantics of erratic SPCF in  $\mathcal{OS}_{cx}$  (and thus by projection, in  $\mathcal{OC}$  and  $\mathcal{OC}_{fin}$ ) is a straightforward extension of the interpretation of PCF in a cpo-enriched Cartesian closed category. Specifically, we have the following:

- Ordered concrete data structures  $\widetilde{\mathbf{B}}$  with which to interpret the basic type  $B$  over the value-set  $\mathbf{B}$ , with associated arithmetic and conditional operations.
- A morphism  $\text{strict?} : (\llbracket T \rrbracket \rightarrow B) \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$  with which to interpret  $\text{strict?} : (T \rightarrow B) \rightarrow \text{bool}$ ,
- A sequential algorithm  $(\{(c, \bullet)\}, \{\}) \in D_{cx}(\llbracket T \rrbracket)$  for each  $T$ , with which to interpret  $\top$ .
- A binary operation (pairwise union) with which to interpret nondeterministic choice.

We refer to e.g. [12] for more details of the semantics of SPCF.

**Proposition 7.** *The interpretation of ESPCF in  $\mathcal{OS}_{cx}$  is sound with respect to  $\lesssim_{\perp}^{\top}$ : if  $\pi_1 \llbracket M \rrbracket \subseteq \llbracket N \rrbracket$  then  $M \lesssim^{\top} N$ , and if  $\pi_2 \llbracket M \rrbracket \supseteq \pi_2 \llbracket N \rrbracket$  then  $M \lesssim_{\perp} N$ .*

*Proof.* We prove that if  $M$  is not a value, or  $\top$ , then  $\pi_i \llbracket M \rrbracket = \bigcup \{\pi_i \llbracket N \rrbracket \mid M \longrightarrow N\}$  for  $i \in \{1, 2\}$ . Computational adequacy, and thus inequational soundness, follows the standard impredicative argument.

Full abstraction with respect to may-and-must testing is shown by establishing that (each type in) the convex model has a basis of elements which are definable (i.e. denotations of terms). We may derive such a basis from the fact that all elements of finite types are ESPCF definable, which has a simple proof based on the claim that every such type is a retract of a product of Boolean types. This is similar to results for SPCF itself — we refer to [14, 12, 13] for further details and discussion.

A term  $M$  of SPCF is *affinely typed* if (i) it contains no instances of the fixpoint combinator (ii) for every subterm of the form  $N N'$ , the free variables of  $N$  and  $N'$  are disjoint. Sharing of variables is permitted between paired subterms, and thus across the conditional. An affinely definable retraction  $T \trianglelefteq U$  is given by a pair of terms  $\text{inj} : T \rightarrow U$  and  $\text{proj} : U \rightarrow T$  which are affinely typable, and such that  $\lambda x. \text{proj}(\text{inj } x)$  denotes the identity.

**Lemma 7.** *There is an affinely definable retraction  $\text{bool} \rightarrow \text{bool} \trianglelefteq \text{bool} \times \text{bool} \times \text{bool} \times \text{bool}$ .*

*Proof.* Each convex sequential algorithm  $f : \llbracket \text{bool} \rrbracket \rightarrow \llbracket \text{bool} \rrbracket$  can be reconstructed from four pieces of information: whether it represents a strict function (under may-testing, and must-testing interpretations), and the values of  $f(\llbracket \mathbf{tt} \rrbracket)$ ,  $f(\llbracket \mathbf{ff} \rrbracket)$  and  $f(\llbracket \mathbf{tt} \rrbracket \cup \llbracket \mathbf{ff} \rrbracket)$ . If  $f$  is strict, then we compute  $f(x)$  by testing  $x$  and returning the relevant value. Otherwise, return  $f(\llbracket \mathbf{tt} \rrbracket)$ .

Hence the following terms define a retraction of  $\mathbf{bool} \rightarrow \mathbf{bool}$  into  $\mathbf{bool}^4$ .

$$\begin{aligned} \mathbf{inj} &= \lambda f. \langle \mathbf{strict?}f, f \mathbf{tt}, f \mathbf{ff}, f(\mathbf{tt} + \mathbf{ff}) \rangle \\ \mathbf{proj} &= \lambda x. \lambda y. \mathbf{If} \langle \pi_1 x, \langle \mathbf{If} \langle y, \langle \mathbf{If} \langle y, \langle \pi_2 x, \pi_4 x \rangle \rangle, \pi_3 x \rangle \rangle, \pi_2 x \rangle \rangle \end{aligned}$$

Simple inductions yield  $\mathbf{bool}^n \rightarrow \mathbf{bool} \sqsubseteq \mathbf{bool}^{4n}$  and thus:

**Proposition 8.** *For every finite type  $T$ , there exists  $n$  such that  $T \sqsubseteq \mathbf{bool}^n$*

Hence universality holds for all finite types. To prove full abstraction, observe that:

**Lemma 8.** *For any type  $T$ , there exists a chain of types  $\langle T_i \mid i \in \omega \rangle$ , and chains of pairs of ESPCF-definable morphisms  $\langle (f_i : [T_i] \rightarrow [T], g_i : [T] \rightarrow [T_i]) \mid i \in \omega \rangle$  such that for all  $e \in [T]$ ,  $e = \bigsqcup_{i \in \omega} f_i(g_i(e))$ .*

*Proof.* Take e.g.  $\mathbf{nat}_i = \mathbf{bool}^i$ , with the evident chain of embeddings/projections.

**Theorem 2.** *The convex model of SPCF is fully abstract with respect to may-and-must testing.*

So by the definability theorem for finite types, every finitary ESPCF term  $M$  is may-and-must equivalent to an affinely-typed term.

## 5 Conclusions and Further Directions

We have described representations of higher-order functionals with bounded non-determinism as non-deterministic sequential algorithms. Several avenues for further research are open.

**Universal Types.** For the sake of simplicity, we have proved full abstraction via retractions of finite types into finite products of the Boolean type only. However, it is possible to show that the type  $\mathbf{nat} \rightarrow \mathbf{bool}$  is *universal* (with respect to affinely typable retractions).

**Powerdomains.** Erratic SPCF lacks a powerdomain construction; modelling such a construction would provide a clearer link to domain theoretic accounts of nondeterminism. It appears that it is possible to do so by moving to a more explicitly game semantic setting, in which strategies satisfy some confluence properties (a restriction on history sensitivity). We may then give a model of intuitionistic linear logic in which the  $!$  represents a powerdomain.

**Unbounded nondeterminism.** The constructions described here extend to yield an equivalence between stable and monotone but non-continuous functions, and sequential algorithms on unbounded ocds, with the significant difference that proofs of cells become possibly infinite *ordinal* chains of events. These may be used to interpret, for example, countable nondeterminism.

**Bidomains as games.** We have yet to describe algebraically the biorders which arise as sets of states of ocds, as was done for bistable biorders and sequential data structures [10]. A possible route is via the relationship of ocds and nondeterministic sequential algorithms to bistructures [5].

## References

1. Berry, G.: Stable models of typed  $\lambda$ -calculi. In: Ausiello, G., Böhm, C. (eds.) ICALP 1978. LNCS, vol. 62, pp. 72–89. Springer, Heidelberg (1978)
2. Berry, G., Curien, P.-L.: Sequential algorithms on concrete data structures. *Theoretical Computer Science* 20, 265–321 (1982)
3. Cartwright, R., Felleisen, M.: Observable sequentiality and full abstraction. In: *Proceedings of POPL 1992* (1992)
4. Cartwright, R., Curien, P.-L., Felleisen, M.: Fully abstract semantics for observably sequential languages. *Information and Computation* (1994)
5. Curien, P.-L., Winskel, G., Plotkin, G.: Bistructures, bidomains and linear logic. In: *Milner Festschrift*. MIT Press, Cambridge (1997)
6. Harmer, R.: Games and Full Abstraction for Nondeterministic Languages. PhD thesis, Imperial College London (1999)
7. Harmer, R., McCusker, G.: A fully abstract games semantics for finite nondeterminism. In: *Proceedings of the Fourteenth Annual Symposium on Logic in Computer Science, LICS 1999*. IEEE Computer Society Press, Los Alamitos (1998)
8. Hyland, M., Schalk, A.: Games on graphs and sequentially realizable functionals. In: *Proceedings of LICS 2002*. IEEE Press, Los Alamitos (2002)
9. Kahn, G., Plotkin, G.: Concrete domains. *Theoretical Computer Science*, Böhm Festschrift special issue (1993); First appeared as technical report 338 of INRIA-LABORIA (1978)
10. Laird, J.: Locally Boolean domains. *Theoretical Computer Science* 342, 132–148 (2005)
11. Laird, J.: Sequentiality in bounded bidomains. *Fundamenta Informaticae* 65, 173–191 (2005)
12. Laird, J.: Bistability: A sequential domain theory. *Logical Methods in Computer Science* 3 (2007)
13. Laird, J.: On the expressiveness of affine programs with non-local control: The elimination of nesting in SPCF. *Fundamenta Informaticae* 77(4), 511–531 (2007)
14. Longley, J.: Universal types and what they are good for. In: *Domain Theory, Logic and Computation: Proceedings of the 2<sup>nd</sup> International Symposium on Domain Theory*. Kluwer Academic Publishers, Dordrecht (2004)



# A Decidable Spatial Logic with Cone-Shaped Cardinal Directions

Angelo Montanari<sup>1</sup>, Gabriele Puppis<sup>2</sup>, and Pietro Sala<sup>1</sup>

<sup>1</sup> Department of Mathematics and Computer Science, Udine University, Italy

[angelo.montanari,pietro.sala}@dimi.uniud.it](mailto:{angelo.montanari,pietro.sala}@dimi.uniud.it)

<sup>2</sup> Computing Laboratory, Oxford University, England

[gabriele.puppis@comlab.ox.ac.uk](mailto:gabriele.puppis@comlab.ox.ac.uk)

**Abstract.** We introduce a spatial modal logic based on cone-shaped cardinal directions over the rational plane and we prove that, unlike projection-based ones, such as, for instance, Compass Logic, its satisfiability problem is decidable (PSPACE-complete). We also show that it is expressive enough to subsume meaningful interval temporal logics, thus generalizing previous results in the literature, e.g., its decidability implies that of the subinterval/superinterval temporal logic interpreted over the rational line.

## 1 Introduction

Spatial reasoning has both a strong theoretical relevance and applications in many areas of computer science, including robotics, natural language processing, geographical information systems [1, 5, 12]. However, despite the widespread interest in the topic, few techniques have been developed to automatically (and efficiently) reason about spatial relations over infinite structures. As a matter of fact, spatial reasoning has been mainly investigated in quite restricted algebraic settings.

In this paper, we introduce a novel spatial modal logic, called *Cone Logic*, which allows one to reason about cone-shaped directional relations between points in the rational plane. While the satisfiability problem for spatial modal logics with projection modalities turns out to be highly undecidable [7, 9], we prove that Cone Logic enjoys a decidable satisfiability problem (in fact, PSPACE-complete) by taking advantage of a suitable filtration technique. We also show that Cone Logic subsumes interesting interval temporal logics such as the temporal logic of subintervals/superintervals, thus generalizing previous results in the literature [3] and basically disproving a conjecture by Lodaya [6].

## 2 Syntax and Semantics of Cone Logic

In this section, we introduce syntax and semantics of Cone Logic. Let  $\mathbb{P} = \mathbb{Q} \times \mathbb{Q}$  denote the *rational plane* and let  $p = (x, y)$  be one of its points. We denote by



**Fig. 1.** The four quadrants and the cone-shaped cardinal directions

$LL(p)$ ,  $LR(p)$ ,  $UL(p)$ , and  $UR(p)$  the open *lower-left*, *lower-right*, *upper-left*, and *upper-right quadrants* of  $p$ , respectively, which are defined as follows:

$$\begin{aligned}
 LL(p) &= \{(x', y') : x' < x, y' < y\} & LR(p) &= \{(x', y') : x' > x, y' < y\} \\
 UL(p) &= \{(x', y') : x' < x, y' > y\} & UR(p) &= \{(x', y') : x' > x, y' > y\}.
 \end{aligned}$$

Note that, up to a rotation of the axes, these open quadrants can be viewed as the Frank’s cone-shaped cardinal directions ‘North’, ‘West’, ‘East’, ‘South’ [4] (see Figure 1). Similarly, one can denote by  $LL^+(p)$ ,  $LR^+(p)$ ,  $UL^+(p)$ , and  $UR^+(p)$  the semi-closed quadrants of  $p$ , which are defined in the natural way, e.g.,  $LL^+(p) = \{(x', y') : x' \leq x, y' \leq y\} \setminus \{p\}$ .

Given a set  $\text{Prop}$  of propositional variables, formulas of Cone Logic are built up from  $\text{Prop}$  using the boolean connectives  $\neg$  and  $\vee$  and eight modal operators  $\blacklozenge$ ,  $\blacktriangleright$ ,  $\blacktriangleleft$ ,  $\blacktriangle$ ,  $\blacklozenge^+$ ,  $\blacktriangleright^+$ ,  $\blacktriangleleft^+$ , and  $\blacktriangle^+$ . The *size*  $|\varphi|$  of a formula  $\varphi$  is given by the number of its subformulas (for instance,  $\blacklozenge a \vee \neg \blacklozenge \neg b$  is a formula of size 7). Formulas of Cone Logic are evaluated over (labeled regions of) the rational plane. Precisely, let  $\mathcal{P} = (P, \sigma)$  be a *labeled region*, where  $P \subseteq \mathbb{P}$  is a non-empty subset of the rational plane and  $\sigma : P \rightarrow \mathcal{P}(\text{Prop})$  is a *labeling function*. We define the semantics of a formula with respect to a distinguished initial point  $p \in P$  as follows:

- $\mathcal{P}, p \models a$  iff  $a \in \sigma(p)$ ,
- $\mathcal{P}, p \models \neg \varphi$  iff  $\mathcal{P}, p \not\models \varphi$ ,
- $\mathcal{P}, p \models \varphi_1 \vee \varphi_2$  iff  $\mathcal{P}, p \models \varphi_1$  or  $\mathcal{P}, p \models \varphi_2$ ,
- $\mathcal{P}, p \models \blacklozenge \varphi$  (resp.,  $\mathcal{P}, p \models \blacklozenge^+ \varphi$ ) iff  $P$  contains a point  $q$  such that  $q \in LL(p)$  (resp.,  $q \in LL^+(p)$ ) and  $\mathcal{P}, q \models \varphi$  (and similarly for the other modal operators  $\blacktriangleright$ ,  $\blacktriangleleft$ ,  $\blacktriangle$ ,  $\blacklozenge^+$ ,  $\blacktriangleright^+$ , and  $\blacktriangleleft^+$ ).

We further use shorthands such as  $\varphi_1 \wedge \varphi_2 = \neg(\neg \varphi_1 \vee \neg \varphi_2)$ ,  $\blacksquare \varphi = \neg \blacklozenge \neg \varphi$ ,  $\blacklozenge \varphi = \blacklozenge \blacklozenge \varphi$ ,  $\blacksquare \varphi = \blacksquare \blacksquare \varphi$ ,  $\blacktriangleright \varphi = \blacktriangleright \blacktriangleright \varphi$ ,  $\blacktriangleleft \varphi = \blacktriangleleft \blacktriangleleft \varphi$ , etc.

Cone Logic is well-suited for expressing spatial relationships between points, curves, and regions inside the rational plane. Below, we give an intuitive account of its expressiveness through a couple of examples. To begin with, we show how to define an  $a$ -labeled open rectangular region, whose edges are aligned with the  $x$ - and  $y$ -axes, by means of a Cone Logic formula:

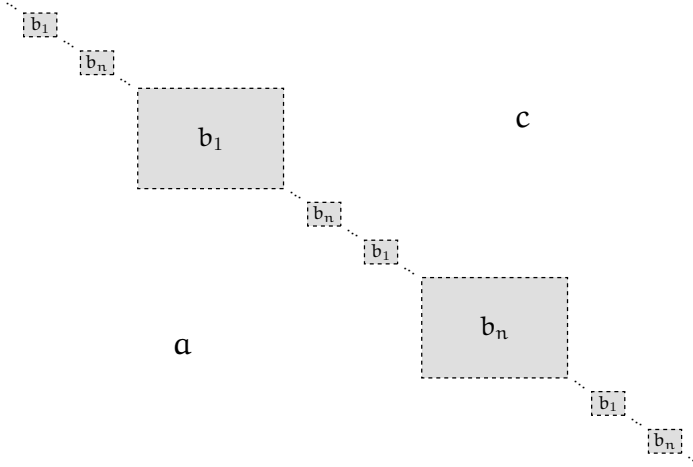


Fig. 2. A labeled rational plane satisfying  $\varphi_{<}$

$$\begin{aligned} \varphi = & \blacklozenge a \wedge \blacklozenge b \wedge \blacklozenge c \wedge \blacklozenge d \wedge \blacklozenge e \\ & \wedge \blacksquare (a \rightarrow \blacklozenge a \wedge \blacklozenge a \wedge \blacklozenge a \wedge \blacklozenge a) \wedge \blacksquare (\neg a \leftrightarrow b \vee c \vee d \vee e) \\ & \wedge \blacksquare (b \rightarrow \blacksquare b) \wedge \blacksquare (c \rightarrow \blacksquare c) \wedge \blacksquare (d \rightarrow \blacksquare d) \wedge \blacksquare (e \rightarrow \blacksquare e). \end{aligned}$$

The second example uses two symmetric modal operators, namely,  $\blacklozenge$  and  $\blacklozenge$ , to enforce non-trivial spatial relationships between labeled regions of the rational plane. Let  $\mathbf{Prop}$  be a signature containing  $n + 2$  propositional variables  $a, b_1, \dots, b_n, c$  and let  $<$  be the partial order over  $\mathbf{Prop}$  such that  $a < b_i < c$ , for every  $1 \leq i \leq n$ , and  $b_i \not< b_j$ , for every pair of distinct indices  $1 \leq i, j \leq n$ . We shortly write  $p \leq q$  (resp.,  $p \geq q$ ) whenever  $p = q$  or  $p < q$  (resp.,  $p > q$ ). Consider now the (Hintikka-like) formula induced by the partial order  $<$ :

$$\begin{aligned} \varphi_{<} = & \blacksquare \bigvee_{p \in \mathbf{Prop}} p \wedge \blacksquare \bigwedge_{p \neq q} \neg(p \wedge q) \\ & \wedge \blacksquare \bigwedge_{p \in \mathbf{Prop}} \left( p \rightarrow \bigwedge_{q \leq p} \blacklozenge q \wedge \bigwedge_{q \geq p} \blacklozenge q \wedge \blacksquare \bigvee_{q \leq p} q \wedge \blacksquare \bigvee_{q \geq p} q \right). \end{aligned}$$

The unique (up to homeomorphism) labeled rational plane that satisfies  $\varphi_{<}$  is depicted in Figure 2. Notice that (i) for every propositional variable  $b_i$ , with  $1 \leq i \leq n$ , the  $b_i$ -labeled region is an (infinite) union of disjoint open rectangles (in fact, the coordinates of their corners are given by pairs of irrational numbers) and (ii) the  $b_i$ -labeled open rectangles are arranged densely in the rational plane, that is, for all indices  $1 \leq i, j, k \leq n$ , with  $i \neq j$ , all  $b_i$ -labeled points  $(x_1, y_1)$ , and all  $b_j$ -labeled points  $(x_2, y_2)$ , with  $x_1 < x_2$  and  $y_1 > y_2$ , there is a  $b_k$ -labeled point  $(x, y)$  such that  $x_1 < x < x_2$  and  $y_1 > y > y_2$ .

The *satisfiability problem* for Cone Logic consists of deciding whether a given formula  $\varphi$  holds at some point of a labeled region of the rational plane. In particular, we are interested in satisfiability problems under interpretation over (open or closed) rectangular regions, namely, regions of the form  $I \times J$ , with  $I$  and  $J$

being two fixed (open or closed) intervals of the rational line<sup>1</sup>. As a matter of fact, note that the whole rational plane  $\mathbb{P}$  is homeomorphic to any open rectangular region of the form  $I \times J$ , with  $I = (x_0, x_1)$  and  $J = (y_0, y_1)$ . Moreover, any formula  $\varphi$  interpreted over an open rectangular region of the form  $I \times \mathbb{Q}$ , with  $I = (x_0, x_1)$ , can be rewritten into an equi-satisfiable formula  $\varphi'$  interpreted over the semi-closed rectangular region  $I' \times \mathbb{Q}$ , where  $I' = [x_0, x_1]$ . Taking advantage of the reducibility of the satisfiability problem over open rectangular regions to that over semi-closed rectangular regions, we can restrict our attention to labeled regions of the form  $\mathcal{P} = (I \times \mathbb{Q}, \sigma)$ , where  $I$  is a closed (non-singleton) interval (hereafter, we call such structures *labeled stripes*).

The relationships between Cone Logic and spatial logics with projection modalities deserve a little discussion. Projection-based spatial logics (most notably, Compass Logic [13]) are two-dimensional modal logics whose accessibility relations allow one to move along one of the two coordinates while keeping the other coordinate constant. On the one hand, Cone Logic inherits from projection-based modal logics some of their desirable features. For instance, it allows one to write suitable formulas that constrain labels to occur along some distinguished axes, e.g., the formula  $\blacklozenge a \wedge \blacksquare \neg a \wedge \blacksquare \neg a \wedge \blacksquare \neg a$  forces  $a$  to hold at the origin or at some point over the positive  $x$ -axis. On the other hand, unlike projection-based modal logics, only a *bounded* number of constraints ‘along the axes’ can be enforced by Cone Logic. We will see that such a limitation can be traded for a positive decidability result.

Hereafter, for the sake of simplicity, we constrain Cone Logic formulas to quantify over open quadrants only, that is, to make use of the modal operators  $\blacklozenge$ ,  $\blacktriangleright$ ,  $\blacktriangleleft$ , and  $\blacklozenge$  only. However, the achieved results (in particular, the tree pseudo-model property proved in Section 4 and the PSPACE decision procedure described in Section 5) can be naturally generalized to the case of unrestricted Cone Logic formulas.

### 3 Basic Machinery: Types, Dependencies, and Shadings

Let us fix a labeled region  $\mathcal{P} = (P, \sigma)$  and a formula  $\varphi$  of Cone Logic. In the sequel, we compare points in  $\mathcal{P}$  with respect to the set of subformulas of  $\varphi$  they satisfy. To do that, we introduce the key notions of  $\varphi$ -atom,  $\varphi$ -type,  $\varphi$ -cluster, and  $\varphi$ -shading.

First of all, we denote by  $\mathcal{Cl}(\varphi)$  the set of all subformulas of  $\varphi$  and of their negations (we identify  $\neg \neg \alpha$  with  $\alpha$ ,  $\neg \blacklozenge \alpha$  with  $\blacksquare \neg \alpha$ , etc.). A  $\varphi$ -atom is any non-empty set  $A \subseteq \mathcal{Cl}(\varphi)$  such that (i) for every formula  $\alpha \in \mathcal{Cl}(\varphi)$ ,  $\alpha \in A$  iff  $\neg \alpha \notin A$  and (ii) for every formula  $\gamma = \alpha \vee \beta \in \mathcal{Cl}(\varphi)$ ,  $\gamma \in A$  iff  $\alpha \in A$  or  $\beta \in A$  (intuitively, a  $\varphi$ -atom is a maximal *locally consistent* set of subformulas of  $\varphi$ ). The cardinality of  $\mathcal{Cl}(\varphi)$  is *linear* in  $|\varphi|$ , while the number of  $\varphi$ -atoms is (at most) *exponential* in  $|\varphi|$ . We then associate with each point  $p \in P$  the set of

<sup>1</sup> Hereafter, square brackets are used to denote closed intervals, e.g.,  $[0, 1]$ , while round brackets are used to denote open intervals, e.g.,  $(0, 1)$ . Semi-open intervals are represented by mixing the two notations, e.g.,  $[0, 1)$ .

all formulas  $\alpha \in \mathcal{Cl}(\varphi)$  such that  $\mathcal{P}, p \models \alpha$ . Such a set is called  $\varphi$ -*type* of  $p$  and it is denoted by  $\mathcal{T}ype_{\mathcal{P}}(p)$ . We have that every  $\varphi$ -type is a  $\varphi$ -atom, but not vice versa.

Given an atom  $A$ , we denote by  $\mathcal{R}eq_{LL}(A)$  (resp.,  $\mathcal{R}eq_{LR}(A)$ ,  $\mathcal{R}eq_{UL}(A)$ ,  $\mathcal{R}eq_{UR}(A)$ ) the set of all formulas  $\alpha \in \mathcal{Cl}(\varphi)$  such that  $\blacktriangleleft \alpha \in A$  (resp.,  $\blacktriangleright \alpha \in A$ ,  $\blacktriangleleft \alpha \in A$ ,  $\blacktriangleright \alpha \in A$ ); similarly, we denote by  $\mathcal{O}bs_{LL}(A)$  (resp.,  $\mathcal{O}bs_{LR}(A)$ ,  $\mathcal{O}bs_{UL}(A)$ ,  $\mathcal{O}bs_{UR}(A)$ ) the set of all formulas  $\alpha$  such that  $\alpha \in A$  and  $\blacktriangleleft \alpha \in \mathcal{Cl}(\varphi)$  (resp.,  $\blacktriangleright \alpha \in \mathcal{Cl}(\varphi)$ ,  $\blacktriangleleft \alpha \in \mathcal{Cl}(\varphi)$ ,  $\blacktriangleright \alpha \in \mathcal{Cl}(\varphi)$ ). We call formulas belonging to one of the first (resp., last) four sets *requests* (resp., *observables*). Taking advantage of these sets, for each direction  $D \in \{LL, LR, UL, UR\}$ , we define two *transitive relations*  $\xrightarrow{D}$  and  $\xrightarrow{\bar{D}}$  between atoms as follows:

$$\begin{aligned}
 A \xrightarrow{D} A' & \quad \text{iff} \quad \begin{cases} \mathcal{R}eq_D(A) \supseteq \mathcal{R}eq_D(A') \cup \mathcal{O}bs_D(A') \\ \mathcal{R}eq_{\bar{D}}(A') \supseteq \mathcal{R}eq_{\bar{D}}(A) \cup \mathcal{O}bs_{\bar{D}}(A) \end{cases} \\
 A \xrightarrow{\bar{D}} A' & \quad \text{iff} \quad \begin{cases} \mathcal{R}eq_D(A) \supseteq \mathcal{R}eq_D(A') \\ \mathcal{R}eq_{\bar{D}}(A') \supseteq \mathcal{R}eq_{\bar{D}}(A) \end{cases}
 \end{aligned}$$

where  $\bar{D}$  denotes the direction opposite to  $D$  (e.g.,  $\overline{LL} = UR$ ). Note that  $A \xrightarrow{D} A'$  (resp.,  $A \xrightarrow{\bar{D}} A'$ ) iff  $A' \xrightarrow{\bar{D}} A$  (resp.,  $A' \xrightarrow{D} A$ ). Moreover, the relations  $\xrightarrow{D}$  and  $\xrightarrow{\bar{D}}$  satisfy the *view-to-type dependency* property, namely, for every pair of points  $p, q$  in  $\mathcal{P}$  and every direction  $D \in \{LL, LR, UL, UR\}$ ,

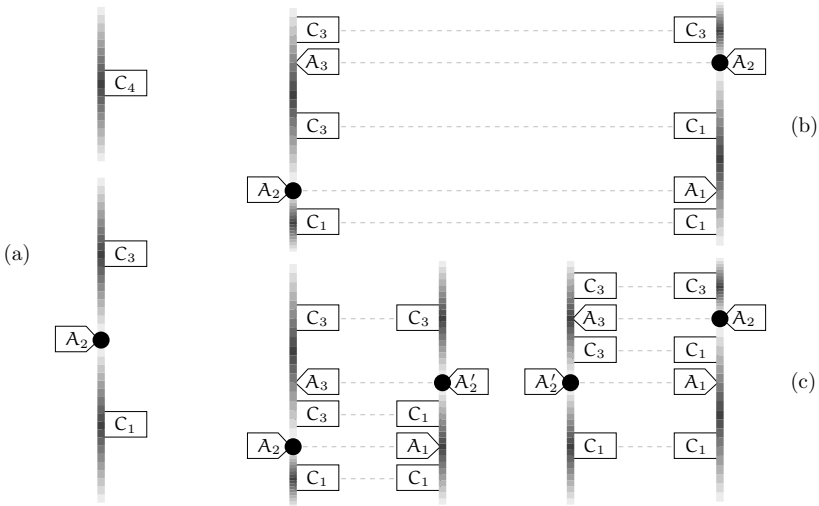
$$\begin{aligned}
 q \in D(p) & \quad \text{implies} \quad \mathcal{T}ype_{\mathcal{P}}(p) \xrightarrow{D} \mathcal{T}ype_{\mathcal{P}}(q) \\
 D(q) \subseteq D(p) & \quad \text{implies} \quad \mathcal{T}ype_{\mathcal{P}}(p) \xrightarrow{\bar{D}} \mathcal{T}ype_{\mathcal{P}}(q).
 \end{aligned}$$

Below, we introduce analogous notions for regions. First, we define a  $\varphi$ -*cluster* as any non-empty set  $C$  of atoms. Then, for a cluster  $C$  and a direction  $D \in \{LL, LR, UL, UR\}$ , we denote by  $\mathcal{R}eq_D(C)$  and  $\mathcal{O}bs_D(C)$ , respectively, the set  $\bigcup_{A \in C} \mathcal{R}eq_D(A)$  and the set  $\bigcup_{A \in C} \mathcal{O}bs_D(A)$ . Moreover, given a pair of clusters  $C, C'$ , we write  $C \xrightarrow{D} C'$  (resp.,  $C \xrightarrow{\bar{D}} C'$ ) whenever  $A \xrightarrow{D} A'$  (resp.,  $A \xrightarrow{\bar{D}} A'$ ) holds for all pairs of atoms  $A \in C$  and  $A' \in C'$ . Finally, we associate with each non-empty region  $P'$  of  $\mathcal{P}$  its  $\varphi$ -*shading*, which is defined as the set  $\mathcal{T}ype_{\mathcal{P}}(P') = \{\mathcal{T}ype_{\mathcal{P}}(p) : p \in P'\}$  of the  $\varphi$ -types of all points of  $P'$ .

Note that, for every labeled region  $\mathcal{P} = (P, \sigma)$ , the formula  $\varphi$  holds at some point  $p$  of  $\mathcal{P}$  iff the shading  $\mathcal{T}ype_{\mathcal{P}}(P)$  contains an atom  $A$  such that  $\varphi \in A$ . Hereafter, we shall omit the argument  $\varphi$ , thus calling a  $\varphi$ -atom (resp., a  $\varphi$ -type, a  $\varphi$ -cluster, etc.) simply an atom (resp., a type, a cluster, etc.).

## 4 From the Rational Plane to the Infinite Binary Tree

In this section, we aim at establishing a tree (pseudo-)model property for satisfiable formulas of Cone Logic. This is done by introducing a suitable notion of decomposition of a labeled region (more precisely, of a labeled stripe) and by iteratively applying it in order to obtain an infinite decomposition tree structure that correctly represents the original model.



**Fig. 3.** Shading sequences (a), stripe expressions (b), and decompositions (c)

### 4.1 Shading Sequences and Stripe Expressions

To start with, we consider the shadings of the vertical straight lines inside a labeled rational plane. A *shading sequence* is a non-empty finite sequence  $S$  of atoms and clusters such that, for every  $1 \leq i \leq |S|$ , if  $S(i)$  is an atom, then  $1 < i < |S|$  and both  $S(i - 1)$  and  $S(i + 1)$  are clusters. Shading sequences allow one to represent the types that appear along some vertical straight lines of a labeled rational plane. As an example, Figure 3(a) depicts a labeled vertical line with an associated shading sequence  $S = C_1 A_2 C_3 C_4$ .

To represent the shadings of the two vertical borders of a labeled stripe, we introduce the notion of *stripe expression*, which is a pair  $E = (L, R)$  of (*left* and *right*) shading sequences having equal length ( $|L| = |R|$ ) and such that, for every  $1 \leq i \leq |E|$  ( $= |L| = |R|$ ),  $L(i)$  is an atom (resp., a cluster) iff  $R(i)$  is an atom (resp., a cluster). We call any pair of the form  $(L(i), R(i))$ , with  $1 \leq i \leq |E|$ , a *matched pair*. As an example, Figure 3(b) depicts the left border and the right border of a labeled stripe, together with the associated stripe expression  $E = (L, R)$ , where  $L = C_1 A_2 C_3 A_3 C_3$  and  $R = C_1 A_1 C_1 A_2 C_3$ . We say that an atom  $A$  is *featured* by the left (resp., right) sequence of a stripe expression  $E = (L, R)$  if there is an index  $1 \leq i \leq |E|$  such that  $A = L(i)$  (resp.,  $A = R(i)$ ) or  $A \in L(i)$  (resp.,  $A \in R(i)$ ), depending on whether  $L(i)$  (resp.,  $R(i)$ ) is an atom or a cluster. By a slight abuse of notation, we denote by  $\bigcup_{1 \leq i \leq |E|} L(i)$  (resp.,  $\bigcup_{1 \leq i \leq |E|} R(i)$ ) the set of all atoms featured by the left (resp., right) sequence of the stripe expression  $E = (L, R)$ .

For every labeled stripe  $\mathcal{P}$ , there is a stripe expression  $E$  whose left (resp., right) sequence features all and only the types of the points along the left (resp., right) border of  $\mathcal{P}$ . However, for a given stripe expression  $E$ , there might exist no labeled

stripe  $\mathcal{P}$  such that the shading of the left (resp., right) border of  $\mathcal{P}$  coincides with the set of atoms featured by the left (resp., right) shading sequences of  $E$ . In the following, we show how to get rid of such a problem. As a first step, we enforce suitable consistency conditions on any stripe expression  $E = (L, R)$ :

- (C1) for every  $1 \leq i < j \leq |E|$ ,  $L(i) \xrightarrow{-D} L(j)$  and  $R(i) \xrightarrow{-D} R(j)$  hold for both  $D = UL$  and  $D = UR$ ;
- (C2) for every  $1 \leq i \leq |E|$ , if  $L(i)$  and  $R(i)$  are clusters, then  $L(i) \xrightarrow{-D} L(i)$  and  $R(i) \xrightarrow{-D} R(i)$  hold for both  $D = UL$  and  $D = UR$ ;
- (C3) for every  $1 \leq i \leq |E|$ ,  $L(i) \xrightarrow{-D} R(i)$  (and hence  $R(i) \xrightarrow{-D} L(i)$ ) holds for both  $D = LR$  and  $D = UR$ ;
- (C4) for every  $1 \leq i \leq |E|$ , if  $L(i)$  and  $R(i)$  are atoms (resp., clusters), then  $L(i) \xrightarrow{LR} R(j)$  and  $L(i) \xrightarrow{UR} R(k)$  (and hence  $R(j) \xrightarrow{UL} L(i)$  and  $R(k) \xrightarrow{UL} L(i)$ ) hold for all  $1 \leq j < i$  (resp.,  $1 \leq j \leq i$ ) and all  $i < k \leq |L|$  (resp.,  $i \leq k \leq |L|$ ).

We compare stripe expressions with respect to their generality by introducing a suitable partial order  $\leq$ . Given two stripe expressions  $E = (L, R)$  and  $E' = (L', R')$ , we write  $E \leq E'$  if  $|E| = |E'|$  and, for every index  $1 \leq i \leq |E|$ , we have either  $L(i) = L'(i)$  and  $R(i) = R'(i)$ , or  $L(i) \subseteq L'(i)$  and  $R(i) \subseteq R'(i)$ , depending on whether  $L(i)$ ,  $R(i)$ ,  $L'(i)$ ,  $R'(i)$  are atoms or clusters. Unless otherwise stated, we tacitly assume that a stripe expression is maximal with respect to the above-defined partial order  $\leq$ . Note that a cluster appearing in a (maximal) stripe expression may contain an exponential number of distinct atoms; however, thanks to consistency conditions, the set of all its atoms can be characterized in terms of the sets of its requests and observables, namely, for every (maximal) stripe expression  $E = (L, R)$ , every index  $1 \leq i \leq |E|$ , and every atom  $A$ , we have that  $A$  belongs to the cluster  $C = L(i)$  (resp.,  $C = R(i)$ ) if and only if  $\mathcal{Req}_D(A) = \mathcal{Req}_D(C)$  and  $\mathcal{Obs}_D(A) \subseteq \mathcal{Obs}_D(C)$  hold for all directions  $D \in \{LL, LR, UL, UR\}$ . This allows us to succinctly represent the two clusters of a matched pair of a (maximal) stripe expression by the sets of their requests and observables, whose size is linear in  $|\varphi|$ . In addition, we can assume every (maximal) stripe expression  $E = (L, R)$  to contain pairwise distinct matched pairs  $(L(i), R(i))$ . From the above, it follows that the length  $|E|$  of any (maximal) stripe expression  $E = (L, R)$  is at most  $4 \cdot |\varphi|$ . At worst, for every pair of distinct indices  $1 \leq i < j \leq |E|$ , if  $L(i)$ ,  $R(i)$ ,  $L(j)$ , and  $R(j)$  are clusters, then, for both  $D = UR$  and  $D = UL$ , we have  $\mathcal{Req}_D(L(j)) \subseteq \mathcal{Req}_D(L(i))$ ,  $\mathcal{Req}_D(R(j)) \subseteq \mathcal{Req}_D(R(i))$ , and either  $\mathcal{Req}_D(L(j)) \subsetneq \mathcal{Req}_D(L(i))$  or  $\mathcal{Req}_D(R(j)) \subsetneq \mathcal{Req}_D(R(i))$ , and in both shading sequences there exist an atom between any pair of consecutive clusters. Hence, every (maximal) stripe expression can be represented using *polynomial space* with respect to  $|\varphi|$ .

## 4.2 Recursive Decompositions of Stripes

Roughly speaking, conditions [C1](#), [C4](#) above provide us with a guarantee that the natural spatial interpretation of a stripe expression  $E$  is *consistent* with view-to-type dependencies. To enforce the *fulfillment* of the existential requests of the

atoms featured by the two shading sequences of  $E$ , we further need to introduce a suitable notion of decomposition. We start by dividing a given labeled stripe into a pair of (thinner) adjacent labeled sub-stripes and then we recursively apply the decomposition to every emerging sub-stripe. This yields an infinite tree-shaped decomposition of the initial structure, where each vertex of the tree represents a labeled (sub-)stripe (and, thus, it is associated with a stripe expression) and each edge represents a containment relationship between two labeled (sub-)stripes.

To start with, we introduce a suitable equivalence relation between shading sequences. Two shading sequences  $S$  and  $S'$  are said to be *equivalent* if

- i) every cluster  $S(i)$  of  $S$  (resp.,  $S'(i')$  of  $S'$ ) is also a cluster of  $S'$  (resp.,  $S$ );
- ii) every atom  $S(i)$  of  $S$  (resp.,  $S'(i')$  of  $S'$ ) either is an atom of  $S'$  (resp.,  $S$ ) or it belongs to the two adjacent clusters  $S(i - 1) = S(i + 1)$  in  $S$  (resp.,  $S'(i' - 1) = S'(i' + 1)$  in  $S'$ ).

As an example, the shading sequences  $S = C_1 A_1 C_1 C_2$  and  $S' = C_1 C_2 C_2$  are equivalent, provided that  $A_1 \in C_1$ , while the shading sequences  $S = C_1 A_1 C_2 C_2$  and  $S' = C_1 C_2 C_2$  are not equivalent (unless  $A_1 \in C_1$  and  $C_1 = C_2$ ).

Decompositions of stripe expressions are defined as follows. Let  $E = (L, R)$  be a stripe expression. A *decomposition* of  $E$  is any pair of stripe expressions  $(E_1, E_2)$ , with  $E_1 = (L_1, R_1)$  and  $E_2 = (L_2, R_2)$ , such that the following *matching* conditions hold:

- (M1)  $L_1$  and  $L$  are equivalent;
- (M2)  $R_2$  and  $R$  are equivalent;
- (M3)  $R_1$  and  $L_2$  are equivalent.

We say that a matched pair  $(L(i), R(i))$  of the stripe expression  $E$  *corresponds* to a matched pair  $(L_1(i_1), R_1(i_1))$  (resp.,  $(L_2(i_2), R_2(i_2))$ ) of the stripe expression  $E_1$  (resp.,  $E_2$ ) under the decomposition  $(E_1, E_2)$  of  $E$  if there exists an index  $1 \leq i_2 \leq |E_2|$  (resp.,  $1 \leq i_1 \leq |E_1|$ ) such that (i)  $L(i) \stackrel{\subseteq}{=} L_1(i_1)$ , (ii)  $R(i) \stackrel{\subseteq}{=} R_2(i_2)$ , and (iii)  $R_1(i_1) \stackrel{\subseteq}{=} L_2(i_2)$ , where  $\stackrel{\subseteq}{=}$  denotes either the equality relation  $=$ , the membership relation  $\in$ , or the containment relation  $\ni$  depending on the form of its left and right arguments (namely, whether they are atoms or clusters). As an example, Figure 3(c) depicts a decomposition of the stripe expression  $E = (L, R)$ , where  $L = C_1 A_2 C_3 A_3 C_3$  and  $R = C_1 A_1 C_1 A_2 C_3$ . Note that, under such a decomposition, the matched pair  $(C_3, C_1)$  of  $E$  corresponds to the matched pairs  $(C_3, C_1)$ ,  $(A_3, A'_2)$ ,  $(C_3, C_3)$  of  $E_1$  and to the matched pairs  $(C_1, C_1)$ ,  $(A'_2, A_1)$ ,  $(C_3, C_1)$  of  $E_2$ . By iteratively applying decompositions, starting from a given stripe expression, one obtains an infinite tree-shaped structure, called decomposition tree.

**Definition 1.** A decomposition tree is an infinite complete binary labeled tree  $\mathcal{T} = (V, E, \downarrow_1, \downarrow_2)$ , where

- $V$  is the set of tree vertices;
- $\downarrow_1$  and  $\downarrow_2$  are the two successor relations;
- $E$  is a labeling function associating a stripe expression  $E(v)$  with each  $v \in V$  such that the pair  $(E(\downarrow_1(v)), E(\downarrow_2(v)))$  is a decomposition of  $E(v)$ .



Note that, for every pair of vertices  $v$  and  $v'$  at the same level of a decomposition tree  $\mathcal{T} = (V, E, \downarrow_1, \downarrow_2)$ , if  $v'$  is right-adjacent to  $v$  (even without being its sibling) and  $E(v) = (L_v, R_v)$  and  $E(v') = (L_{v'}, R_{v'})$  are the associated stripe expressions, then the sequence  $R_v$  turns out to be equivalent to the sequence  $L_{v'}$ .

Let  $\mathcal{T} = (V, E, \downarrow_1, \downarrow_2)$  be a decomposition tree. We impose suitable conditions on  $\mathcal{T}$  which guarantee that every existential request of every atom featured by a stripe expression  $E(v)$  is eventually fulfilled by an observable of an atom featured by a (possibly different) stripe expression  $E(v')$ . Given a stripe expression  $E(v) = (L, R)$ , let us denote by  $E(v)[L]$  (resp.,  $E(v)[R]$ ) its left shading sequence  $L$  (resp., right shading sequence  $R$ ). In the following, we consider a generic vertex  $v$  of  $\mathcal{T}$  and we look at the *right-oriented* (i.e., LR- and UR-oriented) requests of the atoms featured by  $E(v)[L]$ ; symmetrically, we look at the *left-oriented* (i.e., LL- and UL-oriented) requests of the atoms featured by  $E(v)[R]$ .

Let us consider the UR-requests of a left shading sequence  $E(v)[L]$ . Given a vertex  $v$  of  $\mathcal{T}$ , an index  $1 \leq i \leq |E(v)|$ , and a subformula  $\alpha \in \mathcal{R}eq_{UR}(E(v)[L](i))$ , we say that the UR-request  $\alpha$  is

- (F1) *postponed at position  $i$  of vertex  $v$* , if we have  $\alpha \in \mathcal{R}eq_{UR}(E(v)[R](i))$ ;
- (F2) *fulfilled at position  $i$  of vertex  $v$* , if we have  $\alpha \in \mathcal{O}bs_{UR}(E(v)[R](j))$  for some index  $i \leq j \leq |E(v)|$ ;
- (F3) *partially fulfilled at position  $i$  of vertex  $v$* , if there is an index  $1 \leq i_1 \leq |E(\downarrow_1(v))|$  such that (i) the UR-request  $\alpha$  is fulfilled at position  $i_1$  of vertex  $\downarrow_1(v)$  and (ii) the matched pair  $(E(v)[L](i), E(v)[R](i))$  of  $E(v)$  corresponds to the matched pair  $(E(\downarrow_1(v))[L](i_1), E(\downarrow_1(v))[R](i_1))$  of  $E(\downarrow_1(v))$  under the decomposition  $(E(\downarrow_1(v)), E(\downarrow_2(v)))$  of  $E(v)$ .

Similar definitions can be given for the LR-requests of a left shading sequence  $E(v)[L]$  and for the UL-/LL-requests of a right shading sequence  $E(v)[R]$ .

We say that a decomposition tree  $\mathcal{T}$  is *globally fulfilled* if, for every vertex  $v$ , every index  $1 \leq i \leq |E(v)|$ , and every direction  $D \in \{LR, UR\}$  (resp.,  $D \in \{UL, LL\}$ ), the following conditions hold:

- (G1) if  $v$  is the root, then  $\mathcal{R}eq_D(E(v)[R](i)) = \emptyset$  (resp.,  $\mathcal{R}eq_D(E(v)[L](i)) = \emptyset$ );
- (G2) for every subformula  $\alpha \in \mathcal{R}eq_D(E(v)[L](i))$  (resp.,  $\mathcal{R}eq_D(E(v)[R](i))$ ) and every infinite path  $\pi$  that starts at  $v$ , there is a vertex  $v'$  in  $\pi$  (possibly  $v' = v$ ) such that either  $\alpha \notin \mathcal{R}eq_D(E(v')[L](i'))$  (resp.,  $\alpha \notin \mathcal{R}eq_D(E(v')[R](i'))$ ) for all positions  $i'$  of vertex  $v'$  or  $\alpha$  is postponed (E1), fulfilled (E2), or partially fulfilled (E3) at some position  $i'$  of vertex  $v'$ .

We are now ready to establish a tree (pseudo-)model property for satisfiable formulas of Cone Logic. The next theorem states that (i) given a globally fulfilled decomposition tree  $\mathcal{T}$ , there is a labeled stripe  $\mathcal{P} = (I \times \mathbb{Q}, \sigma)$  whose shading coincides with the set of all atoms that are featured by the expressions of  $\mathcal{T}$  (soundness) and (ii) given a labeled stripe  $\mathcal{P} = (I \times \mathbb{Q}, \sigma)$ , there is a globally fulfilled decomposition tree  $\mathcal{T}$  whose expressions feature (at least) the types of all points of  $\mathcal{P}$  (completeness). The proof is omitted for the lack of space (it will be included in the extended version of the paper).

**Theorem 1.** *Soundness. For every globally fulfilled decomposition tree  $\mathcal{T} = (\mathbb{V}, \mathbb{E}, \downarrow_1, \downarrow_2)$ , there is a labeled stripe  $\mathcal{P} = (\mathbb{I} \times \mathbb{Q}, \sigma)$  such that*

$$\text{Type}_{\mathcal{P}}(\mathbb{I} \times \mathbb{Q}) = \bigcup_{\substack{v \in \mathbb{V} \\ 1 \leq i \leq |\mathbb{E}(v)|}} \left( \mathbb{E}(v)[\mathbb{L}](i) \cup \mathbb{E}(v)[\mathbb{R}](i) \right).$$

*Completeness. Conversely, for every labeled stripe  $\mathcal{P} = (\mathbb{I} \times \mathbb{Q}, \sigma)$ , there is a globally fulfilled decomposition tree  $\mathcal{T} = (\mathbb{V}, \mathbb{E}, \downarrow_1, \downarrow_2)$  such that*

$$\text{Type}_{\mathcal{P}}(\mathbb{I} \times \mathbb{Q}) \subseteq \bigcup_{\substack{v \in \mathbb{V} \\ 1 \leq i \leq |\mathbb{E}(v)|}} \left( \mathbb{E}(v)[\mathbb{L}](i) \cup \mathbb{E}(v)[\mathbb{R}](i) \right).$$

## 5 Reducing Cone Logic to a Proper Fragment of CTL

In this section we briefly describe a decision procedure that solves the satisfiability problem for Cone Logic taking advantage of the tree (pseudo-)model property stated in Section 4. According to such a property, the problem of establishing whether or not a Cone Logic formula  $\varphi$  is satisfiable can be reduced to the problem of checking the existence of a globally fulfilled decomposition tree  $\mathcal{T}$  that features a  $(\varphi)$ -atom  $\mathbb{A}$  such that  $\varphi \in \mathbb{A}$ . The effectiveness of such an approach stems from the fact that the properties that characterize a globally fulfilled decomposition tree can be expressed in a proper fragment of CTL. The satisfiability problem for Cone Logic can thus be decided in (at most) exponential time [8]. Given the state of the art of the decision procedures for CTL, deciding the satisfiability problem for Cone Logic turns out to be quite efficient from a practical point of view. In the following, we show that the satisfiability problem for Cone Logic is actually in PSPACE. In the next section, we will prove that the PSPACE complexity bound is strict, namely, that the satisfiability problem for Cone Logic is PSPACE-hard.

**Theorem 2.** *The satisfiability problem for Cone Logic, interpreted over the rational plane, is in PSPACE.*

*Proof (sketch).* We first show how to reduce the satisfiability problem for a Cone Logic formula  $\varphi$  to the satisfiability problem for a suitable CTL formula  $\vec{\varphi}$ , which is a conjunction of CTL formulas of the forms  $\lambda$ , **AG**  $\lambda$ , **EF**  $\lambda$ , **AG EX**  $\lambda$ , **AG**  $\delta$ , and **AG**  $(\lambda \rightarrow \mathbf{AF} \delta)$ , where  $\lambda$  is a propositional formula and  $\delta$  is a CTL formula that uses the modal operator **AX** in a positive way only ((and it has no occurrences of other modal operators). Let us call these formulas *basic* CTL formulas.

To start with, we introduce three distinguished propositional variables, say, 0, 1, and 2, to encode the two successor relations  $\downarrow_1$  and  $\downarrow_2$  of a decomposition tree  $\mathcal{T}$  in a labeled tree structure  $\mathbb{T}$ . For each vertex  $v$  of  $\mathbb{T}$ , we associate either 0, 1, and 2 with  $v$  depending on whether  $v$  is the root,  $v = \downarrow_1(u)$ , or  $v = \downarrow_2(u)$  for some parent vertex  $u$ . Such a labeling can be enforced by means of a suitable conjunction of basic CTL formulas over the signature  $\{0, 1, 2\}$  (see below). Next, the stripe expressions associated with  $\mathcal{T}$  vertices can be encoded as follows. Since the number of shading sequences can be exponential in  $|\varphi|$ , we need to encode *one by*

one the elements that belong to each atom featured by each shading sequence. To this end, we introduce a new set  $\Sigma$  of propositional variables  $l_i^{atom}$ ,  $l_i^{cluster}$ ,  $r_i^{atom}$ ,  $r_i^{cluster}$ ,  $l_{i,\alpha,D}^{obs}$ ,  $l_{i,\alpha,D}^{req}$ ,  $r_{i,\alpha,D}^{obs}$ , and  $r_{i,\alpha,D}^{req}$ , for every index  $1 \leq i \leq 4 \cdot |\varphi|$ , every subformula  $\alpha$  of  $\varphi$ , and every direction  $D \in \{LL, LR, UL, UR\}$ . Intuitively, the propositional variable  $l_i^{atom}$  (resp.,  $l_i^{cluster}$ ) holds at a given vertex  $v$  of  $T$  if and only if the position  $i$  of the left shading sequence  $E(v)[L]$  in  $\mathcal{T}$  contains an atom (resp., a cluster). Similarly, the propositional variable  $l_{i,\alpha,D}^{obs}$  (resp.,  $l_{i,\alpha,D}^{req}$ ) holds at a given vertex  $v$  of  $T$  if and only if the subformula  $\alpha$  belongs to the set of observables  $Obs_D(E(v)[L](i))$  (resp., the set of requests  $Req_D(E(v)[L](i))$ ) of the atom/cluster at position  $i$  of the left shading sequence  $E(v)[L]$ . Analogous encodings are given for the right shading sequence  $E(v)[R]$ . Since the number of subformulas  $\alpha$  of  $\varphi$  is linear in  $|\varphi|$  and the length of a (maximal) stripe expression  $E(v)$  is at most  $4 \cdot |\varphi|$ , the above-defined propositional variables allow one to represent  $E(v)$  in polynomial space. The (local) consistency conditions **C1**–**C4** can be easily expressed by means of a propositional formula  $\lambda_{C1-C4}$  over the signature  $\Sigma$  and hence they can be enforced globally in the labeled tree structure  $T$  by requiring that the basic CTL formula  $\mathbf{AG} \lambda_{C1-C4}$  holds at the root of  $T$ . Similarly, the matching conditions **M1**–**M3**, which impose further restrictions on the stripe expressions associated with pairs of adjacent vertices, can be expressed by means of a basic CTL formula of the form  $\mathbf{AG} \delta_{M1-M3}$  over the signature  $\{0, 1, 2\} \cup \Sigma$ , where  $\delta_{M1-M3}$  contains only positive occurrences of the modal operator  $\mathbf{AX}$  (and no occurrences of other modal operators). As for the conditions of global fulfillment, we can enforce Condition **G1** by a simple propositional formula  $\lambda_{G1}$  over the signature  $\Sigma$  and Condition **G2** by a conjunction of basic CTL formulas of the form  $\mathbf{AG} (\lambda_{i,D,\alpha} \rightarrow \mathbf{AF} \delta_{i,D,\alpha})$ , where  $i$  ranges over  $\{1, \dots, 4 \cdot |\varphi|\}$ ,  $D$  ranges over  $\{LL, LR, UL, UR\}$ ,  $\alpha$  is a subformula of  $\varphi$ ,  $\lambda_{i,D,\alpha}$  is a propositional formula over the signature  $\Sigma$ , and  $\delta_{i,D,\alpha}$  is a CTL formula over the signature  $\{0, 1, 2\} \cup \Sigma$  that contains only positive occurrences of the modal operator  $\mathbf{AX}$  (and no occurrences of other modal operators). Finally, the existence of a ( $\varphi$ -)atom  $A$  in  $\mathcal{T}$  such that  $\varphi \in A$  can be enforced by a basic CTL formula of the form  $\mathbf{EF} \lambda_\varphi$ . The size of all the above formulas is polynomial in  $|\varphi|$ . Altogether, we have that for any formula  $\varphi$  of Cone Logic, we can (effectively) build an *equi-satisfiable* conjunction of basic CTL formulas  $\vec{\varphi}$  over the signature  $\{0, 1, 2\} \cup \Sigma$  in polynomial time.

We conclude the sketch of the proof by outlining a PSPACE procedure that checks whether the resulting formula  $\vec{\varphi}$  is satisfiable or not. First, we write  $\vec{\varphi}$  as the conjunction of the following three CTL formulas:

$$\begin{aligned} \vec{\varphi}_{tree} &= (0 \wedge \neg 1 \wedge \neg 2) \wedge \mathbf{AG} \mathbf{AX} (\neg 0 \wedge \neg(1 \wedge 2)) \wedge \mathbf{AG} (\mathbf{EX} 1 \wedge \mathbf{EX} 2) \\ \vec{\varphi}_{path} &= \mathbf{AG} \lambda_{C1-C4} \wedge \mathbf{AG} \delta_{M1-M3} \wedge \lambda_{G1} \wedge \bigwedge_{i,D,\alpha} \mathbf{AG} (\lambda_{i,D,\alpha} \rightarrow \mathbf{AF} \delta_{i,D,\alpha}) \\ \vec{\varphi}_{init} &= \mathbf{EF} \lambda_\varphi \end{aligned}$$

The formula  $\vec{\varphi}_{tree}$  defines a labeled tree structure  $T$  where each vertex has two distinguishable successors, the formula  $\vec{\varphi}_{path}$  verifies that  $T$  correctly represents a globally fulfilled decomposition tree  $\mathcal{T}$ , and the formula  $\vec{\varphi}_{init}$  checks that  $\mathcal{T}$  features an atom  $A$  such that  $\varphi \in A$ . Since  $\vec{\varphi}_{path}$  contains only *positive*

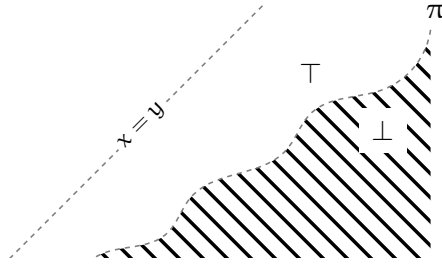
occurrences of the modal operators **AG**, **AF**, and **AX**, we can turn  $\vec{\varphi}_{path}$  into an equivalent LTL formula  $\vec{\varphi}_{path}^{LTL}$  by replacing all occurrences of **AG**, **AF**, and **AX** by **G**, **F**, and **X**, respectively. Formally, we have that for any labeled tree structure  $T$ ,  $\vec{\varphi}$  holds at the root of  $T$  if and only if (i)  $\vec{\varphi}_{tree}$  and  $\vec{\varphi}_{init}$  hold at the root of  $T$  and (ii)  $\vec{\varphi}_{path}^{LTL}$  holds along all infinite paths of  $T$ . By taking advantage of the structure of  $\vec{\varphi}_{path}^{LTL}$  (no **G** operator is nested into an **F** operator), it is possible to show that there exists a *deterministic* Büchi automaton  $\mathcal{A}_{path}$ , which can be computed in polynomial time [2], that recognizes the  $\omega$ -language of all linear models of  $\vec{\varphi}_{path}^{LTL}$ . Given the automaton  $\mathcal{A}_{path}$  over the input alphabet  $\{0, 1, 2\} \times \mathcal{P}(\Sigma)$ , we build a *non-deterministic* Büchi automaton  $\mathcal{A}_{path}^\exists$  that recognizes the projection language  $\pi_{0,1,2}(\mathcal{L}^\omega(\mathcal{A}_{path}))$ . We have that  $\mathcal{A}_{path}^\exists$  recognizes the  $\omega$ -language  $\{0\} \cdot \{1, 2\}^\omega$  if and only if there exists a labeled tree structure  $T$  that satisfies both  $\vec{\varphi}_{tree}$  and  $\vec{\varphi}_{path}^{LTL}$ . Since the inclusion problem for regular  $\omega$ -languages is in PSPACE [10], this gives a procedure that decides, in polynomial space, whether both formulas  $\vec{\varphi}_{tree}$  and  $\vec{\varphi}_{path}$  hold at the root of some labeled tree structure  $T$ . Verifying whether  $\vec{\varphi}_{init}$  holds at the root of  $T$  as well amounts to solve a reachability problem over a slightly modified version of the non-deterministic Büchi automaton  $\mathcal{A}_{path}^\exists$ .  $\square$

## 6 Cone Logic and Interval Temporal Logics

In this section, we prove that Cone Logic subsumes an interesting interval temporal logic, called *DDYY-logic*, which comprises four modal operators  $\langle D \rangle$ ,  $\langle \bar{D} \rangle$ ,  $\langle Y \rangle$ , and  $\langle \bar{Y} \rangle$ . Intuitively, these operators quantify over sub-intervals, super-intervals, ‘younger intervals’, and ‘elder intervals’. From now on, we assume that the underlying temporal domain is (homeomorphic to) the linear ordering  $(\mathbb{Q}, <)$  of the rational numbers and that intervals are non-singleton closed convex subsets of such an ordering, namely, sets of the form  $[x, y] = \{z \in \mathbb{Q} : x \leq z \leq y\}$ , where  $x, y \in \mathbb{Q}$  and  $x < y$ . We shortly denote by  $\mathbb{I}$  the set of all intervals over  $(\mathbb{Q}, <)$ .

The four interval relations  $D$ ,  $\bar{D}$ ,  $Y$ , and  $\bar{Y}$  and the semantics of *DDYY-logic* are defined as follows. Let  $I = [x, y]$  and  $I' = [x', y']$  be two intervals. If  $x < x' < y' < y$ , then we say that  $I'$  is a (strict) *sub-interval* of  $I$ , or, conversely, that  $I$  is a (strict) *super-interval* of  $I'$ . Similarly, if  $x' > x$  and  $y' > y$ , we say that  $I'$  is *younger* than  $I$ , or, conversely, that  $I$  is *elder* than  $I'$ . It is worth noticing that the younger-interval relation  $Y$  and the elder-interval relation  $\bar{Y}$  can be characterized as unions of standard Allen’s relations [2] (for instance, the relation  $Y$  is the union of the ‘later’ relation  $L$ , the ‘immediately after’, or ‘meet’, relation  $A$ , and the ‘overlap’ relation  $O$ ). As for the semantics of *DDYY-logic*, let  $\mathcal{P} = (\mathbb{I}, \sigma)$  be an interval structure, where  $\sigma$  is a valuation function that maps intervals in  $\mathbb{I}$  to

<sup>2</sup> First, we turn each conjunct of  $\vec{\varphi}_{path}^{LTL}$  into an equivalent deterministic Büchi automaton and then we compute the product automaton for the whole formula  $\vec{\varphi}_{path}^{LTL}$ . The resulting automaton has size polynomial in  $|\vec{\varphi}_{path}^{LTL}|$ , provided that the transition labels are symbolically represented by means of suitable propositional formulas over the signature  $\{0, 1, 2\} \cup \Sigma$ .



**Fig. 4.** A  $\pi$ -labeled region delimiting (pseudo)interval-points

sets of propositional variables. The formulas of  $DD\bar{Y}\bar{Y}$ -logic are built up from propositional variables using the boolean connectives and the modal operators  $\langle D \rangle$ ,  $\langle \bar{D} \rangle$ ,  $\langle Y \rangle$ , and  $\langle \bar{Y} \rangle$ , with the obvious semantics (for instance,  $\mathcal{P}, I \models \langle D \rangle \varphi$  holds iff there is a sub-interval  $I'$  of  $I$  such that  $\mathcal{P}, I' \models \varphi$ ).

In [6] Lodaya conjectured the undecidability of the satisfiability problem for the fragment of  $DD\bar{Y}\bar{Y}$ -logic that features the two modal operators  $\langle D \rangle$  and  $\langle \bar{D} \rangle$  only when interpreted over various classes of linear orderings. Here, we partially disprove such a conjecture by showing that formulas of  $DDY\bar{Y}$ -logic, interpreted over the rational line, can actually be translated into equi-satisfiable formulas of Cone Logic. Such a translation exploits the fact that there exists a natural bijection between intervals  $I = [x, y]$  in  $\mathbb{I}$  and points  $p = (x, y)$ , with  $x < y$ , of the rational plane (hereafter, we call these points *interval-points*). Moreover, the region of all and only the interval-points of the rational plane can somehow be described by a suitable formula of Cone Logic. More precisely, let  $\top$ ,  $\perp$ , and  $\pi$  be three fresh propositional variables and let  $\psi_\pi$  be the following formula of Cone Logic:

$$\begin{aligned} \psi_\pi = & \blacksquare(\top \vee \perp \vee \pi) \wedge \blacksquare(\neg\top \vee \neg\perp) \wedge \blacksquare(\neg\top \vee \neg\pi) \wedge \blacksquare(\neg\perp \vee \neg\pi) \\ & \wedge \blacksquare(\top \rightarrow \blacklozenge\top \wedge \blacksquare\top) \wedge \blacksquare(\perp \rightarrow \blacklozenge\perp \wedge \blacksquare\perp) \\ & \wedge \blacksquare(\pi \rightarrow \blacksquare^+\top \wedge \blacksquare^+\perp) \wedge \blacksquare(\blacklozenge\pi \wedge \blacklozenge\pi \rightarrow \pi \vee \blacklozenge\pi \vee \blacklozenge\pi). \end{aligned}$$

Consider now a labeled rational plane  $\mathcal{P} = (\mathbb{P}, \sigma)$  that satisfies  $\psi_\pi$ . We can partition  $\mathcal{P}$  in three regions, namely, (i) the region  $\top_{\mathcal{P}}$  of all  $\top$ -labeled points, (ii) the region  $\perp_{\mathcal{P}}$  of all  $\perp$ -labeled points, and (iii) the region  $\pi_{\mathcal{P}}$  of all  $\pi$ -labeled points (see Figure 4). The region  $\pi_{\mathcal{P}}$  has the form of a ‘thin’ oriented trajectory inside the rational plane such that, for every pair of points  $p, q \in \pi_{\mathcal{P}}$ , there exists another point  $r \in \pi_{\mathcal{P}}$  such that either  $r \in UR(p)$  and  $q \in UR(r)$ , or  $r \in LL(p)$  and  $q \in LL(r)$ . Even though we cannot claim that  $\pi_{\mathcal{P}}$  coincides with the diagonal  $\{(x, x) : x \in \mathbb{Q}\}$  and  $\top_{\mathcal{P}}$  coincides with the set of all interval-points of the rational plane, we can prove the following proposition.

**Proposition 1.** *For every formula  $\varphi$  of Cone Logic and every labeled rational plane  $\mathcal{P} = (\mathbb{P}, \sigma)$  that satisfies  $\varphi_\pi = \varphi \wedge \psi_\pi$ , there is a labeled rational plane  $\mathcal{P}' = (\mathbb{P}, \sigma')$  that satisfies  $\varphi_\pi$  and such that (i) the  $\pi$ -labeled region  $\pi_{\mathcal{P}'}$  coincides*

with the diagonal  $\{(x, x) : x \in \mathbb{Q}\}$  and (ii) the  $\top$ -labeled region  $\top_{\mathcal{P}'}$  coincides with the set of all interval-points of the rational plane.

*Proof.* Let  $\mathcal{P} = (\mathbb{P}, \sigma)$  be a model of the formula  $\varphi_\pi = \varphi \wedge \psi_\pi$ . Without loss of generality, we can assume that for every  $x \in \mathbb{Q}$ , there is a  $\pi$ -labeled point of the form  $\mathbf{p} = (x, \mathbf{y})$ , with  $\mathbf{y} \in \mathbb{Q}$  (this follows easily from Theorem 11). We can thus view the region  $\pi_{\mathcal{P}}$  as the graph of a strictly increasing function  $f_\pi : \mathbb{Q} \rightarrow \mathbb{Q}$  such that, for every point  $\mathbf{p} = (x, \mathbf{y})$ ,  $\pi \in \sigma(\mathbf{p})$  if and only if  $f_\pi(x) = \mathbf{y}$ . Thus, we can denote by  $f_\pi^{-1}$  the inverse of  $f_\pi$ , which is a strictly increasing function as well, and we can introduce the (monotone) transformation  $\mathbf{t}$  that maps any point  $\mathbf{p} = (x, \mathbf{y})$  to the point  $\mathbf{t}(\mathbf{p}) = (x, f_\pi^{-1}(\mathbf{y}))$ . We then exploit such a transformation to define a new labeling function  $\sigma'$  as follows: for every point  $\mathbf{p}$ , we let  $\sigma'(\mathbf{p}) = \sigma(\mathbf{t}(\mathbf{p}))$ . By definition of  $\mathbf{t}$ , the resulting structure  $\mathcal{P}' = (\mathbb{P}, \sigma')$  is homeomorphic to  $\mathcal{P}$  and hence it also satisfies the formula  $\varphi_\pi$ . Moreover, by construction, the region  $\pi_{\mathcal{P}'}$  coincides with the diagonal  $\{(x, x) : x \in \mathbb{Q}\}$  and, similarly, the region  $\top_{\mathcal{P}'}$  coincides with the set of all interval-points of the rational plane.  $\square$

Proposition 11 yields a straightforward translation of any given formula  $\varphi$  of  $D\bar{D}Y\bar{Y}$ -logic into an equi-satisfiable formula  $\varphi'$  of Cone Logic, which is obtained by first replacing in  $\varphi$  every occurrence of the subformula  $\langle D \rangle \alpha$  (resp.,  $\langle \bar{D} \rangle \alpha$ ,  $\langle Y \rangle \alpha$ ,  $\langle \bar{Y} \rangle \alpha$ ) with the formula  $\blacklozenge(\top \wedge \alpha)$  (resp.,  $\blacklozenge(\top \wedge \alpha)$ ,  $\blacklozenge(\top \wedge \alpha)$ ,  $\blacklozenge(\top \wedge \alpha)$ ) and then adding the conjunct  $\psi_\pi$  to the resulting formula. Taking advantage of such a translation and of the decision procedure described in Section 5, we immediately obtain that the satisfiability problem for  $D\bar{D}Y\bar{Y}$ -logic is in PSPACE. As a matter of fact, this subsumes previous results from [3]. Moreover, from [11] we know that the satisfiability problem for D-logic, that is, the interval temporal logic that features the subinterval operator  $\langle D \rangle$  only, and hence that for  $D\bar{D}Y\bar{Y}$ -logic, is PSPACE-hard. Summing up, we have the following corollary.

**Corollary 1.** *The satisfiability problem for Cone Logic, interpreted over the rational plane, and that for  $D\bar{D}Y\bar{Y}$ -logic, interpreted over the rational line, are PSPACE-complete.*

## 7 Conclusions

We would like to conclude by mentioning some natural generalizations of our work. First, we may consider various possible extensions of Cone Logic. For instance, we may think of evaluating formulas of (extended) Cone Logic over *multi-dimensional spaces* (in general,  $2^n$  distinct cone-shaped directions exist in a space with  $n$  dimensions) and/or to partition the two-dimensional space into *more than four* cone-shaped cardinal directions (the same for higher-dimensional spaces). In all such cases, we believe it possible to generalize the achieved results in a rather natural way, preserving the tree pseudo-model property of the logic and, possibly, the PSPACE-completeness of its satisfiability problem. Further generalizations envisage the use of region-based spatial logics. As an example,

the correspondence between intervals over the rational line and points over the rational plane can be lifted to higher-dimensional objects, proving, for instance, that a suitable spatial logic based on rectangular regions, that is, 2-dimensional intervals, is subsumed by a 4-dimensional point-based modal logic very similar to Cone Logic. This establishes an interesting bridge between Cone Logic and modal logics of topological relations. Finally, it is worth studying the satisfiability problem for Cone Logic, and, similarly, for  $\text{D}\bar{\text{D}}\bar{\text{Y}}\bar{\text{Y}}$ -logic, interpreted over different (classes of) structures, e.g., the infinite discrete grid or the Euclidean plane. Even though we expect the satisfiability problem to remain decidable, radically different approaches might be necessary to cope with spaces having discrete or Euclidean topologies.

## References

- [1] Aiello, M., Pratt-Hartmann, I., van Benthem, J.: *Handbook of Spatial Logics*. Springer, Heidelberg (2007)
- [2] Allen, J.F.: Maintaining knowledge about temporal intervals. *Communications of the Association for Computing Machinery* 26(11), 832–843 (1983)
- [3] Bresolin, D., Goranko, V., Montanari, A., Sala, P.: Tableau-based decision procedures for the logics of subinterval structures over dense orderings. *Journal of Logic and Computation* (2008), doi:10.1093/logcom/exn063
- [4] Frank, A.U.: Qualitative spatial reasoning about distances and directions in geographic space. *Journal of Visual Languages and Computing* 3, 343–371 (1992)
- [5] Gabbay, D.M., Kurucz, A., Wolter, F., Zakharyashev, M.: *Many-Dimensional Modal Logics: theory and applications*. Studies in Logic and the Foundations of Mathematics, vol. 148. Elsevier Science Publishers, Amsterdam (2003)
- [6] Lodaya, K.: Sharpening the undecidability of interval temporal logic. In: He, J., Sato, M. (eds.) *ASIAN 2000*. LNCS, vol. 1961, pp. 290–298. Springer, Heidelberg (2000)
- [7] Marx, M., Reynolds, M.: Undecidability of compass logic. *Journal of Logic and Computation* 9(6), 897–914 (1999)
- [8] Meier, A., Mundhenk, M., Thomas, M., Vollmer, H.: The complexity of satisfiability for fragments of CTL and CTL\*. *Electronic Notes in Theoretical Computer Science* 223, 201–213 (2008)
- [9] Morales, A., Navarrete, I., Sciavicco, G.: A new modal logic for reasoning about space: spatial propositional neighborhood logic. *Annals of Mathematics and Artificial Intelligence* 51(1), 1–25 (2007)
- [10] Safra, S.: On the complexity of  $\omega$ -automata. In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pp. 319–327. IEEE Computer Society Press, Los Alamitos (1988)
- [11] Shapirovsky, I., Shehtman, V.B.: Chronological future modality in Minkowski spacetime. In: *Proceedings of the 4th Conference on Advances in Modal Logic*, pp. 437–460. King’s College Publications (2003)
- [12] Stock, O.: *Spatial and Temporal Reasoning*. Kluwer Academic Publishers, Dordrecht (1997)
- [13] Venema, Y.: Expressiveness and completeness of an interval tense logic. *Notre Dame Journal of Formal Logic* 31(4), 529–547 (1990)

# Focalisation and Classical Realisability

Guillaume Munch-Maccagnoni\*

Université Paris 7

**Abstract.** We develop a polarised variant of Curien and Herbelin’s  $\bar{\lambda}\mu\tilde{\mu}$  calculus suitable for sequent calculi that admit a focalising cut elimination (i.e. whose proofs are focalised when cut-free), such as Girard’s classical logic LC or linear logic. This gives a setting in which Krivine’s classical realisability extends naturally (in particular to call-by-value), with a presentation in terms of orthogonality. We give examples of applications to the theory of programming languages.

## 1 Introduction

When Curien and Herbelin unveil in [1] the computational structure of the sequent calculus, they exhibit a model of computation with a simple interaction between *code*  $v$  and *environment*  $e$  inside *commands*  $c = \langle v \parallel e \rangle$  that recalls abstract machines. This is called the  $\bar{\lambda}\mu\tilde{\mu}$  calculus but, following Herbelin [2], we will call it system L, as a reference to the tradition of giving sequent calculi names that begin with this letter.

When the proofs from sequent calculus are represented this way, the symmetry of the logic is reflected in the fact that it is the same syntax that describes code ( $v$ ) and environment ( $e$ ). In particular, each half of the command can bind the other half with the syntax  $\mu x.c'$  (where  $\mu$  is a binder, and the variable  $x$  is bound in the command  $c'$  – we in fact merge in a single letter Curien-Herbelin’s  $\mu$  and  $\tilde{\mu}$ ). This leads to computational ambiguities of the following form:

$$c [\mu y.c'/x] \stackrel{?}{\leftarrow} \langle \mu x.c \parallel \mu y.c' \rangle \stackrel{?}{\rightarrow} c' [\mu x.c/y]$$

In the special case of classical logic,  $x$  and  $y$  can both be fresh in  $c$  and  $c'$ . The above can therefore lead to the identification of  $c$  and  $c'$  without any further assumption (*Lafont’s critical pair*).

If the goal is to find a computational interpretation of classical sequent calculus, then such ambiguities have to be lifted. Curien and Herbelin [1] have achieved an important step in this direction when they have shown that solving the critical pair in favour of the left reduction above yields a computation that corresponds to usual call-by-value (CBV), while the converse choice yields one that corresponds to call-by-name (CBN).

**Focalisation.** Here we tackle this problem from the point of view of focalisation [3,4]. In the realm of logic programming, Andreoli’s focalisation [3] divides the

---

\* Partially funded by INRIA Saclay and the University of Pennsylvania.



connectives of linear logic (LL) among two groups we shall call the *positives* and the *negatives*. The distinction is motivated by the fact that they can be subject to different assumptions during proof-search. Not long after Andreoli's work, Girard [4] considered focalisation as a way to determinise classical sequent calculus with the classical logic LC, which gives an operational status to these polarities. In the first part of the paper (Section 2) we give a syntax for LC and LL derived from Curien-Herbelin's calculus, the *focalising system*  $L$  ( $L_{\text{foc}}$ ). Despite the age of LC and the proximity of this logic with programming languages, it is the first time that such a term language is presented, thus answering a question from Girard [4] (see comparison with other works).

The positives are the *tensor*  $\otimes$ , whose (right-)introduction rule we represent with a pair  $(\cdot, \cdot)$ , and the *plus*  $\oplus$ , whose (right-)introduction rules we represent with the two injections  $\iota_1(\cdot)$  and  $\iota_2(\cdot)$ . A formula whose main connective is positive is decomposed hereditarily until an atom or a negative connective is reached. This means that Andreoli's proof-search recipe builds (normal) terms that belong to the following category of *values*:

$$V ::= x \mid t_- \mid (V, V) \mid \iota_1(V) \mid \iota_2(V)$$

where  $x$  is a positive variable and the term  $t_-$  represents a negative proof.

The negatives are the *par*  $\wp$  and the *with*  $\&$ . Their property is that they are invertible, that is they can be decomposed as early as possible during proof-search, a property better reflected with pattern-matching. Keeping such pattern-matching as little bureaucratic as possible, we represent the (right-)introductions of  $\wp$  and  $\&$  respectively with the binders  $\mu(x, y).c$  and  $\mu(\iota_1(x).c \mid \iota_2(y).c')$ .

The above formulation with values justifies that we see  $\otimes$  and  $\oplus$  as the connectives for the strict pair and the strict sum (the basic datatypes of ML), as much as the invertibility gives  $\wp$  and  $\&$  a lazy computational behaviour. Focalisation therefore gives classical sequent calculus a crisp computational interpretation that goes past the dichotomy between CBV and CBN that prevails in the works on the duality of computation [1, 5]: *lazy* and *strict* no longer qualify strategies of evaluation, but connectives of the logic instead, and CBV and CBN become mixed in the same system.

Credit should be given to the authors who first stated the link between focalisation and the values that underpins our syntax. This was not immediate as Girard's formulation of LC used the *stoup*, a distinguished formula in the sequents. Because the relation was "in the air" before being properly written down, it is hard to go back at the roots of the discovery, but we should mention that the work of Curien and Herbelin [1] had an early occurrence of values explicitly defined as terms in the stoup, though they were not in the above recursive form. The link later became more precise with the works of Dyckhoff-Lengrand [6] and Zeilberger [7].

(As far as classical logic is concerned, we shall in fact present a variant of LC that we call  $LK_{\text{pol}}$  and that, like LL, has the four binary connectives  $\otimes, \wp, \&, \oplus$ . One finds LC back by using the encodings of  $\vee$  and  $\wedge$  found in the original article [4].)

**Realisability.** In the second part of the paper (Section 3) we extend Krivine’s classical realisability for CBN  $\lambda$  calculus [8] to our setting. In realisability we define for each formula  $A$  what it does mean for a term of our language to behave according to (or to *realise*)  $A$  – when formulae are seen as specifications for programs. The definition involves orthogonality between terms and is free from any reference to  $\text{LK}_{\text{pol}}$  or LL. But the main result (*adequacy*) states that a term of type  $A$  also realises  $A$ . It therefore provides a justification of the rules of logic.

The commands of  $L$  remind the computer scientist of the interaction of a *program* with *data* which is found in the theory of automata. We can therefore make a helpful analogy with finite automata in order to introduce classical realisability.

The analogy replaces the terms of  $L$  by words and the states of some NFA  $\mathcal{A} = (S, \Sigma, R, s_0, S_F)$ . Let us write  $\langle \omega \parallel s \rangle$  to symbolise the interaction of a word  $\omega \in \Sigma^*$  with  $\mathcal{A}$  in some state  $s \in S$ . One writes  $\langle a.\omega \parallel s \rangle \rightarrow \langle \omega \parallel s' \rangle$  when  $(s, a, s') \in R$  is a transition of the automaton. Orthogonality between words of  $\Sigma^*$  and states of  $S$  is defined by taking a set of elements of the form  $\langle \omega \parallel s \rangle$  called an *observation*  $\perp$ . This observation has to be *saturated*, that is to say that if  $\langle \omega \parallel s \rangle \rightarrow \langle \omega' \parallel s' \rangle \in \perp$ , then  $\langle \omega \parallel s \rangle \in \perp$ . One writes  $\omega \perp s$  when  $\langle \omega \parallel s \rangle \in \perp$ .

For a given observation  $\perp$ , one then defines  $\mathcal{L}^\perp = \{ s \in S \mid \forall \omega \in \mathcal{L}, \omega \perp s \}$  for all  $\mathcal{L} \subseteq \Sigma^*$  and  $\mathcal{S}^\perp = \{ \omega \in \Sigma^* \mid \forall s \in \mathcal{S}, \omega \perp s \}$  for all  $\mathcal{S} \subseteq S$ . Sets of the form  $\mathcal{L}^\perp$  or  $\mathcal{S}^\perp$  are not ordinary, with the  $\mathcal{S}^\perp$  being regular languages. Moreover, if one takes  $\perp$  to be the smallest observation for which one has  $\varepsilon \perp s_F$  for all  $s_F \in S_F$ , then  $\{s_0\}^\perp$  is the language recognised by  $\mathcal{A}$ . In addition, the co-linearity of  $s$  and  $s'$ , that is to say  $\{s\}^\perp = \{s'\}^\perp$ , corresponds to the Nerode equivalence of states  $s$  and  $s'$ . The equivalence class of  $s$  is therefore given by  $\{s\}^{\perp\perp}$ .

With orthogonality, it is therefore possible to express concisely the main axes of the theory [4]. The intuitions given by orthogonality remain valid with classical realisability, but now we have a much more expressive model of computation that extends  $\lambda$  calculus. Formulae of the logic replace regular expressions, and the sets of terms that realise some formula replace regular languages.

**Applications.** In the third part of the paper (Section 4) come applications. We first show this method allows us to easily show properties of normalisation, type safety or parametricity.

System  $L$  can be compared to  $\lambda$  calculus when it comes to the study of programming: in particular the notion of evaluation order is better treated. In support of this argument, we show classical realisability is discriminating enough to show a clear distinction and relation between the universal quantification coming from proof theory and polymorphism obtained through *value restriction* (Section 4, “*the two quantifications*”). This issue is indeed related to the order of evaluation imposed by quantifications.

We of course do not claim that second order classical predicate calculus is as such a satisfactory model of computation with respect to the current programming

---

<sup>1</sup> See Terui’s [9] for an earlier appearance of notions of automata theory in an orthogonal setting derived from Ludics.

practice. But, as we show, classical realisability accepts in a modular way extensions of the language.

### Comparison with Other Works

**Danos-Joinet-Schellinx’s  $LK_{\text{pol}}^\eta$ .** The paper [10] already considered the four connectives  $\otimes$ ,  $\wp$ ,  $\&$ ,  $\oplus$  at the same time in a derivative of LC called  $LK_{\text{pol}}^\eta$  (which was no more a syntax than LC), but we provide an additional justification for this choice: it is the division of the connectives between strict and lazy that justifies the number of connectives. Also, we chose to get rid of the “ $\eta$ -restriction” of  $LK_{\text{pol}}^\eta$ , hence our choice of the name  $LK_{\text{pol}}$ .

**The “duality of computation”.** The works of Curien-Herbelin and Wadler [15] present a “duality of computation” that appears as the result of a necessary arbitrary choice between CBN and CBV. Laurent established the link with polarities [11], but the duality remained formulated as a dichotomy. On the contrary,  $L_{\text{foc}}$  is a syntax where eager and lazy coexist (as was the case in the non-written two-sided version of LC mentioned by Girard in [4]). The duality of computation is therefore formulated the level of the connectives, as the symmetry between code and environment. This duality is now distinct from the one between positives and negatives.

**Comparison with LLP as a candidate syntax for LC.** The question of giving a representation of LC’s proofs, asked by Girard in [4], is ancient. Laurent gave LLP’s polarised proof nets as an answer [11]; but it should be said LC’s proofs (or equivalently  $LK_{\text{pol}}$ ’s) are not represented directly in proof nets but through a translation into LLP that introduces modalities. This representation overshadows the notions of evaluation order and values that underlie LC and  $LK_{\text{pol}}$ , notions that are important in classical logic as underlined in the syntax we propose.

**Ludics.** We borrow some terminology from Girard’s Ludics [12], as well as the idea of reconstructing types from behaviours. We do not claim however our work should be seen as an alternative version of Ludics, mainly because a syntax based on binders like ours does not offer a proper treatment for the notion of “location” which is prominent in this work.

In addition, we mention the related works of Zeilberger and Terui, from which the present work, which dates back to [13], is independent (except for the more recent Section 4, “*The Two Quantifications*”, where credit is given). Both Zeilberger [7] and Terui [9] proposed focalised calculi inspired by Girard’s Ludics [12]. This prompts a comparison with our proposal.

We share with Zeilberger’s *Calculus of Unity* the computational interpretation of the polarities in a calculus that mixes CBV and CBN. Yet Zeilberger’s syntax, being of higher order, is not a syntax in the conventional and finitary sense of the word.

Terui’s *Computational Ludics* [9] tries to remain closer to Ludics although it does not feature “locations”; and the emphasis is more on the study of complexity. Computational Ludics is fully linear, unlike our setting which can be classical or feature exponentials.

The incentive we have for insisting on using variables and binders, unlike Ludics and like Zeilberger and Terui, is that it allows us to remain conventional. For the same reason we chose here to avoid formal pattern-matching and synthetic connectives, unlike Zeilberger and Terui, and we claim to get a syntax that is closer to the tradition of term syntaxes for logic. (Curien and the author's [14] defines however a variant of our syntax that treats patterns as first-class citizens.)

## 2 Focalising System L

Here we define the syntax and the reduction rules of  $L_{\text{foc}}$ .

**Syntax.** *Positive* and *negative variables* are respectively written  $x, y, z \dots$  and  $\alpha, \beta, \gamma \dots$ . One defines the sets  $\mathcal{T}_+$  and  $\mathcal{T}_-$  of the *positive* and *negative terms*  $t_+$  and  $t_-$ , as well as the set  $\mathcal{C}$  of *commands*  $c$ :

$$\begin{array}{ll}
 \kappa ::= & \alpha \quad | \quad x \\
 t ::= & t_+ \quad | \quad t_- \\
 t_+ ::= & x \quad | \quad \mu\alpha.c \\
 & |(t, t) \quad | \quad \iota_i(t) \text{ (for } i \in \{1, 2\}) \quad (\otimes, \oplus_i) \\
 & | \mu^\lambda(\kappa).c \quad | \quad \{t\} \quad (!, \exists) \\
 t_- ::= & \alpha \quad | \quad \mu x.c \\
 & | \mu(\kappa, \kappa).c \quad | \quad \mu(\iota_1(\kappa).c | \iota_2(\kappa).c) \quad (\wp, \&) \\
 & | \lambda(t) \quad | \quad \mu\{\kappa\}.c \quad (?, \forall) \\
 c ::= & \langle t_+ \parallel t_- \rangle \quad | \quad \langle t_- \parallel t_+ \rangle
 \end{array}$$

with  $\mu(\kappa, \kappa').c$  undefined when  $\kappa = \kappa'$ . Variables that come before a dot in the syntax are bound by the binder  $\mu$ , and terms and commands are always taken modulo  $\alpha$ -equivalence.

$\mathcal{FV}(\cdot)$  denotes the set of the free variables of its argument.  $\mathcal{T}_+^0, \mathcal{T}_-^0, \mathcal{C}^0$  are the sets of the closed terms and commands. In the command  $\langle t \parallel t' \rangle$ ,  $t$  is called the *counter-term* of  $t'$  and  $t'$  the counter-term of  $t$ .

**Formulae.** Positive *atoms* are written  $X, Y$ . Formulae  $A, B$ , positive formulae  $P, Q$  and negative formulae  $N, M$  are given by:

$$\begin{array}{l}
 A ::= P \mid N \\
 P ::= X \mid A \otimes A \mid A \oplus A \mid !A \mid \exists X A \\
 N ::= X^\perp \mid A \wp A \mid A \& A \mid ?A \mid \forall X A
 \end{array}$$

(exponentials are given but will only be used for LL). The polarity of a formula is therefore the polarity of its main connective; but it should be noted that it does not introduce constraints of polarity on the logical systems we introduce: our syntax is only polarised at the level of the dynamics, with shifts of polarities left implicit.

**One-sided sequents.** The literature admits two traditions on sequents: Gentzen’s two-sided sequents ( $\Gamma \vdash \Delta$ ) and Girard’s one-sided sequents ( $\vdash \Gamma$ ). An advantage of the latter is that there is half less rules. The syntax could admit both writings and it might be helpful to clarify the link between the two.

Gentzen’s tradition makes a distinction between being on the right of the sequent ( $\langle t |$ ) and being on the left of the sequent ( $|t \rangle$ ). In  $\langle t \| u \rangle$ , we shall call  $\langle t |$  the code and  $|u \rangle$  the environment as a legacy of the  $\bar{\lambda}\mu\tilde{\mu}$  calculus [1] or of Krivine’s weak head reduction machine [8]. (For instance,  $\langle \nu_1(t) |$  shall represent the first injection of a strict sum applied to  $t$ , while  $|\nu_1(t) \rangle$  shall represent the first projection applied to the lazy pair given by the counter-term.)

Girard’s tradition, with all the formulae on the right, does not make the distinction between  $\langle t |$  and  $|t \rangle$ . As a consequence the syntax has to be quotiented with a new  $\alpha$ -equivalence,  $\langle t \| u \rangle \equiv \langle u \| t \rangle$ . As this paper is in Girard’s tradition, this  $\alpha$ -equivalence will hold. Reasoning as such *modulo* the left-right symmetry blurs the interpretation in terms of abstract machines, but this simplifies the presentation. But a presentation of the present system with two-sided sequents will appear in a long version of the paper.

**Duality.** Girard’s one-sided tradition requires that we replace the connective of negation  $\neg$  by a morphism  ${}^\perp$  on formulae. (The two-sided version will show it is of course possible to have this negation in the syntax, and it will clearly appear that this negation which changes the polarity is different from the ones that appear in works where there is a choice between CBV and CBN, such as Wadler’s Dual Calculus [5].) To each positive formula  $P$  (respectively each negative formula  $N$ ) corresponds a negative *dual* formula  $P^\perp$  (resp. positive  $N^\perp$ ) given by:

$$\begin{array}{ll}
 (X)^\perp \stackrel{\text{def.}}{=} X^\perp & (X^\perp)^\perp \stackrel{\text{def.}}{=} X \\
 (A \otimes B)^\perp \stackrel{\text{def.}}{=} A^\perp \wp B^\perp & (A \wp B)^\perp \stackrel{\text{def.}}{=} A^\perp \otimes B^\perp \\
 (A \oplus B)^\perp \stackrel{\text{def.}}{=} A^\perp \& B^\perp & (A \& B)^\perp \stackrel{\text{def.}}{=} A^\perp \oplus B^\perp \\
 (\exists X A)^\perp \stackrel{\text{def.}}{=} \forall X A^\perp & (\forall X A)^\perp \stackrel{\text{def.}}{=} \exists X A^\perp \\
 (!A)^\perp \stackrel{\text{def.}}{=} ?A^\perp & (?A)^\perp \stackrel{\text{def.}}{=} !(A^\perp)
 \end{array}$$

One therefore has by definition  $A^{\perp\perp} = A$  for each formula  $A$ .

**Contexts, judgements.**  $\Gamma, \Delta \dots$  denote *contexts*: sets of elements of the form  $x : N$  or  $\alpha : P$ . The sequents of  $L_{\text{foc}}$  are judgements of the form:

$$c : (\vdash \Gamma) \qquad \vdash t_+ : P \mid \Gamma \qquad \vdash t_- : N \mid \Gamma$$

In  $\vdash t_+ : P \mid \Gamma$  (resp.  $\vdash t_- : N \mid \Gamma$ ), formula  $P$  (resp.  $N$ ) is said to be *principal*. This should not to be confused with the notion of *stoup*, since the latter requires additional constraints of linearity.

**Substitution.** For each formulae  $A, P$  and each atom  $X$  one defines the formula  $A[P/X]$ ; the important cases are  $X[P/X] = P$  and  $X^\perp[P/X] = P^\perp$ .

**Systems.** Rules for typing  $L_{\text{foc}}$  in one-sided MALL,  $LK_{\text{pol}}$  and LL are given Fig. [1], [2] and [3].

## MALL

**Identity group:**

$$\frac{}{\vdash x : P \mid x : P^\perp} (ax_+) \quad \frac{}{\vdash \alpha : N \mid \alpha : N^\perp} (ax_-)$$

$$\frac{c : (\vdash \kappa : A, \Gamma)}{\vdash \mu \kappa . c : A \mid \Gamma} (\mu) \quad \frac{\vdash t : A \mid \Gamma \quad \vdash u : A^\perp \mid \Delta}{\langle t \parallel u \rangle : (\vdash \Gamma, \Delta)} (\text{cut})$$

**Logic group:**

$$\frac{\vdash t : A \mid \Gamma \quad \vdash u : B \mid \Delta}{\vdash (t, u) : A \otimes B \mid \Gamma, \Delta} (\otimes) \quad \frac{c : (\vdash \kappa : A, \kappa' : B, \Gamma)}{\vdash \mu(\kappa, \kappa') . c : A \wp B \mid \Gamma} (\wp)$$

$$\frac{c : (\vdash \kappa : A, \Gamma) \quad c' : (\vdash \kappa' : B, \Gamma)}{\vdash \mu(\iota_1(\kappa) . c \mid \iota_2(\kappa') . c') : A \& B \mid \Gamma} (\&) \quad \frac{\vdash t : A \mid \Gamma}{\vdash \iota_i(t) : A_1 \oplus A_2 \mid \Gamma} (\oplus_i)$$

$$\frac{\vdash t : A[P/X] \mid \Gamma}{\vdash \{t\} : \exists X A \mid \Gamma} (\exists) \quad \frac{c : (\vdash \kappa : A, \Gamma)}{\vdash \mu\{\kappa\} . c : \forall X A \mid \Gamma} (\forall) \quad (X \notin \mathcal{FV}(\Gamma))$$

**Fig. 1.** The multiplicative additive linear logic MALL

**LK<sub>pol</sub>:** MALL + the following structural group:

$$\frac{c : (\vdash \Gamma)}{c : (\vdash \kappa : A, \Gamma)} (w) \quad \frac{c : (\vdash \kappa : A, \kappa' : A, \Gamma)}{c[\kappa/\kappa'] : (\vdash \kappa : A, \Gamma)} (c)$$

**Fig. 2.** The constructive classical logic LK<sub>pol</sub>

**LL:** MALL + the following structural group:

$$\frac{\vdash t : A \mid \Gamma}{\vdash \lrcorner(t) : ?A \mid \Gamma} (?d) \quad \frac{c : (\vdash \kappa : A, ?\Gamma)}{\vdash \mu^\lrcorner(\kappa) . c : !A \mid ?\Gamma} (!)$$

$$\frac{c : (\vdash \Gamma)}{c : (\vdash x : ?A, \Gamma)} (?w) \quad \frac{c : (\vdash x : ?A, y : ?A, \Gamma)}{c[x/y] : (\vdash x : ?A, \Gamma)} (?c)$$

**Fig. 3.** The linear logic LL

## Focalising Weak Head Reduction

We now move on to defining the cut-elimination protocol based on focalisation.

**Values.** *Values* and *positive values* are defined as follows:

$$V ::= V_+ \mid t_- \quad V_+ ::= x \mid (V, V) \mid \iota_i(V) \mid \mu^\lrcorner(\kappa) . c \mid \{V\}$$

(It therefore holds, by convention, that any negative term is a value.)

The set of values is written  $\mathbb{V}$ .

**Head Reduction.** Execution on the calculus is defined as a relation of head-reduction on  $\mathcal{C}$ .

*$\mu$ -reduction:*

$$\langle \mu \alpha . c \parallel t_- \rangle \rightarrow_{\mu_-} c[t_-/\alpha] \quad \langle \mu x . c \parallel V_+ \rangle \rightarrow_{\mu_+} c[V_+/x]$$

$\beta$ -reduction:

$$\begin{aligned} \langle (V, V') \parallel \mu(\kappa, \kappa').c \rangle &\rightarrow_{\beta} c[V, V'/\kappa, \kappa'] && (\otimes/\wp) \\ \langle \iota_i(V) \parallel \mu(\iota_1(\kappa_1).c_1 \mid \iota_2(\kappa_2).c_2) \rangle &\rightarrow_{\beta} c_i[V/\kappa_i] && (\oplus_i/\&) \\ \langle \{V\} \parallel \mu\{\kappa\}.c \rangle &\rightarrow_{\beta} c[V/\kappa] && (\exists/\forall) \\ \langle \mu^{\wedge}(\kappa).c \parallel \wedge(V) \rangle &\rightarrow_{\beta} c[V/\kappa] && (!/?) \end{aligned}$$

(In case the polarities of the  $V$ 's and of the  $\kappa$ 's do not match each other, the relation  $\rightarrow_{\beta}$  is not defined.)

$\zeta$ -reduction<sup>2</sup>. In case the above rules cannot reduce a command, the following reductions make new cuts appear:

$$\begin{aligned} \text{if } t \notin \mathbb{V} \text{ or } t' \notin \mathbb{V} \text{ then } &\langle (t, t') \parallel u_- \rangle \rightarrow_{\zeta} \langle t \parallel \mu\kappa.\langle t' \parallel \mu\kappa'.\langle (\kappa, \kappa') \parallel u_- \rangle \rangle \rangle \\ \text{if } t \notin \mathbb{V} \text{ then } &\langle \iota_i(t) \parallel u_- \rangle \rightarrow_{\zeta} \langle t \parallel \mu x.\langle \iota_i(x) \parallel u_- \rangle \rangle \\ \text{if } t \notin \mathbb{V} \text{ then } &\langle \{t\} \parallel u_- \rangle \rightarrow_{\zeta} \langle t \parallel \mu x.\langle \{x\} \parallel u_- \rangle \rangle \\ \text{if } t \notin \mathbb{V} \text{ then } &\langle \wedge(t) \parallel V_+ \rangle \rightarrow_{\zeta} \langle t \parallel \mu x.\langle \wedge(x) \parallel V_+ \rangle \rangle \end{aligned}$$

*Head reduction:*

$$\rightarrow \stackrel{\text{def.}}{=} \rightarrow_{\mu_-} \cup \rightarrow_{\mu_+} \cup \rightarrow_{\beta} \cup \rightarrow_{\zeta}$$

**Church-Rosser.** By definition,  $\rightarrow$  has no critical pair. This implies the Church-Rosser property when  $\rightarrow$  is extended to sub-commands. (We have in fact an Orthogonal Pattern Rewrite System, which implies confluence [15].)

**Subject reduction.** Focalising system  $\mathbf{L}$  enjoys *subject reduction* in both  $\mathbf{LK}_{\text{pol}}$  and  $\mathbf{LL}$ . (Proof is routine since each connective has a constructor.)

**Example.** We give the example of the implication, writing  $v$  the code and  $e$  the environment as in Curien-Herbelin's [4]. Take:

$$\begin{aligned} A \rightarrow B &\stackrel{\text{def.}}{=} A^{\perp} \wp B && \lambda\kappa.v \stackrel{\text{def.}}{=} \mu(\kappa, \kappa').\langle v \parallel \kappa' \rangle && (\kappa' \notin \mathcal{FV}(v)) \\ v \cdot e &\stackrel{\text{def.}}{=} (v, e) && v v' \stackrel{\text{def.}}{=} \mu\kappa.\langle v \parallel v' \cdot \kappa \rangle && (\kappa \notin \mathcal{FV}(v, v')) \end{aligned}$$

One has the following derivations:

$$\frac{\vdash v : B \mid \kappa : A^{\perp}, \Gamma}{\vdash \lambda\kappa.v : A \rightarrow B \mid \Gamma} \text{ (abs)} \quad \frac{\vdash v : A \rightarrow B \mid \Gamma \quad \vdash v' : A \mid \Delta}{\vdash v v' : B \mid \Gamma, \Delta} \text{ (app)}$$

We study two particular cases for  $A \rightarrow B$ :

*Case  $A, B$  negative.* This corresponds to CBN. One has, for a positive value  $E$ :

$$\langle v v' \parallel E \rangle \rightarrow \langle v \parallel v' \cdot E \rangle \quad \langle \lambda\alpha.v \parallel v' \cdot E \rangle \rightarrow \langle v [v'/\alpha] \parallel E \rangle$$

<sup>2</sup> Terminology borrowed from Wadler [5]. Forbidding non invertible constructs  $\otimes, \oplus, \exists, ?$  to contain non-values similarly to [4] would be an alternative to the  $\rightarrow_{\zeta}$  reduction, which is therefore available as a convenience. Notice one of course has to arbitrarily decide the evaluation order of the strict pair  $(\cdot, \cdot)$ .

These are the rules of reduction of a Krivine machine in weak head reduction (as in Krivine's [8]), whose *stacks* are values; or again the rules of the  $\bar{\lambda}\mu\tilde{\mu}$  calculus in CBN [11].

*Case A, B positive.* One would expect this to correspond to CBV, and indeed one has:

$$\langle v v' \parallel e \rangle \rightarrow \langle v \parallel v' \cdot e \rangle \qquad \langle \lambda x.v \parallel v' \cdot e \rangle \rightarrow^3 \langle v' \parallel \mu x.\langle v \parallel e \rangle \rangle$$

(where  $\rightarrow$  is the contextual closure of  $\rightarrow$ ). Together with  $\langle V \parallel \mu x.c \rangle \rightarrow c[V/x]$ , this looks like the rules of the CBV  $\bar{\lambda}\mu\tilde{\mu}$  calculus, but the translation of Curien-Herbelin's CBV calculus using this encoding of implication fails because the environment can be of the form  $\mu\alpha.c$  instead of  $\mu x.c$ . As studied by Laurent [11] in the case of LLP, the proper translation of the CBV  $\bar{\lambda}\mu\tilde{\mu}$  calculus in  $\text{LK}_{\text{pol}}$  requires that we define implication with  $\downarrow(P^\perp \wp Q)$ , where the positive connective  $\downarrow$  is an unary tensor that plays in classical logic the role that  $!$  has in LLP: changing the polarity.

### 3 Realisability

This section defines a tool for the study of untyped  $\text{L}_{\text{foc}}$  based on Krivine's classical realisability for CBN  $\lambda$  calculus [8]. We first define a notion of orthogonality between closed terms.

**Definition 1.** A subset  $\perp$  of  $\mathcal{C}^0$  is saturated whenever:

$$c \rightarrow c', c' \in \perp \implies c \in \perp$$

In the rest of the paper,  $\perp$  is some saturated subset of  $\mathcal{C}^0$  and we say that  $t$  is orthogonal to  $u$ , and we write  $t \perp u$ , when  $\langle t \parallel u \rangle \in \perp$ . Because we follow the tradition of single-sided sequents ( $\langle t \parallel u \rangle \equiv \langle u \parallel t \rangle$ ), one has  $t \perp u$  equivalent to  $u \perp t$ .

**Definition 2.** Let  $T \in \mathcal{P}(\mathcal{T}_+^0)$ . One defines  $T^\perp = \{ t_- \in \mathcal{T}_-^0 \mid \forall t_+ \in T, t_+ \perp t_- \}$ . Similarly for  $T \in \mathcal{P}(\mathcal{T}_-^0)$ , one defines  $T^\perp = \{ t_+ \in \mathcal{T}_+^0 \mid \forall t_- \in T, t_+ \perp t_- \}$ . A behaviour (terminology borrowed from Girard's Ludics [12]) is some subset of  $\mathcal{T}_+^0$  or of  $\mathcal{T}_-^0$  of the form  $T^\perp$ .

Depending on the polarity of  $\emptyset$ , one either has  $\emptyset^\perp = \mathcal{T}_+^0$  or  $\emptyset^\perp = \mathcal{T}_-^0$ ; disambiguation will be provided by the context.

**Proposition 3 (Basic properties of the orthogonal).** Let  $T$  and  $U$  be two subsets of  $\mathcal{T}_+^0$  or  $\mathcal{T}_-^0$ . (1) One has  $T \subset T^{\perp\perp}$ . (2) If  $T \subseteq U$  then  $U^\perp \subseteq T^\perp$ . (3) One has  $T^{\perp\perp\perp} = T^\perp$ . (4)  $T$  is a behaviour if and only if  $T = T^{\perp\perp}$ . (5) If  $\mathcal{U}$  is a set of subsets of  $\mathcal{T}_+^0$  (resp. of  $\mathcal{T}_-^0$ ), then one has  $(\bigcup \mathcal{U})^\perp = \bigcap \{ T^\perp \mid T \in \mathcal{U} \}$ .

**Behaviours.** We define for each formula a corresponding behaviour.

**Definition 4.** Parameters  $R, S \dots$  are the members of the set  $\Pi \stackrel{\text{def.}}{=} \mathcal{P}(\mathcal{T}_+^0 \cap \mathbb{V})$ . The language of formulae is extended with parameters: when  $R$  is a parameter,  $R$  is an atomic positive formula and  $R^\perp$  is an atomic negative formula.



**Definition 5.** Let  $T$  and  $U$  be two subsets of  $\mathcal{T}_+^0$  or  $\mathcal{T}_-^0$ . One defines:

$$\begin{aligned} T \times U &= \{ (t, u) \mid t \in T, u \in U \} \\ T + U &= \{ \iota_1(t) \mid t \in T \} \cup \{ \iota_2(u) \mid u \in U \} \\ \downarrow T &= \{ \{u\} \mid u \in T \} \\ !T &= \{ \mu^!(\kappa).c \mid V \in T^{\perp_{\mathbb{V}}} \Rightarrow c[V/\kappa] \in \perp \} \end{aligned}$$

**Definition 6.** To each closed positive formula  $P$  one associates a behaviour  $|P| \in \mathcal{P}(\mathcal{T}_+^0)$  and to each closed negative formula  $N$  one associates a behaviour  $|N| \in \mathcal{P}(\mathcal{T}_-^0)$ . For any term  $t$ , one says  $t$  realises  $A$ , and one writes  $t \Vdash A$ , whenever  $t \in |A|$ . The definition is given by induction on the size of the formula:

$$\begin{aligned} |R| &= R^{\perp\perp} & |R^\perp| &= R^\perp \\ |A \otimes B| &= (|A| \times |B|)^{\perp\perp} & |A \wp B| &= (|A^\perp| \times |B^\perp|)^\perp \\ |A \oplus B| &= (|A| + |B|)^{\perp\perp} & |A \& B| &= (|A^\perp| + |B^\perp|)^\perp \\ |!A| &= (!|A|)^{\perp\perp} & |?A| &= (!|A^\perp|)^\perp \\ |\exists X A| &= \left( \bigcup_{R \in \Pi} \downarrow |A[R/X]| \right)^{\perp\perp} & |\forall X A| &= \left( \bigcup_{R \in \Pi} \downarrow |A^\perp[R/X]| \right)^\perp \end{aligned}$$

We therefore have by definition that for any closed formula  $A$ , one has  $|A|^\perp = |A^\perp|$ . As a consequence we get an equivalent formulation of realisability, closer to the historical definitions [16]:  $t$  realises  $A$  if and only if  $\forall u (u \Vdash A^\perp \Rightarrow t \perp u)$ .

**Generation lemma.** What follows is the main lemma required by the main result of the section.

**Definition 7.**

1. Let  $T$  be a behaviour and  $U \subseteq T$ .  $U$  generates  $T$  if  $T = U^{\perp\perp}$ .
2. Let  $T$  be a behaviour. The set  $T_{\mathbb{V}}$  of the values of  $T$  is  $T \cap \mathbb{V}$ .

**Lemma 8 (Generation).** If  $A$  is a closed formula, then  $|A|$  is generated by the set of its values.

The proof requires the lemmas that follow.

**Lemma 9.** If  $T$  is a behaviour, then  $T_{\mathbb{V}}^{\perp\perp} \cap \mathbb{V} = T_{\mathbb{V}}$ .

**Lemma 10.** Let  $T$  and  $U$  be two behaviours. The following properties hold:

1.  $T_{\mathbb{V}}^{\perp\perp} \times U_{\mathbb{V}}^{\perp\perp} \subseteq (T \times U)_{\mathbb{V}}^{\perp\perp}$ ;
2.  $T_{\mathbb{V}}^{\perp\perp} + U_{\mathbb{V}}^{\perp\perp} \subseteq (T + U)_{\mathbb{V}}^{\perp\perp}$ ;
3.  $\downarrow(T_{\mathbb{V}}^{\perp\perp}) \subseteq (\downarrow T)_{\mathbb{V}}^{\perp\perp}$ .

*Proof.* (1) Let  $t \in T_{\mathbb{V}}^{\perp\perp}$  and  $u \in U_{\mathbb{V}}^{\perp\perp}$ ; let  $v \in (T \times U)_{\mathbb{V}}^\perp$ . If  $t, u \in \mathbb{V}$ , then  $(t, u) \in T_{\mathbb{V}} \times U_{\mathbb{V}}$  by lemma 9. Yet, by definition,  $(T \times U)_{\mathbb{V}} = T_{\mathbb{V}} \times U_{\mathbb{V}}$ , hence  $(t, u) \perp v$ . Otherwise, the result follows by saturation of  $\perp$ , since  $\langle (t, u) \parallel v \rangle$  reduces by  $\rightarrow_\zeta$  to an element of  $\perp$ . (2) and (3): same reasoning.  $\square$

*Proof (Generation lemma).* We sketch some key cases of the proof. By induction on the size of  $A$ . Case  $A$  negative: the result is trivial. Case  $A = R$ :  $|A| = R^{\perp\perp}$  and  $R$  is a set of values. Case  $A = B \otimes C$ :  $|B| \times |C|$  is equal to  $|B|_{\mathbb{V}}^{\perp\perp} \times |C|_{\mathbb{V}}^{\perp\perp}$  by induction hypothesis, and is therefore included in  $(|B| \times |C|)_{\mathbb{V}}^{\perp\perp}$  by lemma 10. Hence  $|A|$  is generated by  $(|B| \times |C|)_{\mathbb{V}}$ .  $\square$

**Corollary 11 (Substitution).** *Let  $A$  a formula with  $\mathcal{FV}(A)$  of the form  $\{X\}$  and  $P$  a closed positive formula.  $|P|_{\mathbb{V}}$  is a parameter and one has:*

$$|A[P/X]| = |A[|P|_{\mathbb{V}}/X]|$$

**Adequacy lemma.** The main result of this section affirms that well-typed terms belong to the behaviours described by their types.

**Theorem 12 (Adequacy lemma,  $\text{LK}_{\text{pol}}$ ).** *Let  $c$  be a command (respectively  $t$  a term) typable in  $\text{LK}_{\text{pol}}$ , of type  $c : (\vdash \kappa_1 : A_1, \dots, \kappa_n : A_n)$  (resp.  $\vdash t : B \mid \kappa_1 : A_1, \dots, \kappa_n : A_n$ ) where the formulae  $A_1, \dots, A_n$  (resp. and  $B$ ) are closed. For all closed terms  $u_1, \dots, u_n$ , if  $u_1 \Vdash A_1^{\perp}, \dots, u_n \Vdash A_n^{\perp}$ , then  $c[\vec{u}_i/\vec{\kappa}_i] \in \perp$  (resp.  $t[\vec{u}_i/\vec{\kappa}_i] \Vdash B$ ).*

*Proof.* By induction on the derivation of  $c$  and  $t$ . The actual induction hypothesis has to be generalised to non-closed formulae, but we can nevertheless sketch the proof with the significant case of activation. Suppose  $\vdash \mu\kappa.c : B \mid \Gamma$  comes from  $c : (\vdash \kappa : B \mid \Gamma)$ . Let  $V \in |B|_{\mathbb{V}}^{\perp}$ . One has  $\langle V \parallel \mu\kappa.c[\vec{u}_i/\vec{\kappa}_i] \rangle \rightarrow c[V, \vec{u}_i/\kappa, \vec{\kappa}_i]$ . Yet  $c[V, \vec{u}_i/\kappa, \vec{\kappa}_i] \in \perp$  by induction hypothesis; hence  $V \perp \mu\kappa.c[\vec{u}_i/\vec{\kappa}_i]$  by saturation. Therefore  $\mu\kappa.c[\vec{u}_i/\vec{\kappa}_i] \in |B|_{\mathbb{V}}^{\perp}$ , which is equal to  $|B|$  by the generation lemma.  $\square$

The adequacy lemma holds if one substitutes “LL” for “ $\text{LK}_{\text{pol}}$ ”. However, classical realisability would give no quantitative result in relation to linearity.

## 4 Applications

We show some of the consequences of the adequacy lemma. Proofs are given to show their brevity. In the following,  $\vdash$  refers equally to typability in  $\text{LK}_{\text{pol}}$  and typability in LL.

Because realisability works with closed terms, we introduce a negative constant,  $\text{tp}$  (for “top-level”), seen as a pattern matching with no branch, that shall serve as an initial environment which is closed.

**Normalisation and type safety.** The following is an instance of the disjunction property. Such a result usually follows from a cut-elimination theorem and a property of subject reduction.

*Example 13.* *Let a formula of the form  $A_1 \oplus A_2$  and  $t \in \mathcal{T}_+^0$  such that  $\vdash t : A_1 \oplus A_2$ . Then there exists  $i \in \{1, 2\}$  and a closed value  $V$  of the same polarity as  $A_i$  such that  $\langle t \parallel \text{tp} \rangle \rightarrow^* \langle i_i(V) \parallel \text{tp} \rangle$ .*

*Proof.* Take  $C$  the set of the  $\langle \nu_i(V) \parallel \text{tp} \rangle$  with  $i \in \{1, 2\}$  and  $V$  a closed value of the same polarity as  $A_i$ . Take  $\perp = \{c \in \mathcal{C}^0 \mid \exists c_0 \in C, c \rightarrow^* c_0\}$ . For all  $V \in |A_i|_{\vee}$  one has  $\nu_i(V) \perp \text{tp}$ , hence  $\text{tp} \in (|A_1|_{\vee} + |A_2|_{\vee})^{\perp}$ . By proposition 10 and the generation lemma, one therefore has  $\text{tp} \Vdash A_1^{\perp} \& A_2^{\perp}$ . Since the adequacy lemma gives  $t \Vdash A_1 \oplus A_2$ , one has  $t \perp \text{tp}$ .  $\square$

This example generalises in two directions: (1) With the positive formula left unspecified ( $\vdash t : P$ ), one gets a result of normalisation in head reduction ( $\langle t \parallel \text{tp} \rangle \rightarrow^* \langle V \parallel \text{tp} \rangle$ )<sup>3</sup> (2) The result generalises to other positive positive formulae: a tensor yields a pair of values, and more generally one has a property of type safety for combinations of  $\otimes$  and  $\oplus$ . This implies type safety for higher-level constructs: a function from  $A$  to  $P$  supplied with the proper argument yields a result of the expected form. We therefore have an alternative to the traditional acceptance of type safety, where one usually proves subject reduction and other syntactical properties.

**Parametricity.** We prove the uniformity of the universal quantification in an example – which of course generalises.

*Example 14.* Let  $t$  be a term typable of type  $\vdash t : \forall X(X \otimes X \rightarrow X \otimes X)$ . Let  $V_1$  and  $V_2$  be two positive values. One has  $\langle t \parallel \{(V_1, V_2) \cdot \text{tp}\} \rangle \rightarrow^* \langle (V_i, V_j) \parallel \text{tp} \rangle$  for some  $i, j \in \{1, 2\}$ .

*Proof.* Indeed, take  $\perp = \{c \in \mathcal{C}^0 \mid \exists i, j \in \{1, 2\}, c \rightarrow^* \langle (V_i, V_j) \parallel \text{tp} \rangle\}$  and  $R = \{V_1, V_2\} \in \Pi$ . One derives  $\langle t \parallel \{(V_1, V_2) \cdot \alpha\} \rangle : (\vdash \alpha : R \otimes R)$ . Yet  $\text{tp} \Vdash R^{\perp} \wp R^{\perp}$ . By the adequacy lemma, one therefore has  $t \perp \perp \{(V_1, V_2) \cdot \text{tp}\}$ .  $\square$

## The Two Quantifications

Zeilberger motivated the use of focalisation in order to explain the “imperfections” of realistic typed programming languages such as the value restriction for intersection types in CBV [18]. We show here how classical realisability concisely accounts for such imperfections.

We have shown above that the adequacy lemma by itself gives some form of type safety and normalisation. We can therefore use it as a criterion to test new rules. One of its major advantages is its modularity. Suppose a feature is added to the system under the form of a new connective, with dual inference rules  $\heartsuit$  and  $\spadesuit$ . Ensuring that adequacy holds refines into two stages:

- (1) Find dual behaviours that correspond to  $\heartsuit$  and  $\spadesuit$ , i.e. for which the induction step of adequacy can be shown.
- (2) Show that the behaviours of  $\heartsuit$  and  $\spadesuit$  are generated by their values, so that the generation lemma holds.

Modularity comes from the fact that, as one can see, only the rules  $\heartsuit$  and  $\spadesuit$  are involved.

<sup>3</sup> As far as strong normalisation is concerned, it should be possible to adapt the technique developed by Lengrand and Miquel [17] for a non-polarised and non-confluent symmetric calculus.

As an example, we apply our method to the possible definitions of  $\forall$  and  $\exists$ . The remarks that follow are however general and apply as well to other “intersection types” such as the binary intersection type and first-order universal quantification.

The first definition that comes to mind for the behaviours of the second-order quantifications  $\forall X A$  and  $\exists X A$  is the following:

$$|\forall X A| \stackrel{?}{=} \bigcap_{R \in \Pi} |A[R/X]| \qquad |\exists X A| \stackrel{?}{=} \left( \bigcup_{R \in \Pi} |A[R/X]| \right)^{\perp\perp}$$

They are dual behaviours by a basic property of the orthogonal, and this definition corresponds to the following inference rules:

$$\frac{\vdash t : A[P/X] \mid \Gamma}{\vdash t : \exists X A \mid \Gamma} \qquad \frac{\vdash t : A \mid \Gamma}{\vdash t : \forall X A \mid \Gamma} \quad (X \notin \mathcal{FV}(\Gamma))$$

Hence this quantification passes the first test. But  $|\forall X P|$  fails to pass the second test, because of:

**Proposition 15.** *An intersection of behaviours generated by their values is not generated by its values in general.*

The proof will be given in a long version of the paper. This remark corresponds in particular to the well-known fact that the first implementations of polymorphism in CBV were unsound in the presence of side-effects (here, control operators of classical logic) [4].

Two distinct solutions that pass the test and that therefore fit the deductive frame of  $\text{LK}_{\text{pol}}$  and  $\text{LL}$  exist.

*A first solution: introducing a shift.* The impossibility of a positive  $\forall$  is noted by Girard when he develops the denotational semantics of classical logic [4]. The connective  $\forall$  is therefore given the negative polarity in LC. The typing rules of second-order quantification of Fig. 11 introduce to this effect a construct that forces the polarity. This corresponds to a Curry-style version of the usual quantification (“ $\lambda X$ ”) of Church-style system  $F$ , which already appeared in Lengrand-Miquel’s symmetric and Curry-style adaptation of  $F_\omega$  [17].

*A second solution: introducing a value restriction.* The second solution restricts the introduction of universal quantification to values, a method found in polymorphism à la ML. It yields quantifications that are different from the first ones, and to make the distinction we shall write them  $\forall$  and  $\exists$ . Value restriction corresponds to the following modification of the above tentative behaviour so that it validates the generation lemma:

$$|\forall X A| \stackrel{\text{def.}}{=} \left( \bigcap_{R \in \Pi} |A[R/X]|_{\forall} \right)^{\perp\perp}$$

(and dually for  $\exists$ ).

<sup>4</sup> Specifically, SML/NJ’s type system was unsound due to the presence of `call/cc`, as discovered by Harper and Lillibridge in 1991 [<http://www.seas.upenn.edu/~sweirich/types/archive/1991/msg00034.html>].

As we will see, they are not the usual quantifications, but they are related to  $\forall$  and  $\exists$  as follows: if we consider that  $\mu\{\kappa\}.c$  – the construct for  $\forall$  – corresponds to a shift of polarity at the level of terms that could be made explicit in the types – with an unary connective (written  $\uparrow$ ) of the negative polarity – then one has the type equality  $|AX \uparrow A| = |\forall X A|$  (and dually for  $\exists$ ).

Now, the adequacy lemma is obtained at the price of the following restriction over the typing rules:

$$\frac{\vdash t : A[P/X] \mid \Gamma}{\vdash t : EX A \mid \Gamma} \quad \frac{\vdash V : A \mid \Gamma}{\vdash V : AX A \mid \Gamma} \quad (X \notin \mathcal{FV}(\Gamma))$$

(Now trying to prove subject reduction for  $LK_{\text{pol}}$  and  $LL$  enriched with these rules would be harder, because there are no corresponding constructs in the syntax. With classical realisability, the fact that there are no constructs makes the proof of adequacy even simpler than for  $\forall$  and  $\exists$ .)

**Comparison of the two solutions.** Although related, the two kinds of quantification are different, since the latter connective will enjoy paradoxical properties such as the fact that  $AX(A \oplus B)$  is the same type as  $(AX A) \oplus (AX B)$ . This shows that  $A$  is not a proper universal quantification for classical logic. (More precisely, one can prove that for a wide range of observations the equality of behaviours  $|AX(A \oplus B)| = |(AX A) \oplus (AX B)|$  holds. By the standards of realisability, this allows one to consider the corresponding type coercions.)

This recalls what Girard called the “*shocking equalities*” of the quantification of Ludics [19]. Now, since our definition of  $\forall$  definitely yields the usual quantification of classical logic, one would tend to question the use of the paradoxical  $A$ . Value restriction and its “shocking equalities” are in fact interesting from the computer scientist’s point of view, because it gives more sub-typing rules. One example is the intersection type, for which a “shocking” equality such as  $|(P_1 \cap P_2) \otimes Q| = |(P_1 \otimes Q) \cap (P_2 \otimes Q)|$ , that is to say the possibility of introducing coercions between these two types, is desirable.

## 5 Conclusion

The present work is not only a concise synthesis but also an extension of distinct works of proof theory: the development of proof syntaxes for sequent calculi initiated by Herbelin and Curien [11]; the study of focalisation and polarisation initiated by Andreoli [3] and Girard [4]; and Krivine’s realisability [16,8] that exposes the computational content of proofs.

Yet the result is surprisingly close to the theory of programming languages, as shown by the analogy of Section 1, the status given to values, or the presence of a distinction between “eager” and “lazy” connectives.

There are many leads for future works, the most obvious being: (1) We would like to study the recent results on the computational content of specific formulae [20,8,21] from the point of view of polarisation. (2) We would like to study the practical counterparts to the good theoretical properties of  $LC$ ’s translation of  $\wedge$  and  $\vee$  that are exposed in [4]. For instance, it should be possible to base on

the present work an extraction procedure for your favourite theorem prover that relies on this translation, which should be compared to extant procedures.

**Acknowledgements.** This work was started and completed at LIX and PPS, but the main part was carried out at Penn. I am grateful to Pierre-Louis Curien and Hugo Herbelin, for helpful interactions and numerous comments around this work, to Stephan Zdancewic and Jeffrey Vaughan for valuable discussions, as well as to the anonymous referees for their comments.

## References

1. Curien, P.L., Herbelin, H.: The duality of computation. *ACM SIGPLAN Notices* 35, 233–243 (2000)
2. Herbelin, H.: Duality of computation and sequent calculus: a few more remarks (manuscript, 2008)
3. Andreoli, J.M.: Logic programming with focusing proof in linear logic. *Journal of Logic and Computation* 2, 297–347 (1992)
4. Girard, J.Y.: A new constructive logic: Classical logic. *Math. Struct. Comp. Sci.* (1991)
5. Wadler, P.: Call-by-value is dual to call-by-name. *SIGPLAN Not.* 38, 189–201 (2003)
6. Dyckhoff, R., Lengrand, S.: LJQ: A strongly focused calculus for intuitionistic logic. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) *CiE 2006. LNCS*, vol. 3988, pp. 173–185. Springer, Heidelberg (2006)
7. Zeilberger, N.: On the unity of duality. *Ann. Pure and App. Logic* 153, 1 (2008)
8. Krivine, J.L.: Realizability in classical logic. To appear in *Panoramas et synthèses*, Société Mathématique de France (2004)
9. Terui, K.: Computational Ludics. To appear in *TCS* (2008)
10. Danos, V., Joinet, J.B., Schellinx, H.: A new deconstructive logic: Linear logic. *Journal of Symbolic Logic* 62 (3), 755–807 (1995)
11. Laurent, O.: Etude de la polarisation en logique. Thèse de doctorat, Université Aix-Marseille II (2002)
12. Girard, J.Y.: Locus solum: From the rules of logic to the logic of rules. *Mathematical Structures in Computer Science* 11, 301–506 (2001)
13. Munch-Maccagnoni, G.: Étude polarisée du système L. Master's thesis (2008)
14. Curien, P.L., Munch-Maccagnoni, G.: The duality of computation under focus (2009)
15. Nipkow, T.: Higher-order critical pairs. In: *Proc. 6th IEEE Symp. Logic in Computer Science*, pp. 342–349. IEEE Press, Los Alamitos (1991)
16. Krivine, J.L.: *Lambda-calculus, types and models*. Ellis Horwood (1993)
17. Lengrand, S., Miquel, A.: Classical  $F_\omega$ , orthogonality and symmetric candidates. *Ann. Pure Appl. Logic* 153(1-3), 3–20 (2008)
18. Zeilberger, N.: Refinement types and computational duality. In: *PLPV 2009* (2009)
19. Girard, J.Y.: *Le Point Aveugle, Cours de logique, Tome II: Vers l'imperfection*. Visions des Sciences. Hermann (2007)
20. Beffara, E., Danos, V.: Disjunctive normal forms and local exceptions. In: *ACM SIGPLAN Int. Conf. Func. Prog., Uppsala, Sweden*, pp. 203–211 (2003)
21. Krivine, J.L.: Structures de réalisabilité, RAM et ultrafiltre sur  $\mathbb{N}$  (to appear) (2008)
22. Herbelin, H.: C'est maintenant qu'on calcule, au cœur de la dualité, Habilitation thesis (2005)
23. Girard, J.Y.: Linear logic. *Theoretical Computer Science* 50, 1–102 (1987)

# Decidable Extensions of Church’s Problem

Alexander Rabinovich

The Blavatnik School of Computer Science, Tel Aviv University, Israel  
rabinoa@post.tau.ac.il

**Abstract.** For a two-variable formula  $B(X,Y)$  of Monadic Logic of Order (MLO) the Church Synthesis Problem concerns the existence and construction of a finite-state operator  $Y=F(X)$  such that  $B(X,F(X))$  is universally valid over  $\text{Nat}$ .

Büchi and Landweber (1969) proved that the Church synthesis problem is decidable.

We investigate a parameterized version of the Church synthesis problem. In this extended version a formula  $B$  and a finite-state operator  $F$  might contain as a parameter a unary predicate  $P$ .

A large class of predicates  $P$  is exhibited such that the Church problem with the parameter  $P$  is decidable.

Our proofs use Composition Method and game theoretical techniques.

## 1 Introduction

Two fundamental results of classical automata theory are decidability of the monadic second-order logic of order (MLO) over  $\omega = (\mathbb{N}, <)$  and computability of the Church synthesis problem. These results have provided the underlying mathematical framework for the development of formalisms for the description of interactive systems and their desired properties, the algorithmic verification and the automatic synthesis of correct implementations from logical specifications, and advanced algorithmic techniques that are now embodied in industrial tools for verification and validation.

**Decidable Expansions of  $\omega$ .** Büchi [1] proved that the monadic theory of  $\omega = (\mathbb{N}, <)$  is decidable. Even before the decidability of the monadic theory of  $\omega$  has been proved, it was shown that the expansions of  $\omega$  by “interesting” functions have undecidable monadic theory. In particular, the monadic theory of  $(\mathbb{N}, <, +)$  and the monadic theory of  $(\mathbb{N}, <, \lambda x.2 \times x)$  are undecidable [15,20]. Therefore, most efforts to find decidable expansions of  $\omega$  deal with expansions of  $\omega$  by monadic predicates.

Elgot and Rabin [5] found many interesting predicates  $P$  for which MLO over  $(\mathbb{N}, <, P)$  is decidable. Among these predicates are the set of factorial numbers  $\{n! \mid n \in \mathbb{N}\}$ , the sets of  $k$ -th powers  $\{n^k \mid n \in \mathbb{N}\}$  and the sets  $\{k^n \mid n \in \mathbb{N}\}$  (for  $k \in \mathbb{N}$ ).

The Elgot and Rabin method has been generalized and sharpened over the years and their results were extended to a variety of unary predicates

(see e.g., [18,16,3]). In [11,14] we provided necessary and sufficient conditions for the decidability of monadic (second-order) theory of expansions of the linear order of the naturals  $\omega$  by unary predicates.

**Church’s Problem.** What is known as the “Church synthesis problem” was first posed by A. Church in [4] for the case of  $(\omega, <)$ . The Church problem is much more complicated than the decidability problem for *MLO*. Church uses the language of automata theory. It was McNaughton (see [9]) who first observed that the Church problem can be equivalently phrased in game-theoretic language and in recent years many authors took up the generalizations of such games for various applications of the algorithmic theory of infinite games (see e.g., [6,10]). McNaughton considered games over  $\omega$ . We consider such games over expansions of  $\omega$  by unary predicates.

Let  $\mathcal{M} = (\mathbb{N}, <, P)$  be the expansion of  $\omega$  by a unary predicate  $P$ . Let  $\varphi(X_1, X_2, Z)$  be a formula, where  $X_1, X_2$  and  $Z$  are set (monadic predicate) variables. The *McNaughton game*  $\mathcal{G}_\varphi^{\mathcal{M}}$  is defined as follows.

1. The game is played by two players, called Player I and Player II.
2. A *play* of the game has  $\omega$  rounds.
3. At round  $i \in \mathbb{N}$ : first, Player I chooses  $\rho_{X_1}(i) \in \{0, 1\}$ ; then, Player II chooses  $\rho_{X_2}(i) \in \{0, 1\}$ . Both players can observe whether  $i \in P$ .
4. By the end of the play two predicates  $\rho_{X_1}, \rho_{X_2} \subseteq \mathbb{N}$  have been constructed □
5. Then, Player I wins the play if  $\mathcal{M} \models \varphi(\rho_{X_1}, \rho_{X_2}, P)$ ; otherwise, Player II wins the play.

What we want to know is: Does either one of the players have a *winning strategy* in  $\mathcal{G}_\varphi^{\mathcal{M}}$ ? If so, which one? That is, can Player I choose his moves so that, whatever way Player II responds we have  $\varphi(\rho_{X_1}, \rho_{X_2}, P)$ ? Or can Player II respond to Player I’s moves in a way that ensures the opposite?

At round  $i$ , Player I has access only to  $\rho_{X_1}(0) \dots \rho_{X_1}(i-1), \rho_{X_2}(0) \dots \rho_{X_2}(i-1)$  and  $P(0) \dots P(i)$ .

Hence, a strategy of Player I can be defined as a function which assigns to any finite sequence

$$(\rho_{X_1}(0), \rho_{X_2}(0), P(0)) \dots (\rho_{X_1}(i-1), \rho_{X_2}(i-1), P(i-1)) (*, *, P(i))$$

a value in  $\{0, 1\}$  which is taken to be  $\rho_{X_1}(i)$ .

At round  $i$ , Player II has access only to  $\rho_{X_1}(0) \dots \rho_{X_1}(i), \rho_{X_2}(0) \dots \rho_{X_2}(i-1)$  and  $P(0) \dots P(i)$ .

Hence, a strategy of Player II can be defined as a function which assigns to any finite sequence

$$(\rho_{X_1}(0), \rho_{X_2}(0), P(0)) \dots (\rho_{X_1}(i-1), \rho_{X_2}(i-1), P(i-1)) (\rho_{X_1}(i), *, P(i))$$

a value in  $\{0, 1\}$  which is taken to be  $\rho_{X_2}(i)$ .

---

<sup>1</sup> We identify monadic predicates with their characteristic functions.



Since strategies are functions from finite strings (over a finite alphabet) to  $\{0, 1\}$  we can classify them according to their complexity. The recursive strategies, the finite-memory strategies, i.e., the strategies computable by finite-state transducers are defined in a natural way (see Sect. 3).

We investigate the following parameterized version of the Church synthesis problem.

**Synthesis Problems for  $\mathcal{M} = (\mathbb{N}, <, P)$ , where  $P \subseteq \mathbb{N}$**

*Input:* an MLO formula  $\varphi(X_1, X_2, Z)$ .

*Task:* Check whether Player I has a finite-memory winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$  and if there is such a strategy - construct it.

To simplify notations, games and the synthesis problem were previously defined for formulas with three free variables  $X_1, X_2$  and  $Z$ . It is easy to generalize all definitions and results to formulas  $\psi(X_1, \dots, X_m, Y_1, \dots, Y_n, Z_1, \dots, Z_l)$  with many variables. In this generalization at round  $\beta$ , Player I chooses values for  $X_1(\beta), \dots, X_m(\beta)$ , then Player II replies by choosing the values to  $Y_1(\beta), \dots, Y_n(\beta)$  and the structure  $\mathcal{M}$  provides the interpretation for  $Z_1, \dots, Z_l$ . Note that, strictly speaking, the input to the synthesis problem is not only a formula, but a formula plus a partition of its free-variables to Player I's variables and Player II's variables and parameter's variables.

In [2], Büchi and Landweber prove the computability of the synthesis problem in  $\omega = (\mathbb{N}, <)$  (no parameters).

**Theorem 1.1 (Büchi-Landweber, 1969).** *Let  $\varphi(\bar{X}, \bar{Y})$  be a formula, where  $\bar{X}$  and  $\bar{Y}$  are disjoint lists of variables. Then:*

**Determinacy:** *One of the players has a winning strategy in the game  $\mathcal{G}_\varphi^\omega$ .*

**Decidability:** *It is decidable which of the players has a winning strategy.*

**Finite-state strategy:** *The player who has a winning strategy, also has a finite-state winning strategy.*

**Synthesis algorithm:** *We can compute for the winning player in  $\mathcal{G}_\varphi^\omega$  a finite-state winning strategy.*

The determinacy part of the theorem follows from the topological arguments. In particular for every expansion  $\mathcal{M}$  of  $\omega$  by unary predicates, the game  $\mathcal{G}_\varphi^{\mathcal{M}}$  is determinate.

Let  $\mathcal{M}$  be an expansion of  $\omega$  by unary predicates. We proved in [12], that there is an algorithm which for every MLO formula  $\varphi$  decides who wins  $\mathcal{G}_\varphi^{\mathcal{M}}$  if and only if the monadic theory of  $\mathcal{M}$  is decidable. Moreover, we proved that if the monadic theory of  $\mathcal{M}$  is decidable, then the player who has a winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$  has a recursive MLO-definable winning strategy which is computable from  $\varphi$ .

The finite-state strategy part of Theorem 1.1 fails for decidable expansions of  $\omega$ . For example, let  $\mathbf{Fac} = \{n! \mid n \in \mathbb{N}\}$  be the set of factorial numbers. The monadic theory of  $\mathcal{M}_{\mathbf{fac}} := (\mathbb{N}, <, \mathbf{Fac})$  is decidable by [5]. Let  $\varphi(X_1, X_2, Z)$  be a formula which specifies that  $t \in X_1$  iff  $t + 1 \in Z$  (hence for the game  $\mathcal{G}_\varphi^{\mathcal{M}_{\mathbf{fac}}}$  the moves of Player II are irrelevant). It is easy to see that Player I has a winning

strategy in  $\mathcal{G}_\varphi^{\mathcal{M}_{fac}}$ , yet Player I has no finite-state winning strategy in this game. The results of this paper imply that the synthesis problem for  $(\mathbb{N}, <, \mathbf{Fac})$  is decidable.

**Main Result.** Our main result describes a large class of predicates  $P$  such that the synthesis problem for  $(\mathbb{N}, <, P)$  is decidable.

An  $\omega$ -sequence  $a_i$  is said to be ultimately periodic with lag  $l$  and period  $d$  if  $a_i = a_{i+d}$  for  $i > l$ .

**Definition 1.2.** Let  $\bar{k} = (k_1 < k_2 < \dots k_i < \dots)$  be an increasing  $\omega$ -sequence of integers.

1.  $\bar{k}$  is sparse if for each  $d$  there is  $n$  such that  $k_{i+1} - k_i > d$  for each  $i > n$ .  
 $\bar{k}$  is effectively sparse if there is an algorithm that for each  $d$  computes  $n$  such that  $k_{i+1} - k_i > d$  for each  $i > n$ .
2.  $\bar{k}$  is ultimately reducible if for every  $m > 1$  the sequence  $k_i \bmod m$  is ultimately periodic.  $\bar{k}$  is effectively ultimately reducible if there is an algorithm that for each  $m$  computes a lag and a period of  $k_i \bmod m$ .

**Definition 1.3.** Let  $ER$  be the class of increasing recursive  $\omega$ -sequences of integers which are effectively sparse and effectively ultimately reducible.

Let  $P \subseteq \mathbb{N}$  be a predicate. We denote by  $Enum(P)$  the sequence  $(k_1, k_2 \dots k_i \dots)$  which enumerates the elements of  $P$  in the increasing order. Often we do not distinguish between  $P$  and  $Enum(P)$ , In particular we say that a predicate is  $ER$  predicate if  $Enum(P)$  is in  $ER$ . The class  $ER$  contains many interesting predicates. It contains the set  $\mathbf{Fact} = \{n! \mid n \in \mathbb{N}\}$  of factorial numbers, the sets  $\{k^n \mid n \in \mathbb{N}\}$ , the sets  $\{n^k \mid n \in \mathbb{N}\}$ . It has nice closure properties, e.g. if  $\bar{k}$  and  $\bar{l}$  are in  $ER$  then  $\{k_i + l_i \mid i \in \mathbb{N}\}$ ,  $\{k_i \times l_i \mid i \in \mathbb{N}\}$ , and  $\{k_i^{l_i} \mid i \in \mathbb{N}\}$  are in  $ER$ .

In [18], Siefkes introduced  $ER$  predicates and generalized Elgot-Rabin contraction method to prove that for every  $ER$  predicate  $P$  the monadic theory of  $\mathcal{M} = (\mathbb{N}, <, P)$  is decidable. Our main results show that the synthesis problem for each predicate  $P \in ER$  is decidable.

**Theorem 1.4 (Main).** Let  $P$  be an  $ER$  predicate and let  $\mathcal{M} = (\mathbb{N}, <, P)$ . There is an algorithm that for every MLO formula  $\varphi(X_1, X_2, Z)$  decides whether Player I has a finite-memory winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$ , and if so constructs such a strategy.

Our algorithm is based on game theoretical techniques and the composition method developed by Feferman-Vaught, Shelah and others.

**Organization of the paper.** The article is organized as follows. The next section recalls standard definitions about the monadic second-order logic of order, and summarizes elements of the composition method. In Section 3, we introduce game-types, define games on game types and show that these game are reducible to the McNaughton games. Section 4 consider games over finite chains. Sufficient conditions are provided for existence of a finite state strategies which uniformly wins over a class of finite chains.

Section 5 describes an algorithm for the synthesis problem over the expansions of  $\omega$  by ER predicates, and proves the soundness of the algorithm, i.e., if the algorithm outputs a strategy for  $\mathcal{G}_\varphi^{\mathcal{M}}$ , then it is a finite state strategy which wins  $\varphi$  over  $\mathcal{M}$ . The proof of completeness appears in the full version of this paper [13]. Further results and open questions are discussed in Sect. 6.

## 2 Preliminaries and Background

We use  $i, j, n, k, l, m, p, q$  for natural numbers. We use  $\mathbb{N}$  for the set of natural numbers and  $\omega$  for the first infinite ordinal. We use the expressions “chain” and “linear order” interchangeably. A chain with  $m$  elements will be denoted by  $m$ .

We use  $\mathbb{P}(A)$  for the set of subsets of  $A$ .

### 2.1 The Monadic Logic of Order (MLO)

**Syntax.** The syntax of the monadic second-order logic of order - MLO has in its vocabulary *individual* (first order) variables  $t_1, t_2 \dots$ , monadic *second-order* variables  $X_1, X_2 \dots$  and one binary relation  $<$  (the order).

Atomic formulas are of the form  $X(t)$  and  $t_1 < t_2$ . Well formed formulas of the monadic logic MLO are obtained from atomic formulas using Boolean connectives  $\neg, \vee, \wedge, \rightarrow$  and the first-order quantifiers  $\exists t$  and  $\forall t$ , and the second-order quantifiers  $\exists X$  and  $\forall X$ . The quantifier depth of a formula  $\varphi$  is denoted by  $qd(\varphi)$ .

We use upper case letters  $X, Y, Z, \dots$  to denote second-order variables; with an overline,  $\bar{X}, \bar{Y}$ , etc., to denote finite tuples of variables.

**Semantics.** A *structure* is a tuple  $\mathcal{M} := (A, <^{\mathcal{M}}, \bar{P}^{\mathcal{M}})$  where:  $A$  is a non-empty set,  $<^{\mathcal{M}}$  is a binary relation on  $A$ , and  $\bar{P}^{\mathcal{M}} := (P_1^{\mathcal{M}}, \dots, P_l^{\mathcal{M}})$  is a *finite* tuple of subsets of  $A$ .

If  $\bar{P}^{\mathcal{M}}$  is a tuple of  $l$  sets, we call  $\mathcal{M}$  an *l-structure*. If  $<^{\mathcal{M}}$  linearly orders  $A$ , we call  $\mathcal{M}$  an *l-chain*. When the specific  $l$  is unimportant, we simply say that  $\mathcal{M}$  is a *labeled* chain.

Suppose  $\mathcal{M}$  is an  $l$ -structure and  $\varphi$  a formula with free-variables among  $X_1, \dots, X_l$ . We define the relation  $\mathcal{M} \models \varphi$  (read:  $\mathcal{M}$  satisfies  $\varphi$ ) as usual, understanding that the second-order quantifiers range over *subsets* of  $A$ .

Let  $\mathcal{M}$  be an  $l$ -structure. The *monadic theory* of  $\mathcal{M}$ ,  $MTh(\mathcal{M})$ , is the set of all formulas with free-variables among  $X_1, \dots, X_l$  satisfied by  $\mathcal{M}$ .

From now on, we omit the superscript in ‘ $<^{\mathcal{M}}$ ’ and ‘ $\bar{P}^{\mathcal{M}}$ ’. We often write  $(A, <) \models \varphi(\bar{P})$  meaning  $(A, <, \bar{P}) \models \varphi$ .

For a chain  $\mathcal{M} = (A, <, \bar{P})$  and a subset  $I$  of  $A$ , we denote by  $\mathcal{M} \upharpoonright I$  the subchain of  $\mathcal{M}$  over the set  $I$ .

### 2.2 Elements of the Composition Method

Our proofs make use of the technique known as the composition method developed by Feferman-Vaught and Shelah [8,17]. To fix notations and to aid the

reader unfamiliar with this technique, we briefly review the definitions and results that we require. A more detailed presentation can be found in [19] or [7].

Let  $n, l \in \mathbb{N}$ . We denote by  $\mathfrak{Form}_l^n$  the set of MLO formulas with free variables among  $X_1, \dots, X_l$  and of quantifier depth  $\leq n$ .

**Definition 2.1.** Let  $n, l \in \mathbb{N}$  and let  $\mathcal{M}, \mathcal{N}$  be  $l$ -structures. The  $n$ -theory of  $\mathcal{M}$  is  $Th^n(\mathcal{M}) := \{\varphi \in \mathfrak{Form}_l^n \mid \mathcal{M} \models \varphi\}$ . If  $Th^n(\mathcal{M}) = Th^n(\mathcal{N})$ , we say that  $\mathcal{M}$  and  $\mathcal{N}$  are  $n$ -equivalent and write  $\mathcal{M} \equiv^n \mathcal{N}$ .

Clearly,  $\equiv^n$  is an equivalence relation. For any  $n \in \mathbb{N}$  and  $l > 0$ , the set  $\mathfrak{Form}_l^n$  is infinite. However, it contains only finitely many semantically distinct formulas. So, there are finitely many  $\equiv^n$ -equivalence classes of  $l$ -structures. In fact, we can compute characteristic formulas for the  $\equiv^n$ -equivalence classes:

**Lemma 2.2 (Hintikka Lemma).** For  $n, l \in \mathbb{N}$ , we can compute a finite set  $Char_l^n \subseteq \mathfrak{Form}_l^n$  such that:

- For every  $\equiv^n$ -equivalence class  $C$  there is a unique  $\tau \in Char_l^n$  such that for every  $l$ -structure  $\mathcal{M}$ :  $\mathcal{M} \in C$  iff  $\mathcal{M} \models \tau$ .
- Every MLO formula  $\varphi(X_1, \dots, X_l)$  with  $qd(\varphi) \leq n$  is equivalent to a (finite) disjunction of characteristic formulas from  $Char_l^n$ . Moreover, there is an algorithm which for every formula  $\varphi(X_1, \dots, X_l)$  computes a finite set  $G \subseteq Char_l^{qd(\varphi)}$  of characteristic formulas, such that  $\varphi$  is equivalent to the disjunction of all the formulas from  $G$ .

Any member of  $Char_l^n$  we call a  $(n, l)$ -Hintikka formula or  $(n, l)$ -characteristic formula. We use  $\tau, \tau_i, \tau^j$  to range over the characteristic formulas and  $G, G_i, G'$  to range over sets of characteristic formulas.

**Definition 2.3 ( $n$ -Type).** For  $n, l \in \mathbb{N}$  and an  $l$ -structure  $\mathcal{M}$ , we denote by  $type_n(\mathcal{M})$  the unique member of  $Char_l^n$  satisfied by  $\mathcal{M}$  and call it the  $n$ -type of  $\mathcal{M}$ .

Thus,  $type_n(\mathcal{M})$  determines  $Th^n(\mathcal{M})$  and, indeed,  $Th^n(\mathcal{M})$  is computable from  $type_n(\mathcal{M})$ .

**Definition 2.4 (Sum of chains).** (1) Let  $l \in \mathbb{N}$ ,  $\mathcal{I} := (I, <^{\mathcal{I}})$  a chain and  $\mathfrak{S} := (\mathcal{M}_\alpha \mid \alpha \in I)$  a sequence of  $l$ -chains. Write  $\mathcal{M}_\alpha := (A_\alpha, <^\alpha, P_1^\alpha, \dots, P_l^\alpha)$  and assume  $A_\alpha \cap A_\beta = \emptyset$  whenever  $\alpha \neq \beta$  are in  $I$ . The ordered sum of  $\mathfrak{S}$  is the  $l$ -chain

$$\sum_{\mathcal{I}} \mathfrak{S} := \left( \bigcup_{\alpha \in I} A_\alpha, <^{\mathcal{I}, \mathfrak{S}}, \bigcup_{\alpha \in I} P_1^\alpha, \dots, \bigcup_{\alpha \in I} P_l^\alpha \right), \text{ where}$$

if  $\alpha, \beta \in I$ ,  $a \in A_\alpha$ ,  $b \in A_\beta$ , then  $b <^{\mathcal{I}, \mathfrak{S}} a$  iff  $\beta <^{\mathcal{I}} \alpha$  or  $\beta = \alpha$  and  $b <^\alpha a$ .

If the domains of the  $\mathcal{M}_\alpha$ ’s are not disjoint, replace them with isomorphic  $l$ -chains that have disjoint domains, and proceed as before.

(2) If for all  $\alpha \in I$ ,  $\mathcal{M}_\alpha$  is isomorphic to  $\mathcal{M}$  for some fixed  $\mathcal{M}$ , we denote  $\sum_{\mathcal{I}} \mathfrak{S}$  by  $\mathcal{M} \times \mathcal{I}$ .

(3) If  $\mathcal{I} = (\{0, 1\}, <)$  and  $\mathfrak{S} = (\mathcal{M}_0, \mathcal{M}_1)$ , we denote  $\sum_{\mathcal{I}} \mathfrak{S}$  by  $\mathcal{M}_0 + \mathcal{M}_1$ .

We will use only special cases of this definition in which the index chain  $\mathcal{I}$  and the summand chains  $\mathcal{M}_\alpha$  are finite or of the order type  $\omega$ .

The next proposition says that taking ordered sums preserves  $\equiv^n$ -equivalence.

**Proposition 2.5.** *Let  $n, l \in \mathbb{N}$ . Assume:*

1.  $(I, <^{\mathcal{I}})$  is a linear order,
2.  $(\mathcal{M}_\alpha^0 \mid \alpha \in I)$  and  $(\mathcal{M}_\alpha^1 \mid \alpha \in I)$  are sequences of  $l$ -chains, and
3. for every  $\alpha \in I$ ,  $\mathcal{M}_\alpha^0 \equiv^n \mathcal{M}_\alpha^1$ .

Then,  $\sum_{\alpha \in I} \mathcal{M}_\alpha^0 \equiv^n \sum_{\alpha \in I} \mathcal{M}_\alpha^1$ .

This allows us to define the sum of formulas in  $\text{Char}_l^n$  with respect to any linear order.

**Definition 2.6.** (1) *Let  $n, l \in \mathbb{N}$ ,  $\mathcal{I} := (I, <^{\mathcal{I}})$  a chain,  $\mathfrak{H} := (\tau_\alpha \mid \alpha \in I)$  a sequence of  $(n, l)$ -Hintikka formulas. The ordered sum of  $\mathfrak{H}$ , (notations  $\sum_{\mathcal{I}} \mathfrak{H}$  or  $\sum_{\alpha \in \mathcal{I}} \tau_\alpha$ ), is an element  $\tau$  of  $\text{Char}_l^n$  such that:*

*if  $\mathfrak{G} := (\mathcal{M}_\alpha \mid \alpha \in I)$  is a sequence of  $l$ -chains and  $\text{type}_n(\mathcal{M}_\alpha) = \tau_\alpha$  for  $\alpha \in I$ , then*

$$\text{type}_n\left(\sum_{\mathcal{I}} \mathfrak{G}\right) = \tau.$$

(2) *If for all  $\alpha \in I$ ,  $\tau_\alpha = \tau$  for some fixed  $\tau \in \text{Char}_l^n$ , we denote  $\sum_{\alpha \in \mathcal{I}} \tau_\alpha$  by  $\tau \times \mathcal{I}$ .*

(3) *If  $\mathcal{I} = (\{0, 1\}, <)$  and  $\mathfrak{H} = (\tau_0, \tau_1)$ , we denote  $\sum_{\alpha \in \mathcal{I}} \tau_\alpha$  by  $\tau_0 + \tau_1$ .*

The following fundamental result of Shelah can be found in [17]:

**Theorem 2.7 (Addition Theorem).** *The function which maps the pairs of characteristic formulas to their sum is a recursive function. Formally, the function  $\lambda n, l \in \mathbb{N}. \lambda \tau_0, \tau_1 \in \text{Char}_l^n. \tau_0 + \tau_1$  is recursive.*

We often use the following well-known lemmas:

**Lemma 2.8.** *For every  $n \in \mathbb{N}$  there is  $N_0(n)$  such that for every sentence  $\varphi$  of the quantifier depth at most  $n$  and every  $m \geq N_0$ :*

*$\varphi$  is satisfiable over the  $m$ -element chain iff it is satisfiable over the  $m + N_0$ -element chain, i.e.,  $m \equiv^n m + N_0$ .*

*Furthermore,  $N_0$  is computable from  $n$ .*

**Lemma 2.9.** *For every  $n \in \mathbb{N}$  there is  $N_1(n)$  such that for every  $\mathcal{M} = (A, <, P)$ : if  $n_1 > n_2 \geq N_1$  and  $n_1 = n_2 \bmod N_1$ , then  $\mathcal{M} \times n_1 \equiv^n \mathcal{M} \times n_2$ . Moreover,  $N_1$  is computable from  $n$ .*

### 3 Game Types

In this section we introduce game-types; their role for games is similar to the role of types for *MLO*. We define games on game types and show that these game are reducible to the McNaughton games. But first we introduce a terminology, define finite-memory strategies and fix some notational conventions.

Let  $\mathcal{M} := (\mathbb{N}, <, \bar{P})$  be an  $l$ -chain and let  $\rho := (\rho_{X_1}(0), \rho_{X_2}(0)) \dots (\rho_{X_1}(i), \rho_{X_2}(i)) \dots$  be a play. We denote by  $\mathcal{M}^\wedge \rho$  the expansion of  $\mathcal{M}$  by the predicates  $\rho_{X_1}$  and  $\rho_{X_2}$ . We say that the  $m$ -type of  $\rho$  is  $\tau$  if  $\tau = \text{type}_m(\mathcal{M}^\wedge \rho)$ . Whenever  $\mathcal{M}$  is clear from the context we write  $\text{type}_m(\rho)$  for  $\text{type}_m(\mathcal{M}^\wedge \rho)$ .

A strategy for Player I for games over  $l$ -chains is a transducer which consists of a set  $Q$  - memory states, an initial state  $q_{init}$ , the memory update functions  $\mu_1 : Q \times \{0, 1\}^l \rightarrow Q$  and  $\mu_2 : Q \times \{0, 1\} \rightarrow Q$ , and the output function  $\theta : Q \rightarrow \{0, 1\}$ .

A strategy is finite-memory (or finite-state) if its set of memory states is finite.

During a play at round  $i$ , Player I first updates the state according to  $\mu_1$  and the values of predicates  $\bar{P}(i)$ , then outputs its value according to  $\theta$ , and then after a move of Player II update the state according to  $\mu_2$ . Hence, a play  $\rho := (\rho_{X_1}(0), \rho_{X_2}(0)) \dots (\rho_{X_1}(i), \rho_{X_2}(i)) \dots$  is consistent with such a strategy if there are  $q_0, q'_0 \dots q_i, q'_i$  such that  $q_0 = \mu_1(q_{init}, \bar{P}(0))$ ,  $\rho_{X_1}(i) = \theta(q_i)$ ,  $q'_i = \mu_2(q_i, \rho_{X_2}(i))$  and  $q_{i+1} = \mu_1(q'_i, \bar{P}(i + 1))$ .

**Notational Conventions**

1. In Hintikka’s Lemma we considered formulas with the free variables among  $X_1, \dots, X_l$ . It is trivially can be extended to formulas with free second-order variables in any finite list  $\bar{V}$ . In particular we use  $\text{Char}^k(X, Y, Z)$  for the set of Hintikka formulas of quantifier depth  $k$  with free variables  $X, Y, Z$ .
2. Whenever we deal with the synthesis problem over an  $l$ -chain  $\mathcal{M} = (\mathbb{N}, <, P_1, \dots, P_l)$ , we will often replace variables  $Z_i$  by the predicate  $P_i$ ; in particular we will write “ $\varphi(X_1, X_2, P_1, \dots, P_l)$ ” instead of “ $\varphi(X_1, X_2, Z_1, \dots, Z_l)$ ”
3. By Lemma 2.2, for every formula  $\varphi(X_1, X_2, P)$  of a quantifier depth  $n$  there is  $G \subseteq \text{Char}^n(X_1, X_2, P)$  such that  $\varphi$  is equivalent to the disjunction of all formulas from  $G$ . Moreover,  $G$  is computable from  $\varphi$ . We often identify  $\varphi$  with this set  $G$  and write “ $\mathcal{G}_G^{\mathcal{M}}$ ” instead of “ $\mathcal{G}_\varphi^{\mathcal{M}}$ ”.

**Definition 3.1.** Let  $\mathcal{M}$  be an  $l$ -chain,  $st$  be a strategy, and  $G \subseteq \text{Char}^n(X_1, X_2, \bar{P})$ .  $st$  wins  $G$  over  $\mathcal{M}$  iff the  $m$ -type of every play (on  $\mathcal{M}$ ) consistent with  $st$  is in  $G$ .

**Definition 3.2 (Game Types).** Let  $n \in \mathbb{N}$ .

**Game type of a chain.** Let  $\mathcal{M} := (A, < \bar{P})$  be an  $l$ -chain, where  $(A, <)$  is finite or of order type  $\omega$ . The  $n$ -game-type of  $\mathcal{M}$  is defined as:  
 $\text{game-type}_n(\mathcal{M}) := \{G \subseteq \text{Char}^n(X_1, X_2, \bar{P}) \mid \text{Player I wins } \mathcal{G}_G^{\mathcal{M}}\}$ .

**Formal game-type.** A formal  $(n, l)$ -game-type is an element<sup>2</sup> of  $\mathbb{P}(\mathbb{P}(\text{Char}^n(X_1, X_2, \bar{P})))$ , where  $\bar{P}$  is an  $l$ -tuple  $(P_1, \dots, P_l)$  of variables. We denote by  $\text{Gtype}_l^n$  the set of formal  $(n, l)$ -game-types.

Let  $F$  be a function from  $\mathbb{N}$  into  $\text{Gtype}_1^n$  and  $G \subseteq \text{Char}^n(X_1, X_2, P)$ . We consider the following  $\omega$ -game  $\text{Game}(F, G)$ .

<sup>2</sup> Recall that  $\mathbb{P}(A)$  stands for the set of subsets of  $A$ .

**Game**( $F, G$ ): The game has  $\omega$  rounds and it is defined as follows:

**Round i:** Player I chooses  $G_i \in F(i)$ . Then, Player II chooses  $\tau_i \in G_i$ .

**Winning conditions:** Let  $\tau_i$  ( $i \in \mathbb{N}$ ) be the sequence of moves of Player II in the play. Player I wins the play if  $\Sigma_i \tau_i \in G$ .

The following lemma is immediate:

**Lemma 3.3.** *if  $\forall i (F_1(i) \subseteq F_2(i))$ ,  $G_1 \subseteq G_2$  and Player I wins  $\text{Game}(F_1, G_1)$ , then Player I wins  $\text{Game}(F_2, G_2)$ .*

The following proposition plays an important role in our proofs:

**Proposition 3.4.** *Assume that  $F(i)$  ( $i \in \mathbb{N}$ ) is ultimately periodic. Then, it is decidable which of the players wins  $\text{Game}(F, G)$ . Moreover, the winner has a finite-memory winning strategy which is computable from  $G$ .*

## 4 Winning Strategies over Classes of Finite Chains

In the introduction we defined McNaughton’s games over expansions of  $\omega$ . In this subsection we will consider the games over expansions of finite chains. These games are defined similarly. The only difference is that these games are of finite length. The games over an  $l$ -chains with  $m$  elements have  $m$  rounds.

The following lemma says that there is a sentence which uniformly expresses that Player I has a winning strategy in the game with winning condition  $\varphi$ .

**Lemma 4.1.** *For every  $\varphi$  there is a formula  $\text{win}(\varphi)$  such that for every finite  $l$ -chain  $\mathcal{M}$ , Player I has a winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$  iff  $\mathcal{M} \models \text{win}(\varphi)$ . Furthermore,  $\text{win}(\varphi)$  is computable from  $\varphi$ .*

*Proof.* (Sketch) In [11] we proved much stronger result (Theorem 2.3 in [11]) which says that there is a formula  $\text{win}_\varphi$  such if  $\mathcal{M}$  is an expansion of  $\omega$ , then Player I has a winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$  iff  $\mathcal{M} \models \text{win}_\varphi$ . □

Recall that we identify a subset  $G$  of  $\text{Char}^m(X_1, X_2, \bar{P})$  with the disjunction  $\bigvee_{\tau \in G} \tau$ . In particular, for  $G \subseteq \text{Char}^m(X_1, X_2, \bar{P})$  we write  $\text{win}(G)$  for  $\text{win}(\bigvee G)$ .

For  $C \subseteq \mathbb{P}(\text{Char}^m(X_1, X_2, \bar{P}))$  we write  $\text{Win}(C)$  for  $\bigwedge_{G \in C} \text{win}(G)$ .  $\text{Win}(C)$  expresses that Player I has a winning strategy for every  $G \in C$ .

**Definition 4.2 (Residual).** *For  $\tau \in \text{Char}^m$  and  $G \subseteq \text{Char}^m$ , define  $\text{res}_\tau(G)$  as  $\text{res}_\tau(G) := \{\tau' \mid \tau + \tau' \in G\}$ ; define  $\text{Res}(G)$  as  $\text{Res}(G) := \{\text{res}_\tau(G) \mid \tau \in G\}$ .*

Assume that  $\rho$  is a partial play of type  $\tau$ . Player I can win  $\text{res}_\tau(G)$  after  $\rho$  iff he has a strategy which ensures that every extension of  $\rho$  wins  $G$ .

Let  $st$  be a strategy of Player I and  $\mathcal{C}$  be a class of chains. We say that  $st$  wins  $\varphi$  over  $\mathcal{C}$  iff  $st$  is a winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}$  for every  $\mathcal{M} \in \mathcal{C}$ .

**Lemma 4.3.** *Assume that  $\mathcal{M}_0$  and  $\mathcal{M}_1$  are finite  $l$ -chains. If  $\mathcal{M}_0 \models \text{win}(G)$  and  $\mathcal{M}_1 \models \text{Win}(\text{Res}(G))$  then Player I has a finite-memory strategy which wins  $G$  over the class  $\{\mathcal{M}_0 + \mathcal{M}_1 \times k \mid k \in \mathbb{N}\}$  of  $l$ -chains.*

*Proof.* Let  $k_0$  and  $k_1$  be the length of  $\mathcal{M}_0$  and  $\mathcal{M}_1$  respectively. Consider the following strategy of Player I:

Play first  $k_0$  rounds according to his winning strategy for  $\text{win}(G)$ . For every  $j \in \mathbb{N}$  if the  $m$ -type of the play after  $k_0 + jk_1$  rounds is  $\tau$  then play the next  $k_1$  rounds according to the winning strategy for  $\text{win}(\text{res}_\tau(G))$ .

It is easy to show by the induction on  $j$  that if a play  $\rho$  is played according to this strategy, then after  $k_0 + jk_1$  rounds its  $m$ -type is in  $G$ . Therefore, it is a winning strategy for Player I.

Player I needs only a finite memory to keep the information about the  $m$ -type of the play  $\tau_i$  up to each round  $i$ . After a round  $i$  he should add to  $\tau_{i-1}$  the type of the play during the round  $i$ , i.e., to add to  $\tau_{i-1}$  the  $m$ -type of one element chain expanded by the predicates  $\rho_{X_1}(i)$ ,  $\rho_{X_2}(i)$  and  $P(i)$ . Player I can calculate in a finite memory whether the current round number is  $k_0 + jk_1$  for some  $j \in \mathbb{N}$ . Hence, this strategy is a finite-memory strategy.  $\square$

**Definition 4.4.** Let  $\mathcal{M}$  be an  $l$ -chain,  $st$  be a strategy, and  $G \subseteq \text{Char}^m(X_1, X_2, \bar{P})$ .  $st$  realizes  $G$  on  $\mathcal{M}$  if it wins  $\mathcal{G}_G^M$  and for every  $m$ -type  $\tau \in G$  there is a play  $\rho$  consistent with  $st$  such that  $\text{type}_m(\mathcal{M} \cap \rho) = \tau$ ,

In other words  $st$  realizes  $G$  in  $\mathcal{M}$ , if  $st$  wins  $\mathcal{G}_G^M$  and there is no  $G_1 \subsetneq G$  such that  $st$  wins  $\mathcal{G}_{G_1}^M$ . Recall that for  $n \in \mathbb{N}$  we also denote by  $n$  the finite chain with  $n$  elements.

- Lemma 4.5.** 1. If for  $n_1 < n_2$  a strategy realizes  $G$  over chains  $n_1$  and  $n_2$ , then  $\text{Win}(\text{Res}(G))$  is satisfiable over the chain  $n_2 - n_1$ .  
 2. If for  $n_1 < n_2$  a strategy realizes  $G$  over  $n_1$  and wins  $G$  over  $n_2$ , then  $\text{Win}(\text{Res}(G))$  is satisfiable over  $n_2 - n_1$ .

*Proof.* (1) follows from (2). (2) follows from the definition of  $\text{Win}$  and Definitions 4.2 and 4.4.  $\square$

**Proposition 4.6.** For  $m \in \mathbb{N}$ , let  $n$  be an upper bound on the quantifier depth of  $\text{win}(G)$  for every  $G \subseteq \text{Char}_2^m$ , and let  $N_0$  be computed from  $n$  as in Lemma 2.8. For every  $i \in [0, N_0 - 1]$  the following are equivalent:

1. Player I has a finite-memory strategy which wins  $G$  over the class  $\{t > N_0 \mid t \bmod N_0 = i\}$  of finite chains.
2. Player I has a finite-memory strategy which wins  $G$  over an infinite subclass of  $\{t > N_0 \mid t \bmod N_0 = i\}$ .
3. There is a finite-memory strategy which realizes  $G_1 \subseteq G$  over  $n_1$  and over  $n_2$  for some  $n_2 > n_1 \geq N_0$  such that  $n_1 \bmod N_0 = n_2 \bmod N_0 = i$ .
4. There is  $G_1 \subseteq G$  such that  $N_0 \models \text{win}(G')$  for every  $G' \in \text{Res}(G_1)$ , and  $N_0 + i \models \text{win}(G_1)$ .

*Proof.* The implication (1)  $\Rightarrow$  (2) is immediate.

(2)  $\Rightarrow$  (3). If a strategy wins  $G$  over  $\mathcal{M}$  then it realizes a subset of  $G$ . Since the set of subset of  $G$  is finite, it follows that there is a subset of  $G$  which is realized infinitely often and therefore at least twice.

(3)  $\Rightarrow$  (4) follows from Lemmas 2.8 and 4.5.

(4)  $\Rightarrow$  (1) follows from Lemma 4.3.  $\square$

Proposition 4.6 is crucial for the design of our algorithm, due the decidability of (4).



## 5 Algorithm

In this section we describe an algorithm for the synthesis problem for the expansions of  $\omega$  by ER predicates. For every MLO formula  $\varphi(X_1, X_2, P)$ , first construct a set of the characteristic formulas  $G$  such that  $\varphi$  is equivalent to their disjunction and then use the following algorithm.

Synthesis algorithm over  $\mathcal{M} := (\mathbb{N}, <, P)$  where  $P$  is in ER

*Instance:*  $m \in \mathbb{N}$ .

*Task:* Find the set  $Out = \{G \subseteq Char^m(X_1, X_2, P) \mid \text{Player I has a finite-memory winning strategy in } \mathcal{G}_G^M\}$ , and for each  $G \in Out$  construct a finite-memory strategy  $st(G)$  which wins  $G$  over  $\mathcal{M}$ .

We prove the soundness of the algorithm, i.e., if  $G \in Out$ , then there is a finite-state strategy which wins  $G$  over  $\mathcal{M}$ . The proof of the reverse implication appears in the full version of this paper [13].

Let us first illustrate some ideas of the algorithm for  $\mathcal{M}_{ex} := (\mathbb{N}, <, P_{ex})$ , where  $P_{ex} := (k_l \mid l \in \mathbb{N})$  and  $k_{l+1} - k_l = l!$ . Let  $st$  be a finite-memory strategy. Note that there is  $l_{st}(m)$  such that for every  $G \subseteq Char^m$ :  $st$  wins  $G$  on  $\mathcal{M}_{ex} \upharpoonright [k_{l_{st}}, \infty)$  iff  $st$  wins  $G$  on  $\mathcal{M}_{ex} \upharpoonright [k_l, \infty)$  for every  $l \geq l_{st}$ . Recall that  $\mathcal{M} \upharpoonright I$  is the subchain of  $\mathcal{M}$  over the set  $I$ .

We can compute  $U_{st}^\infty := \{G \subseteq Char^m \mid st \text{ wins } G \text{ on } \mathcal{M}_{ex} \upharpoonright [k_l, \infty) \text{ for every } l \geq l_{st}\}$ . For  $l \in \mathbb{N}$  we can compute  $V_{st}^l := \{G \subseteq Char^m \mid st \text{ wins } G \text{ on } \mathcal{M}_{ex} \upharpoonright [0, k_l)\}$  which is a periodic sequence. From  $U_{st}^\infty$  and  $\{V_{st}^l\}_{l=0}^\infty$  we can compute  $Out_{st} := \{G \subseteq Char^m \mid st \text{ wins } G \text{ on } \mathcal{M}_{ex}\}$ . Of course, we could compute  $Out_{st}$  directly from the description of  $st$ . However, our algorithm computes  $U := \{G \subseteq Char^m \mid \text{there is a finite-memory strategy } st \text{ such that } G \in U_{st}^\infty\}$ , and the sequence  $V^l := \{G \subseteq Char^m \mid \text{there is a finite-memory strategy } st \text{ which wins } G \text{ on } \mathcal{M}_{ex} \upharpoonright [0, k_l)\}$ . The sequence  $\{V^l\}_{l=0}^\infty$  is periodic. From  $U$  and  $\{V^l\}_{l=0}^\infty$  we can compute the desirable  $Out$ .

An important property of  $P_{ex}$  is that for every  $n$ , and every  $l > n$  the distance between  $l$ -th and  $l + 1$ -th elements of  $P_{ex}$  is equal modulo  $n$  to the distance between  $n$ -th and the  $n + 1$ -th elements of  $P_{ex}$ . Usually, this property fails for ER predicates; however, the sequence of the distances modulo  $n$  behaves periodically. Our algorithm is more subtle than the above sketch for  $P_{ex}$  and relies on this periodicity.

**Conventions.** Let  $\tau(X_1, X_2)$  be an  $m$ -type for  $m > 0$ . There is a unique  $m$ -type  $\tau^*(X_1, X_2, P)$  such that  $\tau \rightarrow (\tau^*(X_1, X_2, P) \wedge \forall t \neg P(t))$ . We often will not distinguish between  $\tau$  and the corresponding  $\tau^*$ . In particular, for  $m$ -type  $\tau_1(X_1, X_2, P)$  we write  $\tau + \tau_1$  instead of  $\tau^* + \tau_1$ . We also lift this correspondence to sets of  $m$ -types;; for a set  $G \subseteq Char_2^m$  we denote by  $G$  the set  $G^* := \{\tau^* \mid \tau \in G\}$ .

Now we are going to describe our algorithm.

**Step 1**

1. Compute  $One := \{G \subseteq Char^m(X_1, X_2, P) \mid \text{Player I has a strategy which wins } G \text{ over one element structure } (0, <, \{0\})\}$ .  
 For  $G \in One$ , we denote by  $st_1(One, G)$  the corresponding winning strategy.
2. Let  $N_0$  be defined from  $m$  as in Proposition 4.6. For  $i = 0, \dots, N_0 - 1$  compute  $CWIN^i := \{G \subseteq Char^m(X_1, X_2) \mid \text{Player I has a finite-memory strategy which wins } G \text{ over the class } \{t > N_0 \mid t \bmod N_0 = i\}\}$ . This set is computable by condition (4) of Proposition 4.6.  
 For  $G \in CWIN^i$ , we denote by  $st_1(i, G)$  the corresponding finite-memory winning strategy; this strategy is computable by Lemma 4.3, since the condition (4) of Proposition 4.6 holds.

**Step 2.** Let  $\bar{k} := k_0 < k_1 < \dots < k_i < \dots$  be the enumeration of the elements of  $P$  in the increasing order. Compute  $l$  and  $p$  such that for every  $n$  greater than  $l$ :

1.  $k_{n+1} - k_n > N_0$  and
2.  $(k_{n+1} - k_n) \bmod N_0 = (k_{n+p+1} - k_{n+p}) \bmod N_0$
3. For  $j < p$ , set  $d_j := k_{l+j+1} - k_{l+j} \bmod N_0$ .

(To compute such  $l$  and  $p$  we need our assumption that  $P \in ER$ .)

**Step 3.** Let  $F : \mathbb{N} \rightarrow \text{Gtype}^m(X_1, X_2, P)$  be defined as follows:

$$F(i) = \begin{cases} One & \text{if } i \text{ is even} \\ CWIN^{d_j} & \text{if } i = 2s + 1 \text{ and } s \bmod p = j \end{cases}$$

Note that  $F$  is a periodic sequence.

Use Proposition 3.4 to compute the set  $U := \{G \subseteq Char^m(X_1, X_2, P) \mid \text{Player I has a finite-memory strategy which wins } Game(F, G)\}$ .

For  $G \in U$ , we denote by  $st_{main}(F, G)$  the corresponding finite-memory winning strategy.

Now, for  $G \in U$  we describe a finite-memory strategy  $st_3(F, G)$  which wins  $G$  over the class  $\{\mathcal{M}_i := \mathcal{M} \upharpoonright [k_{l+pi}, \infty) \mid i \in \mathbb{N}\}$  of chains.

We organize our description of how strategy  $st_3(F, G)$  behaves on  $\mathcal{M}_i := \mathcal{M} \upharpoonright [k_{l+pi}, \infty)$  in sessions. For  $s \in \mathbb{N}$ , the session  $2s$  is played on the one element subchain of  $\mathcal{M}_i$  isomorphic to  $(0, < \{0\})$ ; the session  $2s + 1$  will be played on the subchain  $\mathcal{M} \upharpoonright (k_{l+pi+s}, k_{l+pi+s+1})$  which is isomorphic to the  $(k_{l+pi+s+1} - k_{l+pi+s})$ -element chain expanded by the empty predicate.

*Session 0.* Let  $G_0$  be the first move of  $st_{main}(F, G)$ . Then Player I will move according to his winning strategy in  $st_1(One, G_0)$ . After a move of Player II, the  $m$ -type of the partial play  $\rho_0$  is some  $\tau_0 \in G_0$ .

*Session  $2s + 1$ .* Let  $G_{2s+1}$  be the move of Player I according to  $st_{main}(F, G)$  after a partial play  $G_0\tau_0G_1\tau_1 \dots G_{2s}\tau_{2s}$ . Then Player I will play according to his strategy in  $st_1(d_{(s \bmod p)}, G_{2s+1})$  until he reads one on  $P$  (recall that  $d_j$ , were

defined in Step 2). At this point the type of a subplay  $\rho_{2s+1}$  during this round will be  $\tau_{2s+1} \in G_{2s+1}$ .

*Session 2s.* ( $s > 0$ ) Let  $G_{2s}$  be the move of Player I according to  $st_{main}(F, G)$  after a partial play  $G_0\tau_0G_1\tau_1 \dots G_{2s-1}\tau_{2s-1}$ . Player I will move according to his winning strategy in  $st_1(One, G_{2s})$ . After a move of Player II, the  $m$ -type of the partial play  $\rho_{2s}$  during this session will be some  $\tau_{2s} \in G_{2s}$ .

Observe that this is indeed a finite-memory strategy. Like in the proof of Lemma 4.3, Player I can compute in a finite memory at each session  $s$  the  $m$ -type  $\tau_s$  of the subplay during session  $s$ , and then after this session to supply only this  $m$ -type to  $st_{main}(F, G)$  (and not the whole history  $G_0\tau_0 \dots G_s\tau_s$ ).

This strategy wins  $G$  because the sequence  $G_0\tau_0 \dots G_s\tau_s \dots$  played over the sessions is consistent with the winning strategy  $st_{main}(F, G)$  in  $Game(F, G)$ .

**Step 4.** We are going to compute the set  $V := \{G \subseteq Char^m(X_1, X_2, P) \mid \text{Player I has a strategy which wins } G \text{ over } \mathcal{M} \upharpoonright [0, k_{l+pi}) \text{ for some } i \in \mathbb{N}\}$ .

Let  $n$  be the quantifier depth of  $win(G)$ .

By our choice of  $N_0, l$  and  $p$  (in Step 1 and Step 2) we know that for every  $i$ :

$$\mathcal{M} \upharpoonright [k_{l+i}, k_{l+i+1}) \equiv^n \mathcal{M} \upharpoonright [k_{l+i+p}, k_{l+i+1+p})$$

Hence, for every  $i$ :

$$\begin{aligned} \mathcal{M} \upharpoonright [k_{l+pi}, k_{l+pi+p}) &= \sum_{s=0}^{p-1} \mathcal{M} \upharpoonright [k_{l+pi+s}, k_{l+pi+s+1}) \equiv^n \\ &\equiv^n \sum_{s=0}^{p-1} \mathcal{M} \upharpoonright [k_{l+s}, k_{l+s+1}) = \mathcal{M} \upharpoonright [k_l, k_{l+p}) \end{aligned}$$

Let  $N_1 := N_1(n)$  be defined as in Lemma 2.9. From the above equivalence, Lemma 2.9 and Proposition 2.5, it follows that for every  $i$  there is  $j \leq N_1$  such that

$$\mathcal{M} \upharpoonright [k_l, k_{l+pi}) \equiv^n \mathcal{M} \upharpoonright [k_l, k_{l+pj})$$

and hence,  $\mathcal{M} \upharpoonright [0, k_{l+pi}) \equiv^n \mathcal{M} \upharpoonright [0, k_{l+pj})$ .

Therefore,  $V = \{G \subseteq Char^m(X_1, X_2, P) \mid \mathcal{M} \upharpoonright [0, k_{l+pj}) \models win(G) \text{ for some } j \leq N_1\}$ . To compute the right hand side we need to check a finite set of games over finite chains. Hence, this is computable and therefore,  $V$  is computable.

For  $G \in V$ , let  $l_G \leq N_1$  be such that  $\mathcal{M} \upharpoonright [0, k_{l+pl_G}) \models win(G)$  and let  $st_4(V, G)$  be the corresponding strategy which wins  $G$  over  $\mathcal{M} \upharpoonright [0, k_{l+pl_G})$ .

**Step 5.** Output  $Out := \{G \subseteq Char^m(X, Y, P) \mid \exists G_1 \in V \text{ such that } res_\tau(G) \in U \text{ for every } \tau \in G_1\}$ .

For every  $G \in Out$  we describe a finite-memory strategy  $st(G)$  which wins  $G$  over  $\mathcal{M}$ . Assume  $G \in Out$  and let  $G_1 \in V$  be such that  $res_\tau(G) \in U$  for every  $\tau \in G_1$ . Since  $G_1 \in V$ , there is  $l_{G_1}$  and a strategy  $st_4(V, G_1)$  which wins  $G_1$  over  $\mathcal{M} \upharpoonright [0, k_{l+pl_{G_1}})$ .

Player I will play the first  $l+p \times l_{G_1}$  rounds according to this winning strategy. Let  $\rho$  be a play according to this strategy, and let  $\tau$  be its  $m$ -type and let  $G_2 = res_\tau(G)$ . The rest of the game Player I will play according to his finite-memory

strategy  $st_3(F, G_2)$  computed in the Step 3 Clearly, the described strategy is a finite-memory strategy.

The  $m$ -type of the whole play is in  $\tau + G_2 = G$ . Therefore, the described strategy is winning in  $\mathcal{G}_G^M$ . This completes the description of our algorithm and the proof that if  $G \in Out$ , then Player I has a finite-memory winning strategy in  $\mathcal{G}_G^M$ .

## 6 Further Results and Open Questions

We proved that the finite-memory synthesis problem is decidable for the expansions of  $\omega$  by the predicates from  $ER$ . In [12] it was proved that the decidability of the monadic theory of  $\mathcal{M}$  is equivalent to the decidability of the recursive synthesis problem for  $\mathcal{M}$ .

The question whether the decidability of the monadic theory of  $\mathcal{M}$  implies the decidability of the finite-memory synthesis problem for  $\mathcal{M}$  remains open.

A natural question to consider is the synthesis problem for strategies between finite-memory and recursive ones, e.g., the strategies computable by push-down automata.

There are some minor modifications of the McNaughton games to the games with look-ahead. Let  $\mathcal{M} = (\mathbb{N}, <, P)$  be the expansion of  $\omega$  by a unary predicate  $P$ . Let  $h_1, h_2$  be integers (look-ahead) of the players. Let  $\varphi(X_1, X_2, Z)$  be a formula. The game  $\mathcal{G}_\varphi^M(h_1, h_2)$  with look-ahead  $h_1$  for Player I and look-ahead  $h_2$  for Player II is defined as follows. The game is played by two players and each of its plays has  $\omega$  rounds.

1. At round  $i \in \mathbb{N}$ : first, Player I chooses  $\rho_{X_1}(i) \in \{0, 1\}$ ; then, Player II chooses  $\rho_{X_2}(i) \in \{0, 1\}$ . Player I can observe whether  $i + h_1 \in P$  and Player II can observe whether  $i + h_2 \in P$
2. By the end of the play two predicates  $\rho_{X_1}, \rho_{X_2} \subseteq \mathbb{N}$  have been constructed.
3. Then, Player I wins the play if  $\mathcal{M} \models \varphi(\rho_{X_1}, \rho_{X_2}, P)$ ; otherwise, Player II wins the play.

The proof of the next proposition is similar to the proof of Theorem [4].

**Proposition 6.1.** *Let  $P$  be an ER predicate, and  $h_1, h_2$  integers and let  $\mathcal{M} = (\mathbb{N}, <, P)$ . There is an algorithm that for every MLO formula  $\varphi(X_1, X_2, Z)$  decides whether Player I has a finite-memory winning strategy in  $\mathcal{G}_\varphi^M(h_1, h_2)$ , and if so, constructs such a strategy.*

It is easy to modify our proofs and to show that it is decidable whether Player II has a finite-memory winning strategy.

Section [1] (page [426]) gives an example of the game  $\mathcal{G}_\varphi^{M_{fac}}$  where Player I has a winning strategy, yet he has no finite-memory winning strategy. Note that for this particular game, Player I has a finite-memory one-look-ahead winning strategy, i.e., he has a finite-memory winning strategy in  $\mathcal{G}_\varphi^{M_{fac}}(1, h_2)$  for every  $h_2$ .

Relying on the definability results in [12] we can prove the following Proposition.

**Proposition 6.2 (Determinacy for look-ahead finite-memory strategy).**

Let  $P$  be an ER predicate, and let  $\mathcal{M} = (\mathbb{N}, <, P)$ . For every MLO formula  $\varphi(X_1, X_2, Z)$  there is  $h$  such that one of the players has a finite-memory winning strategy in  $\mathcal{G}_\varphi^{\mathcal{M}}(h, h)$ . Furthermore, there is an algorithm that computes such  $h$  and a finite-memory winning strategy for the winner in  $\mathcal{G}_\varphi^{\mathcal{M}}(h, h)$ .

It is plausible that in our proofs the compositional methods can be hidden and a presentation can be given based on automata theoretic concepts. The logical  $n$ -types can be replaced by “ $n$ -types”, using semigroups or automata rather than formulas to describe properties of words. The only place where automata based techniques might fail is in the proof of Proposition [3.4](#).

## References

1. Büchi, J.R.: On a decision method in restricted second order arithmetic. In: Nagel, E., et al. (eds.) Proc. International Congress on Logic, Methodology and Philosophy of Science, pp. 1–11. Stanford University Press (1960)
2. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finitestate strategies. Transactions of the AMS 138(27), 295–311 (1969)
3. Carton, O., Thomas, W.: The Monadic Theory of Morphic Infinite Words and Generalizations. Inf. Comput. 176(1), 51–65 (2002)
4. Church, A.: Logic, Arithmetic and Automata. In: Proc. Internat. Cong. Math. 1963, Almqvist and Wilksells, Uppsala (1963)
5. Elgot, C., Rabin, M.O.: Decidability and Undecidability of Extensions of Second (First) Order Theory of (Generalized) Successor. J. Symb. Log. 31(2), 169–181 (1966)
6. Grädel, E., Thomas, W., Wilke, T.: Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
7. Gurevich, Y.: Monadic second-order theories. In: Barwise, J., Feferman, S. (eds.) Model-Theoretic Logics, pp. 479–506. Springer, Heidelberg (1985)
8. Feferman, S., Vaught, R.L.: The first-order properties of products of algebraic systems. Fundamenta Mathematica 47, 57–103 (1959)
9. McNaughton, R.: Finite-state infinite games. Project MAC Rep. MIT, Cambridge (1965)
10. Perrin, D., Pin, J.E.: Infinite Words Automata, Semigroups, Logic and Games. In: Pure and Applied Mathematics, vol. 141. Elsevier, Amsterdam (2004)
11. Rabinovich, A.: On decidability of Monadic logic of order over the naturals extended by monadic predicates. Information and Computation 205(6), 870–889 (2007)
12. Rabinovich, A.: Church Synthesis Problem with Parameters. Logical Methods in Computer Science 3(4:9), 1–24 (2007)
13. Rabinovich, A.: Decidable Extensions of Church’s Problem (full version) (2009), <http://www.cs.tau.ac.il/~rabinoac/sl09a-full>
14. Rabinovich, A., Thomas, W.: Decidable Theories of the Ordering of Natural Numbers with Unary Predicates. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 562–574. Springer, Heidelberg (2006)
15. Robinson, R.M.: Restricted Set-Theoretical Definitions in Arithmetic. Proceedings of the AMS 9(2), 238–242 (1958)

16. Semenov, A.: Logical theories of one-place functions on the set of natural numbers. *Mathematics of the USSR - Izvestia* 22, 587–618 (1984)
17. Shelah, S.: The monadic theory of order. *Ann. of Math.* 102, 379–419 (1975)
18. Siefkes, D.: The recursive sets in certain monadic second order fragments of arithmetic. *Arch. Math. Logik* 17, 71–80 (1975)
19. Thomas, W.: Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In: Mycielski, J., Rozenberg, G., Salomaa, A. (eds.) *Structures in Logic and Computer Science*. LNCS, vol. 1261, pp. 118–143. Springer, Heidelberg (1997)
20. Trakhtenbrot, B.A.: *Finite automata and the logic of one-place predicates* (Russian version 1961). *AMS Transl.* 59, 23–55 (1966)

# Nested Hoare Triples and Frame Rules for Higher-Order Store

Jan Schwinghammer<sup>1</sup>, Lars Birkedal<sup>2</sup>, Bernhard Reus<sup>3</sup>, and Hongseok Yang<sup>4</sup>

<sup>1</sup> Saarland University, Saarbrücken

<sup>2</sup> IT University of Copenhagen

<sup>3</sup> University of Sussex, Brighton

<sup>4</sup> Queen Mary University of London

**Abstract.** Separation logic is a Hoare-style logic for reasoning about programs with heap-allocated mutable data structures. As a step toward extending separation logic to high-level languages with ML-style general (higher-order) storage, we investigate the compatibility of nested Hoare triples with several variations of higher-order frame rules.

The interaction of nested triples and frame rules can be subtle, and the inclusion of certain frame rules is in fact unsound. A particular combination of rules can be shown consistent by means of a Kripke model where worlds live in a recursively defined ultrametric space. The resulting logic allows us to elegantly prove programs involving stored code. In particular, it leads to natural specifications and proofs of invariants required for dealing with recursion through the store.

## 1 Introduction

Many programming languages permit not only the storage of first-order data, but also forms of higher-order store. Examples are code pointers in C, and ML-like general references. It is therefore important to have modular reasoning principles for these language features. Separation logic is an effective formalism for modular reasoning about pointer programs, in low-level C-like programming languages and, more recently, also in higher-level languages [6,7,9,13]. However, its assertions are usually limited to talk about first-order data.

In previous work, we have begun the study of separation logic for languages with higher-order store [2,12]. A challenge in this research is the combination of proof rules from separation logic for modular reasoning, and proof rules for code stored on the heap. Ideally, a program logic for higher-order store provides sufficiently expressive proof rules that, e.g., can deal with recursion through the store, and at the same time interact well with (higher-order) frame rules, which enable modular program verification.

Our earlier work shows that separation logic is consistent with higher-order store. However, the formulation of [2,12] has a shortcoming: code is treated like any other data in that assertions can only mention concrete commands. For modular reasoning, it is clearly desirable to abstract from particular code and instead (partially) specify its behaviour. For example, when verifying mutually

recursive procedures on the heap, one would like to consider each procedure in isolation, relying on properties but not the implementations of the others. The recursion rule in [212] does not achieve this. A second, and less obvious consequence of lacking behavioural specifications for code in assertions is that one cannot take full advantage of the frame rules of separation logic. For instance, the language in [2] can simulate higher-order procedures by passing arguments through the heap, but the available (higher-order) frame rules are not useful here because an appropriate specification for this encoding is missing.

In this article, we address these shortcomings by investigating a program logic in which stored code can be specified using Hoare triples, i.e., an assertion language with *nested triples*. This is an obvious idea, but the combination of nested triples and frame rules turns out to be tricky: the most natural combination turns out to be unsound.

The main technical contributions of this paper are therefore: (1) the observation that certain “deep” frame rules can be unsound, (2) the suggestion of a “good” combination of nested Hoare triples and frame rules, and (3) the verification of those by means of an elegant Kripke model, where the worlds are themselves world-dependent sets of heaps. The worlds form a complete metric space and (the denotation of) the tensor  $\otimes$ , needed to generically express higher-order frame rules, is contractive; as a consequence, our logic permits recursively defined assertions.

After introducing the syntax of language and assertions in Section 2 we discuss some unsound combinations of rules in Section 3, which also contains the suggested set of rules for our logic. The soundness of the logic is then shown in Section 4.

## 2 Syntax of Programs and Assertions

We consider a simple imperative programming language extended with operations for stored code and heap manipulation. The syntax of the language is shown in Fig. 1. The expressions in the language are integer expressions, variables, and the quote expression ‘ $C$ ’ for representing an unevaluated command  $C$ . The integer or code value denoted by expression  $e_1$  can be stored in a heap cell  $e_0$  using  $[e_0]:=e_1$ , and this stored value can later be looked up and bound to the (immutable) variable  $y$  by  $\text{let } y=[e_0] \text{ in } D$ . In case the value stored in cell  $e_0$  is code ‘ $C$ ’, we can run (or “evaluate”) this code by executing  $\text{eval}[e_0]$ . Our language also provides constructs for allocating and disposing heap cells such as  $e_0$  above. We point out that, as in ML, all variables  $x, y, z$  in our language are *immutable*, so that once they are bound to a value, their values do not change. This property of the language lets us avoid side conditions on variables when studying frame rules. Finally, we do not include while loops in our language, since they can be expressed by stored code (using Landin’s knot).

Our assertion language is standard first-order intuitionistic logic, extended with separating connectives *emp*,  $*$ , the points-to predicate  $\mapsto$  [13], and with recursively defined assertions. The syntax of assertions appears in Fig. 1. Each



---

$d \in Exp ::= 0 \mid -1 \mid 1 \mid \dots \mid d_1+d_2 \mid \dots \mid x$	integer expressions, variable
$\quad \mid 'C'$	quote (command as expression)
$C \in Com ::= [e_1]:=e_2 \mid \text{let } y=[e] \text{ in } C \mid \text{eval } [e]$	assignment, lookup, unquote
$\quad \mid \text{let } x=\text{new } (e_1, \dots, e_n) \text{ in } C \mid \text{free } e$	allocation, disposal
$\quad \mid \text{skip} \mid C_1; C_2 \mid \text{if } (e_1=e_2) \text{ then } C_1 \text{ else } C_2$	no op, sequencing, conditional
$P, Q \in Assn ::= \text{false} \mid \text{true} \mid P \vee Q \mid P \wedge Q \mid P \Rightarrow Q$	intuitionistic-logic connectives
$\quad \mid \forall x.P \mid \exists x.P \mid \text{int}(e) \mid e_1=e_2 \mid e_1 \leq e_2$	quantifiers, atomic formulas
$\quad \mid e_1 \mapsto e_2 \mid \text{emp} \mid P * Q$	separating connectives
$\quad \mid \{P\}e\{Q\} \mid P \otimes Q \mid \dots$	Hoare triple, invariant extension

---

**Fig. 1.** Syntax of expressions, commands and assertions

assertion describes a property of states, which consist of an immutable stack and a mutable heap. Formula  $\text{emp}$  means that the heap component of the state is empty, and  $P * Q$  means that the heap component can be split into two, one satisfying  $P$  and the other satisfying  $Q$ , both evaluated with respect to the same stack. The points-to predicate  $e_0 \mapsto e_1$  states that the heap component consists of only one cell  $e_0$  whose contents is (some approximation of)  $e_1$ .

One interesting aspect of our assertion language is that it includes Hoare triples  $\{P\}e\{Q\}$  and invariant extensions  $P \otimes Q$ ; previous work [42] does not treat them as assertions, but as so-called *specifications*, which form a different syntactic category. Intuitively,  $\{P\}e\{Q\}$  means that  $e$  denotes code satisfying  $\{P\}\_-\{Q\}$ , and  $P \otimes Q$  denotes a modification of  $P$  where all the pre- and post-conditions of triples inside  $P$  are  $*$ -extended with  $Q$ . For instance, the assertion  $(\exists k. (1 \mapsto k) \wedge \{\text{emp}\}k\{\text{emp}\}) \otimes (2 \mapsto 0)$  is equivalent to  $(\exists k. (1 \mapsto k) \wedge \{2 \mapsto 0\}k\{2 \mapsto 0\})$ . This assertion says that cell 1 is the only cell in the heap and it stores code  $k$  that satisfies the triple  $\{2 \mapsto 0\}\_-\{2 \mapsto 0\}$ . This intuition of the  $\otimes$  operator can also be seen in the set of axioms in Fig. 2, which let us distribute  $\otimes$  through all the constructs of the assertion language.

Note that since triples are assertions, they can appear in pre- and post-conditions of triples. This *nested* use of triples is useful in reasoning, because it allows one to specify stored code behaviourally, in terms of properties that it satisfies. Another important consequence of having these new constructs as assertions is that they allow us to study proof rules for exploiting locality of stored code systematically, as we will describe shortly.

The last case  $\dots$  in Fig. 2 represents pre-defined assertions, including recursively defined ones. In particular, it contains all recursively defined assertions

---


$$\begin{aligned}
 & P \circ R \stackrel{\text{def}}{=} (P \otimes R) * R \\
 \{P\}e\{Q\} \otimes R & \Leftrightarrow \{P \circ R\}e\{Q \circ R\} \quad (\kappa x.P) \otimes R \Leftrightarrow \kappa x.(P \otimes R) \quad (\kappa \in \{\forall, \exists\}, x \notin \text{fv}(R)) \\
 (P \otimes R) \otimes R' & \Leftrightarrow P \otimes (R \circ R') \quad (P \oplus Q) \otimes R \Leftrightarrow (P \otimes R) \oplus (Q \otimes R) \quad (\oplus \in \{\Rightarrow, \wedge, \vee, *\}) \\
 P \otimes R & \Leftrightarrow P \quad (P \text{ is one of } \text{true}, \text{false}, \text{emp}, e=e', e \mapsto e' \text{ and } \text{int}(e))
 \end{aligned}$$


---

**Fig. 2.** Axioms for distributing  $\_ \otimes R$

of the form  $R = P \otimes R$ , where  $R$  does not appear in  $P$ . These assertions are always well-defined (because  $\otimes$  is “contractive” in its second argument, as shown in Lemma 4), and they let us reason about self-applying stored code, without using specialized rules 2. We will say more about the use of recursively defined predicates and their existence in Sections 3 and 4<sup>1</sup>

We shall make use of two abbreviations. The first is  $P \circ R$ , which stands for  $(P \otimes R) * R$  (already used in Fig. 2). This abbreviation is often used to add an invariant  $R$  to a Hoare triple  $\{P\}e\{Q\}$ , so as to obtain  $\{P \circ R\}e\{Q \circ R\}$ . We use  $\circ$  instead of  $*$  here to extend not only  $P$  by  $R$  but also ensure, via  $\otimes$ , that all Hoare triples nested inside  $P$  preserve  $R$  as an invariant. The  $\circ$  operator has been introduced in 11, where it is credited to Paul-André Mellies and Nicolas Tabareau. The second abbreviation is for the “ $\mapsto$ ” operator:  $e_1 \mapsto P[e_2] \stackrel{\text{def}}{=} e_1 \mapsto e_2 \wedge P[e_2]$  and  $e_1 \mapsto P[\_] \stackrel{\text{def}}{=} \exists x. e_1 \mapsto P[x]$ . Here  $x$  is a fresh (logic) variable and  $P[\_]$  is an assertion with an expression hole, such as  $\{Q\} \cdot \{R\}$ ,  $\text{int}(\cdot)$ ,  $\cdot = e$  or  $\cdot \leq e$ .

### 3 Proof Rules for Higher-Order Store

In our formal setting, reasoning about programs is done by deriving the judgement  $\Gamma \vdash P$ , where  $P$  is an assertion expressing properties of programs and  $\Gamma$  is a list of variables containing all the free variables in  $P$ . For instance, to prove that command  $C$  stores at cell 1 the code that initializes cell 10 to 0, 2 we need to derive  $\Gamma \vdash \{1 \mapsto \_ \} C \{1 \mapsto \{10 \mapsto \_ \} \_ \{10 \mapsto 0\}\}$ . In this section, we describe inference rules and axioms for assertions that let one efficiently reason about programs. We focus on those related to higher-order store.

**Standard proof rules.** The proof rules include the standard proof rules for intuitionistic logic and the logic of bunched implications 8 (not repeated here). Moreover, the proof rules include variations of standard separation logic proof rules, see Fig. 3<sup>3</sup>. The figure neither includes the rule for executing stored code with  $\text{eval}[e]$  nor the frame rule for adding invariants to triples; the reason for this omission is that these two rules raise nontrivial issues in the presence of higher-order store and nested triples, as we will now discuss.

**Frame rule for higher-order store.** The frame rule is the most important rule in separation logic, and it formalizes the intuition of local reasoning, where

<sup>1</sup> More generally, we may need to solve mutually recursive assertions  $\langle R_1, \dots, R_n \rangle = \langle P_1 \otimes (R_1 * \dots * R_n), \dots, P_n \otimes (R_1 * \dots * R_n) \rangle$  in order to deal with mutually recursive stored procedures. For brevity we omit formal syntax for such; see Theorem 12 for the semantic existence proof.

<sup>2</sup> One concrete example of such a command  $C$  is  $[1] := [10] := 0$ .

<sup>3</sup> The UPDATE, FREE and SKIP rules in the figure are not the usual small axioms in separation logic, since they contain assertion  $P$  describing the unchanged part. Since we have the standard frame rule for  $*$ , we could have used small axioms instead here. But we chose not to do this, because the current non-small axioms make it easier to follow our discussions on frame rules and higher-order store in the next subsection.

---

<b>DEREF</b>	$\frac{\Gamma, x \vdash \{P * e \mapsto x\} \{C\} \{Q\}}{\Gamma \vdash \{\exists x. P * e \mapsto x\} \{ \text{let } x = [e] \text{ in } C \} \{Q\}} \quad (x \notin \text{fv}(e, Q))$	$\frac{\text{UPDATE}}{\Gamma \vdash \{e \mapsto \_ * P\} \{ [e] := e_0 \} \{ e \mapsto e_0 * P \}}$
<b>NEW</b>	$\frac{\Gamma, x \vdash \{P * x \mapsto e\} \{C\} \{Q\}}{\Gamma \vdash \{P\} \{ \text{let } x = \text{new } e \text{ in } C \} \{Q\}} \quad (x \notin \text{fv}(P, e, Q))$	$\frac{\text{FREE}}{\Gamma \vdash \{e \mapsto \_ * P\} \{ \text{free}(e) \} \{P\}}$
<b>SKIP</b>	$\frac{}{\Gamma \vdash \{P\} \{ \text{skip} \} \{P\}}$	$\frac{\text{SEQ} \quad \Gamma \vdash \{P\} \{C\} \{R\} \quad \Gamma \vdash \{R\} \{D\} \{Q\}}{\Gamma \vdash \{P\} \{C; D\} \{Q\}}$
<b>IF</b>	$\frac{\Gamma \vdash \{P \wedge e_0 = e_1\} \{C\} \{Q\} \quad \Gamma \vdash \{P \wedge e_0 \neq e_1\} \{D\} \{Q\}}{\Gamma \vdash \{P\} \{ \text{if } (e_0 = e_1) \text{ then } C \text{ else } D \} \{Q\}}$	$\frac{\text{CONSEQ} \quad \Gamma \vdash P' \Rightarrow P \quad \Gamma \vdash Q \Rightarrow Q'}{\Gamma \vdash \{P\} e \{Q\} \Rightarrow \{P'\} e \{Q'\}}$

---

**Fig. 3.** Proof rules from separation logic

proofs focus on the footprints of the programs we verify. Developing a similar rule in our setting is challenging, because nested triples allow for several choices regarding the shape of the rule. Moreover, the recursive nature of the higher-order store muddies the water and it is difficult to see which choices actually make sense (i.e., do not lead to inconsistency).

To see this problem more clearly, consider the rules below:

$$\frac{\Gamma \vdash \{P\} e \{Q\}}{\Gamma \vdash \{P \square R\} e \{Q \square R\}} \quad \text{and} \quad \frac{}{\Gamma \vdash \{P\} e \{Q\} \Rightarrow \{P \square R\} e \{Q \square R\}} \quad \text{for } \square \in \{*, \circ\}.$$

Note that we have four choices, depending on whether we use  $\square = *$  or  $\square = \circ$  and on whether we have an inference rule or an axiom. If we choose  $*$  for  $\square$ , we obtain *shallow* frame rules that add  $R$  to the outermost triple  $\{P\} e \{Q\}$  only; they do not add  $R$  in nested triples appearing in pre-condition  $P$  and post-condition  $Q$ . On the other hand, if we choose  $\circ$  for  $\square$ , since  $(A \circ R) = (A \otimes R * R)$ , we obtain *deep* frame rules that add the invariant  $R$  not just to the outermost triple but also to all the nested triples in  $P$  and  $Q$ .

The distinction between inference rule and axiom has some bearing on where the frame rule can be applied. With the axiom version, we can apply the frame rule not just to valid triples, but also to nested triples appearing in pre- or post-conditions. With the inference rule version, however, we cannot add invariants to (or remove invariants from) nested triples.

Ideally, we would like to have the axiom versions of the frame rules for both  $*$  and  $\circ$ . Unfortunately, this is not possible for  $\circ$ . Adding the axiom version for  $\circ$  makes our logic unsound. The source of the problem is that with the axiom version for  $\circ$ , one can add invariants selectively to some, but not necessarily all, nested triples. This flexibility can be abused to derive incorrect conclusions.

Concretely, with the axiom version for  $\circ$  we can make the following derivation:

$$\frac{\frac{\Gamma \vdash \{P \circ S\}e\{Q \circ S\}}{\Gamma \vdash \{P\}e\{Q\} \otimes S} \otimes\text{-DIST} \quad \frac{\Gamma \vdash \{P\}e\{Q\} \Rightarrow \{P \circ R\}e\{Q \circ R\}}{\Gamma \vdash \{P\}e\{Q\} \otimes S \Rightarrow \{P \circ R\}e\{Q \circ R\} \otimes S} \text{FRAME}}{\Gamma \vdash \{P\}e\{Q\} \otimes S \Rightarrow \{P \circ R\}e\{Q \circ R\} \otimes S} \otimes\text{-MONO} \quad \text{MODUSPON}}{\frac{\Gamma \vdash \{P \circ R\}e\{Q \circ R\} \otimes S}{\Gamma \vdash \{(P \circ R) \circ S\}e\{(Q \circ R) \circ S\}} \otimes\text{-DIST}}$$

Here we use the distribution axioms for  $\otimes$  in Fig. 2 and the monotonicity of  $-\otimes R$ . This derivation means that when adding  $R$  to nested triples, we can skip the triples in the  $S$  part of the pre- and post-conditions of  $\{P \circ S\}e\{Q \circ S\}$ . This flexibility leads to the unsoundness:

**Proposition 1.** *Adding the axiom version of the frame rule for  $\circ$  renders our logic unsound.*

*Proof.* Let  $R$  be the predicate defined by  $R = (3 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \}) \otimes R$ . Then, we can derive the triple:

$$(\dagger) \quad \frac{k \vdash \{2 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \} \circ R\}k\{2 \mapsto \_ \_ \circ R\}}{k \vdash \{(2 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \} \circ 1 \mapsto \_ \_ \) \circ R\}k\{(2 \mapsto \_ \_ \circ 1 \mapsto \_ \_ \) \circ R\}} \quad \frac{}{k \vdash \{2 \mapsto \text{'free}(-1)\}' * 1 \mapsto \_ \_ * R\}k\{2 \mapsto \_ \_ * 1 \mapsto \_ \_ * 3 \mapsto \{1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * R\}\}}$$

Here the first step uses the derivation above for adding invariants selectively, and the last step uses the Consequence axiom with the below two implications:

$$\begin{aligned} 2 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \} \circ 1 \mapsto \_ \_ \circ R &\iff 2 \mapsto \{1 \mapsto \_ \_ * 1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * 1 \mapsto \_ \_ * R\} * 1 \mapsto \_ \_ * R \\ &\iff 2 \mapsto \{\text{false}\} \_ \{\text{false}\} * 1 \mapsto \_ \_ * R \\ &\iff 2 \mapsto \text{'free}(-1)\}' * 1 \mapsto \_ \_ * R \\ 2 \mapsto \_ \_ \circ 1 \mapsto \_ \_ \circ R &\iff 2 \mapsto \_ \_ * 1 \mapsto \_ \_ * R \iff 2 \mapsto \_ \_ * 1 \mapsto \_ \_ * ((3 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \}) \otimes R) \\ &\iff 2 \mapsto \_ \_ * 1 \mapsto \_ \_ * 3 \mapsto \{1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * R\}. \end{aligned}$$

Consider  $C \equiv \text{let } x=[2] \text{ in } [3]:=x$ . When  $P[y] \equiv \{1 \mapsto \_ \_ \}y\{1 \mapsto \_ \_ \} \otimes R$ ,

$$\frac{\vdash \{2 \mapsto P[\_] * 3 \mapsto P[\_]\}'C'\{2 \mapsto P[\_] * 3 \mapsto P[\_]\}}{\vdash \{2 \mapsto \{1 \mapsto \_ \_ \} \_ \{1 \mapsto \_ \_ \} \circ R\}'C'\{2 \mapsto \_ \_ \circ R\}}$$

Now we instantiate  $k$  in  $(\dagger)$  with  $C$ , discharge the premise of the resulting derivation with the above derivation for  $C$ , and obtain

$$\frac{\vdash \{2 \mapsto \text{'free}(-1)\}' * 1 \mapsto \_ \_ * R\}'C'\{2 \mapsto \_ \_ * 1 \mapsto \_ \_ * 3 \mapsto \{1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * R\}\}}{\vdash \{2 \mapsto \text{'free}(-1)\}' * 1 \mapsto \_ \_ * R\}'C'; \text{free}(2)\{1 \mapsto \_ \_ * 3 \mapsto \{1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * R\}\}}$$

Here the second step uses the rules FREE and SEQ in Fig. 3. But the post-condition of the conclusion here is equivalent to  $1 \mapsto \_ \_ * R$  by the definition of  $R$  and the distribution axioms for  $\otimes$ . Thus, as our rule for eval will show later, we should be able to conclude that

$$\vdash \{2 \mapsto \text{'free}(-1)\}' * 1 \mapsto \_ \_ * R\}'C'; \text{free}(2); \text{eval } [3]'\{1 \mapsto \_ \_ * 3 \mapsto \{1 \mapsto \_ \_ * R\} \_ \{1 \mapsto \_ \_ * R\}\}$$

However, since  $-1$  is not even an address, the program  $(C; \text{free}(2); \text{eval}[3])$  always faults, contradicting the requirement of separation logic that proved programs run without faulting.  $\square$

Notice that in the derivation above it is essential that  $R$  is a recursively defined assertion, otherwise we would not obtain that 2 and 3 point to code satisfying the same  $P$ .

Fortunately, the second best choice leads to a consistent proof system:

**Proposition 2.** *Both the inference rule version of the frame rule for  $\circ$  and the axiom version for  $*$  are sound in our semantics, which will be given in Section 4. In fact, the semantics also validates the following more general version of the rule for  $\circ$ :*

$$\frac{\Gamma \vdash P}{\Gamma \vdash P \otimes R}$$

**Rule for executing stored code.** An important and challenging part of the design of a program logic for higher-order store is the design of a proof rule for  $\text{eval}[e]$ , the command that executes code stored at  $e$ . Indeed, the rule should overcome two challenges directly related to the recursive nature of higher-order store: (1) implicit recursion through the store (i.e., Landin’s knot), and (2) extensional specifications of stored code.

These two challenges are addressed, using the expressiveness of our assertion language, by the following rule for  $\text{eval}[e]$ :

$$\frac{\text{EVAL} \quad \Gamma, k \vdash R[k] \Rightarrow \{P * e \mapsto R[\_]\}k\{Q\}}{\Gamma \vdash \{P * e \mapsto R[\_]\}'\text{eval}[e]\{Q\}}$$

This rule states that in order to prove  $\{P * e \mapsto R[\_]\}'\text{eval}[e]\{Q\}$  for executing stored code in  $[e]$  under the assumption that  $e$  points to arbitrary code  $k$  (expressed by the  $\_$  which is an abbreviation for  $\exists k. e \mapsto R[k]$ ), it suffices to show that the specification  $R[k]$  implies that  $k$  itself fulfils triple  $\{P * e \mapsto R[\_]\}k\{Q\}$ .

In the above rule we do not make any assumptions about what code  $e$  actually points to, as long as it fulfils the specification  $R$ . It may even be updated between recursive calls. However, for recursion through the store,  $R$  must be recursively defined as it needs to maintain itself as an invariant of the code in  $e$ .

The EVAL rule crucially relies on the expressiveness of our assertion language, especially the presence of nested triples and recursive assertions. In our previous work, we did not consider nested triples. As a result, we had to reason explicitly with stored code, rather than properties of the code, as illustrated by one of our old rules for  $\text{eval}$  [2]:

$$\frac{\text{OLDEVAL} \quad \Gamma \vdash \{P\}'\text{eval}[e]\{Q\} \Rightarrow \{P\}'C\{Q\}}{\Gamma \vdash \{P * e \mapsto 'C'\}'\text{eval}[e]\{Q * e \mapsto 'C'\}}$$

$$\begin{array}{c}
\text{EVALNONREC1} \\
\hline
\Gamma \vdash \{P * e \mapsto \forall \mathbf{y}. \{P\}_-\{Q\}\}' \text{eval}[e]' \{Q * e \mapsto \forall \mathbf{y}. \{P\}_-\{Q\}\} \\
\text{EVALNONRECUPD} \\
\hline
\Gamma \vdash \{P * e \mapsto \forall \mathbf{y}. \{P * e \mapsto \_-\}\}' \text{eval}[e]' \{Q\} \\
\text{EVALREC} \\
\hline
\Gamma \vdash \{P \circ R\}' \text{eval}[e]' \{Q \circ R\} \quad (\text{where } R = (e \mapsto \forall \mathbf{y}. \{P\}_-\{Q\} * P_0) \otimes R)
\end{array}$$

**Fig. 4.** Derived rules from EVAL

Here the actual code  $C$  is specified explicitly in the pre- and post-conditions of the triple. In both rules the intuition is that the premise states that the body of the recursive procedure fulfils the triple, under the assumption that the recursive call does so as well. In the EVAL rule this is done without direct reference to the code itself, but rather via a  $k$  satisfying  $R$ . The soundness proof of OLDEVAL proceeded along the lines of Pitts' method for establishing relational properties of domains [10]. On the other hand, EVAL relies on the availability of recursive assertions, the existence of which is guaranteed by Banach's fixpoint theorem.

From the EVAL rule one can easily derive the axioms of Fig. 4. The first two axioms are for non-recursive calls. This can be seen from the fact that in the pre-condition of the nested triples  $e$  does not appear at all or does not have a specification, respectively. Only the third axiom EVALREC allows for recursive calls. The idea of this axiom is that one assumes that the code in  $[e]$  fulfils the required triple provided the code that  $e$  points to at call-time fulfils the triple as well. Let us look at the actual derivation of EVALREC to make this evident. We write  $S[k] \equiv \forall \mathbf{y}. \{P \circ R\}k\{Q \circ R\}$  such that for the original  $R$  of rule EVALREC we have  $R \Leftrightarrow (e \mapsto S[-] * (P_0 \otimes R))$ . Note that  $\Gamma$  contains the variables  $\mathbf{y}$  which may appear freely in  $P$  and  $Q$ .

$$\begin{array}{c}
\frac{\Gamma, k \vdash (\forall \mathbf{y}. \{P \circ R\}k\{Q \circ R\}) \Rightarrow \{P \circ R\}k\{Q \circ R\}}{\text{FOL}} \\
\frac{\Gamma, k \vdash S[k] \Rightarrow \{(P \otimes R) * (P_0 \otimes R) * e \mapsto S[-]\}k\{Q \circ R\}}{\text{CONSEQ}} \\
\frac{\Gamma \vdash \{(P \otimes R) * (P_0 \otimes R) * e \mapsto S[-]\}' \text{eval}[e]' \{Q \circ R\}}{\text{EVAL}} \\
\hline
\Gamma \vdash \{P \circ R\}' \text{eval}[e]' \{Q \circ R\} \quad \text{CONSEQ}
\end{array}$$

In the derivation tree above, the axiom used at the top is simply a first-order axiom for  $\forall$  elimination. The quantified variables  $\mathbf{y}$  are substituted by the variables with the same name from the context. After an application of the EVALREC rule those variables  $\mathbf{y}$  can then be substituted further.

The use of recursive specification  $R = (e \mapsto \forall \mathbf{y}. \{P\}_-\{Q\} * P_0) \otimes R$  is essential here as it allows us to unroll the definition so that the EVAL rule can be applied. Note that in the logic of [5], which also uses nested triples but features neither a specification logic nor expresses any frame rules or axioms, such recursive specifications are avoided. This is possible under the assumption that code does not change during recursion. One can then express the recursive  $R$  above as

$\otimes$ -FRAME	$*$ -FRAME	EVAL
$\frac{\Gamma \vdash P}{\Gamma \vdash P \otimes R}$	$\frac{\Gamma \vdash \{P\}e\{Q\}}{\Gamma \vdash \{P * R\}e\{Q * R\}}$	$\frac{\Gamma, k \vdash R[k] \Rightarrow \{P * e \mapsto R[\_]\}k\{Q\}}{\Gamma \vdash \{P * e \mapsto R[\_]\}'eval[e]'\{Q\}}$

**Fig. 5.** Proof rules specific to higher-order store

follows (we can omit the  $P_0$  now, as this is only needed for mutually recursively defined triples):

$$e \mapsto \{e \mapsto k * P\}k\{e \mapsto k * Q\}.$$

The question however remains how the assertion can be proved for some concrete ‘ $C$ ’ in  $[e]$ . In *loc.cit.* this is done by an induction on some appropriate argument, as total correctness is considered only. Note that our old OLDEVAL can be viewed as a fixpoint induction rule for proving such specifications, if one quantifies away the concrete appearances of ‘ $C$ ’. In any case, our new EVAL is obviously elegant to use, and it does not only allow for recursion through the store but also disentangles the reasoning from the concrete code stored in the heap.

To conclude this section, Fig. 5 summarizes a particular choice of proof-rule set from the current and previous subsections. Soundness is proved in Section 4.

## 4 Semantics of Nested Triples

This section develops a model for the programming language and logic we have presented. The semantics of programs, given in the next subsection using an untyped domain-theoretic model, is standard. The following semantics of the logic is, however, unusual; it is a possible world semantics where the worlds live in a recursively defined *metric* space. Finally, we discuss the existence of recursively defined assertions, which have been used in the previous sections.

**Semantics of expressions and commands.** The interpretation of the programming language is given in the category  $\mathbf{Cppo}_\perp$  of pointed cpos and strict continuous functions, and is the same as in our previous work [2]. That is, commands denote strict continuous functions  $\llbracket C \rrbracket_\eta \in \mathit{Heap} \multimap T_{err}(\mathit{Heap})$  where

$$\mathit{Heap} = \mathit{Rec}(\mathit{Val}) \quad \mathit{Val} = \mathit{Integers}_\perp \oplus \mathit{Com}_\perp \quad \mathit{Com} = \mathit{Heap} \multimap T_{err}(\mathit{Heap}) \quad (1)$$

In these equations,  $T_{err}(D) = D \oplus \{\mathit{error}\}_\perp$  denotes the error monad, and  $\mathit{Rec}(D)$  denotes records with entries from  $D$  and labelled by positive natural numbers. Formally,  $\mathit{Rec}(D) = (\sum_{N \subseteq_{\text{fin}} \mathit{Nats}^+} (N \rightarrow D_\downarrow))_\perp$  where  $(N \rightarrow D_\downarrow)$  is the cpo of maps from the finite address set  $N$  to the cpo  $D_\downarrow = D - \{\perp\}$  of non-bottom elements of  $D$ . We use some evident record notations, such as  $\{\ell_1=d_1, \dots, \ell_n=d_n\}$  for the record mapping label  $\ell_i$  to  $d_i$ , and  $\mathit{dom}(r)$  for the set of labels of a record  $r$ . The *disjointness predicate*  $r \# r'$  on records holds if  $r$  and  $r'$  are not  $\perp$  and have disjoint domains, and a partial *combining operation*  $r \cdot r'$  is defined by

$$r \cdot r' \stackrel{\text{def}}{=} \text{if } r \# r' \text{ then } r \cup r' \text{ else } \perp.$$

---


$$\begin{aligned}
& \llbracket \text{skip} \rrbracket_\eta h \stackrel{\text{def}}{=} h \\
& \llbracket C_1; C_2 \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } \llbracket C_1 \rrbracket_\eta h \in \{\perp, \text{error}\} \text{ then } \llbracket C_1 \rrbracket_\eta h \text{ else } \llbracket C_2 \rrbracket_\eta (\llbracket C_1 \rrbracket_\eta h) \\
& \llbracket \text{if } e=e' \text{ then } C_1 \text{ else } C_2 \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } \{\llbracket e_1 \rrbracket_\eta, \llbracket e_2 \rrbracket_\eta\} \subseteq \text{Com}_\perp \text{ then } \perp \\
& \quad \text{else if } (\llbracket e \rrbracket_\eta = \llbracket e' \rrbracket_\eta) \text{ then } \llbracket C_1 \rrbracket_\eta h \text{ else } \llbracket C_2 \rrbracket_\eta h \\
& \llbracket \text{let } x=\text{new } e_1, \dots, e_n \text{ in } C \rrbracket_\eta h \stackrel{\text{def}}{=} \text{let } \ell = \min\{\ell \mid \forall \ell'. (\ell \leq \ell' < \ell+n) \Rightarrow \ell' \notin \text{dom}(h)\} \\
& \quad \text{in } \llbracket C \rrbracket_{\eta[x \mapsto \ell]} (h \cdot \{\ell = \llbracket e_1 \rrbracket_\eta, \dots, \ell+n-1 = \llbracket e_n \rrbracket_\eta\}) \\
& \llbracket \text{free } e \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } \llbracket e \rrbracket_\eta \notin \text{dom}(h) \text{ then } \text{error} \\
& \quad \text{else } (\text{let } h' \text{ s.t. } h = h' \cdot \{\llbracket e \rrbracket_\eta = h(\llbracket e \rrbracket_\eta)\} \text{ in } h') \\
& \llbracket [e_1] := e_2 \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } \llbracket e_1 \rrbracket_\eta \notin \text{dom}(h) \text{ then } \text{error} \text{ else } (h[\llbracket e_1 \rrbracket_\eta \mapsto \llbracket e_2 \rrbracket_\eta]) \\
& \llbracket \text{let } x=[e] \text{ in } C \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } \llbracket e \rrbracket_\eta \notin \text{dom}(h) \text{ then } \text{error} \text{ else } \llbracket C \rrbracket_{\eta[x \mapsto h(\llbracket e \rrbracket_\eta)]} h \\
& \llbracket \text{eval } [e] \rrbracket_\eta h \stackrel{\text{def}}{=} \text{if } (\llbracket e \rrbracket_\eta \notin \text{dom}(h) \vee h(\llbracket e \rrbracket_\eta) \notin \text{Com}) \text{ then } \text{error} \\
& \quad \text{else } (h(\llbracket e \rrbracket_\eta))(h)
\end{aligned}$$


---

**Fig. 6.** Interpretation of commands  $\llbracket C \rrbracket_\eta \in \text{Heap} \multimap \text{Err}(\text{Heap})$

The interpretation of commands is repeated in Fig. 6 (assuming  $h \neq \perp$ ). The interpretation of the quote operation, ‘ $C$ ’, uses the injection of  $\text{Com}$  into  $\text{Val}$ . The interpretation of the remaining expressions is entirely standard and omitted.

A solution to equation (1) for  $\text{Heap}$  can be obtained by the usual inverse limit construction [14] in the category  $\mathbf{Cppo}_\perp$ . This solution is an SFP domain (e.g., [15]), and thus comes equipped with an increasing chain  $\pi_n : \text{Heap} \rightarrow \text{Heap}$  of continuous projection maps, satisfying  $\pi_0 = \perp$ ,  $\bigsqcup_{n \in \omega} \pi_n = \text{id}_{\text{Heap}}$ , and  $\pi_n \circ \pi_m = \pi_{\min\{n, m\}}$ . The image of each  $\pi_n$  is finite, hence each  $\pi_n(h)$  is a compact element of  $\text{Heap}$ . Moreover, the projections are compatible with composition of heaps: we have  $\pi_n(h \cdot h') = \pi_n(h) \cdot \pi_n(h')$  for all  $h, h'$ .

**Semantic domain for assertions.** A subset  $p \subseteq \text{Heap}$  is *admissible* if  $\perp \in p$  and  $p$  is closed under taking least upper bounds of  $\omega$ -chains. It is *uniform* [3] if it is closed under the projections, i.e., if  $p$  satisfies that  $h \in p \Rightarrow \pi_n(h) \in p$  for all  $n$ . We write  $UAdm$  for the set of all uniform admissible subsets of  $\text{Heap}$ . For  $p \in UAdm$ ,  $p_{[n]}$  denotes the image of  $p$  under  $\pi_n$ . Note that also  $p_{[n]} \in UAdm$ .

The uniform admissible subsets will form the basic building block when interpreting the assertions of our logic. Since assertions in general depend on invariants for stored code, the space of semantic predicates  $\text{Pred}$  will consist of functions  $W \rightarrow UAdm$  from a set of “worlds,” describing the invariants, to the collection of uniform admissible subsets of heaps. But, the invariants for stored code are themselves semantic predicates, and the interaction between  $\text{Pred}$  and  $W$  is governed by (the semantics of)  $\otimes$ . Hence we seek a space of worlds  $W$  that is “the same” as  $W \rightarrow UAdm$ . We obtain such a  $W$  using metric spaces.

Recall that a 1-bounded ultrametric space  $(X, d)$  is a metric space where the distance function  $d : X \times X \rightarrow \mathbb{R}$  takes values in the closed interval  $[0, 1]$  and satisfies the strong triangle inequality  $d(x, y) \leq \max\{d(x, z), d(z, y)\}$ , for all  $x, y, z \in X$ . An (ultra-)metric space is complete if every Cauchy sequence



has a limit. A function  $f : X_1 \rightarrow X_2$  between metric spaces  $(X_1, d_1), (X_2, d_2)$  is *non-expansive* if for all  $x, y \in X_1, d_2(f(x), f(y)) \leq d_1(x, y)$ . It is *contractive* if for some  $\delta < 1, d_2(f(x), f(y)) \leq \delta \cdot d_1(x, y)$  for all  $x, y \in X_1$ .

The complete, 1-bounded ultrametric spaces and non-expansive functions between them form a Cartesian closed category *CBUlt*. Products in *CBUlt* are given by the set-theoretic product where the distance is the maximum of the componentwise distances, and exponentials are given by the non-expansive functions equipped with the sup-metric. A functor  $F : \text{CBUlt}^{op} \times \text{CBUlt} \rightarrow \text{CBUlt}$  is *locally non-expansive* if  $d(F(f, g), F(f', g')) \leq \max\{d(f, f'), d(g, g')\}$  for all non-expansive  $f, f', g, g'$ , and it is *locally contractive* if  $d(F(f, g), F(f', g')) \leq \delta \cdot \max\{d(f, f'), d(g, g')\}$  for some  $\delta < 1$ . By multiplication of the distances of  $(X, d)$  with a shrinking factor  $\delta < 1$  one obtains a new ultrametric space,  $\delta \cdot (X, d) = (X, d')$  where  $d'(x, y) = \delta \cdot d(x, y)$ . By shrinking, a locally non-expansive functor  $F$  yields a locally contractive functor  $(\delta \cdot F)(X_1, X_2) = \delta \cdot (F(X_1, X_2))$ .

The set *UAdm* of uniform admissible subsets of *Heap* becomes a complete, 1-bounded ultrametric space when equipped with the following distance function:  $d(p, q) \stackrel{\text{def}}{=} \text{if } (p \neq q) \text{ then } (2^{-\max\{i \in \omega \mid p_{[i]} = q_{[i]}\}}) \text{ else } 0$ . Note that  $d$  is well-defined: first, because  $\pi_0 = \perp$  and  $\perp \in p$  for all  $p \in \text{UAdm}$  the set  $\{i \in \omega \mid p_{[i]} = q_{[i]}\}$  is non-empty; second, this set is finite, because  $p \neq q$  implies  $p_{[i]} \neq q_{[i]}$  for all sufficiently large  $i$  by the uniformity of  $p, q$  and using  $\bigsqcup_{n \in \omega} \pi_n = \text{id}_{\text{Heap}}$ .

**Theorem 3.** *There exists an ultrametric space  $W$  and an isomorphism  $\iota$  from  $\frac{1}{2} \cdot (W \rightarrow \text{UAdm})$  to  $W$  in *CBUlt*.*

*Proof.* By an application of America & Rutten’s existence theorem for fixed points of locally contractive functors on complete ultrametric spaces [1], applied to  $F(X, Y) = \frac{1}{2} \cdot (X \rightarrow \text{UAdm})$ . See [3] for details of a similar application.  $\square$

We write *Pred* for  $\frac{1}{2} \cdot (W \rightarrow \text{UAdm})$  and  $\iota^{-1} : W \cong \text{Pred}$  for the inverse to  $\iota$ .

For an ultrametric space  $(X, d)$  and  $n \in \omega$  we use the notation  $x \stackrel{n}{=} y$  to mean that  $d(x, y) \leq 2^{-n}$ . By the ultrametric inequality, each  $\stackrel{n}{=}$  is an equivalence relation on  $X$  [3]. Since all non-zero distances in *UAdm* are of the form  $2^{-n}$  for some  $n \in \omega$ , this is also the case for the distance function on  $W$ . Therefore, to show that a map is non-expansive it suffices to show that  $f(x) \stackrel{n}{=} f(y)$  whenever  $x \stackrel{n}{=} y$ . The definition of *Pred* has the following consequence: for  $p, q \in \text{Pred}, p \stackrel{n}{=} q$  iff  $p(w) \stackrel{n-1}{=} q(w)$  for all  $w \in W$ . This fact is used repeatedly in our proofs.

For  $p, q \in \text{UAdm}$ , the separating conjunction  $p * q$  is defined as usual, by  $h \in p * q \stackrel{\text{def}}{\iff} \exists h_1, h_2. h = h_1 \cdot h_2 \wedge h_1 \in p \wedge h_2 \in q$ . This operation is lifted to non-expansive functions  $p_1, p_2 \in \text{Pred}$  pointwise, by  $(p_1 * p_2)(w) = p_1(w) * p_2(w)$ . This is well-defined, and moreover determines a non-expansive operation on the space *Pred*. The corresponding unit for the lifted  $*$  is the non-expansive function  $\text{emp} = \lambda w. \{\{\}, \perp\}$  (i.e.,  $p * \text{emp} = \text{emp} * p = p$ , for all  $p$ ). We let  $\text{emp} = \iota(\text{emp})$ .

**Lemma 4.** *There exists a non-expansive map  $\circ : W \times W \rightarrow W$  and a map  $\otimes : \text{Pred} \times W \rightarrow \text{Pred}$  that is non-expansive in its first and contractive in its second argument, satisfying  $q \circ r = \iota(\iota^{-1}(q) \otimes r * \iota^{-1}(r))$  and  $p \otimes r = \lambda w. p(r \circ w)$  for all  $p \in \text{Pred}$  and  $q, r \in W$ .*

*Proof.* The defining equations of both operations give rise to contractive maps, which have (unique) fixed points by Banach’s fixed point theorem.  $\square$

**Lemma 5.**  $(W, \circ, emp)$  is a monoid in  $CBUlt$ . Moreover,  $\otimes$  is an action of this monoid on  $Pred$ .

*Proof.* First,  $emp$  is a left-unit for  $\circ$ ,  $emp \circ q = \iota((\lambda w. \iota^{-1}(emp)(q \circ w)) * \iota^{-1}(q)) = \iota(\iota^{-1}(q)) = q$ . Using this, one shows that it is also a right-unit for  $\circ$ . Next, one shows by induction that for all  $n \in \omega$ ,  $\circ$  is associative up to distance  $2^{-n}$ , from which associativity follows. By the 1-boundedness of  $W$  the base case is clear. For the inductive step  $n > 0$ , by definition of the distance function on  $Pred$  it suffices to show that for all  $w \in W$ ,  $\iota^{-1}((p \circ q) \circ r)(w) \stackrel{n-1}{=} \iota^{-1}(p \circ (q \circ r))(w)$ . This equation follows from the definition of  $\circ$  and the inductive hypothesis.

That  $\otimes$  forms an action of  $W$  on  $Pred$  follows from these properties of  $\circ$ . First,  $p \otimes emp = \lambda w. p(emp \circ w) = p$  since  $emp$  is a unit for  $\circ$ . Next,  $(p \otimes q) \otimes r = \lambda w. p(q \circ (r \circ w)) = \lambda w. p((q \circ r) \circ w) = p \otimes (q \circ r)$  by the associativity of  $\circ$ .  $\square$

**Semantics of triples and assertions.** Since assertions appear in the pre- and post-conditions of Hoare triples, and triples can be nested inside assertions, the interpretation of assertions and triples must be defined simultaneously. To achieve this, we first define a notion of semantic triple.

**Definition 6 (Semantic triple).** A semantic Hoare triple consists of predicates  $p, q \in Pred$  and a strict continuous function  $c \in Heap \multimap T_{err}(Heap)$ , written  $\{p\}c\{q\}$ . For  $w \in W$ , a semantic triple  $\{p\}c\{q\}$  is forced by  $w$ , denoted  $w \models \{p\}c\{q\}$ , if for all  $r \in UAdm$  and all  $h \in Heap$ :

$$h \in p(w) * \iota^{-1}(w)(emp) * r \Rightarrow c(h) \in Ad(q(w) * \iota^{-1}(w)(emp) * r),$$

where  $Ad(r)$  denotes the least downward closed and admissible set of heaps containing  $r$ . A semantic triple is valid, written  $\models \{p\}c\{q\}$ , if  $w \models \{p\}c\{q\}$  for all  $w \in W$ . We extend semantic triples from  $Com = Heap \multimap T_{err}(Heap)$  to all  $d \in Val$ , by  $w \models \{p\}d\{q\}$  iff  $d = c$  for some command  $c \in Com$  and  $w \models \{p\}c\{q\}$ . A triple holds approximately up to level  $k$ ,  $w \models_k \{p\}d\{q\}$ , if  $w \models \{p\}\pi_k; d; \pi_k\{q\}$ .

Thus, semantic triples bake in the first-order frame property (by conjoining  $r$ ), and “close” the “open” recursion (by applying the world  $w$ , on which the triple implicitly depends, to  $emp$ ). The admissible downward closure that is applied to the entire post-condition is in line with a partial correctness interpretation of triples. In particular, it entails that the sets  $\{c \in Com \mid w \models_k \{p\}c\{q\}\}$  and  $\{c \in Com \mid w \models \{p\}c\{q\}\}$  are admissible and downward closed subsets of  $Com$ . Finally, semantic triples are non-expansive, in the sense that if  $w \stackrel{n}{=} w'$  for  $n > 0$  and  $w \models \{p\}c\{q\}$ , then  $w' \models_{n-1} \{p\}c\{q\}$ . This observation plays a key role in the following definition of the semantics of nested triples. Another useful observation is that  $w \models \{p\}c\{q\}$  is equivalent to  $\forall k \in \omega. w \models_k \{p\}c\{q\}$ .

Assertions are interpreted as elements  $\llbracket P \rrbracket_\eta \in Pred$ . Note that  $(UAdm, \subseteq)$  is a complete Heyting BI algebra. Using the pointwise extension of the operations of

---

$\llbracket \text{true} \rrbracket_\eta w = \text{Heap}$	$\llbracket P \wedge Q \rrbracket_\eta w = \llbracket P \rrbracket_\eta w \cap \llbracket Q \rrbracket_\eta w$
$\llbracket \text{false} \rrbracket_\eta w = \{\perp\}$	$\llbracket P \vee Q \rrbracket_\eta w = \llbracket P \rrbracket_\eta w \cup \llbracket Q \rrbracket_\eta w$
$\llbracket \text{emp} \rrbracket_\eta w = \{\{\}, \perp\}$	$\llbracket P * Q \rrbracket_\eta w = \llbracket P \rrbracket_\eta w * \llbracket Q \rrbracket_\eta w$
$\llbracket e_1 \mapsto e_2 \rrbracket_\eta w = \{h \mid h \sqsubseteq \{\llbracket e_1 \rrbracket_\eta = \llbracket e_2 \rrbracket_\eta\}\}$	$\llbracket P \otimes Q \rrbracket_\eta w = (\llbracket P \rrbracket_\eta \otimes \iota(\llbracket Q \rrbracket_\eta))w$
$\llbracket e_1 = e_2 \rrbracket_\eta w = \{h \mid h \neq \perp \Rightarrow \llbracket e_1 \rrbracket_\eta = \llbracket e_2 \rrbracket_\eta\}$	$\llbracket \forall x.P \rrbracket_\eta w = \bigcap_{d \in \text{Val}} \llbracket P \rrbracket_{\eta[x:=d]} w$
$\llbracket \exists x.P \rrbracket_\eta w = \{h \mid \forall n \in \omega. \pi_n(h) \in \bigcup_{d \in \text{Val}} \llbracket P \rrbracket_{\eta[x:=d]} w\}$	
$\llbracket P \Rightarrow Q \rrbracket_\eta w = \{h \mid \forall n \in \omega. \pi_n(h) \in \llbracket P \rrbracket_\eta w \text{ implies } \pi_n(h) \in \llbracket Q \rrbracket_\eta w\}$	
$\llbracket \llbracket P \rrbracket e \{Q\} \rrbracket_\eta w = \text{Ad}\{h \in \text{Heap} \mid \text{rnk}(h) > 0 \Rightarrow w \models_{\text{rnk}(h)-1} \{\llbracket P \rrbracket_\eta\} \llbracket e \rrbracket_\eta \{\llbracket Q \rrbracket_\eta\}\}$	

---

Fig. 7. Semantics of assertions

this algebra to the set of non-expansive functions  $W \rightarrow UAdm$ , we also obtain a complete Heyting BI algebra on  $\text{Pred} = \frac{1}{2} \cdot (W \rightarrow UAdm)$  which soundly models the intuitionistic predicate BI part of the assertion logic. Moreover, the monoid action of  $W$  on  $\text{Pred}$  serves to model the invariant extension of the assertion logic. In order to define a non-expansive interpretation of nested triples we will use the following definition:

**Definition 7 (Rank of a heap).** *If  $h$  is a compact element of  $\text{Heap}$ , then the least  $r$  for which  $\pi_r(h) = h$  is the rank of  $h$ , abbreviated  $\text{rnk}(h)$ , otherwise the rank is undefined.*

The interpretation of assertions is spelled out in detail in Fig. 7. The interpretation of a nested triple  $\llbracket P \rrbracket e \{Q\}$  is *not* independent of the heap, unlike the (more traditional) semantics of “top-level” triples, i.e.  $\models \{p\}c\{q\}$ . More precisely, the definition in Fig. 7 means that triples as assertions depend on the rank of the current heap. This is necessary to provide a *non-expansive* function from  $W$  to  $UAdm$  that provides enough “approximation information.” Simpler definitions like  $\{h \in \text{Heap} \mid w \models \{\llbracket P \rrbracket_\eta\} \llbracket e \rrbracket_\eta \{\llbracket Q \rrbracket_\eta\}\}$  are heap independent but not non-expansive. A similar approach has been taken in [3] to force non-expansiveness for a reference type constructor for ML-style references.

As a consequence of this interpretation, the axiom  $\{\{A\}e\{B\} \wedge A\}e\{B\}$  does not hold; the inner triple is only approximately valid up to the level of the rank of the argument heap. Analogously, the following rule does not appear to be sound in our semantics:

$$\frac{\{A\}e\{B\} \Rightarrow \{P\}e'\{Q\}}{\{\{A\}e\{B\} \wedge P\}e'\{Q\}}$$

The opposite direction does actually hold. Axioms and rules like these are used, e.g., in [5] in proofs for recursion through the store; instead we use (EVAL).

**Soundness of the proof rules.** We prove soundness of the proof rules listed in Sections 2 and 3. We first consider the distribution axioms for  $- \otimes R$  in Fig. 2.

**Lemma 8 ( $\otimes$ -Dist, 1).** *The axiom  $(P \otimes Q) \otimes R \Leftrightarrow P \otimes (Q \circ R)$  is valid.*

*Proof.* An instance of the fact that  $\otimes$  is a monoid action (Lemma 5). □

**Lemma 9 ( $\otimes$ -Dist, 2).** *The axiom  $\{P\}e\{Q\} \otimes R \Leftrightarrow \{P \circ R\}e\{Q \circ R\}$  is valid.*

*Proof.* The statement follows from the following claim: for all  $p, q, r \in \text{Pred}$ , strict continuous  $c : \text{Heap} \multimap \text{Terr}(\text{Heap})$  and all  $w \in W$ ,  $\iota(r) \circ w \models \{p\}c\{q\}$  iff  $w \models \{p \otimes \iota(r) * r\}c\{q \otimes \iota(r) * r\}$ . The proof of this claim uses the equation

$$(p \otimes \iota(r) * r)(w) * \iota^{-1}(w)(\text{emp}) = p(\iota(r) \circ w) * \iota^{-1}(\iota(r) \circ w)(\text{emp}),$$

which is a consequence of the definitions of  $\otimes$  and  $\circ$ .  $\square$

The proofs of the remaining distribution axioms are easy since the logical connectives are interpreted pointwise, and  $\text{emp}$  and  $(e_1 \mapsto e_2)$  are constant.

Next, we consider the rules for higher-order store given in Fig. 5.

**Lemma 10 ( $\otimes$ -Frame).** *The  $\otimes$ -FRAME rule is sound: if  $h \in p(w)$  for all  $h \in \text{Heap}$  and  $w \in W$ , then  $h \in (p \otimes \iota(r))(w)$  for all  $h \in \text{Heap}$ ,  $w \in W$  and  $r \in \text{Pred}$ .*

*Proof.* Assume  $\forall h. \forall w. h \in p(w)$ . Let  $r \in \text{Pred}$ ,  $w \in W$  and  $h \in \text{Heap}$ . We show that  $h \in (p \otimes \iota(r))(w)$ . Note that  $(p \otimes \iota(r))(w) = p(\iota(r) \circ w)$  by the definition of  $\otimes$ . So, for  $w' \stackrel{\text{def}}{=} \iota(r) \circ w$ , the assumption shows that  $h \in p(w') = (p \otimes \iota(r))(w)$ .  $\square$

**Lemma 11 ( $*$ -Frame).**  *$\{P\}e\{Q\} \Rightarrow \{P * R\}e\{Q * R\}$  is valid for all  $P, Q, R, e$ .*

*Proof.* We show that for all  $w \in W$ ,  $p, q, r \in \text{Pred}$  and  $c \in \text{Com}$ , if  $w \models \{p\}c\{q\}$ , then  $w \models \{p * r\}c\{q * r\}$ . This implies the lemma as follows. If  $k$  is the rank of  $\pi_n(h)$  and  $\pi_n(h) \in \llbracket \{P\}e\{Q\} \rrbracket w$ , then  $w \models_{k-1} \{\llbracket P \rrbracket_\eta\} \llbracket e \rrbracket_\eta \{\llbracket Q \rrbracket_\eta\}$ . This lets us conclude  $w \models_{k-1} \{\llbracket P * R \rrbracket_\eta\} \llbracket e \rrbracket_\eta \{\llbracket Q * R \rrbracket_\eta\}$ , which in turn implies that  $\pi_n(h)$  belongs to  $\llbracket \{P * R\}e\{Q * R\} \rrbracket w$ .

Assume  $w \models \{p\}c\{q\}$ . We must show that  $w \models \{p * r\}c\{q * r\}$ . Let  $r' \in \text{UAdm}$  and assume  $h \in (p * r)(w) * \iota^{-1}(w)(\text{emp}) * r' = p(w) * \iota^{-1}(w)(\text{emp}) * (r(w) * r')$ . Since  $w \models \{p\}c\{q\}$ , it follows that  $c(h) \in \text{Ad}(q(w) * \iota^{-1}(w)(\text{emp}) * (r(w) * r')) = \text{Ad}((q * r)(w) * \iota^{-1}(w)(\text{emp}) * r')$ , which establishes  $w \models \{p * r\}c\{q * r\}$ .  $\square$

The proofs of the remaining rules (i.e., EVAL and those in Fig. 3) are similar but omitted due to lack of space.

**Semantics of recursive assertions.** The following general fixed point theorem is a consequence of Banach's fixed point theorem, and it allows us to introduce recursively defined assertions in the logic, as used in previous sections.

**Theorem 12 (Mutually recursive predicates).** *Let  $I$  be a set and suppose that, for each  $i \in I$ ,  $F_i : \text{Pred}^I \rightarrow \text{Pred}$  is a contractive function. Then there exists a unique  $\mathbf{p} = (p_i)_{i \in I} \in \text{Pred}^I$  such that  $F_i(\mathbf{p}) = p_i$ , for all  $i \in I$ .*

Note that this theorem is sufficiently general to permit the mutual recursive definition of even infinite families of predicates.

As established in Lemma 4,  $\otimes$  is contractive in its right-hand argument. Thus, for fixed  $P$  and  $\eta$ , the map  $F(r) = \llbracket P \rrbracket_\eta \otimes \iota(r)$  on  $\text{Pred}$  is contractive and has a unique fixed point,  $r$ . Given an equation  $R = P \otimes R$  we take  $r$  as the

interpretation of  $R$ , and note that indeed  $\llbracket R \rrbracket_\eta = r = \llbracket P \rrbracket_\eta \otimes \iota(r) = \llbracket P \otimes R \rrbracket_\eta$ . Along the same lines, we can interpret mutually recursive assertions:  $R_1 = P_1 \otimes (R_1 * \dots * R_n)$ ,  $\dots$ ,  $R_n = P_n \otimes (R_1 * \dots * R_n)$ . Using the non-expansiveness of  $*$  as an operation on  $\text{Pred}$ , these equations give rise to contractive functions  $F_i(r_1, \dots, r_n) = \llbracket P_i \rrbracket_\eta \otimes \iota(r_1 * \dots * r_n)$ .

*Acknowledgments.* We would like to thank François Pottier, Kristian Støvring and Jacob Thamsborg for helpful discussions. Kristian suggested that  $\otimes$  is a monoid action. Partial support has been provided by FNU project 272-07-0305 “Modular reasoning about software”, EPSRC projects EP/G003173/1 “From reasoning principles for function pointers to logics for self-configuring programs” and EP/E053041/1 “Scalable program analysis for software verification”.

## References

1. America, P., Rutten, J.J.M.M.: Solving reflexive domain equations in a category of complete metric spaces. *J. Comput. Syst. Sci.* 39(3), 343–375 (1989)
2. Birkedal, L., Reus, B., Schwinghammer, J., Yang, H.: A simple model of separation logic for higher-order store. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfssdóttir, A., Walukiewicz, I. (eds.) *ICALP 2008, Part II*. LNCS, vol. 5126, pp. 348–360. Springer, Heidelberg (2008)
3. Birkedal, L., Støvring, K., Thamsborg, J.: Realizability semantics of parametric polymorphism, general references, and recursive types. In: *FOSSACS 2009*. LNCS, vol. 5504, pp. 456–470. Springer, Heidelberg (2009)
4. Birkedal, L., Torp-Smith, N., Yang, H.: Semantics of separation-logic typing and higher-order frame rules for Algol-like languages. *Log. Methods Comput. Sci.* 2(5:1) (2006)
5. Honda, K., Yoshida, N., Berger, M.: An observationally complete program logic for imperative higher-order functions. In: *LICS*, pp. 270–279. IEEE Computer Society Press, Los Alamitos (2005)
6. Krishnaswami, N., Birkedal, L., Aldrich, J., Reynolds, J.: Idealized ML and Its Separation Logic (2007). <http://www.cs.cmu.edu/~neelk/>
7. Nanevski, A., Morrisett, G., Shinnar, A., Govereau, P., Birkedal, L.: Ynot: dependent types for imperative programs. In: *ICFP*, pp. 229–240. ACM Press, New York (2008)
8. O’Hearn, P.W., Pym, D.J.: The logic of bunched implications. *B. Symb. Log.* 5(2), 215–244 (1999)
9. Parkinson, M., Biermann, G.: Separation logic, abstraction and inheritance. In: *POPL*, pp. 75–86. ACM Press, New York (2008)
10. Pitts, A.M.: Relational properties of domains. *Inf. Comput.* 127, 66–90 (1996)
11. Pottier, F.: Hiding local state in direct style: a higher-order anti-frame rule. In: *LICS*, pp. 331–340. IEEE Computer Society Press, Los Alamitos (2008)
12. Reus, B., Schwinghammer, J.: Separation logic for higher-order store. In: Ésik, Z. (ed.) *CSL 2006*. LNCS, vol. 4207, pp. 575–590. Springer, Heidelberg (2006)
13. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: *LICS*, pp. 55–74. IEEE Computer Society Press, Los Alamitos (2002)
14. Smyth, M.B., Plotkin, G.D.: The category-theoretic solution of recursive domain equations. *SIAM J. Comput.* 11(4), 761–783 (1982)
15. Streicher, T.: *Domain-theoretic Foundations of Functional Programming*. World Scientific, Singapore (2006)

# A Complete Characterization of Observational Equivalence in Polymorphic $\lambda$ -Calculus with General References<sup>\*</sup>

Eijiro Sumii

Tohoku University  
sumii@ecei.tohoku.ac.jp

**Abstract.** We give the first sound and complete proof method for observational equivalence in full polymorphic  $\lambda$ -calculus with existential types and first-class, higher-order references. Our method is syntactic and elementary in the sense that it only employs simple structures such as relations on terms. It is nevertheless powerful enough to prove many interesting equivalences that can and cannot be proved by previous approaches, including the latest work by Ahmed, Dreyer and Rossberg (POPL 2009).

## 1 Introduction

Data abstraction and local state are both known to introduce interesting properties—in particular, observational equivalences—into computer programs. Methodology for reasoning about such properties has been a major challenge in the fundamental research on programming languages (e.g., [13, 18]).

Recently, Ahmed, Dreyer and Rossberg [3] developed a technique, based on step-indexed Kripke-style logical relations, for proving observational equivalence in polymorphic  $\lambda$ -calculus with *both* abstract types and references. While their technique is (to our knowledge) the first “direct”—i.e., without encoding into other languages such as polymorphic  $\pi$ -calculus [16] or continuation passing style [12]—proof method for observational equivalence in this language, it is incomplete and specialized for particular cases (generative abstract data types). Indeed, some interesting equivalences cannot be proved by their method [3, Section 5.7 and 5.8]. Independently, Birkedal, Støvring and Thamsborg [5, 6] have also developed logical relations in a language with polymorphic (and recursive) types and references. However, they are also incomplete and not yet useful enough for reasoning about observational equivalence involving local state in general [6, Section 1 and 6] [5, Section 1].

In this paper, we take a different approach, based on Sumii et al.’s environmental bisimulation [10, 19, 23, 24], and give the first sound and complete proof method for observational equivalence in call-by-value  $\lambda$ -calculus with impredicative universal and existential types, as well as “full” references (i.e., references

---

<sup>\*</sup> Appendices online: <http://www.kb.ecei.tohoku.ac.jp/~sumii/pub/polyref.pdf>

are first-class values and all values can be referred to, including functions and references themselves). Our development is not only complete, but also arguably simpler than other theories in that it only requires elementary notions such as terms, values, relations and sets without explicit need for metric spaces or step indices. Although observational equivalence is clearly undecidable in any Turing-complete language (in our case, general recursion can be encoded via higher-order references [15, Exercise 13.5.8]), we believe that our approach is useful for understanding and reasoning about information hiding in a wide range of settings including ours.

The rest of this paper is structured as follows. Section 2 discusses related work and our contributions with respect to it. Section 3 describes our language. Section 4 defines environmental bisimulation for this language and Section 5 develops up-to techniques. Section 6 proves the characterization theorem by using the up-to techniques. Section 7 proves examples of equivalences from 3 and Section 8 concludes with more comments.

## 2 Related Work

The classic references on observational equivalences introduced by polymorphic types are Reynolds’ relational parametricity [18] and Mitchell’s representation independence [14]. Establishing a similar theory for local state has turned out to be highly challenging. The classic reference here is Meyer and Sieber [13]. Pitts and Stark [17] developed syntactic logical relations (i.e., logical relations over the term model) for  $\lambda$ -calculus with ML-like references. Their references are limited to the first order in the sense that only integers—not functions, nor references themselves—can be referred to. This is due to a difficulty involved in the circularity of “references to (functions containing) references.” Ahmed, Appel and Virga [2] used step indices [4] to break this circularity, though they worked on a unary (rather than binary) model and type safety (rather than observational equivalence). A recent paper by Ahmed, Dreyer and Rossberg [3]—discussed in Section 1, 7 and 8—follows this line of work. Another line of work has been carried out by Birkedal et al. [5–7]. To the best of our knowledge, our method can prove strictly more examples of equivalences than all of the above approaches (though this is hard to prove generally, because completeness by itself does not always mean an automatic proof; recall that observational equivalence is undecidable in our language).

Abramsky, Honda and McCusker [1] (as well as Laird [11] and Tzevelekos [25]) developed a fully abstract game (or trace) semantics for simply typed  $\lambda$ -calculus with general references. They did not treat polymorphism, however. Our theory is more elementary in the sense that it requires very little machinery other than the syntax and operational semantics of the language itself, enabling the simple treatment of complex combinations like polymorphism and state. (Of course, this does *not* devalue game semantics at all: their whole point is syntax-freedom, while ours is the complete opposite, i.e., “semantics-freedom.”)

Environmental bisimulation was first devised for untyped  $\lambda$ -calculus with encryption [23]. Since then, it has been applied to various languages, including

polymorphic  $\lambda$ -calculus with existential types [24] and untyped  $\lambda$ -calculus with general references [10] [19, Section 4]. The technical contributions of the present paper with respect to these are: (1) the *combination* of polymorphic types and references, which required careful handling of store typing (e.g., Definition 3 and 4), (2) the combination of small-step semantics and existential types, requiring a subtle adjustment to the context closure operation (Definition 9) and therefore to the up-to context technique (Definition 10), (3) a more powerful up-to reduction technique (Definition 7 and 8) that allows renaming of fresh locations, and (4) bisimulation proofs for non-trivial examples of observational equivalence in the present language, many of which have been considered hard traditionally (see, e.g., [3, 13, 17]).

Gordon [9] (as well as a number of papers that followed) considered bisimulations for functional languages with input and output effects (and for object calculi). To our knowledge, none of them treated references. Lassen et al. [12, 21] developed normal form bisimulations for polymorphic  $\lambda$ -calculus with control operators (and references) or in continuation passing style. Normal form bisimulations are generally incomplete with respect to contextual equivalence in languages without control operators (or in direct style) [21, Section 1] [12, Section 1.1].

To summarize, the main thrust of our work is to give actual evidence that environmental bisimulations scale easily to various languages, including the present one with both polymorphism and state, which has been considered difficult in the long history of research in this area (again see [3, 13, 17] for instance).

### 3 The Language

The syntax of our language is given in Figure 1. It is a standard polymorphic  $\lambda$ -calculus with existential types and references. We assume an infinite set of locations  $Loc$  and write  $loc(M)$  for the set of locations that appear in term  $M$ . We use meta-variables  $C, D, \dots$  for location-free (and possibly open) terms. We often omit type annotations when they are unimportant. We adopt the standard notion of bound variables and  $\alpha$ -equivalence, and write  $FV(M)$  and  $FTV(\tau)$  for free variables and free type variables of  $M$  and  $\tau$ , respectively. We use the nullary tuple  $\langle \rangle$  and the nullary product type  $\mathbf{1}$  as the unit value and the unit type.

The typing rules and the left-to-right, call-by-value reduction relation are given by judgments of the forms  $S, \Gamma, \Sigma \vdash M : \tau$  and  $s \triangleright M \rightarrow t \triangleright N$ , where  $S$  is a set of type variables,  $\Gamma$  is a type environment (a partial map from variables to types),  $\Sigma$  is a store typing (a partial map from locations to closed types), and  $s$  and  $t$  are stores (a partial map from locations to values). Their definitions are standard [15]. Key rules involving polymorphism and state are shown in Figure 2 in the appendices. As usual,  $S, \Gamma$ , and  $\Sigma$  are omitted when they are empty. We write  $\rightarrow$  for the reflexive and transitive closure of  $\rightarrow$ . In examples, we use integers and Booleans, which are easy to add as primitives or encode as functions. We write  $\{x \mapsto v\}$  for a finite map  $\{(x, v)\}$  in general. We also write  $f\{x \mapsto v\}$  for  $\{(x, v)\} \cup \{(y, f(y)) \mid y \neq x\}$ , and  $f \uplus \{x \mapsto v\}$  for  $f\{x \mapsto v\}$  only if  $x \notin dom(f)$  (it is undefined otherwise).



$\rho, \sigma, \tau ::=$	type	$U, V, W ::=$	value
$\alpha$	type variable	$x$	variable
$\tau \rightarrow \sigma$	function type	$\lambda x : \tau. M$	function
$\forall \alpha. \tau$	universal type	$\Lambda \alpha. M$	type function
$\exists \alpha. \tau$	existential type	$\text{pack } (\tau, V) \text{ as } \exists \alpha. \sigma$	package
$\tau_1 \times \cdots \times \tau_n$	product type	$\langle V_1, \dots, V_n \rangle$	tuple
$\tau \text{ ref}$	reference type	$\ell$	location
$L, M, N, C, D ::=$	term	$E ::=$	evaluation context
$x$	variable	$[]$	hole
$\lambda x : \tau. M$	abstraction	$EM$	application (left)
$MN$	application	$VE$	application (right)
$\Lambda \alpha. M$	type abstraction	$\text{pack } (\tau, E) \text{ as } \exists \alpha. \sigma$	packing
$M[\tau]$	type application	$\text{open } E \text{ as } (\alpha, x) \text{ in } M$	opening
$\text{pack } (\tau, M) \text{ as } \exists \alpha. \sigma$	packing	$\langle V_1, \dots, V_m, E, M_{m+1}, \dots, M_n \rangle$	tupling
$\text{open } M \text{ as } (\alpha, x) \text{ in } N$	opening	$\#_i(E)$	projection
$\langle M_1, \dots, M_n \rangle$	tupling	$\text{ref } E$	allocation
$\#_i(M)$	projection	$!E$	dereference
$\ell$	location	$E := M$	update (left)
$\text{ref } M$	allocation	$V := E$	update (right)
$!M$	dereference	$E \stackrel{ptr}{=} M ? N_1 : N_2$	pointer equality (left)
$M := N$	update	$V \stackrel{ptr}{=} E ? N_1 : N_2$	pointer equality (right)
$M_1 \stackrel{ptr}{=} M_2 ? N_1 : N_2$	pointer equality		

Fig. 1. Syntax

For simplicity, we (very) often use the abbreviation  $\bar{a}$  to mean the sequence  $a_1, \dots, a_n$  when  $n$  is unimportant, for any kind of meta-variable  $a$ . Furthermore, we often write  $op(\bar{a}, \bar{b}, \dots, \bar{c})$  to mean the sequence  $op(a_1, b_1, \dots, c_1), \dots, op(a_n, b_n, \dots, c_n)$  for various (meta-level) operators  $op$ . For instance,  $\bar{x} : \bar{\tau}$  means  $x_1 : \tau_1, \dots, x_n : \tau_n$ . We always take care that these notations do not create confusion or introduce ambiguity.

The following lemma is important for the “up-to reduction” technique explained in Section 5. Here, we use permutations  $\pi$  on locations because they behave better than substitutions [8].

**Lemma 1.** *Reduction is deterministic up to renaming of fresh locations. That is, if  $s \triangleright M \rightarrow t \triangleright N$  and  $s \triangleright M \rightarrow t_0 \triangleright N_0$ , then  $t_0 \triangleright N_0 = \pi(t \triangleright N)$  for some permutation  $\pi$  on  $\text{Loc} \setminus \text{dom}(s)$ .*

*Proof.* By induction on the derivation of  $s \triangleright M \rightarrow t \triangleright N$ .

Note that the above property is *not* trivial. For instance, reduction would be non-deterministic (even modulo renaming of fresh locations) under the presence of deallocation [22], which disallows the general up-to reduction technique.

The following definition and lemma observe that contexts are reduced either “by themselves without using the value in the hole” or else “by destructing the value in the hole.”

**Definition 1.** Variable  $x$  is at the destruction position in term  $M$  if  $M$  is of the form  $E[xV]$ ,  $E[x[\tau]]$ ,  $E[\text{open } x \text{ as } (\alpha, y) \text{ in } N]$ ,  $E[\#_i(x)]$ ,  $E[!x]$ ,  $E[x := V]$ ,  $E[x \stackrel{ptr}{=} V ? N_1 : N_2]$  or  $E[V \stackrel{ptr}{=} x ? N_1 : N_2]$ .

Note that destruction positions are different from redex positions, e.g., when  $M = Vx$ . Recall also that our reduction is call-by-value.

**Lemma 2 (context reduction).** Suppose  $\bar{\alpha}, \bar{x} : \bar{\tau} \vdash C_1 : \tau$ . If  $C_1$  is not a value and not of the form  $E[\text{ref } V]$ , and if no  $x_i \in \{\bar{x}\}$  is at the destruction position in  $C_1$ , then for some  $C_2$  with  $\bar{\alpha}, \bar{x} : \bar{\tau} \vdash C_2 : \tau$ , we have

$$s \triangleright \theta C_1 \rightarrow s \triangleright \theta C_2$$

for any  $s$  and  $\theta = [\bar{V}/\bar{x}][\bar{\rho}/\bar{\alpha}]$  with  $\Sigma \vdash s$  and  $\Sigma \vdash \bar{V} : \theta \bar{\tau}$ .

*Proof.* By induction on the syntax of  $C_1$ . All cases are trivial, given the standard type soundness theorems (i.e., progress and preservation).

## 4 Environmental Bisimulation

We now define our environmental bisimulation. Readers are referred to previous work for more comprehensive introduction to environmental bisimulations, for polymorphic types (with big-step semantics) [24] or local state (with small-step semantics) [19, Section 1 and 4]. (A subsection in a recent paper [22, Section 1.3] would perhaps be the easiest introduction, even though their language is untyped and includes deallocation.)

**Definition 2.** A concretion environment  $\Delta$  is a partial map from type variables to pairs of closed types. We define  $\Delta^1(\alpha) = \sigma$  and  $\Delta^2(\alpha) = \sigma'$  if  $\Delta(\alpha) = (\sigma, \sigma')$ . We extend their domain from type variables to types in the obvious manner.

Intuitively,  $\Delta(\alpha) = (\sigma, \sigma')$  means that the abstract type  $\alpha$  is implemented by  $\sigma$  on the left hand side of equivalence, and by  $\sigma'$  on the right.

**Definition 3.** A typed value relation  $\mathcal{R}$  is a set of triples of the form  $(V, V', \tau)$ . We write  $\Delta, (\Sigma, \Sigma') \vdash \mathcal{R}$  if  $\Sigma \vdash V : \Delta^1(\tau)$  and  $\Sigma' \vdash V' : \Delta^2(\tau)$  for all  $(V, V', \tau) \in \mathcal{R}$ .

Intuitively,  $\mathcal{R}$  represents the “knowledge” of a context and  $(V, V', \tau) \in \mathcal{R}$  means  $V$  (resp.  $V'$ ) is known under type  $\tau$  to the context on the left (resp. right) hand side. Note that  $\tau$  may be open (with  $FTV(\tau) \subseteq \text{dom}(\Delta)$ ), while  $V$  and  $V'$  are closed (though they may still contain locations). Intuitively, free type variables in  $\tau$  represent names of abstract data types.

**Definition 4.** An environmental relation  $X$  is a set of tuples of the form  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau)$  or  $(\Delta, \mathcal{R}, s, s')$  with appropriate typing, i.e.,

- $\Delta, (\Sigma, \Sigma') \vdash \mathcal{R}$ ,
- $\Sigma \vdash M : \Delta^1(\tau)$  with  $\Sigma \vdash s$ , and
- $\Sigma' \vdash M' : \Delta^2(\tau)$  with  $\Sigma' \vdash s'$

for some  $\Sigma$  and  $\Sigma'$ .

Again, note that  $\tau$  may be open and contain free (i.e., abstract) type variables, while  $M$  and  $M'$  are closed.

Informally,  $X$  is a set of the states of a program and a context.  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X$  means program  $M$  (resp.  $M'$ ) of type  $\tau$  is running under store  $s$  (resp.  $s'$ ) on the left (resp. right) hand side, while  $(\Delta, \mathcal{R}, s, s') \in X$  means that the two programs have stopped with stores  $s$  and  $s'$ , respectively. In both cases,  $\mathcal{R}$  represents the knowledge of the context (i.e., the environment) that has already been given out by the programs.

**Definition 5.** *The context closure  $(\Delta, \mathcal{R})^*$  of  $\mathcal{R}$  under  $\Delta$  is defined as:*  
 $\{ (\overline{V}/\overline{x})\Delta^1(C), [\overline{V}'/\overline{x}]\Delta^2(C), \tau \mid \text{dom}(\Delta), \overline{x} : \overline{\tau} \vdash C : \tau, (\overline{V}, \overline{V}', \overline{\tau}) \in \mathcal{R} \}$

Informally, context closure represents synthesis of knowledge by contexts. With types omitted and infix notation used, it simply says: if  $\overline{V}\mathcal{R}\overline{V}'$ , then  $([\overline{V}/\overline{x}]C)\mathcal{R}^*$   $([\overline{V}'/\overline{x}]C)$ . Recall that our context  $C$  is just a term with free variables  $\overline{x}$ .

The intuitions above lead to the following definition of environmental bisimulation, which asserts that  $X$  is preserved by execution (reduction and evaluation) of the program and by observations (application, type application, opening, projection, allocation, dereference, update, and pointer equality) from the context.

**Definition 6.**  *$X$  is an environmental simulation if:*

1. For any  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X$ ,
  - (a) [Reduction] If  $s \triangleright M \rightarrow t \triangleright N$ , then  $s' \triangleright M' \rightarrow t' \triangleright N'$  for some  $t'$  and  $N'$  with  $(\Delta, \mathcal{R}, t \triangleright N, t' \triangleright N', \tau) \in X$ .
  - (b) [Evaluation] If  $M = V$ , then  $s' \triangleright M' \rightarrow t' \triangleright V'$  for some  $t'$  and  $V'$  with  $(\Delta, \mathcal{R} \cup \{(V, V', \tau)\}, s, t') \in X$ .
2. For any  $(\Delta, \mathcal{R}, s, s') \in X$ ,
  - (a) [Application] If  $(\lambda x : \Delta^1(\tau_1). M, \lambda x : \Delta^2(\tau_1). M', \tau_1 \rightarrow \tau_2) \in \mathcal{R}$ , then  $(\Delta, \mathcal{R}, s \triangleright [W/x]M, s' \triangleright [W'/x]M', \tau_2) \in X$  for any  $(W, W', \tau_1) \in (\Delta, \mathcal{R})^*$ .
  - (b) [Type Application] If  $(\Lambda \alpha. M, \Lambda \alpha. M', \forall \alpha. \tau) \in \mathcal{R}$ , then  $(\Delta, \mathcal{R}, s \triangleright [\Delta^1(\sigma)/\alpha]M, s' \triangleright [\Delta^2(\sigma)/\alpha]M', [\sigma/\alpha]\tau) \in X$  for any  $\sigma$  with  $FTV(\sigma) \subseteq \text{dom}(\Delta)$ .
  - (c) [Opening] If  $(\text{pack } (\sigma, V) \text{ as } \exists \alpha. \Delta^1(\tau), \text{pack } (\sigma', V') \text{ as } \exists \alpha. \Delta^2(\tau), \exists \alpha. \tau) \in \mathcal{R}$ , then  $(\Delta \cup \{\alpha \mapsto (\sigma, \sigma')\}, \mathcal{R} \cup \{(V, V', \tau)\}, s, s') \in X$  for some  $\alpha$ .
  - (d) [Projection] If  $(\langle V_1, \dots, V_n \rangle, \langle V'_1, \dots, V'_n \rangle, \tau_1 \times \dots \times \tau_n) \in \mathcal{R}$ , then  $(\Delta, \mathcal{R} \cup \{(V_i, V'_i, \tau_i)\}, s, s') \in X$  for any  $i \in \{1, \dots, n\}$ .
  - (e) [Allocation]  $(\Delta, \mathcal{R} \cup \{(\ell, \ell', \tau \text{ ref})\}, s \uplus \{\ell \mapsto W\}, s' \uplus \{\ell' \mapsto W'\}) \in X$  for any  $\ell \notin \text{dom}(s)$ ,  $\ell' \notin \text{dom}(s')$  and  $(W, W', \tau) \in (\Delta, \mathcal{R})^*$ .
  - (f) If  $(\ell, \ell', \tau \text{ ref}) \in \mathcal{R}$ , then
    - i. [Dereference]  $(\Delta, \mathcal{R} \cup \{(s(\ell), s'(\ell'), \tau)\}, s, s') \in X$ .
    - ii. [Update]  $(\Delta, \mathcal{R}, s\{\ell \mapsto W\}, s'\{\ell' \mapsto W'\}) \in X$  for any  $(W, W', \tau) \in (\Delta, \mathcal{R})^*$ .
  - (g) [Pointer Equality] If  $(\ell, \ell_1, \tau \text{ ref}) \in \mathcal{R}$  and  $(\ell, \ell_2, \tau \text{ ref}) \in \mathcal{R}$ , then  $\ell_1 = \ell_2$ .

$X$  is an environmental bisimulation if both  $X$  and  $X^{-1}$  are environmental simulations, where

$$\begin{aligned} X^{-1} &= \{(\Delta^{-1}, \mathcal{R}^{-1}, s' \triangleright M', s \triangleright M, \tau) \mid (\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X\} \\ &\quad \cup \{(\Delta^{-1}, \mathcal{R}^{-1}, s', s) \mid (\Delta, \mathcal{R}, s, s') \in X\} \\ \mathcal{R}^{-1} &= \{(V', V, \tau) \mid (V, V', \tau) \in \mathcal{R}\} \end{aligned}$$

and  $\Delta^{-1}$  is defined (at the risk of confusion with the inverse map) by  $\text{dom}(\Delta^{-1}) = \text{dom}(\Delta)$  and  $\Delta^{-1}(\alpha) = (\sigma', \sigma)$  for any  $\alpha$  with  $\Delta(\alpha) = (\sigma, \sigma')$ . Environmental bisimilarity  $\sim$  is the largest environmental bisimulation, which exists because all the conditions above are monotone on  $X$ , i.e., the union of environmental (bi)simulations is again an environmental (bi)simulation.

## 5 Up-to Techniques

As we shall prove in Section 6, environmental bisimilarity characterizes observational equivalence. However, the above definition by itself is not yet convenient enough for proving instances of observational equivalence between programs. As in concurrency theory [20], various up-to techniques are useful for getting rid of this inconvenience. Below, we report a few of such up-to techniques in our setting.

**Definition 7.** The reduction (and renaming) closure  $X^\rightarrow$  of  $X$  is defined as

$$\begin{aligned} X^\rightarrow &= \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid \\ &\quad s \triangleright M \twoheadrightarrow t \triangleright N, \quad s' \triangleright M' \twoheadrightarrow t' \triangleright N', \quad (\Delta, \mathcal{R}, t \triangleright N, t' \triangleright N', \tau) \in \pi^1(X)\} \\ &\quad \cup \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid s \triangleright M \text{ diverges}\} \\ &\quad \cup \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid \\ &\quad s \triangleright M \twoheadrightarrow t \triangleright V, \quad s' \triangleright M' \twoheadrightarrow t' \triangleright V', \quad (\Delta, \mathcal{R} \cup \{(V, V', \tau)\}, t, t') \in \pi^1(X)\} \\ &\quad \cup \{(\Delta, \mathcal{R}, s, s') \mid (\Delta, \mathcal{R}, s, s') \in \pi^1(X)\} \end{aligned}$$

where

$$\begin{aligned} \pi^1(X) &= \{(\Delta, \pi^1(\mathcal{R}), \pi(s) \triangleright \pi(M), s' \triangleright M', \tau) \mid (\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X\} \\ &\quad \cup \{(\Delta, \pi^1(\mathcal{R}), \pi(s), s') \mid (\Delta, \mathcal{R}, s, s') \in X\} \\ \pi^1(\mathcal{R}) &= \{(\pi(V), V', \tau) \mid (V, V', \tau) \in \mathcal{R}\}. \end{aligned}$$

In short,  $X^\rightarrow$  is the set of elements that reduce or evaluate to some element of  $X$  (modulo renaming of locations). We “cross-sell” up-to reduction and up-to renaming, because our reduction is deterministic only up to renaming of fresh locations (Lemma 11).

**Definition 8.**  $X$  is an environmental simulation up-to reduction (and renaming) if the conditions of Definition 6 hold with all the positive occurrences of  $X$  replaced by  $X^\rightarrow$ .

**Lemma 3 (soundness of up-to reduction).** Suppose  $X$  is an environmental simulation up-to reduction. Then  $X^\rightarrow$  is an environmental simulation.

*Proof.* We check each condition of Definition 6 for each element of  $X^\rightarrow$  by expanding Definition 7. Details are found in Appendix A.

The next up-to technique is the most powerful one.

**Definition 9.** *The context (and environment) closure  $X^*$  of  $X$  is defined as:*

$$\begin{aligned} X^* = & \{(\Delta, \mathcal{R}, s \triangleright [\overline{V}/\overline{y}]\Delta_0^1(E)[M], s' \triangleright [\overline{V}'/\overline{y}]\Delta_0^2(E)[M'], \tau) \mid \\ & (\Delta_0, \mathcal{S}, s \triangleright M, s' \triangleright M', \tau_0) \in X, \\ & \Delta \subseteq \Delta_0, \mathcal{R} \subseteq (\Delta_0, \mathcal{S})^*, FTV(\mathcal{R}) \subseteq \text{dom}(\Delta), \\ & (\overline{V}, \overline{V}', \overline{\tau}) \in \mathcal{S}, \text{dom}(\Delta_0), \overline{y} : \overline{\tau} \vdash E[\tau_0] : \tau, FTV(\tau) \subseteq \text{dom}(\Delta)\} \\ \cup & \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid \\ & (\Delta_0, \mathcal{S}, s, s') \in X, \Delta \subseteq \Delta_0, \mathcal{R} \subseteq (\Delta_0, \mathcal{S})^*, FTV(\mathcal{R}) \subseteq \text{dom}(\Delta) \\ & (M, M', \tau) \in (\Delta_0, \mathcal{S})^*, FTV(\tau) \subseteq \text{dom}(\Delta)\} \\ \cup & \{(\Delta, \mathcal{R}, s, s') \mid \\ & (\Delta_0, \mathcal{S}, s, s') \in X, \Delta \subseteq \Delta_0, \mathcal{R} \subseteq (\Delta_0, \mathcal{S})^*, FTV(\mathcal{R}) \subseteq \text{dom}(\Delta)\} \end{aligned}$$

Here,  $E[\tau_0]$  denotes an evaluation context  $E$  with a hole of type  $\tau_0$ , rather than a type application.

As in Definition 5, the above definition is easier to understand if we omit types (and stores), and use an infix notation  $\mathcal{S} \models M X M'$  for  $(\mathcal{S}, M, M') \in X$ .

- If  $\mathcal{S} \models M X M'$  and  $\overline{V} \mathcal{S} \overline{V}'$ , then  $\mathcal{R} \models ([\overline{V}/\overline{y}]E[M]) X^* ([\overline{V}'/\overline{y}]E[M'])$  for any  $\mathcal{R} \subseteq \mathcal{S}^*$ .
- If  $\mathcal{S} \in X$  and  $M \mathcal{S}^* M'$ , then  $\mathcal{R} \models M X^* M'$  for any  $\mathcal{R} \subseteq \mathcal{S}^*$ .
- If  $\mathcal{S} \in X$ , then  $\mathcal{R} \in X^*$  for any  $\mathcal{R} \subseteq \mathcal{S}^*$ .

Here, up-to context and up-to environment (the subset inclusion  $\mathcal{R} \subseteq \mathcal{S}^*$ ) are cross-sold because of small-step semantics: during reduction under context  $E$  or  $C$ , newly known values need to be substituted into their holes, but they cannot be added to  $\mathcal{R}$  until the reduction terminates.

In the first item above, the restriction to evaluation contexts  $E$  is important (the up-to technique would otherwise be unsound in general) but is *not* a weakness of our approach: see Section 6.

**Definition 10.**  *$X$  is an environmental simulation up-to reduction and context (and environment) if the conditions of Definition 6 hold with all the positive occurrences of  $X$  replaced by  $(X^*)^\rightarrow$ .*

It is also possible to consider just  $X^*$  in place of  $(X^*)^\rightarrow$ . We here consider the latter because we often want to use up-to reduction and up-to context at the same time.

**Lemma 4 (soundness of up-to reduction and context).** *Suppose  $X$  is an environmental simulation up-to reduction and context. Then  $X^*$  is an environmental simulation up-to reduction (so  $(X^*)^\rightarrow$  is an environmental simulation).*

*Proof.* We check each condition of Definition 8 for each element of  $X^*$  by expanding Definition 9. Details are in Appendix B.

The last one is specific to calculi with generative names (like our locations).

**Definition 11.** The allocation closure  $X^\nu$  of  $X$  is defined as

$$X^\nu = \{(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \mid (\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X\} \\ \cup \{(\Delta, \mathcal{R}, s, s') \mid (\Delta, \mathcal{S}, t, t') \in X, (\mathcal{R}, s, s') \in (\Delta, \mathcal{S}, t, t')^\nu\}$$

where

$$(\Delta, \mathcal{S}, t, t')^\nu = \{(\mathcal{R}, s, s') \mid \mathcal{R} = \mathcal{S} \cup \{(\bar{\ell}, \bar{\ell}', \bar{\tau} \text{ ref})\}, (\bar{V}, \bar{V}', \bar{\tau}) \in (\Delta, \mathcal{R})^*, \\ s = t \uplus \{\bar{\ell} \mapsto \bar{V}\}, s' = t' \uplus \{\bar{\ell}' \mapsto \bar{V}'\}\}.$$

Informally,  $X^\nu$  is an extension of  $X$  with extra locations  $\bar{\ell}$  and  $\bar{\ell}'$  allocated (and initialized with  $\bar{V}$  and  $\bar{V}'$ ) by the context. (This extension is limited only to elements of the form  $(\Delta, \mathcal{R}, s, s')$  in the definition above. A similar extension is also possible for  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau)$ , but is less useful because  $M$  and  $M'$  often contain the extended locations  $\bar{\ell}$  and  $\bar{\ell}'$ . See, e.g., Example 2 and 3.)

This time, the definition of up-to can *almost* be obtained by replacing positive  $X$  with  $((X^\nu)^*)^\rightarrow$  in Definition 6. However, Condition 2a, 2b and 2(f)ii require adjustments, as underlined below. Such adjustments were unnecessary in up-to reduction (and context) roughly because reduction (and context) closure does not essentially increase  $(\Delta, \mathcal{R}, s, s') \in X$ . This is not the case in allocation closure. Note also that the underlined conditions are still necessary for soundness, e.g., when the observed terms are functions that take  $n$  locations as arguments and return **true** if and only if these locations are pairwise different.

**Definition 12.**  $X$  is an environmental simulation up-to reduction, context, and allocation (or just an environmental simulation up-to in short) if:

1. For any  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in X$ ,
  - (a) If  $s \triangleright M \rightarrow t \triangleright N$ , then  $s' \triangleright M' \rightarrow t' \triangleright N'$  for some  $t'$  and  $N'$  with  $(\Delta, \mathcal{R}, t \triangleright N, t' \triangleright N', \tau) \in ((X^\nu)^*)^\rightarrow$ .
  - (b) If  $M = V$ , then  $s' \triangleright M' \rightarrow t' \triangleright V'$  for some  $t'$  and  $V'$  with  $(\Delta, \mathcal{R} \cup \{(V, V', \tau)\}, s, t') \in ((X^\nu)^*)^\rightarrow$ .
2. For any  $(\Delta, \mathcal{R}, s, s') \in X$ ,
  - (a) If  $(\lambda x: \Delta^1(\tau_1). M, \lambda x: \Delta^2(\tau_1). M', \tau_1 \rightarrow \tau_2) \in \mathcal{R}$ , then  $(\Delta, \mathcal{S}, t \triangleright [W/x]M, t' \triangleright [W'/x]M', \tau_2) \in ((X^\nu)^*)^\rightarrow$  for any  $(\mathcal{S}, t, t') \in (\Delta, \mathcal{R}, s, s')^\nu$  and  $(W, W', \tau_1) \in (\Delta, \mathcal{S})^*$ .
  - (b) If  $(\Lambda \alpha. M, \Lambda \alpha. M', \forall \alpha. \tau) \in \mathcal{R}$ , then  $(\Delta, \mathcal{S}, t \triangleright [\Delta^1(\sigma)/\alpha]M, t' \triangleright [\Delta^2(\sigma)/\alpha]M', [\sigma/\alpha]\tau) \in ((X^\nu)^*)^\rightarrow$  for any  $(\mathcal{S}, t, t') \in (\Delta, \mathcal{R}, s, s')^\nu$  and  $\sigma$  with  $FTV(\sigma) \subseteq \text{dom}(\Delta)$ .
  - (c) If  $(\text{pack}(\sigma, V) \text{ as } \exists \alpha. \Delta^1(\tau), \text{pack}(\sigma', V') \text{ as } \exists \alpha. \Delta^2(\tau), \exists \alpha. \tau) \in \mathcal{R}$ , then  $(\Delta \cup \{\alpha \mapsto (\sigma, \sigma')\}, \mathcal{R} \cup \{(V, V', \tau)\}, s, s') \in ((X^\nu)^*)^\rightarrow$  for some  $\alpha$ .
  - (d) If  $(\langle V_1, \dots, V_n \rangle, \langle V'_1, \dots, V'_n \rangle, \tau_1 \times \dots \times \tau_n) \in \mathcal{R}$ , then  $(\Delta, \mathcal{R} \cup \{(V_i, V'_i)\}, s, s') \in ((X^\nu)^*)^\rightarrow$  for any  $i \in \{1, \dots, n\}$ .
  - (e) No condition required (item left only for the sake of consistent numbering).
  - (f) If  $(\ell, \ell', \tau \text{ ref}) \in \mathcal{R}$ , then
    - i.  $(\Delta, \mathcal{R} \cup \{(s(\ell), s'(\ell')), \tau\}, s, s') \in ((X^\nu)^*)^\rightarrow$ .

ii.  $(\Delta, \mathcal{S}, t\{\ell \mapsto W\}, t'\{\ell' \mapsto W'\}) \in ((X^\nu)^*)^\rightarrow$  for any  $(\mathcal{S}, t, t') \in (\Delta, \mathcal{R}, s, s')^\nu$  and  $(W, W', \tau) \in (\Delta, \mathcal{S})^*$ .

(g) If  $(\ell, \ell'_1, \tau \text{ ref}) \in \mathcal{R}$  and  $(\ell, \ell'_2, \tau \text{ ref}) \in \mathcal{R}$ , then  $\ell'_1 = \ell'_2$ .

**Lemma 5 (soundness of up-to reduction, context, and allocation).** *Suppose  $X$  is an environmental simulation up-to reduction, context, and allocation. Then  $X^\nu$  is an environmental simulation up-to reduction and context (so  $((X^\nu)^*)^\rightarrow$  is an environmental simulation).*

*Proof.* We check each condition of Definition 10 for each element of  $X^\nu$  by expanding Definition 11. Details in Appendix C.

## 6 The Characterization Theorem

We now prove that environmental bisimilarity coincides with a form of observational equivalence. Let  $\equiv$  be the largest environmental relation such that  $\equiv^* \subseteq \equiv$  and for any  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in \equiv$ ,  $s \triangleright M$  converges if and only if  $s' \triangleright M'$  does. It exists because the union of all such environmental relations trivially satisfies the same property ( $X$  appears only once in a negative position in each clause of Definition 9).

The relation  $\equiv$  corresponds to the conventional definition of contextual equivalence in the following way. Take two closed values  $V$  and  $V'$  of type  $\tau$ . If  $(\emptyset, \{(V, V', \tau)\}, \emptyset, \emptyset) \in \equiv$ , then  $(\emptyset, \emptyset, \emptyset \triangleright [V/x]C, \emptyset \triangleright [V'/x]C) \in \equiv^*$  by Definition 9 for any well-typed  $C$ . Therefore,  $[V/x]C$  and  $[V'/x]C$  coterminate by the definition above. Conversely, if  $V$  and  $V'$  coterminate under arbitrary (well-typed) contexts, then  $\{(\emptyset, \{(V, V', \tau)\}, \emptyset, \emptyset)\}^*$  satisfies the above property. Hence  $(\emptyset, \{(V, V', \tau)\}, \emptyset, \emptyset) \in \equiv$ .

Although the argument above assumed closed values, it is also straightforward to treat open terms  $N$  and  $N'$ , by taking  $V = \lambda \bar{x}. N$  and  $V' = \lambda \bar{x}. N'$  for  $\{\bar{x}\} \supseteq FV(N) \cup FV(N')$  as in previous work [24, Section 6] [10, Appendix A.2].

**Lemma 6 (soundness of environmental bisimulation).** *Environmental bisimilarity  $\sim$  is included in  $\equiv$ .*

*Proof.* Let  $\preceq$  be the environmental similarity. By Lemma 4,  $(\preceq^*)^\rightarrow$  is an environmental simulation and therefore  $\preceq^* \subseteq (\preceq^*)^\rightarrow \subseteq \preceq$ . By symmetry,  $\succeq^* \supseteq \succeq$  (where  $\succeq$  denotes  $\preceq^{-1}$ ). Hence  $\sim^* = (\preceq \cap \succeq)^* \subseteq (\preceq^* \cap \succeq^*) \subseteq (\preceq \cap \succeq) = \sim$ . Also, by Condition 1a) and 1b) of Definition 6, for any  $(\Delta, \mathcal{R}, s \triangleright M, s' \triangleright M', \tau) \in \sim$ ,  $s \triangleright M$  converges if and only if  $s' \triangleright M'$  does. Hence  $\sim \subseteq \equiv$ .

**Lemma 7 (completeness).**  *$\equiv$  is an environmental bisimulation.*

*Proof.* By checking each condition of Definition 6 for each element of  $\equiv$ , expanding Definition 9. Again, details are found in Appendix D.

**Theorem 1 (characterization).** *Environmental bisimilarity  $\sim$  equals observational equivalence  $\equiv$ .*

## 7 Examples

Below, we present examples of equivalence proofs by our environmental bisimulation. In each example, we prove the equivalence of the first two terms by constructing an environmental bisimulation  $X$  up-to reduction, context, and allocation. More examples are given in Appendix E.

*Example 1 (abstract counters with and without bounds checking [3, Section 5.1]).*  
Let

$$\begin{aligned} mkCnt &= (\text{let } x = \text{ref } 0 \text{ in } cnt_x) & mkCnt' &= (\text{let } x = \text{ref } 0 \text{ in } cnt'_x) \\ cnt_x &= \text{pack}(\text{int}, \langle incr_x, chk_x \rangle) \text{ as } \tau & cnt'_x &= \text{pack}(\text{int}, \langle incr_x, chk'_x \rangle) \text{ as } \tau \\ chk_x &= \lambda z. z \leq !x & chk'_x &= \lambda z. \text{true} \\ incr_x &= \lambda \_. ++x & \tau &= \exists \alpha. (\mathbf{1} \rightarrow \alpha) \times (\alpha \rightarrow \text{bool}) \end{aligned}$$

using standard syntactic sugar such as  $++x = (x := !x + 1; !x)$ . Then  $mkCnt$  and  $mkCnt'$  are equivalent because of the following  $X$ .

$$\begin{aligned} X &= \{(\emptyset, \emptyset, \emptyset \triangleright mkCnt, \emptyset \triangleright mkCnt', \tau)\} \\ &\cup \{(\Delta, \mathcal{R}, s, s') \mid \\ &\quad \mathcal{R} = \{(\text{incr}_\ell, \text{incr}_{\ell'}, \mathbf{1} \rightarrow \alpha), (\text{chk}_\ell, \text{chk}'_\ell, \alpha \rightarrow \text{bool}), (1, 1, \alpha), \dots, (n, n, \alpha)\}, \\ &\quad \Delta = \{\alpha \mapsto (\text{int}, \text{int}), s = \{\ell \mapsto n\}, s' = \{\ell' \mapsto n\}, n \in \{0, 1, 2, \dots\}\} \end{aligned}$$

Let us check that  $X$  is indeed an environmental bisimulation up-to reduction, context, and allocation. Most conditions hold just by construction. The only cases that need to be checked are Condition 2a for the first and second elements of the above  $\mathcal{R}$ . Both of them are straightforward, given the fact that  $(W, W', \mathbf{1}) \in (\Delta, \mathcal{R})^*$  implies  $W = W' = \langle \rangle$  and  $(W, W', \alpha) \in (\Delta, \mathcal{R})^*$  implies  $W = W' \in \{1, 2, \dots, n\}$  (immediate from Definition 5).

*Example 2 (irreversible state change [3, Section 5.5], or the “awkward” example [17, Example 5.9]).* Below is an example that can be proved by recent work [3] but cannot by classic one [17]. This example uses no existential types, but is still interesting because of local state. (It can easily be turned into an equivalence involving packages, like Example 3.)

$$\begin{aligned} M &= (\text{let } x = \text{ref } 0 \text{ in } V_x) & M' &= V' \\ V_x &= \lambda f. x := 1; f \langle \rangle; !x & V' &= \lambda f. f \langle \rangle; 1 \quad \tau = (\mathbf{1} \rightarrow \mathbf{1}) \rightarrow \text{int} \end{aligned}$$

for which we take

$$\begin{aligned} X &= \{(\emptyset, \emptyset, \emptyset \triangleright M, \emptyset \triangleright M', \tau)\} \\ &\cup \{(\emptyset, \mathcal{R}, s, \emptyset) \mid \mathcal{R} = \{(V_\ell, V', \tau)\}, s = \{\ell \mapsto i\}, i \in \{0, 1\}\} \\ &\cup \{(\emptyset, \mathcal{R}, s \uplus t \triangleright N, t' \triangleright N', \text{int}) \mid \\ &\quad \mathcal{R} = \{(V_\ell, V', \tau), (\bar{k}, \bar{k}', \bar{\rho} \text{ ref})\}, s = \{\ell \mapsto 1\}, \\ &\quad \text{dom}(t) = \{\bar{k}\}, \text{dom}(t') = \{\bar{k}'\}, (t(\bar{k}), t'(\bar{k}'), \bar{\rho}) \in (\emptyset, \mathcal{R})^*, \\ &\quad (N, N', \text{int}) \in \{(E_1[\dots[E_n[C; !\ell]; !\ell]\dots]; !\ell, E_1[\dots[E_n[C; 1]; 1]\dots]; 1)\}^{(\emptyset, \mathcal{R})}. \end{aligned}$$

Here,  $\mathcal{T}^{(\Delta, \mathcal{R})}$  denotes closure under contexts in  $\mathcal{T}$ , i.e.,

$$\mathcal{T}^{(\Delta, \mathcal{R})} = \{([\bar{V}/\bar{x}]\Delta^1(C), [\bar{V}'/\bar{x}]\Delta^2(C'), \tau) \mid (C, C') \in \mathcal{T}, (\bar{V}, \bar{V}', \bar{\tau}) \in \mathcal{R}, \text{dom}(\Delta), \bar{x} : \bar{\tau} \vdash C : \tau, \text{dom}(\Delta), \bar{x} : \bar{\tau} \vdash C' : \tau\}.$$



The irreversible change of state is represented by the requirement  $s = \{\ell \mapsto 1\}$  in the third subset of  $X$  above. Stores  $t$  and  $t'$  account for locations  $\bar{k}$  and  $\bar{k}'$  (and their contents) created by the contexts. The most important technique here is the inclusion of all contexts of the form  $E_1[\dots[E_n[C; !\ell]; !\ell]\dots]; !\ell$  and  $E_1[\dots[E_n[C; 1]; 1]\dots]; 1$ , representing nested calls to  $V_\ell$  and  $V'$ . Then, the only non-trivial case to prove is Condition [1a](#) for  $N$  and  $N'$ , which follows from Lemma [2](#).

*Example 3 (well-bracketed state change [3](#), Section 5.7), credited to Jacob Thamsborg*). This is an existential variant of a negative example in [3](#) (i.e., they could not prove it).

$$\begin{array}{ll}
 M = \text{pack}(\text{int ref}, \langle \text{ref } 1, \lambda x. V_x \rangle) \text{ as } \sigma & M' = \text{pack}(\mathbf{1}, \langle \langle \rangle, \lambda \_ . V' \rangle) \text{ as } \sigma \\
 V_x = \lambda f. (x := 0; f \langle \rangle; x := 1; f \langle \rangle; !x) & V' = \lambda f. (f \langle \rangle; f \langle \rangle; 1) \\
 \sigma = \exists \alpha. \alpha \times (\alpha \rightarrow \tau) & \tau = (\mathbf{1} \rightarrow \mathbf{1}) \rightarrow \text{int}
 \end{array}$$

The difficulty of this example lies in how to represent the fact that the mutations  $x := 0$  and  $x := 1$  are “well-bracketed,” i.e., whenever  $x$  is mutated to 0, it will eventually be restored to 1.

Consider first the context’s observations on  $M$ . By opening and projection, the context learns some location  $\ell$  under an abstract type  $\alpha$  (with store  $\{\ell \mapsto 1\}$ ) and the function  $\lambda x. V_x$  of type  $\alpha \rightarrow \tau$ . By applying the latter to the former, it then learns function  $V_\ell$ . This function can be applied to some  $f = \lambda \_ . [\overline{V}/\overline{x}]C$  (where  $\overline{V}$  are taken from the context’s knowledge), yielding a term  $N_1$  of the form  $[\overline{V}/\overline{x}]C_1; \ell := 1; [\overline{V}/\overline{x}]D_1; !\ell$  with store  $\{\ell \mapsto 0\}$ . By Lemma [2](#), any non-value of the form  $[\overline{V}/\overline{x}]C_1$  either reduces to another term of the same form, or else “uses” some  $V_i$ . In the former case, the form of  $N_1$  does not change. In the latter case, suppose  $V_i = V_\ell$  and  $N_1 = E[V_\ell W]$  for some  $E$  and  $W$ , that is,  $N_1$  makes a nested call to  $V_\ell$  (otherwise, the form of  $N_1$  does not change, either). Then  $N_1$  reduces to a term of the form  $E[[\overline{V}/\overline{x}]C_2; \ell := 1; [\overline{V}/\overline{x}]D_2; !\ell]$  and the above arguments can be repeated for the subterm  $[\overline{V}/\overline{x}]C_2; \ell := 1; [\overline{V}/\overline{x}]D_2; !\ell$ . (Similar arguments apply to  $V'$  as well.) To enumerate all such terms that are possible under the store  $\{\ell \mapsto 0\}$ , we define a binary relation  $\mathcal{T}_\ell^0$  on contexts by induction, using free variable  $v$  as a hole to substitute  $V_\ell$  (or  $V'$ ).

- $(C; \ell := 1; D; !\ell) \mathcal{T}_\ell^0 (C; D; 1)$
- If  $E[vW] \mathcal{T}_\ell^0 E'[vW]$ , then  $E[C; \ell := 1; D; !\ell] \mathcal{T}_\ell^0 E'[C; D; 1]$

On the other hand, if  $[\overline{V}/\overline{x}]C$  converges to value  $\langle \rangle$ , then  $[\overline{V}/\overline{x}]C; \ell := 1; [\overline{V}/\overline{x}]D; !\ell$  reduces to  $[\overline{V}/\overline{x}]D; !\ell$  with store  $\{\ell \mapsto 1\}$ . Similarly,  $E[[\overline{V}/\overline{x}]C; \ell := 1; [\overline{V}/\overline{x}]D; !\ell]$  reduces to  $E[[\overline{V}/\overline{x}]D; !\ell]$ . We therefore define another binary relation  $\mathcal{T}_\ell^1$  on contexts to enumerate possible terms under the store  $\{\ell \mapsto 1\}$ .

- $(D; !\ell) \mathcal{T}_\ell^1 (D; 1)$
- If  $E[vW] \mathcal{T}_\ell^0 E'[vW]$ , then  $E[D; !\ell] \mathcal{T}_\ell^1 E'[D; 1]$

Again by Lemma [2](#), any non-value of the form  $[\overline{V}/\overline{x}]D$  either reduces to the same form or makes a nested call to  $V_\ell$ . Hence the additional rules:

- If  $E[vW] \mathcal{T}_\ell^1 E'[vW]$ , then  $E[C; \ell := 1; D; !\ell] \mathcal{T}_\ell^0 E'[C; D; 1]$
- If  $E[vW] \mathcal{T}_\ell^1 E'[vW]$ , then  $E[D; !\ell] \mathcal{T}_\ell^1 E'[D; 1]$

This concludes the definition of  $\mathcal{T}_\ell^0$  and  $\mathcal{T}_\ell^1$ . The following lemmas—proved by simple case analysis on the derivations of  $E[vW] \mathcal{T}_\ell^0 E'[vW]$  and  $E[vW] \mathcal{T}_\ell^1 E'[vW]$ —are used when  $[\overline{V}/\overline{x}]D$  converges to value  $\langle \rangle$  and therefore  $E[[\overline{V}/\overline{x}]D; !\ell]$  reduces to  $E[1]$ .

- If  $E[vW] \mathcal{T}_\ell^1 E'[vW]$ , then  $E[1] \mathcal{T}_\ell^1 E'[1]$ .
- If  $E[vW] \mathcal{T}_\ell^0 E'[vW]$ , then  $E[1] \mathcal{T}_\ell^1 E'[1]$ .

Then, take:

$$\begin{aligned}
X = & \{(\emptyset, \emptyset, \emptyset \triangleright M, \emptyset \triangleright M', \sigma)\} \\
& \cup \{(\Delta, \mathcal{R}, s, \emptyset) \mid \\
& \quad s = \{\ell \mapsto 1\}, \quad \Delta = \{\alpha \mapsto (\text{int ref}, \mathbf{1})\}, \\
& \quad \mathcal{R} = \{(\ell, \langle \rangle, \alpha), (\lambda x. V_x, \lambda \_. V', \alpha \rightarrow \tau), (V_\ell, V', \tau)\}\} \\
& \cup \{(\Delta, \mathcal{R}, s \uplus t \triangleright N, t' \triangleright N', \text{int}) \mid \\
& \quad (N, N', \text{int}) \in \mathcal{T}_\ell^0(\Delta, \mathcal{R}), \quad s = \{\ell \mapsto 0\}, \quad \Delta = \{\alpha \mapsto (\text{int ref}, \mathbf{1})\}, \\
& \quad \mathcal{R} = \{(\ell, \langle \rangle, \alpha), (\lambda x. V_x, \lambda \_. V', \alpha \rightarrow \tau), (V_\ell, V', \tau), (\overline{k}, \overline{k'}, \overline{\rho} \text{ ref})\} \\
& \quad \text{dom}(t) = \{\overline{k}\}, \quad \text{dom}(t') = \{\overline{k'}\}, \quad (t(\overline{k}), t'(\overline{k'}), \overline{\rho}) \in (\Delta, \mathcal{R})^*\} \\
& \cup \{(\Delta, \mathcal{R}, s \uplus t \triangleright N, t' \triangleright N', \text{int}) \mid \\
& \quad (N, N', \text{int}) \in \mathcal{T}_\ell^1(\Delta, \mathcal{R}), \quad s = \{\ell \mapsto 1\}, \quad \Delta = \{\alpha \mapsto (\text{int ref}, \mathbf{1})\}, \\
& \quad \mathcal{R} = \{(\ell, \langle \rangle, \alpha), (\lambda x. V_x, \lambda \_. V', \alpha \rightarrow \tau), (V_\ell, V', \tau), (\overline{k}, \overline{k'}, \overline{\rho} \text{ ref})\} \\
& \quad \text{dom}(t) = \{\overline{k}\}, \quad \text{dom}(t') = \{\overline{k'}\}, \quad (t(\overline{k}), t'(\overline{k'}), \overline{\rho}) \in (\Delta, \mathcal{R})^*\}
\end{aligned}$$

Thanks to the construction of  $\mathcal{T}_\ell^0$  and  $\mathcal{T}_\ell^1$  with the lemmas above, the proof that  $X$  is a bisimulation up-to is routine, using Lemma 2 for reduction of the terms  $N$  and  $N'$ .

## 8 Conclusion

We have presented the first complete, purely syntactic (“semantics-free,” as opposed to syntax-free) proof technique for observational equivalences in polymorphic  $\lambda$ -calculus with full references, with non-trivial examples that could not be proved previously. Although we omitted explicit recursion, recursive functions can be encoded [15, Exercise 13.5.8]. Treatment of recursive types (either equi-recursive or iso-recursive) is also straightforward (see, e.g., [24]). Deallocation and pointer arithmetic (by defining stores as partial maps from locations to *arrays* of values) can also be added without essential difficulty, though deallocation introduces non-determinism [22] and invalidates the up-to reduction technique in general (but “up-to *deterministic* reduction” is still possible).

Of course, the above facts do *not* mean other approaches are useless. On the contrary, the inclusion of an infinite number of contexts in examples with callbacks suggests that, at least for some special cases, more convenient techniques (like [3]) can be devised to reduce the “size” of the set to be constructed by the user. (On the other hand, those examples have also shown that, with the help of Lemma 2, our “brute-force” method is often simple enough.) Logical relations are also better at giving a compositional model of universal types, as in [3] and [5, 6].

As we have shown in recent work [22], our approach is applicable to more general properties other than observational equivalence, such as memory safety and space improvement. It would also be possible to adapt them to our typed setting. Contrary to the previous (too negative) conjecture [24, Section 8], it *is* possible as well to use our method to prove free theorems à la Wadler [26] based on parametricity [18]. Work is ongoing on this topic.

*Acknowledgements.* Derek Dreyer first referred me to his work with Ahmed and Rossberg [3], from which I took examples in the present paper. Some of the anonymous reviewers found (and even corrected) a number of serious typos in a previous version of this paper.

## References

- [1] Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: Thirteenth Annual IEEE Symposium on Logic in Computer Science, pp. 334–344 (1998)
- [2] Ahmed, A., Appel, A.W., Virga, R.: An indexed model of impredicative polymorphism and mutable references (2003), <http://www.cs.princeton.edu/~amal/papers/impred.pdf>
- [3] Ahmed, A., Dreyer, D., Rossberg, A.: State-dependent representation independence. In: Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 340–353 (2009)
- [4] Appel, A.W., McAllester, D.: An indexed model of recursive types for foundational proof-carrying code. ACM Transactions on Programming Languages and Systems 23(5), 657–683 (2001)
- [5] Birkedal, L., Støvring, K., Thamsborg, J.: Realizability semantics of parametric polymorphism, references, and recursive types. In: Foundations of Software Science and Computation Structures. LNCS, vol. 5504, pp. 456–470. Springer, Heidelberg (2009)
- [6] Birkedal, L., Støvring, K., Thamsborg, J.: Relational parametricity for references and recursive types. In: Types in Language Design and Implementation, pp. 91–104 (2009)
- [7] Bohr, N., Birkedal, L.: Relational reasoning for recursive types and references. In: Kobayashi, N. (ed.) APLAS 2006. LNCS, vol. 4279, pp. 79–96. Springer, Heidelberg (2006)
- [8] Gabbay, M., Pitts, A.: A new approach to abstract syntax involving binders. In: 14th Annual IEEE Symposium on Logic in Computer Science, pp. 214–224 (1999)
- [9] Gordon, A.D.: Functional Programming and Input/Output. PhD thesis, University of Cambridge (1993)
- [10] Koutavas, V., Wand, M.: Small bisimulations for reasoning about higher-order imperative programs. In: Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 141–152 (2006)
- [11] Laird, J.: A fully abstract trace semantics for general references. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 667–679. Springer, Heidelberg (2007)
- [12] Lassen, S.B., Levy, P.B.: Typed normal form bisimulation for parametric polymorphism. In: 23rd Annual IEEE Symposium on Logic in Computer Science, pp. 341–352 (2008)

- [13] Meyer, A.R., Sieber, K.: Towards fully abstract semantics for local variables: Preliminary report. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 191–203 (1988)
- [14] Mitchell, J.C.: On the equivalence of data representations. In: Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pp. 305–330. Academic Press, London (1991)
- [15] Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
- [16] Pierce, B.C., Sangiorgi, D.: Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM* 47(3), 531–586 (2000); Extended abstract appeared in: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 531–584 (1997)
- [17] Pitts, A.M., Stark, I.: Operational reasoning for functions with local state. In: Higher Order Operational Techniques in Semantics, pp. 227–273. Cambridge University Press, Cambridge (1998)
- [18] Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: Information Processing 1983, Proceedings of the IFIP 9th World Computer Congress, pp. 513–523 (1983)
- [19] Sangiorgi, D., Kobayashi, N., Sumii, E.: Environmental bisimulations for higher-order languages. In: Twenty-Second Annual IEEE Symposium on Logic in Computer Science, pp. 293–302 (2007)
- [20] Sangiorgi, D., Milner, R.: The problem of “weak bisimulation up to”. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 32–46. Springer, Heidelberg (1992)
- [21] Støvring, K., Lassen, S.B.: A complete, co-inductive syntactic theory of sequential control and state. In: Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 161–172 (2007)
- [22] Sumii, E.: A theory of non-monotone memory (or: Contexts for **free**). In: 18th European Symposium on Programming. LNCS, vol. 5502, pp. 237–251. Springer, Heidelberg (2009)
- [23] Sumii, E., Pierce, B.C.: A bisimulation for dynamic sealing. *Theoretical Computer Science* 375(1–3), 169–192 (2007); Extended abstract appeared in: Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 161–172 (2004)
- [24] Sumii, E., Pierce, B.C.: A bisimulation for type abstraction and recursion. *Journal of the ACM* 54(5-26), 1–43 (2007); Extended abstract appeared in: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 63–74 (2005)
- [25] Tzevelekos, N.: Full abstraction for nominal general references. In: Twenty-Second Annual IEEE Symposium on Logic in Computer Science, pp. 399–410 (2007)
- [26] Wadler, P.: Theorems for free! In: Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming Languages and Computer Architecture, pp. 347–359. ACM Press, New York (1989)

# Non-Commutative First-Order Sequent Calculus

Makoto Tatsuta

National Institute of Informatics  
2-1-2 Hitotsubashi, Tokyo 101-8430, Japan  
tatsuta@nii.ac.jp

**Abstract.** This paper investigates a non-commutative first-order sequent calculus NCLK. For that, this paper extends a non-commutative positive fragment to a full first-order sequent calculus  $LK^-$  having antecedent-grouping and no right exchange rule. This paper shows (1) NCLK is equivalent to LJ, (2) NCLK with the exchange rule is equivalent to LK, (3)  $LK^-$  is equivalent to LJ, and (4) translations between  $LK^-$  and NCLK.

## 1 Introduction

Substructural logics, which are logical systems without some of the contraction rule, the weakening rule, and the exchange rule, have been actively studied in both mathematical logic and computer science. For example, linear logic, which is a logical system without the contraction rule, is successful [3].

We will present a first-order sequent calculus NCLK without the exchange rule, called Non-Commutative First-Order Sequent Calculus. The system has the same language of the first-order classical sequent calculus LK, but has only a restricted set of inference rules. We will show the system is equivalent to the first-order intuitionistic sequent calculus LJ. We will also show the system NCLK becomes equivalent to LK when the exchange rule is added to the system. This shows the exchange rule gives a classical principle. We respect order of formulas in a sequent in the system, but conjunction and disjunction are proved to be commutative according to its inference rules.

Substructural logic without the exchange rule that has non-commutative conjunction and disjunction has been studied, but substructural logic without the exchange rule that has commutative conjunction and disjunction has not been fully studied yet. Recently several interesting results have been discovered for this kind of substructural logic. [1] showed a positive fragment of infinitary Peano Arithmetic without the exchange rule has 1-backtracking game theoretic semantics. [2] showed a positive fragment of infinitary Peano Arithmetic without the exchange rule is equivalent to a positive fragment of infinitary Heyting Arithmetic with the law of excluded middle for  $\Sigma_1^0$ -formulas.

This paper will first investigate an underlying logic for those papers. Those papers discussed arithmetic, but we restrict our attention to only its underlying logic, and show the logic itself has a surprising property, that is, the equivalence to LJ. Those papers discussed only a positive system that does not have implication, but we extend a positive fragment to a full logic  $LK^-$  with implication. The

system  $LK^-$  has a sequent having antecedent-grouping and does not have the right exchange rule. Formulas in the antecedent are grouped and structural rules can be used only inside a group. This system is proved to be equivalent to LJ. A key of the equivalence proof is analyzing the minimum length of succedents of sequents in a given proof.

Secondly, we will give the system NCLK, which is obtained from  $LK^-$  by coding grouping information by the length of a sequence of formulas. We will give translations between NCLK and  $LK^-$  and show they preserve provability. Combining the equivalence between  $LK^-$  and LJ, these translations will prove the equivalence between NCLK and LJ. On the other hand, when we add the exchange rule to NCLK, the coding information will be lost and it is proved to become equivalent to LK.

Technical novelties of this paper are (1) the extension of the non-commutative positive fragment [12] to the full non-commutative logic  $LK^-$  with implication, (2) the equivalence between  $LK^-$  and LJ, (3) the definition of NCLK by coding grouping information by the length of a sequence of formulas, and (4) translations between  $LK^-$  and NCLK.

[2] showed the fragment of arithmetic without the exchange rule is equivalent to the fragment of intuitionistic arithmetic with the law  $EM_1$  of excluded middle for  $\Sigma_1^0$  formulas. On the other hand, our system  $LK^-$  is equivalent to LJ. We can explain reasons for the difference for  $EM_1$  in the following way. The first reason is that the minimum length of the succedents in the sequents in the proof is a key for proving the equivalence. When a proof is given in  $LK^-$ , we can immediately find the minimum length. On the other hand, when a proof is given in the system in [12], since it is an infinitary system, we cannot find the minimum, and instead we can only have flag formulas that are some  $\Pi_1^0$ -formulas. For case analysis by flag formulas, [2] needed  $EM_1$ . The second reason is that we can directly show  $LK^-$  does not derive  $EM_1$ , and on the other hand we can drive  $EM_1$  in the system in [12] by using infinitary logic and true atomic formulas.

[45] investigated the sequent calculus obtained from LK by restricting the implication right rule to only intuitionistic sequents and showed the system is equivalent to LJ. Our system NCLK will give another way of restriction to LK so that the resulting system becomes equivalent to LJ.

A potential application of these systems  $LK^-$  and NCLK will be program extraction, since it is equivalent to first-order intuitionistic logic.

Section 2 defines and discusses  $LK^-$ . We give definitions of LK and LJ in Section 3. Section 4 proves the implication from  $LK^-$  to LJ and Section 5 proves the other implication from LJ to  $LK^-$ . We define and discuss NCLK in Section 6. Section 7 gives the translations between NCLK and  $LK^-$ , and shows the equivalence between NCLK and LJ.

## 2 The System $LK^-$

**Definition 2.1 (language).** The language is a first-order language generated from the following symbols. We have variables  $x, y, z, \dots$ . We have constants and function symbols. Terms are constructed from variables, constants, and function

symbols, and denoted by  $s, t, \dots$ . We have predicate symbols including 0-ary predicate symbols  $\top$  and  $\perp$ , which mean the truth and the falsity respectively. Atomic formulas are constructed from predicate symbols and terms, and denoted by  $a, b, \dots$ . Formulas are defined by

$$A, B, C, D, \dots ::= a \mid A \wedge B \mid A \vee B \mid A \rightarrow B \mid \forall x A \mid \exists x A.$$

We will write  $\neg A$  for  $A \rightarrow \perp$ .  $A[t/x]$  is the formula obtained from  $A$  by replacing  $x$  by  $t$ .

A sequent is of the form  $\Gamma \vdash A_1, \dots, A_n$  where  $\Gamma$  is a sequence of formulas and  $n$  occurrences of the symbol  $-$ .

In the sequent  $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \dots, -, \Gamma_n \vdash A_1, \dots, A_n$  where  $\Gamma_i$  is a sequence of formulas and does not contain the symbol  $-$ , the group  $\Gamma_0$  means the initial group, and the  $i$ -th group  $\Gamma_i$  corresponds to  $A_i$ .

$\Gamma, \Delta, \Pi, \Sigma, \dots$  denote a sequence of both formulas and symbols  $-$ . We will write  $-^n$  for  $-, \dots, -$  ( $n$  times).  $|\Gamma|$  denotes the number of the formulas in  $\Gamma$ .  $\#_-\Gamma$  denotes the number of the  $-$  symbols in  $\Gamma$ .

We respect order of formulas in a sequence and a sequent.

We have the following inference rules:

$$\begin{array}{c} \frac{}{\Gamma_1, A, \Gamma_2 \vdash \Delta, A} (Ax) \quad \frac{}{\Gamma \vdash \Delta, \top} (Ax\top) \quad \frac{}{\Gamma_1, \perp, \Gamma_2 \vdash \Delta} (Ax\perp) \\ \frac{\Gamma, - \vdash \Delta, A \wedge B, A \quad \Gamma, - \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B} (\wedge R) \\ \frac{\Gamma_1, A \wedge B, \Gamma_2, A \vdash \Delta}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta} (\wedge L1) \quad \frac{\Gamma_1, A \wedge B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta} (\wedge L2) \\ \frac{\Gamma, - \vdash \Delta, A \vee B, A}{\Gamma \vdash \Delta, A \vee B} (\vee R1) \quad \frac{\Gamma, - \vdash \Delta, A \vee B, B}{\Gamma \vdash \Delta, A \vee B} (\vee R2) \\ \frac{\Gamma_1, A \vee B, \Gamma_2, A \vdash \Delta \quad \Gamma_1, A \vee B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \vee B, \Gamma_2 \vdash \Delta} (\vee L) \\ \frac{\Gamma, A \vdash \Delta, A \rightarrow B}{\Gamma \vdash \Delta, A \rightarrow B} (\rightarrow R1) \quad \frac{\Gamma, - \vdash \Delta, A \rightarrow B, B}{\Gamma \vdash \Delta, A \rightarrow B} (\rightarrow R2) \\ \frac{\Gamma_1, A \rightarrow B, \Gamma_2, - \vdash \Delta, A \quad \Gamma_1, A \rightarrow B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \Delta} (\rightarrow L) \\ \frac{\Gamma, - \vdash \Delta, \forall x A, A}{\Gamma \vdash \Delta, \forall x A} (\forall R) \quad \frac{\Gamma_1, \forall x A, \Gamma_2, A[t/x] \vdash \Delta}{\Gamma_1, \forall x A, \Gamma_2 \vdash \Delta} (\forall L) \\ \frac{\Gamma, - \vdash \Delta, \exists x A, A[t/x]}{\Gamma \vdash \Delta, \exists x A} (\exists R) \quad \frac{\Gamma_1, \exists x A, \Gamma_2, A \vdash \Delta}{\Gamma_1, \exists x A, \Gamma_2 \vdash \Delta} (\exists L) \\ \frac{\Gamma \vdash \Delta}{\Gamma, - \vdash \Delta, A} (weak R) \quad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} (weak L) \end{array}$$

where the conclusion does not contain free occurrences of  $x$  in the rules  $(\forall R)$  and  $(\exists L)$ .

Intuitive meaning of provable sequents is given as follows: If  $\Gamma_0, -, \Gamma_1, \dots, -, \Gamma_n \vdash A_1, \dots, A_n$  is provable, then (1)  $\Gamma_0 \vdash$  is true, or (2)  $\Gamma_0, -, \Gamma_1, \dots, -, \Gamma_i \vdash A_i$  is true for some  $i$ . Each inference rule is sound by this interpretation. Theorem 4.1 will provide more information.

We explain this system with some examples.

**Example 2.2.** The first example shows its conjunction is commutative.

$$\frac{\frac{\frac{}{-, A \wedge B, -, B \vdash B \wedge A, B} (Ax)}{-, A \wedge B, - \vdash B \wedge A, B} (\wedge L2)}{-, A \wedge B \vdash B \wedge A} \quad \frac{\frac{\frac{}{-, A \wedge B, -, A \vdash B \wedge A, A} (Ax)}{-, A \wedge B, - \vdash B \wedge A, A} (\wedge L1)}{-, A \wedge B \vdash B \wedge A} (\wedge R)$$

**Example 2.3.** The next example shows how this system respects the order of formulas. We have three provable sequents

$$\begin{aligned} & -, A, -, B \vdash A, \perp, \\ & -, A, -, B \vdash \perp, A, \\ & -, A, -, B \vdash \perp, B. \end{aligned}$$

On the other hand the sequent

$$-, A, -, B \vdash B, \perp$$

is not provable. The first sequent is provable since the initial and the first groups give the assumption  $A$ , which proves the first formula  $A$ . The second sequent is provable since the initial, the first, and the second groups give the assumptions  $A, B$ , which prove the second formula  $A$ . The third sequent is provable similarly to the second sequent, since the initial, the first, and the second groups give the assumptions  $A, B$ , which prove the second formula  $B$ . Formally the first sequent is proved by

$$\frac{\frac{\frac{}{-, A \vdash A} (Ax)}{-, A, - \vdash A, \perp} (weak R)}{-, A, -, B \vdash A, \perp} (weak L)$$

and the second and the third sequents are proved by  $(Ax)$ .

On the other hand, the fourth sequent is not provable, since we have neither of the following cases: (1) the initial group is empty, which proves the contradiction, nor (2) the initial and the first groups give the assumption  $A$ , which proves the first formula  $B$ , nor (3) the initial, the first, and the second groups give the assumptions  $A, B$ , which prove the second formula  $\perp$ .

**Example 2.4.** The third example gives an example with implication.

$$\frac{\frac{\vdots \pi_1}{-, \neg(A \vee B), - \vdash \neg A \wedge \neg B, \neg A} \quad \frac{\vdots \pi_2}{-, \neg(A \vee B), - \vdash \neg A \wedge \neg B, \neg B}}{-, \neg(A \vee B) \vdash \neg A \wedge \neg B} (\wedge R)$$



where the proof  $\pi_1$  is

$$\frac{\frac{\frac{\frac{}{\neg, \neg(A \vee B), -, A, -, - \vdash \neg A \wedge \neg B, \neg A, A \vee B, A}}{(Ax)} \quad \frac{}{\neg, \neg(A \vee B), -, A, \perp \vdash \neg A \wedge \neg B, \neg A}}{(Ax \perp)}}{\frac{}{\neg, \neg(A \vee B), -, A, - \vdash \neg A \wedge \neg B, \neg A, A \vee B}}{(\vee R1)}} \quad \frac{}{\neg, \neg(A \vee B), -, A, \perp \vdash \neg A \wedge \neg B, \neg A}}{(Ax \perp)}}{\frac{}{\neg, \neg(A \vee B), -, A \vdash \neg A \wedge \neg B, \neg A}}{(\rightarrow R1)}} \quad \frac{}{\neg, \neg(A \vee B), -, - \vdash \neg A \wedge \neg B, \neg A}}{(\rightarrow L)}$$

and the proof  $\pi_2$  is similar to  $\pi_1$ .

We will show some structural rules are admissible in this system.  $(\Gamma)_0$  is defined to be  $\Gamma$  if  $\Gamma$  does not contain  $-$ .  $(\Gamma, -, \Pi)_0$  is defined to be  $\Gamma$  if  $\Gamma$  does not contain  $-$ .

**Proposition 2.5.** *The following are admissible.*

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta} \text{ (weak L2)}$$

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}{\Gamma_1, -, \Gamma_2 \vdash \Delta_1, A, \Delta_2} \text{ (weak R2)} \quad (\#_- \Gamma_2 = |\Delta_2|, (\Gamma_2)_0 = \phi)$$

$$\frac{\Gamma \vdash \Delta}{\Pi, \Gamma \vdash \Sigma, \Delta} \text{ (weak R3)} \quad (\#_- \Pi = |\Sigma|) \quad \frac{\Gamma_1, -, A, \Gamma_2 \vdash \Delta}{\Gamma_1, A, -, \Gamma_2 \vdash \Delta} \text{ (move)}$$

$$\frac{\Gamma_1, A, \Gamma_2, A, \Gamma_3 \vdash \Delta}{\Gamma_1, A, \Gamma_2, \Gamma_3 \vdash \Delta} \text{ (cont L)} \quad \frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta} \text{ (exch L)}$$

These are proved by induction on proofs. For example, in order to prove (weak L2) is admissible, we assume a proof  $\pi$  of  $\Gamma_1, \Gamma_2 \vdash \Delta$  and construct a proof of  $\Gamma_1, A, \Gamma_2 \vdash \Delta$ . The idea is just adding  $A$  to the antecedent in each sequent in  $\pi$ . Formally we consider cases according to the last rule used in  $\pi$ .

Let the rule be (Ax) and its conclusion be  $\Gamma'_1, B, \Gamma'_2 \vdash \Delta', B$ . We have to show  $\Gamma''_1, B, \Gamma''_2 \vdash \Delta', B$  where  $\Gamma''_1, \Gamma''_2$  is obtained from  $\Gamma'_1, \Gamma'_2$  by adding  $A$ .  $\Gamma''_1, B, \Gamma''_2 \vdash \Delta', B$  is provable by (Ax).

Let the rule be ( $\vee R1$ ) that derives  $\Gamma_1, \Gamma_2 \vdash \Delta', B \vee C$  from  $\Gamma_1, \Gamma_2, - \vdash \Delta', B \vee C, B$ . We have to show  $\Gamma_1, A, \Gamma_2 \vdash \Delta', B \vee C$ . By induction hypothesis we have  $\Gamma_1, A, \Gamma_2, - \vdash \Delta', B \vee C, B$ . By ( $\vee R1$ ), we have  $\Gamma_1, A, \Gamma_2 \vdash \Delta', B \vee C$ . Other cases are proved similarly.

### 3 First-Order Sequent Calculi

This section gives definitions of LK and LJ in a familiar way.

We define the first-order classical sequent calculus LK. The language is defined to be the same as that of LK<sup>-</sup> except its sequents are of the form  $A_1, \dots, A_n \vdash B_1, \dots, B_m$  ( $n, m \geq 0$ ) for formulas  $A_1, \dots, A_n, B_1, \dots, B_m$ .  $\Gamma, \Delta, \Pi, \Sigma, \dots$  denote a sequence of formulas.

We have the following inference rules.

$$\begin{array}{c}
 \frac{}{A \vdash A} (Ax) \quad \frac{}{\top \vdash} (Ax\top) \quad \frac{}{\perp \vdash} (Ax\perp) \\
 \\
 \frac{\Gamma \vdash \Delta, A \quad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B} (\wedge R) \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L1) \quad \frac{\Gamma, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} (\wedge L2) \\
 \\
 \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} (\vee R1) \quad \frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B} (\vee R2) \quad \frac{\Gamma, A \vdash \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} (\vee L) \\
 \\
 \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B} (\rightarrow R) \quad \frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Sigma}{\Gamma, A \rightarrow B \vdash \Delta, \Sigma} (\rightarrow L) \\
 \\
 \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \forall x A} (\forall R) \quad \frac{\Gamma, A[t/x] \vdash \Delta}{\Gamma, \forall x A \vdash \Delta} (\forall L) \\
 \\
 \frac{\Gamma \vdash \Delta, A[t/x]}{\Gamma \vdash \Delta, \exists x A} (\exists R) \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, \exists x A \vdash \Delta} (\exists L) \\
 \\
 \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} (weak R) \quad \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} (weak L) \\
 \\
 \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A} (cont R) \quad \frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} (cont L) \\
 \\
 \frac{\Gamma \vdash \Delta_1, B, A, \Delta_2}{\Gamma \vdash \Delta_1, A, B, \Delta_2} (exch R) \quad \frac{\Gamma_1, B, A, \Gamma_2 \vdash \Delta}{\Gamma_1, A, B, \Gamma_2 \vdash \Delta} (exch L)
 \end{array}$$

where the conclusion does not contain free occurrences of  $x$  in the rules  $(\forall R)$  and  $(\exists L)$ .

We define the first-order intuitionistic sequent calculus LJ. The language is the same as that of LK except that its sequents are intuitionistic sequents  $A_1, \dots, A_n \vdash B$  or  $A_1, \dots, A_n \vdash$ . The inference rules are the same as those of LK except that their sequents are restricted to intuitionistic sequents.

## 4 Implication from $LK^-$ to LJ

This section proves the direction from  $LK^-$  to LJ.

For a given proof  $\pi$ , we define  $|\pi|$  as the minimum length of the succedents of the sequents in  $\pi$ . We define  $(\Gamma_0, -, \Gamma_1, -, \Gamma_2, \dots, -, \Gamma_n)^+$  as  $\Gamma_0, \Gamma_1, \Gamma_2, \dots, \Gamma_n$  if  $\Gamma_0, \dots, \Gamma_n$  do not contain any  $-$  symbol.

**Theorem 4.1.** *If we have a proof of the sequent  $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \dots, -, \Gamma_n \vdash A_1, A_2, \dots, A_n$  in  $LK^-$  where  $n \geq 0$  and  $\Gamma_0, \dots, \Gamma_n$  do not contain any  $-$  symbol, and  $i$  is the minimum length of the succedents of the sequents in the proof, then we have the following:*

- (1)  $i = 0$  and  $\Gamma_0 \vdash$  is provable in LJ, or
- (2)  $i > 0$  and  $\Gamma_0, \dots, \Gamma_i \vdash A_i$  is provable in LJ.

The idea is analyzing the uppermost sequent  $\Gamma \vdash \Delta, A$  with its succedent of length  $i$  in the proof. We sketch the proof. Suppose the sequent  $\Gamma \vdash \Delta, A$  is such a sequent. Since left rules and  $(\rightarrow R1)$  do not change the length of some succedent, right logical rules except  $(\rightarrow R1)$  decrease the length of succedents by 1, and  $(weak R)$  increases the length of the succedent by 1, the inference rule deriving  $\Gamma \vdash \Delta, A$  must be axioms or right logical rules except  $(\rightarrow R1)$ . Then we can show  $\Gamma \vdash A$  is provable in LJ. If it is an axiom,  $\Gamma \vdash A$  is provable in LJ by the corresponding axiom. If it is a right logical rule, we use induction hypothesis for its subproofs. For example, if it is

$$\frac{\frac{A, B, -, - \vdash A \wedge B, A \quad (Ax)}{A, B, -, - \vdash A \wedge B, B} \quad (Ax)}{A, B, - \vdash A \wedge B} \quad (\wedge R)$$

and  $i = 1$ , then the minimum length of succedents in each subproof is 2. By induction hypothesis for each subproof, we have  $A, B \vdash A$  and  $A, B \vdash B$  in LJ, so we have  $A, B \vdash A \wedge B$  in LJ by  $(\wedge R)$ .

Since right logical rules except  $(\rightarrow R1)$  decrease the length of succedents by 1, possible inference rules under  $\Gamma \vdash \Delta, A$  are left rules,  $(\rightarrow R1)$ , or  $(weak R)$ . Those inference rules preserve provability of the  $i$ -th formula  $A$  from the first  $i + 1$  groups of its antecedent in LJ. Hence, in the conclusion of the given proof, the sequent of the  $i$ -th formula  $A$  from the first  $i + 1$  groups of its antecedent is provable in LJ.

**Theorem 4.2.** *If  $LK^-$  proves  $\Gamma_0, -, \Gamma_1 \vdash A$ , then LJ proves  $\Gamma_0, \Gamma_1 \vdash A$ .*

*Proof.* Let  $\pi$  be the proof and  $i$  be  $\|\pi\|$ . Then  $i$  is 0 or 1. By Theorem 4.1 with  $n = 1$ , we have (1)  $i = 0$  and  $\Gamma_0 \vdash$  is provable in LJ, or (2)  $i = 1$  and  $\Gamma_0, \Gamma_1 \vdash A$  is provable in LJ. If  $i = 1$ , we have the claim. If  $i = 0$ , by weakening to  $\Gamma_0 \vdash$ , we have the claim.  $\square$

## 5 Implication from LJ to $LK^-$

This section proves the implication from LJ to  $LK^-$ .

**Proposition 5.1.** *If  $\Gamma \vdash \Delta$  is provable in LJ where  $|\Delta| = 0, 1$ , then  $-^{|\Delta|}, \Gamma \vdash \Delta$  is provable in  $LK^-$ .*

The proof idea is simulating each inference rule of LJ by inference rules of  $LK^-$ . One difference is that a logical rule in  $LK^-$  has a redundant principal formula. For example, the right conjunction rule in  $LK^-$  is

$$\frac{\Gamma, - \vdash \Delta, A \wedge B, A \quad \Gamma, - \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B} \quad (\wedge R)$$

and on the other hand the right conjunction rule in LJ is

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \quad (\wedge R)$$

This difference is covered by putting  $A \wedge B$  by  $(weak R2)$  in Proposition 2.5. The other difference is the existence of  $-$ , which is handled by moving  $-$  by  $(move)$  in Proposition 2.5.

**Theorem 5.2 (Equivalence between  $LK^-$  and LJ).**  $\neg, \Gamma \vdash A$  is provable in  $LK^-$  if and only if  $\Gamma \vdash A$  is provable in LJ.

*Proof.* The implication from the left-hand side to the right-hand side is proved by Theorem 4.2. The implication from the right-hand side to the left-hand side is proved by Proposition 5.1.  $\square$

## 6 Non-Commutative Sequent Calculus NCLK

This section discusses NCLK and shows it becomes equivalent to LK when we add the exchange rule to it.

We define Non-Commutative First-Order Sequent Calculus NCLK.

**Definition 6.1 (NCLK).** Its language is the same as that of LK. Note that its sequents are of the form  $A_1, \dots, A_n \vdash B_1, \dots, B_m$  ( $n, m \geq 0$ ) where  $A_i, B_i$  are formulas.  $\Gamma, \Delta, \Pi, \Sigma \dots$  denote a sequence of formulas.

The inference rules are given as follows.

$$\begin{array}{c}
 \frac{}{\Gamma_1, A, \Gamma_2 \vdash \Delta, A} (Ax) \quad \frac{}{\Gamma \vdash \Delta, \top} (Ax\top) \quad \frac{}{\Gamma_1, \perp, \Gamma_2 \vdash \Delta} (Ax\perp) \\
 \frac{\Gamma, \top \vdash \Delta, A \wedge B, A \quad \Gamma, \top \vdash \Delta, A \wedge B, B}{\Gamma \vdash \Delta, A \wedge B} (\wedge R) \\
 \frac{\Gamma_1, A \wedge B, \Gamma_2, A \vdash \Delta, D, D}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta, D} (\wedge L1) \quad \frac{\Gamma_1, A \wedge B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \wedge B, \Gamma_2 \vdash \Delta, D} (\wedge L2) \\
 \frac{\Gamma, \top \vdash \Delta, A \vee B, A}{\Gamma \vdash \Delta, A \vee B} (\vee R1) \quad \frac{\Gamma, \top \vdash \Delta, A \vee B, B}{\Gamma \vdash \Delta, A \vee B} (\vee R2) \\
 \frac{\Gamma_1, A \vee B, \Gamma_2, A \vdash \Delta, D, D \quad \Gamma_1, A \vee B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \vee B, \Gamma_2 \vdash \Delta, D} (\vee L) \\
 \frac{\Gamma, A \vdash \Delta, A \rightarrow B, A \rightarrow B}{\Gamma \vdash \Delta, A \rightarrow B} (\rightarrow R1) \quad \frac{\Gamma, \top \vdash \Delta, A \rightarrow B, B}{\Gamma \vdash \Delta, A \rightarrow B} (\rightarrow R2) \\
 \frac{\Gamma_1, A \rightarrow B, \Gamma_2, \top \vdash \Delta, D, A \quad \Gamma_1, A \rightarrow B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \Delta, D} (\rightarrow L) \\
 \frac{\Gamma, \top \vdash \Delta, \forall x A, A}{\Gamma \vdash \Delta, \forall x A} (\forall R) \quad \frac{\Gamma_1, \forall x A, \Gamma_2, A[t/x] \vdash \Delta, D, D}{\Gamma_1, \forall x A, \Gamma_2 \vdash \Delta, D} (\forall L) \\
 \frac{\Gamma, \top \vdash \Delta, \exists x A, A[t/x]}{\Gamma \vdash \Delta, \exists x A} (\exists R) \quad \frac{\Gamma_1, \exists x A, \Gamma_2, A \vdash \Delta, D, D}{\Gamma_1, \exists x A, \Gamma_2 \vdash \Delta, D} (\exists L) \\
 \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta, B} (sweak) \quad \frac{\top, \Gamma \vdash \Delta}{\Gamma \vdash \Delta} (\top E) \quad \frac{\Gamma \vdash \perp, \Delta}{\Gamma \vdash \Delta} (\perp E)
 \end{array}$$

where the conclusion does not contain free occurrences of  $x$  in the rules  $(\forall R)$  and  $(\exists L)$ .

*(sweak)* means symmetric weakening.

Intuitive meaning of provable sequents is given as follows: If  $\Pi, A_1, \dots, A_n \vdash B_1, \dots, B_n$  is provable, then (1)  $\Pi \vdash$  is true, or (2)  $\Pi, A_1, \dots, A_i \vdash B_i$  is true for some  $i$ . If  $A_1, \dots, A_n \vdash C_1, \dots, C_m, B_1, \dots, B_n$  is provable, then (1)  $\vdash C_i$  is true for some  $i$ , or (2)  $A_1, \dots, A_i \vdash B_i$  is true for some  $i$ .

This system is obtained from  $LK^-$  by coding grouping information by the length of a sequence of formulas. We explain it by example.

**Example 6.2.** The sequent

$$A_1, -, A_2, A_3, -, A_4, -, A_5, A_6 \vdash B_1, B_2, B_3$$

in  $LK^-$  is coded by the sequent

$$A_1, \top, A_2, A_3, \top, A_4, \top, A_5, A_6 \vdash B_1, B_1, B_1, B_2, B_2, B_3, B_3, B_3$$

in NCLK. The atomic formula  $\top$  is used for separating groups. The group  $\top, A_5, A_6$  corresponds to  $B_3, B_3, B_3$ . The group  $\top, A_4$  corresponds to  $B_2, B_2$ . The group  $\top, A_2, A_3$  corresponds to  $B_1, B_1, B_1$ . We can decode this information by counting formulas from the right to the left on both sides. This translation is formally defined in Definition 7.5

We explain this system by the same examples as those in Section 2.

**Example 6.3.** The first example shows its conjunction is commutative.

$$\frac{\frac{A \wedge B, \top, B \vdash B \wedge A, B, B \quad (Ax)}{A \wedge B, \top \vdash B \wedge A, B} \quad (\wedge L2) \quad \frac{A \wedge B, \top, A \vdash B \wedge A, A, A \quad (Ax)}{A \wedge B, \top \vdash B \wedge A, A} \quad (\wedge L1)}{A \wedge B \vdash B \wedge A} \quad (\wedge R)$$

**Example 6.4.** The next example shows how this system respects the order of formulas. We have three provable sequents

$$\begin{aligned} A, B \vdash A, \perp, \\ A, B \vdash \perp, A, \\ A, B \vdash \perp, B. \end{aligned}$$

On the other hand the sequent

$$A, B \vdash B, \perp$$

is not provable. The first sequent is provable since  $A \vdash A$  is true. The second sequent is provable since  $A, B \vdash A$  is true. The third sequent is provable since  $A, B \vdash B$  is true. Formally the first sequent is proved by

$$\frac{\overline{A \vdash A} \quad (Ax)}{A, B \vdash A, \perp} \quad (sweak)$$

and the second and the third sequents are proved by  $(Ax)$ . On the other hand, the fourth sequent is not provable, since  $A \vdash B$  is not true and  $A, B \vdash \perp$  is not true.

**Example 6.5.** The third example gives an example with implication.

$$\frac{\frac{\vdots \pi_1}{\neg(A \vee B), \top \vdash \neg A \wedge \neg B, \neg A} \quad \frac{\vdots \pi_2}{\neg(A \vee B), \top \vdash \neg A \wedge \neg B, \neg B}}{\neg(A \vee B) \vdash \neg A \wedge \neg B} (\wedge R)$$

where the proof  $\pi_1$  is

$$\frac{\frac{\frac{\neg(A \vee B), \top, A, \top, \top \vdash \neg A \wedge \neg B, \neg A, \neg A, A \vee B, A}{\neg(A \vee B), \top, A, \top \vdash \neg A \wedge \neg B, \neg A, \neg A, A \vee B} (Ax)}{\neg(A \vee B), \top, A, \top \vdash \neg A \wedge \neg B, \neg A, \neg A} (\vee R1)}{\frac{\neg(A \vee B), \top, A \vdash \neg A \wedge \neg B, \neg A, \neg A}{\neg(A \vee B), \top \vdash \neg A \wedge \neg B, \neg A} (\rightarrow R1)} (Ax \perp) \quad (\rightarrow L)$$

and the proof  $\pi_2$  is similar to  $\pi_1$ .

**Remark.** (1) Every rule except  $(\top E)$  and  $(\perp E)$  preserves  $|F| - |\Delta|$ .

(2)  $(\perp E)$  is necessary for making a binary left logical rule for the empty succedent admissible. It is used in the proof of Theorem 7.6. For example, the following is admissible.

$$\frac{\Gamma_1, A \vee B, \Gamma_2, A \vdash \quad \Gamma_1, A \vee B, \Gamma_2, B \vdash}{\Gamma_1, A \vee B, \Gamma_2 \vdash} (\vee L)$$

(3)  $(\top E)$  is necessary since  $\vdash \top \vee \perp, \perp$  would not be provable otherwise, though it is indeed provable by

$$\frac{\frac{\frac{\frac{\top \vdash \top \vee \perp, \top}{\vdash \top \vee \perp} (Ax \top)}{\top \vdash \top \vee \perp} (\vee R1)}{\top \vdash \top \vee \perp, \perp} (sweak)}{\vdash \top \vee \perp, \perp} (\top E)$$

**Proposition 6.6.** (1) The following are admissible.

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2} (sweak2) \quad (|\Gamma_2| = |\Delta_2|)$$

$$\frac{\Gamma_1, A, A, \Gamma_2 \vdash \Delta_1, B, B, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2} (scont) \quad (|\Gamma_2| = |\Delta_2|)$$

$$\frac{\Gamma_1, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta} (weak L) \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \perp, \Delta} (\perp I) \quad \frac{\Gamma_1, \top, \Gamma_2 \vdash \Delta}{\Gamma_1, A, \Gamma_2 \vdash \Delta} (replace L)$$

(2) The following is admissible.

$$\frac{\Gamma \vdash \Delta_1, A, A, \Delta_2}{\Gamma \vdash \Delta_1, A, \Delta_2} (cont R)$$

(3) The following is admissible.

$$\frac{\Gamma_1, \top, A, \Gamma_2 \vdash \Delta_1, B, B, \Delta_2}{\Gamma_1, A, \Gamma_2 \vdash \Delta_1, B, \Delta_2} (\top E2) \quad (|\Gamma_2| = |\Delta_2|)$$

The claims in (1) are proved by induction on the proof. The claim (2) is proved by induction on the proof by using (*weak L*) in (1). The claim (3) is proved by (*scont*) and (*replace L*) in (1).

We define the system NCLK+EX as NCLK with (*exch L*) and (*exch R*).

$$\frac{\Gamma \vdash \Delta_1, A, B, \Delta_2}{\Gamma \vdash \Delta_1, B, A, \Delta_2} \text{ (exch R)} \quad \frac{\Gamma_1, A, B, \Gamma_2 \vdash \Delta}{\Gamma_1, B, A, \Gamma_2 \vdash \Delta} \text{ (exch L)}$$

**Proposition 6.7.** *The following are admissible in NCLK+EX.*

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta} \text{ (cont L)} \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} \text{ (weak R)}$$

*Proof.* (*cont L*) is proved by

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A, A \vdash \perp, \perp, \Delta} \text{ (}\perp\text{I)twice}$$

$$\frac{\Gamma, A, A \vdash \perp, \perp, \Delta}{\Gamma, A, A \vdash \Delta, \perp, \perp} \text{ (exch R)several times}$$

$$\frac{\Gamma, A \vdash \Delta, \perp}{\Gamma, A \vdash \perp, \Delta} \text{ (scont)}$$

$$\frac{\Gamma, A \vdash \perp, \Delta}{\Gamma, A \vdash \perp, \Delta} \text{ (exch R)several times}$$

$$\frac{\Gamma, A \vdash \perp, \Delta}{\Gamma, A \vdash \Delta} \text{ (}\perp\text{E)}$$

(*weak R*) is proved by

$$\frac{\Gamma \vdash \Delta}{\Gamma, \top \vdash \Delta, A} \text{ (sweak)}$$

$$\frac{\Gamma, \top \vdash \Delta, A}{\top, \Gamma \vdash \Delta, A} \text{ (exch L)several times}$$

$$\frac{\top, \Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A} \text{ (}\top\text{E)}$$

□

We will write  $\Gamma \vdash_T \Delta$  to denote that the sequent  $\Gamma \vdash \Delta$  is provable in the system  $T$ .

**Theorem 6.8 (Equivalence between NCLK+EX and LK).**

$\Gamma \vdash_{NCLK+EX} \Delta$  if and only if  $\Gamma \vdash_{LK} \Delta$ .

*Proof.* From the right-hand side to the left-hand side. The claim is proved by induction on the proof.

If the last rule is a logical rule, then add some formulas by weakening in Propositions 6.6 and 6.7 and use the corresponding logical rule. We give some interesting cases.

Case ( $\rightarrow R$ ). We suppose

$$\frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B} \text{ (}\rightarrow\text{R)}$$

Then we have

$$\frac{\vdots IH}{\Gamma, A \vdash \Delta, B} \text{ (weak L)(weak R)(exch R)}$$

$$\frac{\Gamma, A, \top \vdash \Delta, A \rightarrow B, A \rightarrow B, B}{\Gamma, A \vdash \Delta, A \rightarrow B, A \rightarrow B} \text{ (}\rightarrow\text{R2)}$$

$$\frac{\Gamma, A \vdash \Delta, A \rightarrow B, A \rightarrow B}{\Gamma \vdash \Delta, A \rightarrow B} \text{ (}\rightarrow\text{R1)}$$

Case ( $\rightarrow L$ ). We suppose

$$\frac{\Gamma \vdash \Delta, A \quad \Gamma, B \vdash \Sigma}{\Gamma, A \rightarrow B \vdash \Delta, \Sigma} (\rightarrow L)$$

Then we have

$$\frac{\frac{\frac{\vdots IH}{\Gamma \vdash \Delta, A}}{\Gamma, A \rightarrow B, \top \vdash \Delta, \Sigma, \perp, A} (W) \quad \frac{\frac{\vdots IH}{\Gamma, B \vdash \Sigma}}{\Gamma, A \rightarrow B, B \vdash \Delta, \Sigma, \perp, \perp} (W)}{\frac{\frac{\Gamma, A \rightarrow B \vdash \Delta, \Sigma, \perp}{\Gamma, A \rightarrow B \vdash \perp, \Delta, \Sigma} (exch R) \text{several times}}{\Gamma, A \rightarrow B \vdash \Delta, \Sigma} (\perp E)} (\rightarrow L)$$

where ( $W$ ) denotes several steps of (*weak L*), (*weak R*), and (*exch R*).

If the last rule is a structural rule, then it is covered by Propositions [6.6](#) and [6.7](#)

From the left-hand side to the right-hand side. It is proved by induction on a proof since every rule is sound in LK.  $\square$

## 7 Translations between LK<sup>-</sup> and NCLK

This section shows the equivalence between LK<sup>-</sup> and NCLK by giving translations which preserve provability.

First we prepare several admissible rules in LK<sup>-</sup> for the equivalence proof.

**Proposition 7.1.** *The following are admissible in LK<sup>-</sup>.*

$$\frac{\Gamma_1, \top, \Gamma_2 \vdash \Delta}{\Gamma_1, \Gamma_2 \vdash \Delta} (\top E) \quad \frac{\Gamma_1, -, \Gamma_2 \vdash \Delta_1, A, A, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A, \Delta_2} (cont R) \quad (\#_- \Gamma_2 = |\Delta_2|)$$

$$\frac{\Gamma_1, -, \Gamma_2 \vdash \Delta_1, \perp, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} (\perp E) \quad (\#_- \Gamma_2 = |\Delta_2|)$$

They are proved by induction on the proof.

We give a translation from NCLK to LK<sup>-</sup>. To translate  $\Gamma \vdash \Delta$ , we insert the same number of  $-$  symbols as  $|\Delta|$  into  $\Gamma$  in front of each formula from the rightmost formula of  $\Gamma$ . For example, the sequent  $A_1, A_2, A_3, A_4 \vdash B_1, B_2$  in NCLK is translated into the sequent  $A_1, A_2, -, A_3, -, A_4 \vdash B_1, B_2$  in LK<sup>-</sup>.

**Definition 7.2 (Translation from NCLK to LK<sup>-</sup>).**  $\Gamma \vdash \Delta$  is mapped to  $\Gamma^{-|\Delta|} \vdash \Delta$ , where  $(\Gamma_0, A_1, A_2, \dots, A_n)^{-n}$  is defined as  $\Gamma_0, -, A_1, -, A_2, \dots, -, A_n$  and  $(A_1, A_2, \dots, A_m)^{-n}$  ( $m < n$ ) is defined as  $^{-n-m}, -, A_1, -, A_2, \dots, -, A_m$ .

**Theorem 7.3.**  $\Gamma \vdash_{NCLK} \Delta$  implies  $\Gamma^{-|\Delta|} \vdash_{LK^-} \Delta$ .



*Proof.* By induction on the proof. Cases are considered according to the last rule.

If the last rule is  $(Ax)$ ,  $(Ax\top)$ , and  $(Ax\perp)$ , it is proved by  $(Ax)$ ,  $(Ax\top)$ , and  $(Ax\perp)$  respectively.

If the last rule is a right logical rule except  $(\rightarrow R1)$ , we first use  $(\top E)$  in Proposition 7.1, and then use the corresponding logical rule.

If the last rule is a left logical rule except  $(\rightarrow L)$  or  $(\rightarrow R1)$ , we first use  $(cont R)$  in Proposition 7.1, and then use the corresponding logical rule.

If the last rule is  $(\rightarrow L)$ , we suppose

$$\frac{\Gamma_1, A \rightarrow B, \Gamma_2, \top \vdash \Delta, D, A \quad \Gamma_1, A \rightarrow B, \Gamma_2, B \vdash \Delta, D, D}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \Delta, D} (\rightarrow L)$$

Let  $n$  be  $|\Delta, D|$ . We have

$$\frac{\frac{\vdots IH}{(\Gamma_1, A \rightarrow B, \Gamma_2)^{-n}, -, \top \vdash \Delta, D, A} (\top E) \quad \frac{\vdots IH}{(\Gamma_1, A \rightarrow B, \Gamma_2)^{-n}, -, B \vdash \Delta, D, D} (cont R)}{\frac{(\Gamma_1, A \rightarrow B, \Gamma_2)^{-n}, - \vdash \Delta, D, A}{(\Gamma_1, A \rightarrow B, \Gamma_2)^{-n}, B \vdash \Delta, D} (\rightarrow L)}{(\Gamma_1, A \rightarrow B, \Gamma_2)^{-n} \vdash \Delta, D}$$

If the last rule is  $(sweak)$ , we have

$$\frac{\frac{\Gamma \vdash \Delta}{\Gamma, - \vdash \Delta, B} (weak R)}{\Gamma, -, A \vdash \Delta, B} (weak L)$$

If the last rule is  $(\top E)$  and  $(\perp E)$ , the claim is proved by  $(\top E)$  and  $(\perp E)$  in Proposition 7.1 respectively.  $\square$

**Example 7.4.** The NCLK-proof of  $A \wedge B \vdash B \wedge A$  in Example 6.3 is translated into the  $LK^-$ -proof

$$\frac{\frac{\frac{\frac{\frac{-, A \wedge B, -, \top, -, B \vdash B \wedge A, B, B}{- , A \wedge B, -, \top, B \vdash B \wedge A, B} (Ax)}{-, A \wedge B, -, \top, B \vdash B \wedge A, B} (cont R)}{-, A \wedge B, -, \top \vdash B \wedge A, B} (\wedge L2)}{-, A \wedge B, -, \top \vdash B \wedge A, B} (\top E)}{-, A \wedge B, - \vdash B \wedge A, B} (\wedge R) \quad \frac{\frac{\frac{\frac{\frac{-, A \wedge B, -, \top, -, A \vdash B \wedge A, A, A}{- , A \wedge B, -, \top, A \vdash B \wedge A, A} (Ax)}{-, A \wedge B, -, \top, A \vdash B \wedge A, A} (cont R)}{-, A \wedge B, -, \top \vdash B \wedge A, A} (\wedge L1)}{-, A \wedge B, -, \top \vdash B \wedge A, A} (\top E)}{-, A \wedge B, - \vdash B \wedge A, A} (\wedge R)}$$

Next, we define a translation from  $LK^-$  to NCLK. To translate  $\Gamma \vdash \Delta$ , we replace  $-$  by  $\top$  in  $\Gamma$ , and the succedent is produced from  $\Delta$  by multiplying the  $i$ -th formula by  $n_i + 1$  when the  $i$ -th group in  $\Gamma$  has  $n_i$  formulas. An example is given in Example 6.2.  $A^n$  denotes  $A, \dots, A$  ( $n$  times).

**Definition 7.5 (Translation from  $LK^-$  to NCLK).**  $\Gamma \vdash \Delta$  is mapped to  $\Gamma^\top \vdash \Delta^\Gamma$ , where  $\Gamma^\top$  is defined as  $\Gamma_0, \top, \Gamma_1, \top, \Gamma_2, \dots, \top, \Gamma_n$  and  $(A_1, \dots, A_n)^\Gamma$  is defined as  $A_1^{|\Gamma_1|+1}, A_2^{|\Gamma_2|+1}, \dots, A_n^{|\Gamma_n|+1}$  if  $\Gamma$  is  $\Gamma_0, -, \Gamma_1, -, \Gamma_2, \dots, -, \Gamma_n$  and  $\Gamma_i$  does not contain  $-$ .

**Theorem 7.6.**  $\Gamma \vdash_{LK^-} \Delta$  implies  $\Gamma^\top \vdash_{NCLK} \Delta^\Gamma$ .

*Proof.* By induction on the proof.

Case  $(\rightarrow L)$ . We suppose

$$\frac{\Gamma_1, A \rightarrow B, \Gamma_2, - \vdash \Delta, A \quad \Gamma_1, A \rightarrow B, \Gamma_2, B \vdash \Delta}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \Delta} (\rightarrow L)$$

Case 1.  $\Delta$  is not empty. Let  $D$  be the last formula of  $\Delta$  and  $\Gamma$  be  $\Gamma_1, A \rightarrow B, \Gamma_2$ . We use  $(\rightarrow L)$  with induction hypothesis in the following way.

$$\frac{\begin{array}{c} \vdots \\ IH \end{array} \quad \begin{array}{c} \vdots \\ IH \end{array} \quad \frac{\Gamma_1^\top, A \rightarrow B, \Gamma_2^\top, \top \vdash \Delta^F, A \quad \Gamma_1^\top, A \rightarrow B, \Gamma_2^\top, B \vdash \Delta^F, D}{\Gamma_1^\top, A \rightarrow B, \Gamma_2^\top \vdash \Delta^F}}{\Gamma_1^\top, A \rightarrow B, \Gamma_2^\top \vdash \Delta^F} (\rightarrow L)$$

Case 2.  $\Delta$  is empty.

$$\frac{\begin{array}{c} \vdots \\ IH \end{array} \quad \frac{\Gamma_1, A \rightarrow B, \Gamma_2, \top \vdash A}{\Gamma_1, A \rightarrow B, \Gamma_2, \top \vdash \perp, A} (\perp I) \quad \frac{\begin{array}{c} \vdots \\ IH \end{array} \quad \frac{\Gamma_1, A \rightarrow B, \Gamma_2, B \vdash}{\Gamma_1, A \rightarrow B, \Gamma_2, B \vdash \perp, \perp} (\perp I) \text{twice}}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \perp} (\rightarrow L)}{\frac{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash \perp}{\Gamma_1, A \rightarrow B, \Gamma_2 \vdash} (\perp E)}$$

Case  $(\wedge L1)$  and  $\Delta = \phi$ .

$$\frac{\frac{\Gamma_1, A \wedge B, \Gamma_2, A \vdash}{\Gamma_1, A \wedge B, \Gamma_2, A \vdash \perp, \perp} (\perp I) \text{twice}}{\frac{\Gamma_1, A \wedge B, \Gamma_2 \vdash \perp}{\Gamma_1, A \wedge B, \Gamma_2 \vdash} (\perp E)} (\wedge L1)$$

Cases of other left logical rules with the empty succedent are similarly proved.

Case  $(weak R)$ .

$$\frac{\Gamma^\top \vdash \Delta^F}{\Gamma^\top, \top \vdash \Delta^F, A} (sweak)$$

Case  $(weak L)$ . If  $\Delta$  is not empty, this is proved by  $(sweak)$ . If  $\Delta$  is empty, this is proved by  $(weak L)$  in Proposition 6.6.  $\square$

**Example 7.7.** The  $LK^-$ -proof of  $-, A \wedge B \vdash B \wedge A$  in Example 2.2 is translated into the NCLK-proof

$$\frac{\frac{\frac{\top, A \wedge B, \top, B \vdash B \wedge A, B \wedge A, B, B}{\top, A \wedge B, \top \vdash B \wedge A, B \wedge A, B} (\wedge L2)}{\top, A \wedge B \vdash B \wedge A, B \wedge A} (Ax)}{\top, A \wedge B \vdash B \wedge A, B \wedge A} (\wedge L1) \quad \frac{\frac{\frac{\top, A \wedge B, \top, A \vdash B \wedge A, B \wedge A, A, A}{\top, A \wedge B, \top \vdash B \wedge A, B \wedge A, A} (\wedge L1)}{\top, A \wedge B \vdash B \wedge A, B \wedge A} (Ax)}{\top, A \wedge B \vdash B \wedge A, B \wedge A} (\wedge R)$$

Actually these two translations are the inverses of each other with respect to provability. Let the sequent  $\Gamma \vdash \Delta$  in NCLK be translated into the sequent  $(\Gamma \vdash \Delta)^1$  in  $LK^-$  and the sequent  $\Pi \vdash \Sigma$  in  $LK^-$  be translated into the sequent  $(\Pi \vdash \Sigma)^2$  in NCLK. By  $(\top E2)$  in Proposition 6.6 (3) and  $(\top E)$ , we can show that if  $((\Gamma \vdash \Delta)^1)^2$  is provable in NCLK, then  $\Gamma \vdash \Delta$  is provable in NCLK. By  $(cont R)$  and  $(\top E)$  in Proposition 7.1, we can also show that if  $((\Pi \vdash \Sigma)^2)^1$  is provable in  $LK^-$ , then  $\Pi \vdash \Sigma$  is provable in  $LK^-$ .

Finally we show the equivalence.

**Theorem 7.8 (Equivalence between NCLK and LJ).**  $\Gamma \vdash_{NCLK} A$  if and only if  $\Gamma \vdash_{LJ} A$ .

*Proof.* From the left-hand side to the right-hand side.

By Theorem 7.3,  $\Gamma^{-1} \vdash_{LK^-} A$ . By Theorem 4.2,  $\Gamma \vdash_{LJ} A$ .

From the right-hand side to the left-hand side.

By Proposition 5.1,  $\neg, \Gamma \vdash_{LK^-} A$ . By Theorem 7.6,  $\top, \Gamma \vdash_{NCLK} A^{|\Gamma|+1}$ . By  $(\top E)$  and  $(cont R)$  in Proposition 6.6 (2), we have  $\Gamma \vdash_{NCLK} A$ .  $\square$

## 8 Concluding Remarks

We gave the non-commutative first-order sequent calculus NCLK and showed that it is equivalent to the first-order intuitionistic sequent calculus LJ. We also showed that it becomes equivalent to the first-order classical sequent calculus LK when we add the exchange rule to the system. In order to do that, we extended the non-commutative positive fragment to the system  $LK^-$  having antecedent-grouping and no right exchange rule. We showed the equivalence between  $LK^-$  and LJ. We also gave translations between  $LK^-$  and NCLK.

The cut elimination theorem holds in  $LK^-$ . It is proved by extending Theorem 4.1 and using the cut elimination theorem for LJ. The cut elimination theorem is also proved to hold for NCLK by using that for  $LK^-$  and the translations.

The systems NCLK and  $LK^-$  give a starting point for research on logical systems based on non-commutative sequents. They will clarify those systems and enable us to extend them.

## Acknowledgments

We would like to thank Prof. Stefano Berardi and Prof. Kazushige Terui for discussions and comments. We would also like to thank the anonymous referees for valuable comments.

## References

1. Berardi, S., Yamagata, Y.: A sequent calculus for Limit Computable Mathematics. *Annals of Pure and Applied Logic* 153(1-3), 111–126 (2008)
2. Berardi, S., Tatsuta, M.: Positive arithmetic without exchange is a subclassical logic. In: Shao, Z. (ed.) *APLAS 2007*. LNCS, vol. 4807, pp. 271–285. Springer, Heidelberg (2007)
3. Girard, J.Y.: Linear logic. *Theoretical Computer Science* 50(1), 1–102 (1987)
4. Maehara, S.: Eine Darstellung der intuitionistischen Logik in der Klassischen. *Nagoya Mathematical Journal* 7, 45–64 (1954)
5. Nakano, H.: A Constructive Formalization of the Catch and Throw Mechanism. In: *Proceedings of Seventh Annual IEEE Symposium on Logic in Computer Science*, pp. 82–89 (1992)

# Model Checking $\text{FO}(\text{R})$ over One-Counter Processes and beyond

Anthony Widjaja To

LFCS, School of Informatics, University of Edinburgh  
anthony.w.to@ed.ac.uk

**Abstract.** One-counter processes are pushdown processes over a singleton stack alphabet (plus a stack-bottom symbol). We study the problems of model checking asynchronous products of one-counter processes against 1) first-order logic  $\text{FO}(\text{R})$  with reachability predicate, 2) the finite variable fragments  $\text{FO}^k(\text{R})$  ( $k \geq 2$ ) of  $\text{FO}(\text{R})$ , 3) EF-logic which is a fragment of  $\text{FO}^2(\text{R})$ , and 4) all these logics extended with simple component-wise synchronizing predicates. We give a rather complete picture of their combined, expression, and data complexity. To this end, we show that these problems are poly-time reducible to two syntactic restrictions of Presburger Arithmetic, which are equi-expressive with first-order modulo counting theory of  $(\mathbb{N}, <)$ , for which we give optimal quantifier elimination procedures. In particular, these problems are all shown to be in PSPACE, which is in sharp contrast to the closely related problem of model checking  $\text{FO}(\text{R})$  over pushdown processes (with one stack) which has nonelementary complexity. Finally, we apply our proof method to give a fixed automatic (and so rational) graph whose modal logic theory has nonelementary complexity, solving a recently posed open question.

## 1 Introduction

Pushdown automata (PDA) are a natural model for sequential programs with recursive calls and their model checking problems have been studied extensively. It is well-known that, over PDA, the problems of model checking first-order logic with reachability predicate  $\text{FO}(\text{R})$  and monadic second-order logic MSO are decidable in nonelementary recursive time [20]. In fact, PDA (even with one control state) can easily generate natural numbers  $(\mathbb{N}, +1)$  with a successor relation (a.k.a. S1S) and the infinite binary tree with two successor relations (a.k.a. S2S). Since the  $\text{FO}(\text{R})$  theory of S2S and the MSO theory of S1S have nonelementary complexity [4,24], the same lower bound can be deduced for PDA. In contrast, when considering modal and temporal logics — such as EF-logic, LTL, CTL, and  $\mu$ -calculus — the complexity of model checking PDA is at most EXPTIME [3,5,27,28].

One-counter processes (OCPs) are PDA over a singleton stack alphabet (plus a non-removable stack-bottom symbol). The problems of model checking basic modal logic, EF-logic, and  $\mu$ -calculus over one-counter processes have been studied, and their complexity are lower than the corresponding problems for general

PDA [8,10,11,23]. On the other hand, since OCPs can easily generate S1S, it follows from [24] that model checking MSO over OCPs is nonelementary. The complexity of model checking FO(R) over OCPs is, however, unknown. Unlike the case of general PDA, it is easy to show that OCPs cannot generate a graph isomorphic to S2S. Furthermore, the FO theory of S1S with linear order is only PSPACE-complete [7,24].

PDA are well-known to be incapable of modeling concurrent programs. One common way to obtain concurrent behavior from pushdown processes is to consider (*finite*) *products* of graphs generated by PDA. An *asynchronous product* of  $k$  PDA  $\mathcal{P}_1, \dots, \mathcal{P}_k$  can be construed as a concurrent system with processes  $\mathcal{P}_1, \dots, \mathcal{P}_k$ , each behaving independently (i.e. the processes do not interact). Therefore, reachability for an asynchronous product of PDA can trivially be reduced to the reachability problems for each of its components. Note that taking synchronized products — the most general notion of products — of PDA (resp. OCPs) easily yield a model that is as powerful as Turing machines (resp. Minsky counter machines). There are several reasons for studying model checking problems over simple models of concurrent programs such as asynchronous products of PDA and OCPs. First, Wöhrle and Thomas [29] have recently shown that, when combined with logics such as FO(R) and EF-logic, asynchronous products are powerful enough for modeling *a finite amount of synchronization* (synchronization can be embedded in the formulas). Second, asynchronous products of PDA and OCPs are some of the most basic nontrivial concurrent models that are subsumed by more complex models such as ground tree rewrite systems [16], PAD [18], automatic graphs [2], and rational graphs [19]. Some open problems in the more general settings (such as the complexity of model checking EF-logic [18,26]) seem difficult already in the restricted settings.

In this paper, we consider model checking problems of OCPs, as well as asynchronous products of OCPs (ΠOCPs), with respect to specifications in 1) FO(R), 2) the  $k$ -variable fragments  $\text{FO}^k(\text{R})$  ( $k \geq 2$ ) of FO(R), and 3) EF-logic which is a fragment of  $\text{FO}^2(\text{R})$ . We also study these logics extended with simple component-wise synchronizing unary predicates testing whether component  $i$  and  $j$  are the same elements, which we denote by  $\text{FO}_S(\text{R})$ ,  $\text{FO}_S^k(\text{R})$ , and  $\text{EF}_S$ -logic. We give a rather complete picture of their *combined complexity* (i.e. inputs consist of systems and specifications), *expression complexity* (i.e. inputs consist only of specifications with a fixed system), and *data complexity* (i.e. inputs consist only of systems with a fixed specification).

Our results are summarized in Table 1 and Table 2 together with the recent result from [8]. In particular, all our results are within PSPACE, in contrast to

Table 1. Results for OCPs

	FO(R) FO <sup>4</sup> (R)	FO <sup>2</sup> (R)	EF-logic
Combined	PSPACE	PSPACE	in P <sup>NP</sup> & P <sup>NP[log]</sup> -hard [8]
Expression	PSPACE	in P	in P [8]
Data	PH	PH	in P <sup>NP</sup> & P <sup>NP[log]</sup> -hard [8]

**Table 2.** Results for  $\Pi$ OCPs

	FO(R) FO <sup>4</sup> (R) FO <sub>S</sub> (R)	FO <sup>2</sup> (R)	EF-logic	EF <sub>S</sub> -logic
Combined	PSPACE	PSPACE	PSPACE	PSPACE
Expression	PSPACE	in P	in P	PSPACE
Data	PH	PH	in P <sup>NP</sup> & P <sup>NP[log]</sup> -hard	PH

PDA whose expression complexity for FO(R) is nonelementary [4]. Our upper bounds are shown by first introducing two syntactic restrictions  $\mathcal{L}$  and  $\mathcal{L}'$  of Presburger Arithmetic, for which we give optimal quantifier elimination procedures, and showing that the  $\Pi$ OCPs model checking problems are poly-time reducible to either  $\mathcal{L}$  or  $\mathcal{L}'$ . Note that, to obtain a sharp upper bound, we cannot consider only OCPs (without products) and apply Feferman-Vaught type of composition methods (e.g. see [17,22,29]) as the resulting algorithm will run in time that is nonelementary in the formula size. Concerning our lower bound results, in contrast to the result in [8] that model checking EF-logic is in P<sup>NP</sup>, data complexity of FO<sup>2</sup>(R) over OCPs is already hard for every level of PH. On the other hand, the expression complexity of FO<sup>2</sup>(R) over  $\Pi$ OCPs is in P. This generalizes one of the key results in [8] that the expression complexity of EF-logic over OCPs (without products) is in P. However, for each  $k > 3$ , we can show that the expression complexity of FO<sup>k</sup>(R) is PSPACE-complete already for OCPs. Also, notice that the combined complexity of EF-logic becomes PSPACE, which holds already for products of two OCPs. Finally, notice that adding simple synchronization relations to EF-logic causes the expression and data complexity to increase significantly. In fact, we shall use its proof method to give a fixed automatic (and so rational) graph whose modal logic theory has nonelementary complexity, answering a recently posed question in [1,26].

The paper is organized as follows. We fix some necessary notations and definitions in Section 2. In Section 3 we define two syntactic restrictions  $\mathcal{L}$  and  $\mathcal{L}'$  of Presburger Arithmetic and prove that model checking problems for OCPs and  $\Pi$ OCPs are poly-time reducible checking formulas in these restricted logics. In Section 4 we give optimal quantifier elimination procedures for  $\mathcal{L}$  and  $\mathcal{L}'$  and deduce optimal upper bounds for all problems in Table 1 and Table 2. We prove our lower bounds results for OCPs and  $\Pi$ OCPs in Section 5. In Section 6 we give a fixed automatic graph whose modal logic has nonelementary complexity. Finally, we conclude in Section 7 with future work. Due to space constraints, most proofs have been relegated to the full version.

## 2 Preliminary

**General Notations.** Let  $\mathbb{Z}$  denote the set of integers. Let  $\mathbb{N} = \mathbb{Z}_{\geq 0}$ . For  $i, j \in \mathbb{N}$ , we use  $[i, j]$  to denote  $\{i, i + 1, \dots, j\}$ . As usual, the notations  $(i, j)$ ,  $(i, j]$ , and  $[i, j)$  denote the subsets of  $[i, j]$  with the appropriate endpoints omitted. We

use  $a + b\mathbb{N}$  to mean the arithmetic progression  $\{a + bk : k \in \mathbb{N}\}$ . Given  $n$  sets  $S_1, \dots, S_n$ , their product  $\prod_{i=1}^n S_i$  is the set  $\{(s_1, \dots, s_n) : \forall i \in [1, n](s_i \in S_i)\}$ . As usual, for a set  $S$ , we use  $S^*$  to denote the set of all finite strings over  $S$ . In the sequel, we use  $p_i$  to denote the  $i$ th prime number, e.g.,  $p_1 = 2$ .

**Computational Complexity.** We assume familiarity with complexity classes  $L, P, \Sigma_k^P, PH, NP, PSPACE$  and  $EXPTIME$  (see [12]). The class  $P^{NP}$  (resp.  $P^{NP[\log]}$ ) consists of problems solvable by deterministic poly-time Turing machines with polynomially (resp. logarithmically) many calls to an  $NP$  oracle [13]. As usual, for each  $n \in \mathbb{Z}_{>0}$ , we use  $n$ - $EXPTIME$  to denote the class of problems solvable in  $n$ -fold exponential time. A problem is said to be *elementary* if it is in  $n$ - $EXPTIME$  for some  $n \in \mathbb{Z}_{>0}$ ; otherwise, it is *nonelementary*. We assume familiarity with the alternating Turing machines (ATMs) [12]. Recall that the class of problems solvable by logspace (resp. poly-time) ATMs coincides with  $P$  (resp.  $PSPACE$ ). An ATM with not-states (i.e. states that invert the outcome of the run) can be simulated by one without them without any extra space or time [12].

**Graphs.** Let  $\Sigma$  be a finite set of *actions*. A  $\Sigma$ -labeled *graph* is a tuple  $G = (V, \{E_a\}_{a \in \Sigma})$ , where  $V$  is a set of *vertices* of  $G$ , and each  $E_a \subseteq V \times V$  is a binary *edge relation* (a.k.a. transition relation) over  $V$ . Whenever  $\Sigma$  is clear from the context, we shall omit mention of  $\Sigma$ . We also denote  $E_a$  by  $\rightarrow_a$ , and write  $v \rightarrow_a v'$  instead of  $(v, v') \in \rightarrow_a$ . For each  $\Sigma' \subseteq \Sigma$ , the transitive closure of  $(\bigcup_{a \in \Sigma'} \rightarrow_a)$  is denoted by  $\rightarrow_{\Sigma'}^*$ . We also write  $\rightarrow$  for  $\rightarrow_{\Sigma}$ .

**Asynchronous products.** Let  $\Sigma_1, \dots, \Sigma_r$  be  $r$  pairwise disjoint sets of actions. Let  $\Sigma$  be their union. For each  $i \in [1, r]$ , let  $G_i = (V_i, \{E_a\}_{a \in \Sigma_i})$  be a  $\Sigma_i$ -labeled graph. An *asynchronous product* of  $G_1, \dots, G_r$  is the graph  $\prod_{i=1}^r G_i := (V, \{\bar{E}_a\}_{a \in \Sigma})$ , where  $V := \prod_{i=1}^r V_i$  and, whenever  $a \in \Sigma_i$ ,  $\bar{u} = (u_1, \dots, u_r)$ , and  $\bar{v} = (v_1, \dots, v_r)$ , we have  $(\bar{u}, \bar{v}) \in \bar{E}_a$  iff  $(u_i, v_i) \in E_a$  and  $u_j = v_j$  for all  $j \neq i$ . Intuitively, the product is “asynchronous” as each edge relation in  $\prod_{i=1}^r G_i$  changes at most one component in each vertex of  $\prod_{i=1}^r G_i$ , i.e., causing no interaction between different components. See [22,29] for more details.

**Other logical structures.** We define  $S1S_{<}$  to be the structure  $(\mathbb{N}, <)$ , i.e., natural numbers with a (binary) linear order relation  $<$ . The structure  $(\mathbb{N}, +)$  consists of natural numbers with a ternary relation  $+$  interpreted as additions over  $\mathbb{N}$ . See [12,25] for more details.

**One-counter processes.** A *one-counter process* (OCP) over an action alphabet  $\Sigma$  is a tuple  $\mathcal{O} = (Q, \delta^+, \delta^0)$ , where  $Q$  is a finite set of *control states*,  $\delta^+ \subseteq Q \times \Sigma \times Q \times \{-1, 0, 1\}$  is a finite set of *non-zero transitions*, and  $\delta^0 \subseteq Q \times \Sigma \times Q \times \{0, 1\}$  is a finite set of *zero transitions*. Transitions of the form  $(q, a, q', -1)$ ,  $(q, a, q', 0)$ , and  $(q, a, q', 1)$  are, respectively, called *pop transitions*, *internal transitions*, and *push transitions*. The *size*  $|\mathcal{O}|$  of  $\mathcal{O}$  is defined as  $|Q| + |\delta^0| + |\delta^+|$ . The OCP  $\mathcal{O}$  generates the graph  $\mathcal{G}(\mathcal{O}) = (Q \times \mathbb{N}, \{E_a\}_{a \in \Sigma})$ , where  $((q, n), (q', n + k)) \in E_a$  iff either  $n = 0$  and  $(q, a, q', k) \in \delta^0$ , or  $n > 0$  and  $(q, a, q', k) \in \delta^+$ .

An *asynchronous product*  $\mathcal{O}$  of  $r$  OCPs is simply a tuple of  $r$  OCPs  $\mathcal{O}_1, \dots, \mathcal{O}_r$  over pairwise disjoint action alphabets  $\Sigma_1, \dots, \Sigma_r$ , whose control states need not be pairwise disjoint. The product  $\mathcal{O}$  has action labels  $\Sigma := \Sigma_1 \cup \dots \cup \Sigma_r$ . Then, the graph  $\mathcal{G}(\mathcal{O})$  generated by  $\mathcal{O}$  is defined to be the  $\Sigma$ -labeled graph  $\prod_{i=1}^r \mathcal{G}(\mathcal{O}_i)$ . The system  $\mathcal{O}$  also defines another graph  $\mathcal{G}_S(\mathcal{O})$ , which is simply  $\mathcal{G}(\mathcal{O})$  expanded with the “synchronizing” edge relations  $\{=_{i,j}\}_{1 \leq i \neq j \leq r}$  that are defined as

$$=_{i,j} := \{(\bar{c}, \bar{c}) : c_i = c_j\},$$

where  $\bar{c} = (c_1, \dots, c_r)$ . In other words, the relation  $=_{i,j}$  contains all self-loops in  $\mathcal{G}(\mathcal{O})$  restricted to tuples, where  $i$ th and  $j$ th component agree. The graph  $\mathcal{G}_S(\mathcal{O})$  has action labels  $\Sigma \cup \{(i, j)\}_{1 \leq i \neq j \leq r}$ .

**Logics.** We assume familiarity with first-order logic FO (see [15]). If the free variables of  $\phi \in \text{FO}$  are amongst  $x_1, \dots, x_n$ , we may write  $\phi(x_1, \dots, x_n)$  instead of  $\phi$ . Given a graph  $G = (V, \{E_a\}_{a \in \Sigma})$  and a tuple  $\bar{v} = (v_1, \dots, v_n) \in V^n$ , we write  $G \models \phi[\bar{v}]$  to mean that  $\phi$  is true in  $G$  over the valuation which assigns  $v_i$  to  $x_i$ . The same notations can be easily defined when dealing with  $(\mathbb{N}, +)$ . The *quantifier rank* of  $\phi \in \text{FO}$  is the maximum quantifier nesting depth in  $\phi$ .

The  $k$ -variable first-order logic  $\text{FO}^k$  is the restriction of FO to formulas using at most  $k$  variables. Over  $\Sigma$ -labeled graphs, the logic FO(R) (resp.  $\text{FO}^k(\text{R})$ ) is the extension of FO (resp.  $\text{FO}^k$ ) with binary relations  $R_{\Sigma'}$  (for each  $\Sigma' \subseteq \Sigma$ ) interpreted as the transitive closure relation  $\rightarrow_{\Sigma'}^*$  (see [29]). Denote  $R_\Sigma$  by  $R$ . In the sequel, we often use  $\rightarrow_{\Sigma'}^*$  to denote  $R_{\Sigma'}$ .

Formulas in the basic modal logic ML over  $\Sigma$ -labeled graphs are built from the following grammar:  $\phi, \psi ::= \top \mid \neg\psi \mid \phi \vee \psi \mid \langle a \rangle \phi$  ( $a \in \Sigma$ ). Given a graph  $G = (V, \{E_a\}_{a \in \Sigma})$  and each  $\phi \in \text{ML}$ , define a set  $\llbracket \phi \rrbracket_G \subseteq V$  as follows:

- (1)  $\llbracket \top \rrbracket_G = V$ ;
- (2)  $\llbracket \neg\phi \rrbracket_G = V - \llbracket \phi \rrbracket_G$
- (3)  $\llbracket \phi \vee \psi \rrbracket_G = \llbracket \phi \rrbracket_G \cup \llbracket \psi \rrbracket_G$
- (4)  $\llbracket \langle a \rangle \phi \rrbracket_G = \{u \in V : \exists v \in V (u \rightarrow_a v \text{ and } v \in \llbracket \phi \rrbracket_G)\}$

As usual, use  $\perp$ ,  $\phi \wedge \psi$ , and  $[a]\phi$  to denote  $\neg\top$ ,  $\neg(\neg\phi \vee \neg\psi)$ , and  $\neg\langle a \rangle\neg\phi$ , respectively. The logic ML(R) is the extension of ML with reachability modalities  $\langle R_{\Sigma'} \rangle$  (for each  $\Sigma' \subseteq \Sigma$ ), where  $\llbracket \langle R_{\Sigma'} \rangle \phi \rrbracket_G := \{u \in V : \exists v \in V (u \rightarrow_{\Sigma'}^* v \text{ and } v \in \llbracket \phi \rrbracket_G)\}$ . For the purpose of this paper, the EF-logic is the logic ML(R). [This is a slightly more general logic than the commonly considered EF-logic, which is more convenient to work with in our cases. However, all our results will hold as well for the restricted EF-logic.] There is an easy standard translation (see [15]) from formulas in ML (resp. ML(R)) to formulas in  $\text{FO}^2$  (resp.  $\text{FO}^2(\text{R})$ ) with one free variable. Over IOCPs, we shall use  $\text{FO}_S(\text{R})$ ,  $\text{FO}_S^k(\text{R})$ ,  $\text{ML}_S(\text{R})$ , and the  $\text{EF}_S$ -logic to denote the logics FO(R),  $\text{FO}^k(\text{R})$ , ML(R), and the EF-logic with synchronizing predicates  $\{=_{i,j}\}$ , interpreted over graphs of the form  $\mathcal{G}_S(\mathcal{O})$ .

The model checking problems for any of the above logic  $L$  over OCPs (resp. IOCPs) can be defined in the obvious way, i.e., with respect to the graph  $\mathcal{G}(\mathcal{O})$  generated by the input OCP (resp. IOCPs). The input formulas are permitted to have free variables, which are to be interpreted as configurations in  $\mathcal{G}(\mathcal{O})$ , where numbers are *represented in binary*. Also, if  $L$  is  $\text{FO}_S(\text{R})$ ,  $\text{FO}_S^k(\text{R})$ ,  $\text{ML}_S(\text{R})$ , or the  $\text{EF}_S$ -logic, the interpretation is over  $\mathcal{G}_S(\mathcal{O})$ .



The *first-order modulo counting logic*  $\text{FO}_{\text{MOD}}$  extends FO with the modulo counting quantifiers  $\exists^{p,q}$ , for each  $q \in \mathbb{Z}_{>0}$  and  $p \in [0, q)$ . In this paper, we consider  $\text{FO}_{\text{MOD}}$  only over  $(\mathbb{N}, <)$ . The semantics of  $\text{FO}_{\text{MOD}}$  is defined over  $(\mathbb{N}, <)$  as follows:  $(\mathbb{N}, <) \models \exists^{p,q} x \phi(x, \bar{b})$  iff the number  $l := |\{a \in \mathbb{N} : (\mathbb{N}, <) \models \phi(a, \bar{b})\}|$  is either infinite or finite and  $l \equiv p \pmod{q}$ . See [21] for more details.

**Alternation rank.** Given an FO formula  $\phi$ , push all the negations to atomic propositions level. The alternation rank  $\text{AL}(\phi)$  of  $\phi$  is then defined as the maximum number of alternations of operators in  $\{\forall, \wedge\}$  and operators in  $\{\exists, \vee\}$  over all paths from the root to the leaves in the parse tree of  $\phi$ .

**Gödel encoding.** For the purpose of this paper, we define the Gödel function  $\mathfrak{G} : \mathbb{Z}_{>0} \rightarrow \{0, 1\}^\omega$  mapping positive integers to infinite binary words as follows: if  $n = \prod_{i>0} p_i^{j_i}$ , where  $j_i \in \mathbb{N}$  and  $p_i$  the  $i$ th prime, then define  $\mathfrak{G}(n) = j'_1 j'_2 \dots$ , where  $j'_i = 0$  if  $j_i = 0$  and  $j'_i = 1$  if  $j_i > 0$ .

### 3 The Logics $\mathcal{L}$ and $\mathcal{L}'$

We define our first syntactic restriction  $\mathcal{L}$  of Presburger Arithmetic, to which we will reduce the model checking of  $\text{FO}(\mathbb{R})$  over  $\Pi\text{OCPS}$ .

**Definition 1.** *The syntax of the logic  $\mathcal{L}$  is as follows. Atomic propositions are of the form:*

- $x \sim y + c$ , where  $\sim \in \{\leq, \geq, =\}$ ,
- $x \sim c$ , where  $\sim \in \{\leq, \geq, =\}$ ,
- $x \equiv y + c \pmod{d}$ , where  $c \in [0, d - 1]$ , and
- $x \equiv c \pmod{d}$ , where  $c \in [0, d - 1]$ .

Here,  $x$  and  $y$  can take any variables, while  $c$  and  $d$  are constant natural numbers, given in binary representations. We then close the logic under boolean combinations, and existential and universal quantifications. The semantics is given directly from Presburger Arithmetic. The expression  $x \equiv y + c \pmod{d}$  is to be interpreted as the Presburger formula  $\exists z (x = y + c + dz \vee x + dz = y + c)$ .

Intuitively, the logic  $\mathcal{L}$  is the fragment of Presburger Arithmetic that permits only inequality tests, addition with constants, and modulo tests. We now impose some further syntactic restrictions to our logic  $\mathcal{L}$ , to which model checking  $\text{FO}^2(\mathbb{R})$  over  $\Pi\text{OCPS}$  is still poly-time reducible.

**Definition 2.** *Define the logic  $\mathcal{L}'$  as follows. The only variables allowed are  $x_i$  and  $y_i$ , where  $i \in \mathbb{Z}_{>0}$ . The atomic propositions of  $\mathcal{L}'$  are given as follows for each  $i \in \mathbb{Z}_{>0}$ :*

- $x_i \sim y_i + c$  and  $y_i \sim x_i + c$ ,
- $x_i \sim c$  and  $y_i \sim c$ ,
- $x_i \equiv y_i + c \pmod{d}$  and  $y_i \equiv x_i + c \pmod{d}$ , and
- $x_i \equiv c \pmod{d}$  and  $y_i \equiv c \pmod{d}$ .

Here,  $c$  and  $d$  are constant natural numbers given in binary. We then close the logic under boolean combinations, and existential and universal quantifications.

The logic  $\mathcal{L}'$  allows only two variables  $x_i$  and  $y_i$  to be related. In fact, if we only allow  $x_1$  and  $y_1$  as variables, then  $\mathcal{L}'$  coincides with  $\text{FO}^2$  fragment of  $\mathcal{L}$ .

We shall briefly discuss the expressive power of  $\mathcal{L}$  in terms of subsets of  $\mathbb{N}^k$  that can be defined in the logics. It can be shown that  $\mathcal{L}$  coincides with the  $\text{FO}_{\text{MOD}}$  theory over  $(\mathbb{N}, <)$ . In fact, [21] shows that  $\text{FO}_{\text{MOD}}$  theory over  $(\mathbb{N}, <)$  admits a quantifier elimination, when the vocabulary is expanded with congruence tests. Therefore,  $\mathcal{L}$  subsumes  $\text{FO}_{\text{MOD}}$  over  $(\mathbb{N}, <)$ . To show that  $\mathcal{L} \subseteq \text{FO}_{\text{MOD}}(\mathbb{N}, <)$ , observe that expressions of the form  $x \sim y + c$  can easily be replaced by equivalent FO formulas over  $(\mathbb{N}, <)$ . Also, the atomic formula  $x \equiv y + c \pmod{d}$  can be defined as  $\bigwedge_{a=0}^{d-1} (y \equiv a \pmod{d} \leftrightarrow x \equiv a + c \pmod{d})$ , and congruence tests  $x \equiv a \pmod{d}$  can be defined in  $\text{FO}_{\text{MOD}}$  over  $(\mathbb{N}, <)$  as  $\exists^{a,d} y (y < x)$ . The expressive power of  $\text{FO}_{\text{MOD}}$  over  $(\mathbb{N}, <)$  was shown in [21] to be strictly in between FO over  $(\mathbb{N}, <)$  and Presburger Arithmetic. For example, it was shown that Presburger formulas of the form  $x = 2y$  is not definable in  $\text{FO}_{\text{MOD}}$  over  $(\mathbb{N}, <)$ . Finally, we shall emphasize that the proof in [21] of quantifier elimination for  $\text{FO}_{\text{MOD}}$  over  $(\mathbb{N}, <)$  expanded with congruence tests is nonconstructive.

The *membership problem of the logic  $\mathcal{L}$*  is as follows: given  $\phi(\bar{x}) \in \mathcal{L}$ , where  $\bar{x} = (x_1, \dots, x_n)$  and a tuple  $\bar{a} \in \mathbb{N}^n$  in binary, decide whether  $\mathbb{N} \models \phi(\bar{a})$ . The membership problem for  $\mathcal{L}'$  can be defined similarly. We now state a proposition, which can be proved easily (but somewhat tedious) using the result in [8, Lemma 4.6].

**Proposition 3.** *There is a poly-time reduction from the problem of model checking  $\text{FO}_{\mathbb{S}}(\text{R})$  (resp.  $\text{FO}^2(\text{R})$ ) over  $\Pi\text{OCPs}$  to the membership problem for  $\mathcal{L}$  (resp.  $\mathcal{L}'$ ). Furthermore, the alternation rank of the output formula in  $\mathcal{L}$  (resp.  $\mathcal{L}'$ ) is the same as the alternation rank of the input formula in  $\text{FO}_{\mathbb{S}}(\text{R})$  (resp.  $\text{FO}^2(\text{R})$ ) up to addition by a small constant.*

## 4 Upper Bounds

In this section, we shall show that the combined and data complexity of  $\text{FO}_{\mathbb{S}}(\text{R})$  over  $\Pi\text{OCPs}$  are, respectively, in PSPACE and PH. We then show that the expression complexity of  $\text{FO}^2(\text{R})$  is in P. To deduce a  $\text{P}^{\text{NP}}$  upper bound for data complexity of EF-logic over  $\Pi\text{OCPs}$ , it suffices to invoke the Feferman-Vaught type of composition method for EF-logic [22] and use the  $\text{P}^{\text{NP}}$  algorithm for model checking EF-logic over OCPs from [8]. Observe that these will give the claimed upper bounds in Table 1 and Table 2.

### 4.1 Combined and Data Complexity of $\text{FO}_{\mathbb{S}}(\text{R})$

**Theorem 4.** *The combined and data complexity  $\text{FO}_{\mathbb{S}}(\text{R})$  over  $\Pi\text{OCPs}$  are in PSPACE and in PH, respectively.*

By Proposition 3, to deduce this theorem it suffices to prove the following proposition.

**Proposition 5.** *The membership problem of  $\mathcal{L}$ -formulas is in PSPACE. Moreover, fixing the alternation rank of input formulas, the problem is in PH.*

The proof is done via a quantifier elimination technique (e.g. see [12] for an overview). Loosely speaking, our proof can be thought of as an extension of Ehrenfeucht-Fraïssé games on linear orders (e.g. see [15]) with modulo tests. We first define an equivalence relation  $\equiv_{p,m}^k$  on tuples of natural numbers.

**Definition 6.** *Given two  $(k + 1)$ -tuples  $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$  of natural numbers such that  $a_0 = b_0 = 0$  and two numbers  $p, m > 0$ , we write  $\bar{a} \equiv_{p,m}^k \bar{b}$  iff for all  $i, j \in [0, k]$  the following statements hold:*

1.  $|a_i - a_j| < pm$  implies  $|a_i - a_j| = |b_i - b_j|$ ,
2.  $|b_i - b_j| < pm$  implies  $|a_i - a_j| = |b_i - b_j|$ ,
3.  $|a_i - a_j| \geq pm$  iff  $|b_i - b_j| \geq pm$ ,
4.  $a_i \equiv b_i \pmod{p}$ ,
5.  $a_i \leq a_j$  iff  $b_i \leq b_j$ .

It is easy to see that, given  $m' \geq m > 0$ , we have  $\bar{a} \equiv_{p,m'}^k \bar{b}$  implies  $\bar{a} \equiv_{p,m}^k \bar{b}$ . Similarly, if  $p|p'$ , then  $\bar{a} \equiv_{p',m}^k \bar{b}$  implies  $\bar{a} \equiv_{p,m}^k \bar{b}$ . The following lemma can be used to eliminate a quantifier.

**Lemma 7.** *Given two  $(k + 1)$ -tuples  $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$  of natural numbers such that  $a_0 = b_0 = 0$  and two numbers  $p, m > 0$ , if  $\bar{a} \equiv_{p,3m}^k \bar{b}$ , then for all  $a' \in \mathbb{N}$ , there exists  $b' \in \mathbb{N}$  such that  $\bar{a}, a' \equiv_{p,m}^{k+1} \bar{b}, b'$ .*

Let us consider only tuples  $\bar{a} = (a_0, \dots, a_k)$  of natural numbers satisfying  $a_0 = 0$ . Given an  $\equiv_{p,3m}^k$ -equivalence class  $C$  and an  $\equiv_{p,m}^{k+1}$ -equivalence class  $C'$ , we say that  $C'$  is *consistent with  $C$*  if there exist a tuple  $\bar{a} = (a_0, \dots, a_k)$  of natural numbers and a number  $a' \in \mathbb{N}$  such that  $a_0 = 0$ ,  $\bar{a} \in C$ , and  $(\bar{a}, a') \in C'$ . The following lemma shows that we need not consider large numbers when eliminating a quantifier.

**Lemma 8.** *Let  $\bar{a} = (a_0, \dots, a_k)$  be a tuple of natural numbers and  $C$  be its  $\equiv_{p,3m}^k$ -equivalence class. Then, every  $\equiv_{p,m}^{k+1}$ -equivalence class has a representative in the set  $\{(\bar{a}, a') : 0 \leq a' \leq \max(\bar{a}) + pm + p\}$ .*

Define  $r(0, m) := m$  and  $r(n + 1, m) := 3r(n, m)$ , for  $n \in \mathbb{N}$ . By induction, we have  $r(n, m) = 3^n m$ . Let us now define the notion of *offsets* and *periods* of formulas in  $\mathcal{L}$ . If  $\phi$  are atomic formulas of the form  $x \sim y + c$ ,  $x \sim c$ ,  $x \equiv y + c \pmod{d}$ , or  $x \equiv c \pmod{d}$ , then *offsets* of  $\phi$  are defined to be the integer  $c$ . If  $\phi$  is not an atomic formula, then its *offset* is the largest offset of atomic subformulas of  $\phi$ . If  $\phi$  are atomic formulas of the form  $x \sim y + c$  or  $x \sim c$ , then its *period* is defined to be 1. If  $\phi$  are atomic formulas of the form  $x \equiv y + c \pmod{d}$  or  $x \equiv c \pmod{d}$ , then its *period* is defined to be  $d$ . Otherwise, if  $\phi$  is not an atomic formula, its *period* is defined to be the least common multiple of the periods of each of its atomic subformulas. For  $p, m \in \mathbb{Z}_{>0}$ , define  $\mathcal{L}_{p,m}$  to be formulas in  $\mathcal{L}$ , whose periods divide  $p$  and whose offsets are smaller than  $m$ .

**Lemma 9.** *Let  $p, m \in \mathbb{Z}_{>0}$ . Suppose  $\bar{a} = (a_0, \dots, a_k), \bar{b} = (b_0, \dots, b_k)$  are tuples of natural numbers satisfying  $a_0 = b_0 = 0$  and  $\bar{a} \equiv_{p, r(n, m)}^k \bar{b}$ . Then, given a formula  $\phi(x_1, \dots, x_k)$  in  $\mathcal{L}_{p, m}$  of quantifier rank  $n$ ,*

$$(\mathbb{N}, +) \models \phi(a_1, \dots, a_k) \Leftrightarrow (\mathbb{N}, +) \models \phi(b_1, \dots, b_k).$$

We are now ready to prove Proposition 5.

*Proof (of Proposition 5).* We now give a poly-time ATM  $M$  which checks whether  $(\mathbb{N}, +) \models \phi(a_1, \dots, a_n)$  for given a formula  $\phi(x_1, \dots, x_n)$  and a  $n + 1$ -tuple  $\bar{a} = (a_0, \dots, a_n)$ , where  $a_0 = 0$ . First, push all the negations downward to the atomic propositions level, which can be done easily. Suppose that  $p$  and  $m$  be, respectively, the period and offset of the input formula. Now if  $\phi$  is an atomic proposition (i.e. inequality, or modulo tests), it is easy to see that  $M$  can check it in poly-time. If  $\phi$  is  $\psi \vee \psi'$  (resp.  $\psi \wedge \psi'$ ), then existentially (resp. universally) guess  $\psi$  or  $\psi'$  and check the guessed formula. If  $\phi$  is of the form  $\exists x\psi(\bar{y}, x)$  (resp.  $\forall x\psi(\bar{y}, x)$ ) and has quantifier rank  $k$ , then  $M$  existentially (resp. universally) guesses a number  $a_{n+1}$  not exceeding  $\max(\bar{a}) + pr(k, m) + p \leq \max(\bar{a}) + p3^k m + p$  and check whether  $(\mathbb{N}, +) \models \psi(\bar{a}, a_{n+1})$ . The upper bound for  $a_{n+1}$  is sufficient due to Lemma 8.

To analyze the running time of  $M$ , notice that the maximum number that  $M$  can guess on any of its run on input  $\phi$  of quantifier rank  $h$  and a tuple  $\bar{a}$  of natural numbers (in binary) is  $\max(\bar{a}) + \sum_{j=0}^h (pr(j, m) + p) \leq \max(\bar{a}) + p(h + 1)3^h m + p(h + 1)$ , which can be represented using polynomially many bits. [Note that  $p$  and  $m$  are represented in binary and so the guessed number is polynomial in  $\log(p)$  and  $\log(m)$ .] This implies that membership of  $\mathcal{L}$ -formulas is in PSPACE. Finally, notice that the number of alternations used by  $M$  corresponds to the alternation rank of  $\phi$ . Therefore, considering only formulas of fixed alternation rank, the membership problem for  $\mathcal{L}$ -formulas is in PH. □

### 4.2 Expression Complexity of FO<sup>2</sup>(R)

**Theorem 10.** *The expression complexity of FO<sup>2</sup>(R) over IOCPs is in P.*

Define  $\mathcal{L}'_{p, m}$  to be the set of all formulas in  $\mathcal{L}'$  whose periods divide  $p$  and whose offsets do not exceed  $m$ . Let  $\mathcal{L}'_{p, m}(n)$  to be the set of all formulas in  $\mathcal{L}'_{p, m}$  that use only variables in  $\{x_1, \dots, x_n\} \cup \{y_1, \dots, y_n\}$ . For all fixed  $p, m, n \in \mathbb{Z}_{>0}$ , the membership problem of  $\mathcal{L}'_{p, m}(n)$  is as follows: given  $\phi(\bar{x}, \bar{y}) \in \mathcal{L}'_{p, m}(n)$  and two tuples  $\bar{a}, \bar{b} \in \mathbb{N}^n$  of numbers in binary representation, decide whether  $(\mathbb{N}, +) \models \phi(\bar{a}, \bar{b})$ . By Proposition 3, Theorem 10 follows from the following proposition.

**Proposition 11.** *For fixed  $p, m, n \in \mathbb{Z}_{>0}$ , the membership problem of  $\mathcal{L}'_{p, m}(n)$  is in P.*

This proposition can also be proved via quantifier elimination. The intuition that we can obtain a poly-time algorithm is from a two-pebble Ehrenfeucht-Fraïssé

games over linear orders (see [15]), which can only distinguish small linear orders (i.e. only linear in the quantifier rank of the  $\text{FO}^2(\text{R})$  formula). The proof is similar to the case for  $\text{FO}(\text{R})$ , but is much more tedious.

## 5 Lower Bounds

In order to facilitate our lower bound proofs in this section, we shall define a 2-player game, called *the buffer game*, which we shall prove to be PSPACE-complete. First, let  $\mathcal{L}_{\text{DIV}}$  be the set of quantifier-free  $\mathcal{L}$ -formulas in 3-CNF (i.e. in CNF and each clause has exactly three literals) with one free variable  $x$ , whose atomic propositions are of the form  $x \equiv 0 \pmod{p}$  where  $p$  is a prime number. The buffer game is played by Player  $\exists$  and Player  $\forall$ . An arena of the buffer game is a tuple  $(\bar{v}, k, \phi)$ , where  $\bar{v}$  is a finite and strictly increasing sequence of positive integers,  $k$  is the number of integers in  $\bar{v}$ , and  $\phi$  a formula of  $\mathcal{L}_{\text{DIV}}$ . The buffer game with arena  $(\bar{v}, k, \phi)$ , where  $\bar{v} = (v_1, \dots, v_k)$ , has  $k + 1$  rounds and is played as follows. Each round  $r$  defines a *positive* number  $m_r$ , which represents the current buffer value. At round 0, Player  $\exists$  chooses a number  $m_0$  to be written to the buffer. Suppose that  $0 < r \leq k$ , and  $m_0, \dots, m_{r-1}$  are the buffer values chosen from the previous rounds. At even (resp. odd) round  $r$ , Player  $\exists$  (resp. Player  $\forall$ ) rewrites the buffer by a number  $m_r \geq m_{r-1}$  of his choosing such that  $m_r \equiv m_{r-1} \pmod{\prod_{j=1}^{v_r} p_j}$ , i.e.,  $m_r = m_{r-1} + c \left(\prod_{j=1}^{v_r} p_j\right)$  for some  $c \in \mathbb{N}$ . In particular, by Chinese remainder theorem, this condition implies that, for each  $1 \leq j \leq v_r$ ,  $p_j | m_r$  iff  $p_j | m_{r-1}$ . In other words, each player is not allowed to “overwrite” some divisibility information in the buffer. Player  $\exists$  *wins* if  $(\mathbb{N}, +) \models \phi(m_k)$ . Otherwise, Player  $\forall$  *wins*. The problem **BUFFER** is defined as follows: given an arena  $(\bar{v}, k, \phi)$  of the buffer game, where *each number is represented in unary*, decide whether Player  $\exists$  has a winning strategy. For each  $n \in \mathbb{N}$ , we define the problem **BUFFER<sub>n</sub>** to be the restriction of the problem **BUFFER** which takes only an input arena of the form  $(\bar{v}, n, \phi)$ .

**Lemma 12.** *The problem **BUFFER** is PSPACE-complete. The problem **BUFFER<sub>k</sub>** is  $\Sigma_{k+1}^P$ -complete.*

Loosely speaking, by applying Gödel encoding one can encode each truth valuation for boolean formulas into a number. Therefore, boolean formulas can be reduced to statements about divisibility. Furthermore, a *block* of  $\exists$  (resp.  $\forall$ ) quantifiers in a quantified boolean formula can be reduced into a choice of number at a single round in the buffer game for Player  $\exists$  (resp. Player  $\forall$ ).

We now use the buffer game to prove our first lower bound result for the problem of model checking OCPs.

**Proposition 13.** *Combined complexity of  $\text{FO}^2(\text{R})$  on OCPs is PSPACE-hard. For every  $k \in \mathbb{N}$ , there is a fixed formula  $\phi_k$  of  $\text{FO}^2(\text{R})$  with  $k + c$  quantifier alternations, for some small constant  $c \in \mathbb{N}$ , such that checking  $\phi_k$  over OCPs is  $\Sigma_k^P$ -hard.*

To prove this theorem, we first state a standard lemma, whose proof can be found in [8][11] (similar proof techniques have been used earlier in [14]).

**Lemma 14.** *Given a  $\mathcal{L}_{\text{DIV}}$ -formula  $\phi$ , we can compute in polynomial time an OCP  $\mathcal{O}$  with a fixed set  $\Gamma$  of action symbols and an initial state  $q_I$  such that, for each positive integer  $m$ , it is the case that  $\mathcal{G}(\mathcal{O}), (q_I, m) \models \alpha$  iff  $(\mathbb{N}, +) \models \phi(m)$ , where  $\alpha$  is a small fixed EF formula.*

The crucial idea in the proof of the above lemma is that both divisibility and indivisibility tests of the form  $p|x$  or  $p \not|x$  can be reduced to a certain reachability question for an appropriate OCP  $\mathcal{O}$  by embedding a cycle of length  $p$  in  $\mathcal{O}$ .

*Proof (sketch of Proposition 13).* We give a poly-time reduction from BUFFER. Given an arena  $\mathcal{A} = (\bar{v}, k, \phi)$ , we compute an  $\text{FO}^2(\text{R})$  sentence  $\phi'$ , and a OCP  $\mathcal{O} = (Q, \delta^+, \delta^0)$  such that Player  $\exists$  has a winning strategy in  $\mathcal{A}$  iff  $\mathcal{G}(\mathcal{O}) \models \phi'$ . Let  $\bar{v} = (v_1, \dots, v_k)$ . As we shall see,  $\phi'$  depends only on  $k$  and has quantifier rank  $k + c$  for some small constant  $c \in \mathbb{N}$ , which by Lemma 12 will prove the desired lower bound for data complexity.

We now run the algorithm given by Lemma 14 on input  $\phi$  to compute a OCP  $\mathcal{O}_1 = (D, \delta_1^+, \delta_1^0)$  with initial state  $q_I \in D$ . The key now is to build on top of  $\mathcal{O}_1$  and the fixed formula  $\alpha$  (which can be thought of as an  $\text{FO}^2(\text{R})$  formula) so as to encode the initial guessing of numbers.

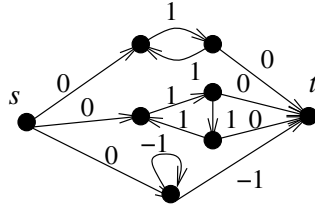
The structure of our output OCP  $\mathcal{O}$  can be visualized as

$$B_0 \rightarrow B_1 \dots \rightarrow B_k \rightarrow \mathcal{O}_1.$$

The number  $k + 1$  of blocks  $B_i$  in  $\mathcal{O}$  corresponds to the number of rounds played in the buffer game. The initial state is in block  $B_0$ . Our output  $\text{FO}^2(\text{R})$  formula will have  $k + 1$  leading (alternating) quantifiers so as to ensure that each player moves in their designated rounds. One variable will be used for storing the last buffer value from the previous round, while the other is used for storing the buffer value after the designated player has made his move. We now describe how to ensure that at each round  $i$  ( $i > 0$ ) the player can only add numbers that are in the set  $H_i := \{c (\prod_{j=1}^{v_i} p_j) : c \in \mathbb{N}\}$ . Define the function  $g : \mathbb{Z}_{>0} \rightarrow \mathbb{Z}_{>0}$  as  $g(s) := \prod_{j=1}^s p_j$ . Note that  $g$  grows exponentially in  $s$ , which is why we cannot simply embed a cycle of length  $g(v_i)$  in  $B_i$ , for each  $i \in [1, k]$ . On the other hand, notice that  $H_i$  is  $\mathbb{Z} - L_i$ , where  $L_i := \left( \bigcup_{1 \leq j \leq v_i} \bigcup_{a \in (0, p_j)} a + \mathbb{N}p_j \right) \cup \mathbb{Z}_{<0}$ .

In turn,  $L_i$  can be characterized as the set of weights of paths in a small finite graph  $G_i$  from a vertex  $s$  to a vertex  $t$ , where the *weight* of a path is the sum of the weights of its edges (which we shall allow to be only either -1, 0, or 1). In fact,  $G_i$  will have  $O(\sum_{j=1}^{v_i} p_j)$  vertices, which is polynomial in  $v_i$ . For example, the set  $(1 + 2\mathbb{N}) \cup (1 + 3\mathbb{N}) \cup (2 + 3\mathbb{N}) \cup \mathbb{Z}_{<0}$  corresponds to the weights of  $s \rightarrow^* t$  paths in the graph in Figure 11.

Furthermore, the graph  $G_i$  can be thought of as an OCP. Adding the self-loop transitions  $(s, \text{loop}_s, s, 0)$  and  $(t, \text{loop}_t, t, 0)$  on states  $s$  and  $t$ , the binary relation  $\{((s, a), (t, a + b)) : b \in H_i\}$  can then be expressed in  $\text{FO}^2(\text{R})$  as  $\neg(x \rightarrow^* y) \wedge E_{\text{loop}_s}(x, x) \wedge E_{\text{loop}_t}(y, y)$ . Therefore, we shall embed the modified OCP  $G_i$  into  $B_i$ , where  $t$  will be the entry state for block  $B_{i+1}$  of  $\mathcal{O}$ . [ $B_{k+1}$  shall be interpreted as  $\mathcal{O}_1$ .]



**Fig. 1.** The  $s \rightarrow^* t$  path-weights in this graph equals  $(1 + 2\mathbb{N}) \cup (1 + 3\mathbb{N}) \cup (2 + 3\mathbb{N}) \cup \mathbb{Z}_{<0}$

Finally, using this idea, it is not difficult to compute the desired  $\text{FO}^2(\mathbb{R})$  sentence by mimicking the  $k + 1$  rounds of the game by using at most  $k + c$  alternating quantifiers (using only the variables  $x$  and  $y$ ). The end buffer value  $m$ , which needs to be checked against  $\phi$ , can be checked against  $\alpha$  instead.  $\square$

We can also apply Lemma 12 to prove the following lower bound.

**Proposition 15.** *The combined complexity of model checking EF-logic over an asynchronous product of two one-counter processes is PSPACE-hard.*

Intuitively, instead of simulating each alternation in the buffer game as values in the two variables  $x$  and  $y$ , we can simulate them as values in two different counters. We can make sure that the divisibility information is not “overwritten” by encoding it as a non-fixed formula.

We saw in the previous section that the expression complexity of  $\text{FO}^2(\mathbb{R})$  over IOCPs is in P. In contrast, we can show that this is not the case for  $\text{FO}^4(\mathbb{R})$  even over OCPs (without products).

**Proposition 16.** *The expression complexity of  $\text{FO}^4(\mathbb{R})$  (without equality relation) over OCPs is PSPACE-hard.*

The fixed graph is in fact  $(\mathbb{N}, <)$ . The proof adapts the technique in 9 of succinctly encoding addition arithmetic on large numbers using the successor relations and linear order  $<$  with only four variables.

We already saw that the data complexity of EF-logic over IOCPs is  $\mathbf{P}^{\text{NP}}$ . In contrast, we can show the following proposition.

**Proposition 17.** *For each  $k \in \mathbb{N}$ , there is a fixed  $\text{EF}_S$ -logic formula  $\phi_k$  such that model checking  $\phi_k$  over IOCPs is  $\Sigma_k^p$ -hard.*

Intuitively, by using the synchronization constraints, one can faithfully simulate two variables  $x$  and  $y$  in any given  $\text{FO}^2(\mathbb{R})$  formula as values of two different counters. This idea can easily be adapted for showing the following proposition by appealing to Proposition 16.

**Proposition 18.** *The expression complexity of  $\text{EF}_S$ -logic over IOCPs is hard for PSPACE.*

## 6 Modal Logic over Automatic Graphs

Automatic graphs [2] are those graphs  $G = (V, \{E_a\}_{a \in \Sigma})$ , where  $V$  is a regular subset of  $\Sigma^*$  for some finite alphabet  $\Sigma$ , and each edge relation  $E_a \subseteq \Sigma^* \times \Sigma^*$  is recognizable by a synchronous transducer over  $\Sigma$ . We briefly recall the definition of synchronous transducers — see [2] for more details. A *synchronous transducer*  $R$  over  $\Sigma$  is a finite word-automaton over the product alphabet  $\Sigma_{\perp} \times \Sigma_{\perp}$ , where  $\Sigma_{\perp} := \Sigma \cup \{\perp\}$  and  $\perp \notin \Sigma$ . Given  $v, w \in \Sigma^*$  where  $v = a_1 \dots a_n$  and  $w = b_1 \dots b_m$ , define  $v \otimes w$  to be the word  $c_1 \dots c_k$  over  $\Sigma_{\perp} \times \Sigma_{\perp}$ , where  $k = \max(n, m)$  and

$$c_i = \begin{cases} (a_i, b_i) & \text{if } i \leq \min(n, m), \\ (\perp, b_i) & \text{if } n < i \leq m, \\ (a_i, \perp) & \text{if } m < i \leq n. \end{cases}$$

The edge relation definable by  $R$  consists of each ordered pair  $(v, w) \in \Sigma^* \times \Sigma^*$  such that the word  $v \otimes w$  is accepted by  $R$  (in the usual automata sense). Rational graphs [19] are similar to automatic graphs, but use a more general notion of transducers.

The FO (resp. ML) theories of automatic (resp. rational) graphs are known to be decidable, e.g., see [2][1]. In fact, the infinite binary tree S2S with a linear order is automatic [2], which implies that model checking FO over automatic graphs is nonelementary. Recently, the authors of [26] and [1] asked whether the complexity of model checking ML over, respectively, automatic graphs and rational graphs is nonelementary. We shall show that this is the case in the stronger sense by establishing a *fixed* automatic graph whose ML theory is nonelementary. Since automatic graphs are rational [19], the same can be said about rational graphs. Our proof uses the proof method for Proposition [17]. Due to space limit, we shall only define a graph  $\mathfrak{T}$  whose modal logic theory we claim to be nonelementary. Its proof can be found in the full version. Furthermore, one can easily check that the graph  $\mathfrak{T}$  is automatic.

We denote by  $\text{S2S}_{<} := (\{0, 1\}^*, \text{succ}_0, \text{succ}_1, <)$  the infinite binary tree with a descendant relation, i.e.,  $\text{succ}_0 := \{(w, w0) : w \in \{0, 1\}^*\}$ ,  $\text{succ}_1 := \{(w, w1) : w \in \{0, 1\}^*\}$ , and  $< := \{(w, wv) : w, v \in \{0, 1\}^*\}$ . Although the FO theory of  $\text{S2S}_{<}$  was proved to be nonelementary in [4], it is not easy to see whether  $\text{FO}^k$  suffices from the proof. Nevertheless, one can easily show that  $\text{FO}^4$  suffices as follows. Using Stockmeyer’s well-known results [24] that equivalence of star-free regular expressions is nonelementary, one can immediately deduce that  $\text{FO}^3$  theory over all finite linear orders with a unary predicate is nonelementary, since there is a linear-time translation from star-free regular expressions to equivalent  $\text{FO}^3$  formulas (e.g. see [6]). One may then use the linear-time reduction given in [4] from FO theory of binary strings to FO theory of  $\text{S2S}_{<}$ , which incurs only an extra variable. So, we have the following proposition.

**Proposition 19.** *The  $\text{FO}^4$  theory of  $\text{S2S}_{<}$  is nonelementary.*

Now define the graph

$$\mathfrak{T} := \langle \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^*, \\ \{\text{succ}_0^i\}_{i=1}^4, \{\text{succ}_1^i\}_{i=1}^4, \{\prec_i\}_{i=1}^4, \{=_{i,j}\}_{1 \leq i < j \leq 4}, \{G_i\}_{i=1}^4 \rangle.$$



where the edge relations are defined as follows:

- $\text{succ}_0^i := \{(\overline{w}, \overline{w}') : w'_i = w_i 0 \text{ and } \forall j \neq i (w_j = w'_j)\}$ . This relation takes the  $i$ th component to its left child.
- $\text{succ}_1^i := \{(\overline{w}, \overline{w}') : w'_i = w_i 1 \text{ and } \forall j \neq i (w_j = w'_j)\}$ . This relation takes the  $i$ th component to its right child.
- $\prec_i := \{(\overline{w}, \overline{w}') : w_i \prec w'_i \text{ and } \forall j \neq i (w_j = w'_j)\}$ . This relation takes the  $i$ th component to its descendant.
- $=_{i,j} := \{(\overline{w}, \overline{w}') : w_i = w_j \text{ and } \forall k (w_k = w'_k)\}$ . This relation simply loops if the  $i$ th component equals the  $j$ th component.
- $\mathbf{G}_i := \{(\overline{w}, \overline{w}') : \forall j \neq i (w_j = w'_j)\}$ . This relation takes the  $i$ th component to any other word (i.e. global modality).

**Proposition 20.** *The graph  $\mathfrak{T}$  is automatic and its ML theory is nonelementary.*

**Theorem 21.** *There exists a fixed automatic (and so rational) graph whose ML theory is nonelementary.*

## 7 Future Work

We conclude now with several future work. We would like to determine the expression complexity of  $\text{FO}^3(\mathbb{R})$  over OCPs and  $\Pi\text{OCPs}$ . Our lower bound proof for  $\text{FO}^4(\mathbb{R})$  does not hold since  $\text{FO}^3(\mathbb{R})$  cannot succinctly encode arithmetic of large numbers using only successors and linear orders [9]. We would also like to study the combined complexity of EF-logic over asynchronous products of PDA. It is known to be PSPACE-hard [27] and decidable (by applying the compositional method [22]), but no better upper or lower bound is known. [In fact, it seems not clear how to adapt the proof of PSPACE upper bound for PDA to asynchronous products.] On the other hand, it follows from the proof of Theorem 21 that model checking  $\text{EF}_{\mathcal{S}}$ -logic over asynchronous products of PDA is nonelementary.

*Acknowledgments.* The author thanks Shunichi Amano, Stefan Göller, Leonid Libkin, and anonymous reviewers for their helpful comments. The author is supported by EPSRC grant E005039 and ORSAS Award.

## References

1. Bekker, W., Goranko, V.: Symbolic model checking of tense logics on rational Kripke models. To appear in proceedings of ILC 2007 (2007)
2. Blumensath, A., Grädel, E.: Automatic structures. In: LICS 2000, pp. 51–62 (2000)
3. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
4. Compton, K.J., Henson, C.W.: A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic* 48(1), 1–79 (1990)
5. Esparza, J., Kucera, A., Schwoon, S.: Model checking LTL with regular valuations for pushdown systems. *Inf. Comput.* 186(2), 355–376 (2003)
6. Etesami, K., Vardi, M.Y., Wilke, T.: First-order logic with two variables and unary temporal logic. *Inf. Comput.* 179(2), 279–295 (2002)

7. Ferrante, J., Rackoff, C.W.: The Computational Complexity of Logical Theories, vol. 718. Springer, Heidelberg (1979)
8. Göller, S., Mayr, R., To, A.W.: On the computational complexity of verifying one-counter processes. To appear in LICS 2009 (2009)
9. Grohe, M., Schweikardt, N.: The succinctness of first-order logic on linear orders. Logical Methods in Computer Science 1(1) (2005)
10. Jancar, P., Sawa, Z.: A note on emptiness for alternating finite automata with a one-letter alphabet. Inf. Process. Lett. 104(5), 164–167 (2007)
11. Jančar, P., Kučera, A., Moller, F., Sawa, Z.: DP lower bounds for equivalence-checking and model-checking of one-counter automata. Inf. Comput. 188(1), 1–19 (2004)
12. Kozen, D.C.: Theory of Computation. Springer, Heidelberg (2006)
13. Krentel, M.W.: The complexity of optimization problems. J. Comput. Syst. Sci. 36(3), 490–509 (1988)
14. Kučera, A.: Efficient verification algorithms for one-counter processes. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 317–328. Springer, Heidelberg (2000)
15. Libkin, L.: Elements Of Finite Model Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2004)
16. Löding, C.: Reachability problems on regular ground tree rewriting graphs. Theory Comput. Syst. 39(2), 347–383 (2006)
17. Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught Theorem. Ann. Pure Appl. Logic 126(1-3), 159–213 (2004)
18. Mayr, R.: Decidability of model checking with the temporal logic EF. Theor. Comput. Sci. 256(1-2), 31–62 (2001)
19. Morvan, C.: On rational graphs. In: Tiuryn, J. (ed.) FOSSACS 2000. LNCS, vol. 1784, pp. 252–266. Springer, Heidelberg (2000)
20. Muller, D.E., Schupp, P.E.: The theory of ends, pushdown automata, and second-order logic. Theor. Comput. Sci. 37, 51–75 (1985)
21. Péladéau, P.: Logically defined subsets of  $N^k$ . Theor. Comput. Sci. 93(2), 169–183 (1992)
22. Rabinovich, A.: On compositionality and its limitations. ACM Trans. Comput. Logic 8(1), 4 (2007)
23. Serre, O.: Parity games played on transition graphs of one-counter processes. In: Aceto, L., Ingólfssdóttir, A. (eds.) FOSSACS 2006. LNCS, vol. 3921, pp. 337–351. Springer, Heidelberg (2006)
24. Stockmeyer, L.J.: The complexity of decision problems in automata theory and logic. PhD thesis, Department of Electrical Engineering, MIT (1974)
25. Thomas, W.: Constructing infinite graphs with a decidable MSO-theory. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 113–124. Springer, Heidelberg (2003)
26. To, A.W., Libkin, L.: Recurrent reachability analysis in regular model checking. In: Cervesato, I., Veith, H., Voronkov, A. (eds.) LPAR 2008. LNCS (LNAI), vol. 5330, pp. 198–213. Springer, Heidelberg (2008)
27. Walukiewicz, I.: Model checking CTL properties of pushdown systems. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000. LNCS, vol. 1974, pp. 127–138. Springer, Heidelberg (2000)
28. Walukiewicz, I.: Pushdown processes: games and model-checking. Inf. Comput. 164(2), 234–263 (2001)
29. Wöhrle, S., Thomas, W.: Model checking synchronized products of infinite transition systems. Logical Methods in Computer Science 3(4) (2007)

# Confluence of Pure Differential Nets with Promotion

Paolo Tranquilli

Laboratoire PPS – Université Paris Diderot - Paris 7  
Case 7014 – 75205 Paris – France  
ptranqui@pps.jussieu.fr

**Abstract.** We study the confluence of Ehrhard and Regnier’s differential nets with exponential promotion, in a pure setting. Confluence fails with promotion and codereliction in absence of associativity of (co)contractions. We thus introduce it as a necessary equivalence, together with other optional ones. We then prove that pure differential nets are Church-Rosser modulo such equivalences. This result generalizes to linear logic regular proof nets, where the same notion of equivalence was already studied in the literature, but only with respect to the problem of normalization in a typed setting. Our proof uses a result of finiteness of developments, which in this setting is given by strong normalization when blocking a suitable notion of “new” cuts.

## 1 Introduction

The inception of Linear Logic (LL, [1]) in the 80’s has reinforced the bridge between logic and computer science already established by the Curry-Howard correspondence years before. LL is in fact a refinement of intuitionistic and classical logic brought forth by a fine semantical analysis. One of its main features is the introduction of two dual modalities, the *exponentials* ! and ?, regulating the use of structural rules (*weakening* and *contraction*), which on the program side correspond to erasure and duplication of resources.

This endeavour, among other things, led the way to a new, parallel syntax of proofs, *proof nets*. These are the syntax of choice for LL, especially when considering cut elimination. In fact one of the main advances of LL over classical logic is that, though preserving an involutive negation (and therefore two-sided sequents), it also preserves properties of intuitionistic logic lacking in the classical framework. One of these, central to our work, is confluence of cut elimination, i.e. the independence of the result of the cut elimination procedure with respect to the actual cuts one decides to reduce.

A further semantical analysis led by Ehrhard [2] has recently provided LL with new models based on topological vector spaces where we can take the derivative of an object. The efforts of the same author and Regnier have permitted to lift such operations to syntax, giving rise to Differential Linear Logic (DiLL, [3]), and their syntax, *differential nets*. Three new rules are introduced to handle the !-modality (*coweakening*, *cocontraction* and *codereliction*) which are duals to the LL rules handling ?. In the proofs as programs paradigm, codereliction allows to introduce depletable resources, which may be asked for many times but may be used just one time, nondeterministically choosing which query they satisfy. This feature configures differential nets as a promising logical framework to extend the Curry-Howard correspondence to nondeterminism and concurrency (see [4]).

Actually, [3] gives the syntax for the promotion free fragment of DiLL only, giving rise to *differential interaction nets*, a nondeterministic example of Lafont’s interaction nets [5]. By modelling nondeterminism by formal sums confluence remains an important property, which is however straightforward in an interaction net paradigm, where no reduction can change the other ones. Here we will extend such property to the whole of differential nets. Promotion in proof nets is handled by *boxes*, synchronized areas of proofs enabling to mark what is to be erased or duplicated. Boxes break the interaction net paradigm: there are cuts (the *commutative* ones) which can be changed by other reductions, so confluence is definitely more delicate.

Part of a previous work of ours [6] was focused on proving confluence for the intuitionistic fragment, which used the recursive types needed to translate  $\lambda$ -calculus. There we observed that confluence fails without keeping into account some semantically grounded equivalences, namely associativity of contractions and cocontractions. A fully quotienting syntax as the one used in [7] for LL is seemingly out of reach in DiLL. Our solution in [6] was employing generalized (co)contraction cells in the style of [8], and some additional reductions.

Here we generalize the result in three ways. By concentrating on the computational contents rather than the logical one, we consider *pure* nets, where types (i.e. formulae) play no role whatsoever, not even recursive ones. Furtherly, the needed equivalences are settled to the maximum extent by means of . . . equivalences on nets. We thus generalize the equivalences and reductions of [9], providing as a byproduct the first proof of confluence<sup>1</sup> for such LL proof nets with equivalences in the completely pure case, as previous works concentrated on normalization in the typed one. Finally, we are able to introduce one more equivalence potentially giving the right to always consider boxes without sums inside (the *bang sum* equivalence).

This result has several ramifications. As is evident in [10], this is the first step in proving strong normalization in the typed case<sup>2</sup>. Furtherly, as can be deduced from [9], this can be the ground for new work on calculi with explicit substitutions: whether by extending some results to untyped calculi; or by considering explicit substitutions for nondeterministic calculi akin to Boudol’s  $\lambda$ -calculus with resources (see [6]).

Our technique, reminiscent of the work done on LL in [10], uses a finite development theorem used to prove a strong confluence property of a suitable notion of parallel reduction.

### 1.1 Rewriting Theory Modulo Equivalence

The aim of this section is making the reader acquainted with the notion of rewriting modulo equivalence, to the extent needed for our purposes. We refer to [11] Section 14.3] for more in-depth details and proofs.

Let  $(S, \rightarrow)$  be an abstract reduction system and let  $\sim$  be an equivalence relation on  $S$ . As usual,  $\overset{\rightarrow}{\Rightarrow}$  and  $\overset{*}{\rightarrow}$  denote the reflexive and reflexive-transitive closures of  $\rightarrow$  respectively. Take a symmetric relation  $\vdash$  such that  $\vdash^* = \sim$ , possibly  $\sim$  itself. Let  $s \vee t$  ( $t$  and  $s$  are **joinable modulo**  $\sim$ ) if  $s \overset{*}{\rightarrow} \sim \overset{*}{\leftarrow} t$ . We say then that  $\rightarrow$  is

<sup>1</sup> To be precise, the stronger result of being Church-Rosser modulo (see Section [11]).

<sup>2</sup> Actually the subject of a future submission by the author and Pagani.

- locally confluent modulo  $\sim$  if  $\leftarrow \rightarrow \subseteq \underline{\forall}$ ;
- confluent modulo  $\sim$  if  $\overset{*}{\leftarrow} \sim \overset{*}{\rightarrow} \subseteq \underline{\forall}$ ;
- locally coherent with  $\vdash$  if  $\vdash \rightarrow \subseteq \underline{\forall}$ ;
- Church-Rosser modulo  $\sim$  (or  $\text{CR}\sim$ ) if  $\approx \subseteq \underline{\forall}$ , where  $\approx := (\rightarrow \cup \leftarrow \cup \sim)^*$ ;
- strongly normalizing modulo  $\sim$  (or  $\text{SN}\sim$ ) if  $\rightarrow$  is  $\text{SN}$ , where  $\overset{\circ}{\rightarrow} := \sim \rightarrow \sim$ ;
- strongly Church-Rosser modulo  $\sim$  if  $\sim \overset{\circ}{\leftarrow} \overset{\circ}{\rightarrow} \subseteq \overset{\circ}{\rightarrow} \sim \overset{\circ}{\leftarrow}$ ;

The last definition is our terminology, while the rest follows [11]. Being Church-Rosser modulo  $\sim$  is the most important property of all those concerning confluence. In particular it implies the unique normal form modulo  $\sim$  property, ( $\approx = \sim$  on normal forms), which again implies that in order to compute the normal form one can use just regular reductions, without ever be forced to  $\sim$ -convert in order to get the result<sup>4</sup>. Contrary to what happens in regular reduction,  $\text{CR}\sim$  is strictly stronger than plain confluence in absence of  $\text{WN}$  [11, Remarks 14.3.6, Exercise 14.3.7]. Following are some important lemmas: the first is a generalization of Newman’s Lemma, the last is a trivial result we did not find in the literature which we will need in our proof.

**Lemma 1 (Huet).** *If  $\rightarrow$  is  $\text{SN}\sim$ , locally confluent modulo  $\sim$  and locally coherent with  $\vdash$ , then it is  $\text{CR}\sim$ .*

**Lemma 2 (van Oostrom).**  *$\rightarrow$  is  $\text{CR}\sim$  iff  $\overset{*}{\rightarrow}$  is strongly  $\text{CR}\sim$ .*

**Lemma 3.** *If  $\rightarrow$  is strongly  $\text{CR}\sim$ , then it is  $\text{CR}\sim$ .*

*Proof.* Straightforward induction: to show that  $\sim \overset{*}{\leftarrow} \overset{*}{\rightarrow} \sim$  is joinable, we proceed by induction first on one side, then on the other.

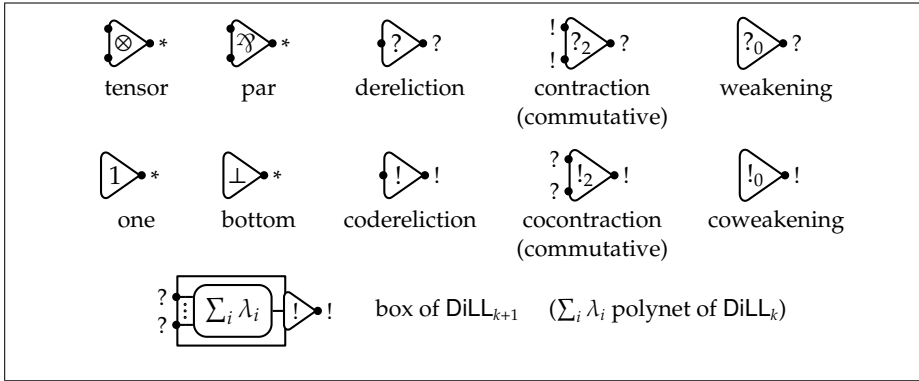
## 2 The System

A **net** is intuitively a network of **cells** linked by **wires** connecting their **ports**. A little more formally, a net  $\pi$  is given by the following data.

- A set  $\mathfrak{p}(\pi)$  of ports.
- A set  $\mathfrak{c}(\pi)$  of cells; to each cell  $c$  is assigned a **symbol**  $\sigma(c)$  in a given alphabet, a port in  $\mathfrak{p}(\pi)$  called **principal**, and a number of other, **auxiliary** ones. How the latter are treated distinguishes between two kinds of cells: in non commutative ones, auxiliary ports are a finite sequence, in **commutative** ones they form a finite set. Every port in  $\mathfrak{p}(\pi)$  can be associated with at most one cell; a port associated with a cell is called **connected**, otherwise it is **free**. Free ports (also called conclusions) are denoted by  $\mathfrak{fp}(\pi)$ . The number of auxiliary ports is determined by the symbol  $\sigma(c)$ .
- A set  $\mathfrak{w}(\pi)$  of wires, which can be either unordered pairs  $\{p, q\}$  of ports, or **deadlocks**, i.e. wires not connecting any port (intuitively short circuited wires). Each port is in exactly one wire. A **directed wire** is an ordered pair  $(p, q)$  such that  $\{p, q\}$  is a wire. **Terminal wires** are the directed ones going to the free ports.

<sup>3</sup> Here like in the rest of the paper,  $:=$  means “defined as”.

<sup>4</sup> This also means there is never the need to perform conversion steps in order to *ready* some redexes, i.e. make them visible.



**Fig. 1.** The cells of differential nets. The labels in  $\{!, ?, *\}$  assigned to ports will be needed only later (see page 507).

An **elementary path** in  $\pi$  is one in the graph trivially obtained by taking cells and free ports as nodes and directed wires as edges, which moreover does not intersect itself<sup>5</sup>. A **polynet** is a formal sum of nets, or equivalently a multiset of nets, all sharing the same free ports. At times we distinguish nets (thus singletons) from polynets by calling them **simple**.

### 2.1 Statics

$\text{DiLL}_0$  nets and polynets are built from all the symbols in Figure 1 but the box one. These are exactly the differential interaction nets presented in [3]. For the moment let us ignore the labels we assign to the ports in the figure, which will be needed only later (see page 507). As usual, the apex of the cell represents the principal port, while the auxiliary ones are depicted on the opposite side.

In order to add boxes, one proceeds by induction, by considering them as cells having a whole polynet as symbol. Let  $\text{DiLL}_{k+1}$  nets and polynets be the ones built from all the cells of Figure 1 where for each box its symbol is a polynet  $\pi$  in  $\text{DiLL}_k$  and there is a bijection between its ports and  $\text{fp}(\pi)$ . The symbol  $\sigma(B)$  of a box  $B$  is also called its **contents**. We will denote by  $! \pi$  a generic box having  $\pi$  as contents. A  $\text{DiLL}$  polynet  $\pi$  is one of  $\text{DiLL}_k$  for any  $k$ ; if such  $k$  is minimal, we say that  $k$  is the depth of  $\pi$  (in fact, the maximal number of nested boxes). A port is **active** if it is either a principal one, or an auxiliary one of a box. A wire linking two active ports is a **cut**.

Figures 5 and 6 will show examples of differential nets. The explicit marking of ports is dropped as they can always be identified with the extremities of wires.

Let  $p_i(\pi)$ ,  $\text{fp}_i(\pi)$ ,  $c_i(\pi)$  and  $w_i(\pi)$  be the set of all occurrences of ports, free ports, cells and wires respectively occurring in  $\pi$ , including in all the contents of the boxes in  $\pi$ . We can slice those sets by depth, so we will denote by  $p_i(\pi)$ ,  $\text{fp}_i(\pi)$ ,  $c_i(\pi)$  and  $w_i(\pi)$  the

<sup>5</sup> Technically, one prohibits the repetition of unoriented wires and that three ports of the same cell be crossed by the path.

corresponding elements of the nets contained in  $i$  nested boxes, where  $i$  is called the depth of the element in  $\pi$ <sup>6</sup>.

**Correctness Criterion.** As usual, the nets blindly built with the cells available are not in general “correct”, where the word can take the meaning of unsequentializable in sequent calculus, or having deranged computational behaviour. Since [12] one of the most used correctness criteria for proof nets is that of switching acyclicity. Given a DiLL net, a **switching path** is an elementary one which does not traverse two auxiliary ports of any  $\mathfrak{N}$  or contraction cell (does not bounce “above” it). A DiLL polynet is called a DiLL **proof net** (or differential proof net) if it is switching acyclic, i.e. it has no deadlocks nor switching cycles, and inductively all box contents are also switching acyclic. From now on we will deal almost only with proof nets.

### 2.2 Dynamics

As with various calculi, the reduction of differential nets can be defined as the context closure of a set of reduction rules, presented as pairs of redexes and contracta. A **linear context**  $\delta[\ ]$  is a simple net  $\delta$  together with a subset  $H_\delta$  of its free ports (the **hole** of  $\delta[\ ]$ ). It is linear as it is not a sum and the hole is not inside a box. Given a simple net  $\lambda$  and a bijection  $\sigma$  between  $H_\delta$  and  $\text{fp}(\lambda)$ , the **plugging**  $\delta[\lambda]$  of a simple net  $\lambda$  in the hole of  $\delta[\ ]$  amounts to identifying the ports according to  $\sigma$  and welding the wires that come together in this way<sup>7</sup>, as shown in Figure 2. This definition is then extended by linearity when we plug a polynet, by setting  $\delta[\sum_i \lambda_i] := \sum_i \delta[\lambda_i]$ .

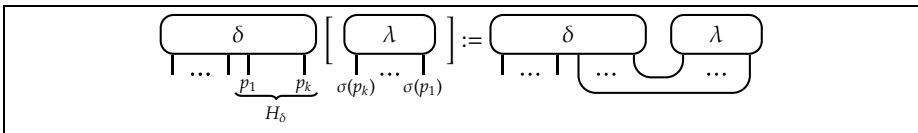


Fig. 2. Plugging of a net in a context

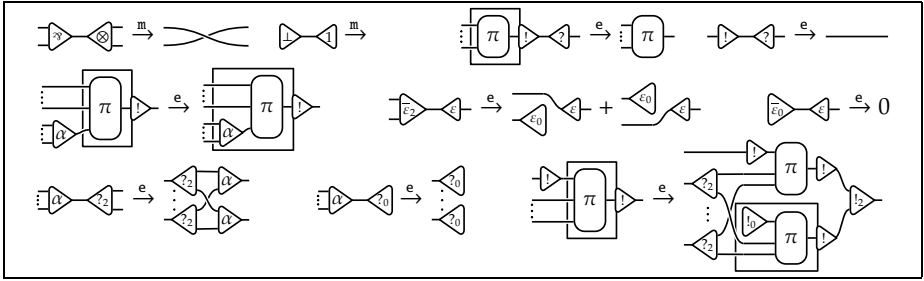
Finally, **contexts** generalize the concept in the following way. A linear context  $\delta[\ ]$  is a context; furthermore if  $\omega[\ ]$  is a context then  $\delta[\omega[\ ]]$  for  $\delta[\ ]$  linear<sup>8</sup>,  $\omega[\ ] + \pi$  for  $\pi$  polynet and  $!\omega[\ ]$  (i.e. a box containing a context) are also contexts. Plugging is easily extended to all contexts. The **context closure**  $\tilde{R}$  of a relation  $R$  is then defined by  $\pi \tilde{R} \sigma$  iff  $\pi = \omega[\lambda]$ ,  $\sigma = \omega[\mu]$  and  $\lambda R \mu$ .

We are now able to define multiplicative reduction  $\xrightarrow{m}$  and the exponential one  $\xrightarrow{e}$  by context closure of the rules of Figure 3, which are pairs consisting of a simple net (the **redex**) and a polynet (the **contractum**). Each redex here is identified by a unique cut. The union  $\xrightarrow{me}$  of the two reduction is the cut elimination of DiLL.

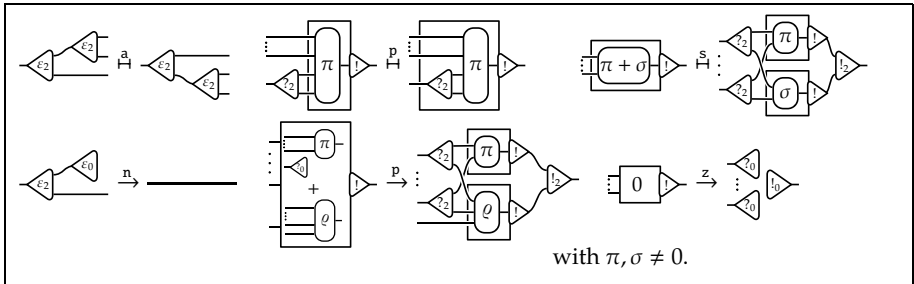
<sup>6</sup> All of this can be defined more formally by an inductive definition. Nevertheless we leave it to the reader as an easy exercise.

<sup>7</sup> For the quite delicate technical details the reader is referred to [13].

<sup>8</sup> Composition of contexts should be defined, but it is trivial once plain plugging is defined.



**Fig. 3.** The multiplicative and exponential reduction rules of DiLL.  $\varepsilon$  and  $\bar{\varepsilon}$  denote either  $?$  and  $!$  or vice versa.  $\alpha$  denotes any symbol among  $!_2$ ,  $!_0$  or a box symbol  $\pi$ . In particular weakening against coweakening reduces to the empty net. We make implicit use of the rule for context plugging of sums: the  $\pi$  inside boxes is a polynet. For example the dereliction on box rule may introduce sums.

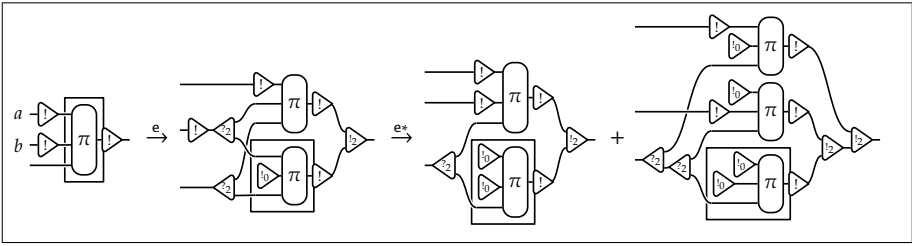


**Fig. 4.** Top: the rules for associative equivalence  $\stackrel{a}{\sim}$ , the push one  $\stackrel{p}{\sim}$  and the bang sum one  $\stackrel{s}{\sim}$ ;  $\dashv$  denotes a one-step conversion. Bottom: the rules for neutral reduction  $\stackrel{n}{\rightarrow}$ , the pull one  $\stackrel{p}{\rightarrow}$  and the bang zero one  $\stackrel{z}{\rightarrow}$ . The condition  $\pi, \sigma \neq 0$  applies to all rules.

**2.3 Equivalences and Canonical Reductions**

As we will show as a remark at page 506, the reductions just presented fail to give a confluent system: we cannot ignore associativity of (co)contractions and neutrality of (co)weakening over (co)contraction. This prompts us to introduce the former as an equivalence and the latter as a reduction. As we need anyway to consider reduction modulo an equivalence, we also study other equivalences (backed by semantical and observational equivalence) which are optional though must be taken together. Each equivalence is accompanied by a reduction which in a sense settles a zeroary case of the equivalence. Reversing each of these gives unwanted looping reductions. The **associative**, **push** and **bang sum** equivalences, together with the **neutral**, **pull** and **bang zero** reductions (which *do not* reduce cuts), are shown in Figure 4. The  $\pi, \sigma \neq 0$  condition is needed, lest one would be able to spawn trees of contractions from nothing, giving looping reductions.





**Fig. 5.** Reduction of a box with two coderelictions on it. Starting with codereliction  $b$  swaps the two linear copies of  $!\pi$  and therefore both the cocontraction and contraction trees in the last addend.

The push equivalence<sup>9</sup> has already been studied in the literature on proof nets and explicit substitutions [8,9]. The pull reduction may seem somewhat complicated, however it is a generalization of the reduction pulling out weakenings from boxes [9]. The usual reduction can be reobtained when having  $\varrho = 0$ , which by means of a  $z$ -reduction and some  $n$  ones gives the expected result. Such form (which in fact contains a sort of on-the-fly  $s$ -conversion) is required in order to get local coherence<sup>10</sup>.

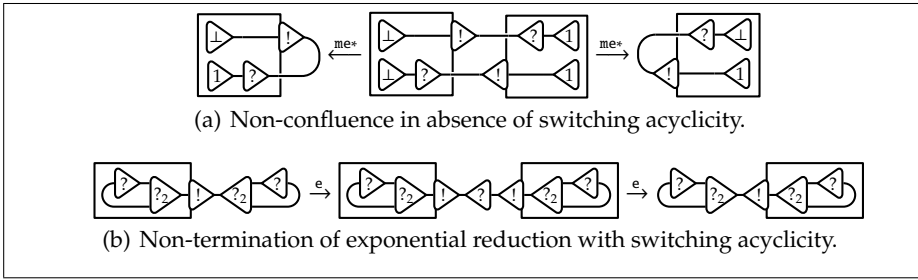
The part about sums inside boxes was already known to be valid semantically and observationally: we give here some syntactic ground to using it. From the point of view of semantics it is interesting to note that it implements the well known exponential isomorphism  $!A \otimes !B \cong !(A \& B)$  from linear logic (see [2]).

From now on  $\sim$  and  $\overset{c}{\rightarrow}$  (**canonical** reduction) will denote either  $\overset{a}{\rightarrow}$  and  $\overset{n}{\rightarrow}$  or the union of the  $a$ ,  $p$  and  $s$  conversions and the  $npz$ -reduction respectively. By checking the cases not already proved in the literature, one gets the following.

**Proposition 4 (stability of correctness).** *If  $\pi$  is switching acyclic and  $\pi \overset{mec}{\rightarrow} \pi'$  or  $\pi \sim \pi'$  then  $\pi'$  is switching acyclic also.*

*Examples and remarks.* Figure 5 gives the reason to employ associative equivalence, showing the reduction of the coderelictions on box critical peak, which cannot be joined with regular reductions. One reduction only is shown, as the other is symmetric. Other critical peaks due to the codereliction on box rule (namely when against dereliction and contraction) show that also the  $n$ -reductions cannot be left out. One can already see two big differences with respect to LL and the work done with it in [10]: firstly, sums may arise even without the “logical” step of dereliction on box; moreover, the codereliction on box rule, which reduces a commutative cut, changes the possible cuts on *all* other cuts of the box. These problems prevent an immediate adaptation of the measures used in [10]. The nets in Figure 6 are examples already known in LL showing issues about correctness and types.

<sup>9</sup> Though an equivalence is not directed, the name comes from [14] and [6] where it was a reduction. We felt like keeping it for its good pairing with the pull reduction.  
<sup>10</sup> The problem arises in a  $p$ -equivalence and  $n$ -reduction critical peak, as the latter may eliminate a contraction of *one* of the addends in the box.



**Fig. 6.** Issues with confluence. Figure 6(a) shows the need for correctness. The example shown is even simply typed. Figure 6(b) shows how in the pure case even the exponential reduction alone is not terminating.

### 3 The Finite Development Theorem

In [15], Danos proved the counterpart of the finite development theorem for MELL, and Pagani and Tortora de Falco did the same for the whole of second order LL in [10]. In this setting the actual definition of what a “new” redex is gets more technical.

#### 3.1 Marking New Cuts

We define a notion of new and old cuts, by leaving a mark on the new ones. **Marks** are cells of a new symbol with two ports and no reduction rule, graphically depicted by little circles. Its main purpose is to block reductions and equivalences (for example a mark between two contractions blocks the  $a$ -conversion).

Ideally, these marks are placed during reduction to block “new” wires. By new we mean two kinds of wires: those that in a typed setting would decrease the logical complexity of the cut formula, and those that before the reduction were exponential clashes. The latter are peculiar to a truly untyped setting, and are brought by the opening box and neutral reductions, which erase an exponential port. For example, if we erase marks from the net shown in Figure 8 and we fire the dereliction against box redex we end up with a valid multiplicative cut which was a clash before. Rather than lock this special kind of “new” wires during reduction, we can lock all potentially dangerous clashes since the start, as markings will prevent new clashes from arising. We thus need to define what an exponential clash is.

Let  $\tau$  be the partial function from  $p_i(\pi)$  to the labels  $\{!, ?, *\}$  thus defined. On ports of cells it gives the values already shown in Figure 1; on the ports of marks it is undefined; for  $p \in fp_i(\pi)$  we set  $\tau(p) = ?$  if  $p$  is over an auxiliary port of a box, we leave it undefined otherwise.  $\tau$  provides for a sort of pre-typing. A directed wire  $(p, q)$  is called a **!-wire** (resp. **?-wire**) if  $\tau(p) = ?$  and  $\tau(q) = !$  (resp. vice versa), where however we let at most one of the two be undefined. In any case **!** and **?**-wires are called **exponential** (which applies to undirected wires also). An **exponential clash** (simply clash from now

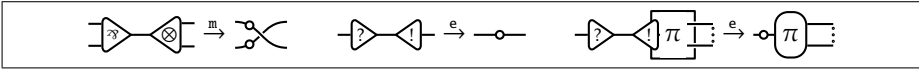


Fig. 7. The modified reduction rules of  $\text{DiLL}^\circ$

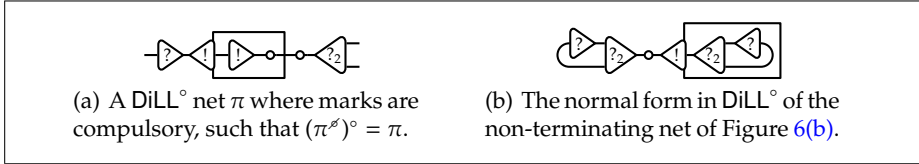


Fig. 8. Examples for  $\text{DiLL}^\circ$

on) is a wire  $\{p, q\}$  such that one of  $\tau(p)$ ,  $\tau(q)$  is ! (resp. ?) and the other is defined but not ? (resp. !)<sup>11</sup>.

$\text{DiLL}^\circ$  is the system given by polynets with marks and without clashes, and by changing some rules to introduce the mark as depicted in Figure 7. It is immediate to see that the absence of clashes is preserved by reduction, as the new wires which could bring close unmatched ports are interrupted by marks. From the point of view of  $\text{DiLL}$ , clash-freeness imposes just that some marks be added: given any  $\text{DiLL}$  polynet  $\pi$ , we define the injection  $\pi^\circ$  in  $\text{DiLL}^\circ$  by placing a mark interrupting each clash. Conversely,  $\text{DiLL}^\circ$  can be clearly surjected on  $\text{DiLL}$  by erasing all marks. We call this surjection  $\pi^\sigma$ . The net  $\pi$  in Figure 8(a) is an example of  $\text{DiLL}^\circ$  net enjoying  $(\pi^\sigma)^\circ = \pi$ . On the other hand, if  $\sigma$  is the net in Figure 6(b), then  $\sigma^\circ = \sigma$  and it is strongly normalizing to the net in Figure 8(b).

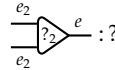
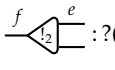
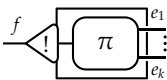
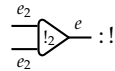
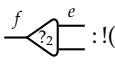
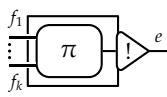

### 3.2 Measuring Exponential Reduction

Ideally, we may regard exponential reduction as a procedure that “slides” cells along exponential paths in the net, with ! and ? cells sliding in opposite direction. We thus assign to each cut a natural number, indicating how far the two cells around it are from the end of the path they are sliding on. After a reduction however many cuts may have arisen. So we will employ the multiset of the weights of the cuts and the multiset order<sup>12</sup>. Global additive duplication poses another problem. In [6] we settled it by employing multisets of multisets. Here however we estimate how many addends can sprout during reduction, so we can use this value and count each cut as many times as there can be addends containing it. We will also need to get an estimate of the number of copies (both regular and linear) of a box.

<sup>11</sup> More intuitively: !-wires and ?-wires are those that in a typing attempt would get an outermost ! or ? respectively, while clashes would give a failure to unify an outermost exponential modality.

<sup>12</sup> Multisets (here presented as  $[a_1, \dots, a_k]$  with additive notation) over a well founded set are well ordered by the transitive closure of  $A + [a] > A + B$  with  $\forall b \in B : b < a$ .

**Table 1.** Rules for the  $?$ -weight (top), the  $!$  one (middle) and the spread  $\text{sp}$  (bottom). In the spread formula (which ranges over derelictions and coderelictions at depth 0) we use the notation  $?(c)$  and  $!(c)$  for the corresponding measure on the wire from the principal port of  $c$ .

 $ : ?(e) = ?(e_1) + ?(e_2);$	 $ : ?(e) = ?(f);$
 $ : ?(e_i) = \begin{cases} ?(f)(1 + \sum_{j=1}^k !(e_j)) & \text{if } \pi = 0, \\ ?(f)(1 + \sum_{j=1}^k !(e_j)) \sum_{\lambda \in \pi} \text{sp}(\lambda) ?(e_i^\lambda) & \text{otherwise;} \end{cases}$	
otherwise $ : ?(e)$ is a variable if $e$ is terminal at depth 0, $?(e) = 1$ otherwise.	
 $ : !(e) = !(e_1) + !(e_2);$	 $ : !(e) = !(f);$
 $ : !(e) = \begin{cases} 1 + \sum_{j=1}^k !(f_j) & \text{if } \pi = 0, \\ (1 + \sum_{j=1}^k !(f_j)) \sum_{\lambda \in \pi} \text{sp}(\lambda) & \text{otherwise.} \end{cases}$	
 $ : !(e) = !(f)$ if $p \in \text{fp}_0(\sigma(B))$ for a box $B$ , $p$ is above an auxiliary port, and $f$ is the wire corresponding to $p$ outside the box,	
otherwise $ : !(e)$ is a variable if $e$ is terminal at depth 0, $!(e) = 1$ otherwise.	
$\text{sp}(\lambda) = \prod_{\substack{c \in \text{CQ}(\lambda) \\ \sigma(c) = ?}} !(c) \cdot \prod_{\substack{c \in \text{CQ}(\lambda) \\ \sigma(c) = !}} ?(c)$	

**Exponential Paths.** An **!-path** (resp. **?-path**) is an elementary path made only of  $!$ -wires (resp.  $?$ -wires), not traversing any mark, dereliction or codereliction (though it may end on them). In either case, the path is called **exponential**. All cells internal to an exponential path must necessarily be contractions, cocontractions or boxes. The main technical advantage of  $\text{DiLL}^\circ$  over  $\text{DiLL}$  is that in it no reduction can open new exponential paths.

Next we define by mutual induction three basic measures on which we will base the measure of the whole net. Two of them, the **?-weight**  $?(e)$  and the **!-weight**  $!(e)$ , are on wires. The third, the **spread**  $\text{sp}(\lambda)$ , is defined on simple subnets of the given net<sup>13</sup>. For the purpose of working modularly with the measures, we introduce *variables* on the terminal wires over  $\text{fp}_0(\pi)$ . We will thus consider *variables*  $!(d)$  (resp.  $?(d)$ ) with  $d$  a terminal wire.

*Weighting wires and estimating addends.* Table 1 provides the laws for  $?(e)$  (resp.  $!(e)$ ), giving them depending on an adjacent cell. By the absence of clashes there is no

<sup>13</sup> Without getting into details, a subnet of  $\pi$  is a properly formed net given by a subset of cells and wires of  $\pi$ , all taken from the same depth (but then including all the elements contained in its boxes).

ambiguity in the definitions. By  $e^\lambda$  we denote the wire corresponding to  $e$  inside a box, in the net  $\lambda$  of the box contents. At the bottom we also show the law for the spread. Notice that all measures are polynomials with natural coefficients. Notice also that there is a circular dependency between the three measures, so the next lemma is not trivial.

**Lemma 5.** *Given a DiLL<sup>o</sup> proof net  $\pi$ ,  $?(e)$ ,  $!(e)$  and  $\text{sp}(\lambda)$  are defined for all  $e \in \mathbf{w}_1(\pi)$  and all  $\lambda$  simple subnets of  $\pi$  at any depth.*

*Proof (sketch).* One proceeds by a primary induction on the depth: supposing all the (polynomial) measures have been defined inside all boxes, one can

- define  $!(e)$  by induction on the maximal length of  $?$ -paths starting from  $e$  (instantiating the variables of the  $?$ -conclusions inside boxes in the process);
- only then, define  $?(e)$  by induction on the maximal length of  $!$ -paths (relying also on the measures inside boxes just instantiated);
- finally define the spread from the two. □

*Weighting nets and polynets.* The **weight**  $|e|$  of a wire is  $?(e) + !(e)$ . Let  $!\mathbf{cw}_0(\lambda)$  (resp.  $\mathbf{b}_0(\lambda)$ ) be the set of exponential cuts (resp. boxes) at depth 0 of a simple net  $\lambda$ . Let us fix a polynet  $\pi$ , and let  $c(B)$  (the **count** of the box  $B$ ) denote  $?(e)(1 + \sum_j !(f_j))$  with  $e$  and  $f_j$  the wires on the principal and the auxiliary ports of  $B$  respectively. Then for each sum (i.e. multiset)  $\rho$  of subnets of  $\pi$  we define by induction on their depth the following multiset ( $\lambda$  will denote a generic simple subnet):

$$\|\rho\| := \sum_{\lambda \in \rho} \text{sp}(\lambda) \|\lambda\|, \quad \|\lambda\| := [ |e| \mid e \in !\mathbf{cw}_0(\lambda) ] + \sum_{B \in \mathbf{b}_0(\lambda)} c(B) \|\sigma(B)\|,$$

Finally, the polynomial measure of the whole net (a special case of the measure already given) can be instantiated with 1 for all variables to get an actual number.

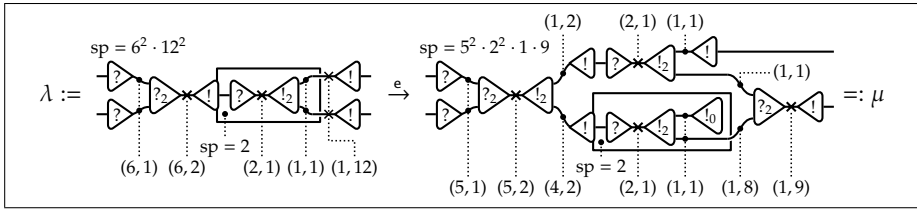
Notice that this measure depends monotonously from the weight of each part of the net. This intuition will be given a solid ground by Lemma 7.

*Intuitive ideas of the measures.* As already hinted at the begin of this section, these measures should help to estimate how far each cut has to go in both directions to arrive at a “dead end” (for DiLL<sup>o</sup>, the logical rules), and how many times each cut should be accounted for.

Morally  $!(e)$  measures the size of the tree of cocontractions above  $e$  (which is invariant under associativity). The most important feature is that it counts all the coderelictions linked to  $e$ . On boxes we count

- the  $!$ -weight on the auxiliary ports because the codereliction against box rule creates a contraction and a codereliction; plus one to count the box itself, especially if it has no auxiliary ports;
- multiplied by the spread of the contents in order to be invariant by  $s$ -conversion, and keep such invariants even if the sum inside... spreads.

Dually  $?(e)$  measures the size of the tree of contractions above  $e$ . The rule when  $e$  is on an auxiliary port of a box  $B$  contains:



**Fig. 9.** An example of calculated measures. Each relevant wire  $e$  is labelled by the pair  $(!(e), ?(e))$ . Cuts are specially marked.

- $?(f)$  because the contractions on the principal port of  $B$  may shift to auxiliary ports during reduction;
- the sum of  $!$ -weights of the auxiliary wires because codereliction against box creates contractions; plus 1 to provide something to decrease when a cut enters a box (box against box and those similar);
- the  $?$ -measures inside because either by opening the box or by  $p$ -conversion the contraction trees inside can pour outside; summed, to respect both  $p$ -conversion and  $s$ -conversion; this sum is weighted with the spread to prevent a reduction generating a sum inside from increasing such weight.

As already hinted,  $sp(\lambda)$  estimates how many addends may have a reduct of  $\lambda$ . This is achieved by morally multiplying all the possible number of choices potentially to be done in  $\lambda$ . Now sums arise

- on (co)dereliction against co(co)ntraction reductions, so the size of the tree of co(co)nteractions on the principal port of a (co)dereliction should estimate what choices that (co)dereliction may do;
- on (co)dereliction against box rules, when the box contains an actual sum; however the spread of a box contents are already accounted for in both the  $!$ -weight and the  $?$ -weight.

Finally the cuts inside a box  $B$  count  $c(B)$  times as this number estimates how many regular and linear copies of its contents may be done, and all cuts count  $sp(\lambda)$  times to account for additive duplication.

Before sketching the proofs, we show in Figure 9 an example of reduction step where we have calculated (in the way indicated by Lemma 5) all the relevant measures of the two nets. It turns out that  $\|\lambda\| = 5184(12[3]+[8, 13, 13])$ , while  $\|\mu\| = 900(9[3]+[7, 10])$ , which is indeed lower (though in a quite coarse way).

**Replacement and Modularity Lemmas.** In the following, we will consider the extensional (i.e. pointwise) order  $\leq$  on non-zero values for all the polynomials.

For different simple nets  $\lambda, \mu$ , we distinguish the weights calculated on one or the other by putting them as superscripts, as in  $?^\lambda(e)$ . Suppose  $\lambda$  and  $\mu$  are two nets with identified terminal wires  $C$ . We say that  $\lambda$  can replace  $\mu$  if for each terminal wire  $d$  we

have that  $!^\lambda(d) \leq !^\mu(d)$  and  $?^\lambda(d) \leq ?^\mu(d)$  (one of the comparisons may be a trivial one among the same variables). An induction on the context reveals the following lemma.

**Lemma 6 (replacement).** *Suppose  $\lambda$  can replace  $\mu$ ,  $\omega[\ ]$  is a linear context with  $\omega[\lambda]$  and  $\omega[\mu]$  proof nets. Then for each  $e$  wire in the context  $\omega$ ,  $?^{\omega[\mu]}(e) \leq ?^{\omega[\lambda]}(e)$  and  $!^{\omega[\mu]}(e) \leq !^{\omega[\lambda]}(e)$ .*

In the following the weight  $|D|$  of a set of wires  $D$  is the multiset of the weights of its wires. A terminal wire is **dormant** if it connects an active port or two free ones. Dormant wires are those that can become cuts when glued in a context. The proof of the following lemma, which we omit, is an induction on the depth of the hole in the context.

**Lemma 7 (modularity).** *Take  $\lambda$  and  $\mu_1, \dots, \mu_n$  simple nets and  $\omega[\ ]$  a context such that  $\omega[\lambda]$  and  $\omega[\mu_i]$  are all  $\text{DiLL}^\circ$  pure proof nets. Suppose moreover that:*

- for every  $i$  we have that  $\mu_i$  can replace  $\lambda$ ;
- $\sum_i \text{sp}(\mu_i) \leq \text{sp}(\lambda)$ ;
- if  $n = 0$ , then  $\|\lambda\| > [\ ]$ , otherwise for every  $i$  we have  $\|\mu_i\| + |D_i|^{\mu_i} < \|\lambda\|$ , (resp.  $\leq$ ) where  $D_i$  is the set of active wires in  $\mu_i$  that are not dormant in  $\lambda$ .

*Then we have the pointwise inequality  $\|\omega[\sum_i \mu_i]\| < \|\omega[\lambda]\|$  (resp.  $\leq$ ).*

Thanks to modularity, the following result is up to mechanical checks which we omit altogether.

**Lemma 8.**  *$\|\cdot\|$  has the following properties.*

- if  $\pi \xrightarrow{e} \pi'$  then  $\|\pi'\| < \|\pi\|$ ;
- if  $\pi \xrightarrow{\text{mc}} \pi'$  then  $\|\pi'\| \leq \|\pi\|$ ;
- if  $\pi \sim \pi'$  then  $\|\pi'\| = \|\pi\|$ .

**Theorem 9 (finite developments).** *Reduction on  $\text{DiLL}^\circ$  is SN.*

*Proof (sketch).* Only **m** and **c** remain to be settled. For all reductions, the pair given by  $(\|\pi\|, \#_m(\pi) + \#_c(\pi))$  strictly decreases for lexicographic ordering, where  $\#_m$  just counts the multiplicative cells in  $\pi$ , and  $\#_c$  weights coweakenings, weakenings and boxes containing 0 in the following inductive way:

$$\#_c(\pi) := 1 + \#_{i_0}(\pi) + \#_{\gamma_0}(\pi) + \sum_{B \in \text{b}_0(\pi)} (1 + \text{deg}(B)) \#_c(\sigma(B))$$

where  $\#_{i_0}$  and  $\#_{\gamma_0}$  count coweakenings and weakenings at depth 0, and the degree  $\text{deg}(B)$  is the number of ports of  $B$ .  $\square$

## 4 Proving Confluence

Recall that  $\sim$  and **c** may be a-equivalence and n-reduction, or full asp-equivalence and nzp-reduction. Some of the diagrams show we cannot separate s-equivalence from the p one (and their associated reductions). Checking all local confluence and local coherence diagrams as indicated by Lemma [11](#), gives the following proposition, which then finally leads the way to the main theorem of the work.

**Proposition 10.** *Reduction in  $\text{DiLL}^\circ$  is  $\text{CR}\sim$ , and so are  $\mathfrak{m}$  and  $\text{ec}$  alone.*

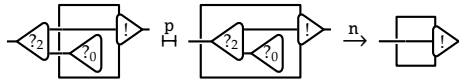
**Main Theorem.** *Reduction of  $\text{DiLL}$  pure proof nets is  $\text{CR}\sim$ , and so are  $\mathfrak{m}$  and  $\text{ec}$  alone.*

*Proof.* Using  $\text{DiLL}^\circ$  we define a parallel reduction  $\twoheadrightarrow$ . Let  $\pi \twoheadrightarrow \sigma$  iff  $\pi^\circ \xrightarrow{\text{mec}^*} \varrho$  in  $\text{DiLL}^\circ$  and  $\sigma = \varrho^\circ$ . Then

- $\xrightarrow{\text{mec}} \subseteq \twoheadrightarrow \subseteq \xrightarrow{\text{mec}^*}$ , so that  $\twoheadrightarrow^* = \xrightarrow{\text{mec}^*}$  (notice  $\pi^\circ$  cannot block any reduction);
- $\twoheadrightarrow$  is strongly  $\text{CR}\sim$  because  $\xrightarrow{\text{mec}^*}$  is so in  $\text{DiLL}^\circ$ : if  $\pi \twoheadrightarrow \sigma_1^\circ, \sigma_2^\circ$  with  $\pi^\circ \xrightarrow{\text{mec}^*} \sigma_1, \sigma_2$ , then  $\sigma_1, \sigma_2 \xrightarrow{\text{mec}^*} \rho$ , and then  $(\sigma_i^\circ)^\circ \xrightarrow{\text{mec}^*} \rho$ , because  $(\sigma_i^\circ)^\circ$  has less marks than  $\sigma_i$  as the latter is clash-free. In the end  $\sigma_1^\circ, \sigma_2^\circ \twoheadrightarrow \rho^\circ$ .

Then we conclude, as  $\twoheadrightarrow$  is  $\text{CR}\sim$  by Lemma 3 ( $\twoheadrightarrow$  is reflexive as  $(\pi^\circ)^\circ = \pi$ ), which means that  $\twoheadrightarrow^* = \xrightarrow{\text{mec}^*}$  is strongly  $\text{CR}\sim$ , which in turn by Lemma 2 gives Church-Rosser modulo  $\sim$  for the ordinary reduction 14. It is not hard to give parallel reductions for the  $\text{ec}$  and  $\mathfrak{m}$  ones and do the same.  $\square$

**A Conclusion: The Case for MELL.** Our system of reductions and equivalences bears close resemblance to the one developed for MELL in 9. In fact, stripping  $\text{DiLL}$  of all its differential features, the only difference is the absence in 9 of anything related to the  $\mathfrak{p}$ -reduction. In MELL the  $\mathfrak{p}$ -reduction is given by simply pulling out a weakening out of a box, which in  $\text{DiLL}$  can be done by a concatenation of  $\mathfrak{p}$ ,  $\mathfrak{z}$  and  $\mathfrak{n}$  reduction steps. In 16, the author calls such a variant a *total*  $\mathfrak{p}$ -reduction, which was here omitted because of its redundancy in  $\text{DiLL}$ . First, we argue that without such a step the  $\text{CR}\sim$  property is broken, as shown by the following coherence critical peak, leading to two normal forms which are not directly equivalent 15:



Then a direct consequence of the Main Theorem is the following, which may prove useful in the study of calculi with explicit substitutions.

**Theorem 11.** *MELL pure proof nets, with  $\mathfrak{a}$  and  $\mathfrak{p}$  equivalences together with  $\mathfrak{n}$  and total  $\mathfrak{p}$  reduction is  $\text{CR}\sim$ .*

## References

1. Girard, J.Y.: Linear logic. *Theor. Comput. Sci.* 50, 1–102 (1987)
2. Ehrhard, T.: Finiteness spaces. *Math. Structures Comput. Sci.* 15(4), 615–646 (2005)
3. Ehrhard, T., Regnier, L.: Differential interaction nets. *Theor. Comput. Sci.* 364(2), 166–195 (2006)

<sup>14</sup> Notice that we cannot infer  $\text{CR}\sim$  of  $\twoheadrightarrow$  directly from the same property in  $\text{DiLL}^\circ$ , as chained parallel reductions are not necessarily in  $\text{DiLL}^\circ$ .

<sup>15</sup> In fact, even the confluence modulo property does not hold, though probably there is confluence for  $\sim \rightarrow \sim$ .



4. Ehrhard, T., Laurent, O.: Interpreting a finitary pi-calculus in differential interaction nets. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 333–348. Springer, Heidelberg (2007)
5. Lafont, Y.: Interaction nets. In: POPL 1990, pp. 95–108. ACM, New York (1990)
6. Tranquilli, P.: Intuitionistic differential nets and lambda calculus. To appear on Theor. Comput. Sci. (2008)
7. Regnier, L.: Lambda-Calcul et Réseaux. Thèse de doctorat, Université Paris 7 (1992)
8. Di Cosmo, R., Guerrini, S.: Strong normalization of proof nets modulo structural congruences. In: Narendran, P., Rusinowitch, M. (eds.) RTA 1999. LNCS, vol. 1631, pp. 75–89. Springer, Heidelberg (1999)
9. Di Cosmo, R., Kesner, D., Polonovski, E.: Proof nets and explicit substitutions. Math. Structures Comput. Sci. 13(3), 409–450 (2003)
10. Pagani, M., Tortora de Falco, L.: Strong normalization property for second order linear logic. To appear on Theor. Comput. Sci. (2008)
11. Terese: Term Rewriting Systems. Cambridge Tracts in Theoretical Computer Science, vol. 55. Cambridge University Press, Cambridge (2003)
12. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Mathematical Logic 28, 181–203 (1989)
13. Vaux, L.:  $\lambda$ -calcul différentiel et logique classique: interactions calculatoires. Thèse de doctorat, Université de la Méditerranée (2007)
14. Di Cosmo, R., Kesner, D.: Strong normalization of explicit substitutions via cut elimination in proof nets. In: LICS, p. 35. IEEE Computer Society Press, Los Alamitos (1997)
15. Danos, V.: La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalement du  $\lambda$ -calcul). Thèse de doctorat, Université Paris 7 (1990)
16. Tranquilli, P.: Nets between determinism and nondeterminism. Ph.D. thesis, Università Roma Tre/Université Paris Diderot, Paris 7 (April 2009)

# Decision Problems for Nash Equilibria in Stochastic Games<sup>\*</sup>

Michael Ummels<sup>1</sup> and Dominik Wojtczak<sup>2</sup>

<sup>1</sup> RWTH Aachen University, Germany  
ummels@logic.rwth-aachen.de

<sup>2</sup> CWI, Amsterdam, The Netherlands  
d.k.wojtczak@cwi.nl

**Abstract.** We analyse the computational complexity of finding Nash equilibria in stochastic multiplayer games with  $\omega$ -regular objectives. While the existence of an equilibrium whose payoff falls into a certain interval may be undecidable, we single out several decidable restrictions of the problem. First, restricting the search space to stationary, or pure stationary, equilibria results in problems that are typically contained in PSPACE and NP, respectively. Second, we show that the existence of an equilibrium with a binary payoff (i.e. an equilibrium where each player either wins or loses with probability 1) is decidable. We also establish that the existence of a Nash equilibrium with a certain binary payoff entails the existence of an equilibrium with the same payoff in pure, finite-state strategies.

## 1 Introduction

We study *stochastic games* [21] played by multiple players on a finite, directed graph. Intuitively, a play of such a game evolves by moving a token along edges of the graph: Each vertex of the graph is either controlled by one of the players, or it is *stochastic*. Whenever the token arrives at a non-stochastic vertex, the player who controls this vertex must move the token to a successor vertex; when the token arrives at a stochastic vertex, a fixed probability distribution determines the next vertex. A measurable function maps plays to payoffs. In the simplest case, which we discuss here, the possible payoffs of a single play are binary (i.e. each player either wins or loses a given play). However, due to the presence of stochastic vertices, a player's *expected payoff* (i.e. her probability of winning) can be an arbitrary probability.

Stochastic games with  $\omega$ -regular objectives have been successfully applied in the verification and synthesis of reactive systems under the influence of random events. Such a system is usually modelled as a game between the system and its environment, where the environment's objective is the complement of the system's objective: the environment is considered hostile. Therefore, the research

---

<sup>\*</sup> This work was supported by the DFG Research Training Group 1298 (ALGO SYN) and the ESF Research Networking Programme "Games for Design and Verification".

in this area has traditionally focused on two-player games where each play is won by precisely one of the two players, so-called *two-player zero-sum games*. However, the system may comprise of several components with independent objectives, a situation which is naturally modelled by a multiplayer game.

The most common interpretation of rational behaviour in multiplayer games is captured by the notion of a *Nash equilibrium* [20]. In a Nash equilibrium, no player can improve her payoff by unilaterally switching to a different strategy. Chatterjee & al. [7] gave an algorithm for computing a Nash equilibrium in a stochastic multiplayer games with  $\omega$ -regular winning conditions. We argue that this is not satisfactory. Indeed, it can be shown that their algorithm may compute an equilibrium where all players lose almost surely (i.e. receive expected payoff 0), while there exist other equilibria where all players win almost surely (i.e. receive expected payoff 1).

In applications, one might look for an equilibrium where as many players as possible win almost surely or where it is guaranteed that the expected payoff of the equilibrium falls into a certain interval. Formulated as a decision problem, we want to know, given a  $k$ -player game  $\mathcal{G}$  with initial vertex  $v_0$  and two thresholds  $\bar{x}, \bar{y} \in [0, 1]^k$ , whether  $(\mathcal{G}, v_0)$  has a Nash equilibrium with expected payoff at least  $\bar{x}$  and at most  $\bar{y}$ . This problem, which we call NE for short, is a generalisation of the *quantitative decision problem* for two-player zero-sum games, which asks whether in such a game player 0 has a strategy that ensures to win the game with a probability that lies above a given threshold.

In this paper, we analyse the decidability of NE for games with  $\omega$ -regular objectives. Although the decidability of NE remains open, we can show that several restrictions of NE are decidable: First, we show that NE becomes decidable when one restricts the search space to equilibria in positional (i.e. pure, stationary), or stationary, strategies, and that the resulting decision problems typically lie in NP and PSPACE, respectively (e.g. if the objectives are specified as Muller conditions). Second, we show that the following *qualitative* version of NE is decidable: Given a  $k$ -player game  $\mathcal{G}$  with initial vertex  $v_0$  and a *binary* payoff  $\bar{x} \in \{0, 1\}^k$ , decide whether  $(\mathcal{G}, v_0)$  has a Nash equilibrium with expected payoff  $\bar{x}$ . Moreover, we prove that, depending on the representation of the objective, this problem is typically complete for one of the complexity classes P, NP, co-NP and PSPACE, and that the problem is invariant under restricting the search space to equilibria in pure, finite-state strategies.

Our results have to be viewed in light of the (mostly) negative results we derived in [26]. In particular, it was shown in [26] that NE becomes undecidable if one restricts the search space to equilibria in pure strategies (as opposed to equilibria in possibly mixed strategies), even for *simple stochastic multiplayer games*. These are games with simple reachability objectives. The undecidability result crucially makes use of the fact that the Nash equilibrium one is looking for can have a payoff that is not binary. Hence, this result cannot be applied to the qualitative version of NE, which we show to be decidable in this paper. It was also proven in [26] that the problems that arise from NE when one restricts the search space to equilibria in positional or stationary strategies are both

NP-hard. Moreover, we showed that the restriction to stationary strategies is at least as hard as the problem SqrtSum [1], a problem which is not known to lie inside the polynomial hierarchy. This demonstrates that the upper bounds we prove for these problems in this paper will be hard to improve.

Due to lack of space, some of the proofs in this paper are either only sketched or omitted entirely. For the complete proofs, see [27].

**Related Work.** Determining the complexity of Nash equilibria has attracted much interest in recent years. In particular, a series of papers culminated in the result that computing a Nash equilibrium of a two-player game in strategic form is complete for the complexity class PPAD [12,8]. However, the work closest to ours is [25], where the decidability of (a variant of) the qualitative version of NE in infinite games without stochastic vertices was proven. Our results complement the results in that paper, and although our decidability proof for the qualitative setting is structurally similar to the one in [25], the presence of stochastic vertices makes the proof substantially more challenging.

Another subject that is related to the study of stochastic multiplayer games are *Markov decision processes with multiple objectives*. These can be viewed as stochastic multiplayer games where all non-stochastic vertices are controlled by one single player. For  $\omega$ -regular objectives, Etessami & al. [16] proved the decidability of NE for these games. Due to the different nature of the restrictions, this result is incomparable to our results.

## 2 Preliminaries

The model of a (*two-player zero-sum*) *stochastic game* [9] easily generalises to the multiplayer case: Formally, a *stochastic multiplayer game (SMG)* is a tuple  $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, \Delta, (\text{Win}_i)_{i \in \Pi})$  where

- $\Pi$  is a finite set of *players* (usually  $\Pi = \{0, 1, \dots, k - 1\}$ );
- $V$  is a finite, non-empty set of *vertices*;
- $V_i \subseteq V$  and  $V_i \cap V_j = \emptyset$  for each  $i \neq j \in \Pi$ ;
- $\Delta \subseteq V \times ([0, 1] \cup \{\perp\}) \times V$  is the *transition relation*;
- $\text{Win}_i \subseteq V^\omega$  is a Borel set for each  $i \in \Pi$ .

The structure  $G = (V, (V_i)_{i \in \Pi}, \Delta)$  is called the *arena* of  $\mathcal{G}$ , and  $\text{Win}_i$  is called the *objective*, or the *winning condition*, of player  $i \in \Pi$ . A vertex  $v \in V$  is *controlled by player  $i$*  if  $v \in V_i$  and a *stochastic vertex* if  $v \notin \bigcup_{i \in \Pi} V_i$ .

We require that a transition is labelled by a probability iff it originates in a stochastic vertex: If  $(v, p, w) \in \Delta$  then  $p \in [0, 1]$  if  $v$  is a stochastic vertex and  $p = \perp$  if  $v \in V_i$  for some  $i \in \Pi$ . Additionally, for each pair of a stochastic vertex  $v$  and an arbitrary vertex  $w$ , we require that there exists precisely one  $p \in [0, 1]$  such that  $(v, p, w) \in \Delta$ . Moreover, for each stochastic vertex  $v$ , the outgoing probabilities must sum up to 1:  $\sum_{(p,w):(v,p,w) \in \Delta} p = 1$ . Finally, we require that for each vertex the set  $v\Delta := \{w \in V : \text{exists } p \in (0, 1] \cup \{\perp\} \text{ with } (v, p, w) \in \Delta\}$  is non-empty, i.e. every vertex has at least one successor.

A special class of SMGs are *two-player zero-sum stochastic games (2SGs)*. These are SMGs played by only two players (player 0 and player 1) and one player’s objective is the complement of the other player’s objective, i.e.  $\text{Win}_0 = V^\omega \setminus \text{Win}_1$ . An even more restricted model are *one-player stochastic games*, also known as *Markov decision processes (MDPs)*, where there is only one player (player 0). Finally, *Markov chains* are SMGs with no players at all, i.e. there are only stochastic vertices.

**Strategies and strategy profiles.** In the following, let  $\mathcal{G}$  be an arbitrary SMG. A *(mixed) strategy of player  $i$  in  $\mathcal{G}$*  is a mapping  $\sigma : V^*V_i \rightarrow \mathcal{D}(V)$  assigning to each possible *history*  $xv \in V^*V_i$  of vertices ending in a vertex controlled by player  $i$  a (discrete) probability distribution over  $V$  such that  $\sigma(xv)(w) > 0$  only if  $(v, \perp, w) \in \Delta$ . Instead of  $\sigma(xv)(w)$ , we usually write  $\sigma(w \mid xv)$ . A *(mixed) strategy profile of  $\mathcal{G}$*  is a tuple  $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$  where  $\sigma_i$  is a strategy of player  $i$  in  $\mathcal{G}$ . Given a strategy profile  $\bar{\sigma} = (\sigma_j)_{j \in \Pi}$  and a strategy  $\tau$  of player  $i$ , we denote by  $(\bar{\sigma}_{-i}, \tau)$  the strategy profile resulting from  $\bar{\sigma}$  by replacing  $\sigma_i$  with  $\tau$ .

A strategy  $\sigma$  of player  $i$  is called *pure* if for each  $xv \in V^*V_i$  there exists  $w \in v\Delta$  with  $\sigma(w \mid xv) = 1$ . Note that a pure strategy of player  $i$  can be identified with a function  $\sigma : V^*V_i \rightarrow V$ . A strategy profile  $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$  is called *pure* if each  $\sigma_i$  is pure.

A strategy  $\sigma$  of player  $i$  in  $\mathcal{G}$  is called *stationary* if  $\sigma$  depends only on the current vertex:  $\sigma(xv) = \sigma(v)$  for all  $xv \in V^*V_i$ . Hence, a stationary strategy of player  $i$  can be identified with a function  $\sigma : V_i \rightarrow \mathcal{D}(V)$ . A strategy profile  $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$  of  $\mathcal{G}$  is called *stationary* if each  $\sigma_i$  is stationary.

We call a pure, stationary strategy a *positional strategy* and a strategy profile consisting of positional strategies only a *positional strategy profile*. Clearly, a positional strategy of player  $i$  can be identified with a function  $\sigma : V_i \rightarrow V$ . More generally, a pure strategy  $\sigma$  is called *finite-state* if it can be implemented by a finite automaton with output or, equivalently, if the equivalence relation  $\sim \subseteq V^* \times V^*$  defined by  $x \sim y$  if  $\sigma(xz) = \sigma(yz)$  for all  $z \in V^*V_i$  has only finitely many equivalence classes. Finally, a *finite-state strategy profile* is a profile consisting of finite-state strategies only.

It is sometimes convenient to designate an initial vertex  $v_0 \in V$  of the game. We call the tuple  $(\mathcal{G}, v_0)$  an *initialised SMG*. A strategy (strategy profile) of  $(\mathcal{G}, v_0)$  is just a strategy (strategy profile) of  $\mathcal{G}$ . In the following, we will use the abbreviation SMG also for initialised SMGs. It should always be clear from the context if the game is initialised or not.

Given an initial vertex  $v_0$  and a strategy profile  $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$ , the *conditional probability of  $w \in V$  given the history  $xv \in V^*V$*  is the number  $\sigma_i(w \mid xv)$  if  $v \in V_i$  and the unique  $p \in [0, 1]$  such that  $(v, p, w) \in \Delta$  if  $v$  is a stochastic vertex. We abuse notation and denote this probability by  $\bar{\sigma}(w \mid xv)$ . The probabilities  $\bar{\sigma}(w \mid xv)$  induce a probability measure on the space  $V^\omega$  in the following way: The probability of a basic open set  $v_1 \dots v_k \cdot V^\omega$  is 0 if  $v_1 \neq v_0$  and the product of the probabilities  $\bar{\sigma}(v_j \mid v_1 \dots v_{j-1})$  for  $j = 2, \dots, k$  otherwise. It is a classical result of measure theory that this extends to a unique probability measure assigning a probability to every Borel subset of  $V^\omega$ , which we denote by  $\text{Pr}_{v_0}^{\bar{\sigma}}$ .

For a strategy profile  $\bar{\sigma}$ , we are mainly interested in the probabilities  $p_i := \Pr_{v_0}^{\bar{\sigma}}(\text{Win}_i)$  of winning. We call  $p_i$  the (expected) payoff of  $\bar{\sigma}$  for player  $i$  and the vector  $(p_i)_{i \in \Pi}$  the (expected) payoff of  $\bar{\sigma}$ .

**Subarenas and end components.** Given an SMG  $\mathcal{G}$ , we call a set  $U \subseteq V$  a subarena of  $\mathcal{G}$  if 1.  $U \neq \emptyset$ ; 2.  $v\Delta \cap U \neq \emptyset$  for each  $v \in U$ , and 3.  $v\Delta \subseteq U$  for each stochastic vertex  $v \in U$ .

A set  $C \subseteq V$  is called an end component of  $\mathcal{G}$  if  $C$  is a subarena, and additionally  $C$  is strongly connected: for every pair of vertices  $v, w \in C$  there exists a sequence  $v = v_1, v_2, \dots, v_n = w$  with  $v_{i+1} \in v_i\Delta$  for each  $0 < i < n$ . An end component  $C$  is maximal in a set  $U \subseteq V$  if there is no end component  $C' \subseteq U$  with  $C \subsetneq C'$ . For any subset  $U \subseteq V$ , the set of all end components maximal in  $U$  can be computed by standard graph algorithms in quadratic time (see e.g. [13]).

The central fact about end components is that, under any strategy profile, the set of vertices visited infinitely often is almost surely an end component. For an infinite sequence  $\alpha$ , we denote by  $\text{Inf}(\alpha)$  the set of elements occurring infinitely often in  $\alpha$ .

**Lemma 1 ([13,10]).** *Let  $\mathcal{G}$  be any SMG, and let  $\bar{\sigma}$  be any strategy profile of  $\mathcal{G}$ . Then  $\Pr_v^{\bar{\sigma}}(\{\alpha \in V^\omega : \text{Inf}(\alpha) \text{ is an end component}\}) = 1$  for each vertex  $v \in V$ .*

Moreover, for any end component  $C$ , we can construct a stationary strategy profile  $\bar{\sigma}$  that, when started in  $C$ , guarantees to visit all (and only) vertices in  $C$  infinitely often.

**Lemma 2 ([13,11]).** *Let  $\mathcal{G}$  be any SMG, and let  $C$  be any end component of  $\mathcal{G}$ . There exists a stationary strategy profile  $\bar{\sigma}$  with  $\Pr_v^{\bar{\sigma}}(\{\alpha \in V^\omega : \text{Inf}(\alpha) = C\}) = 1$  for each vertex  $v \in C$ .*

**Values, determinacy and optimal strategies.** Given a strategy  $\tau$  of player  $i$  in  $\mathcal{G}$  and a vertex  $v \in V$ , the value of  $\tau$  from  $v$  is the number  $\text{val}^\tau(v) := \inf_{\bar{\sigma}} \Pr_v^{\bar{\sigma}-i, \tau}(\text{Win}_i)$ , where  $\bar{\sigma}$  ranges over all strategy profiles of  $\mathcal{G}$ . Moreover, we define the value of  $\mathcal{G}$  for player  $i$  from  $v$  as the supremum of these values, i.e.  $\text{val}_i^{\mathcal{G}}(v) = \sup_{\tau} \text{val}^\tau(v)$ , where  $\tau$  ranges over all strategies of player  $i$  in  $\mathcal{G}$ . Intuitively,  $\text{val}_i^{\mathcal{G}}(v)$  is the maximal payoff that player  $i$  can ensure when the game starts from  $v$ . If  $\mathcal{G}$  is a two-player zero-sum game, a celebrated theorem due to Martin [19] states that the game is determined, i.e.  $\text{val}_0^{\mathcal{G}} = 1 - \text{val}_1^{\mathcal{G}}$  (where the equality holds pointwise). The number  $\text{val}^{\mathcal{G}}(v) := \text{val}_0^{\mathcal{G}}(v)$  is consequently called the value of  $\mathcal{G}$  from  $v$ .

Given an initial vertex  $v_0 \in V$ , a strategy  $\sigma$  of player  $i$  in  $\mathcal{G}$  is called optimal if  $\text{val}^\sigma(v_0) = \text{val}_i^{\mathcal{G}}(v_0)$ . A globally optimal strategy is a strategy that is optimal for every possible initial vertex  $v_0 \in V$ . Note that optimal strategies do not need to exist since the supremum in the definition of  $\text{val}_i^{\mathcal{G}}$  is not necessarily attained. However, if for every possible initial vertex there exists an optimal strategy, then there also exists a globally optimal strategy.

**Objectives.** We have introduced objectives as abstract Borel sets of infinite sequences of vertices; to be amenable for algorithmic solutions, all objectives must be finitely representable. In verification, objectives are usually  $\omega$ -regular sets specified by formulae of the logic S1S (monadic second-order logic on infinite words) or LTL (linear-time temporal logic) referring to unary predicates  $P_c$  indexed by a finite set  $C$  of colours. These are interpreted as winning conditions in a game by considering a colouring  $\chi : V \rightarrow C$  of the vertices in the game. Special cases are the following well-studied conditions:

- *Büchi* (given by a set  $F \subseteq C$ ): the set of all  $\alpha \in C^\omega$  such that  $\text{Inf}(\alpha) \cap F \neq \emptyset$ .
- *co-Büchi* (given by set  $F \subseteq C$ ): the set of all  $\alpha \in C^\omega$  such that  $\text{Inf}(\alpha) \subseteq F$ .
- *Parity* (given by a priority function  $\Omega : C \rightarrow \mathbb{N}$ ): the set of all  $\alpha \in C^\omega$  such that  $\min(\text{Inf}(\Omega(\alpha)))$  is even.
- *Streett* (given by a set  $\Omega$  of pairs  $(F, G)$  where  $F, G \subseteq C$ ): the set of all  $\alpha \in C^\omega$  such that for all pairs  $(F, G) \in \Omega$  with  $\text{Inf}(\alpha) \cap F \neq \emptyset$  it is the case that  $\text{Inf}(\alpha) \cap G \neq \emptyset$ .
- *Rabin* (given by a set  $\Omega$  of pairs  $(F, G)$  where  $F, G \subseteq C$ ): the set of all  $\alpha \in C^\omega$  such that there exists a pair  $(F, G) \in \Omega$  with  $\text{Inf}(\alpha) \cap F \neq \emptyset$  but  $\text{Inf}(\alpha) \cap G = \emptyset$ .
- *Muller* (given by a family  $\mathcal{F}$  of sets  $F \subseteq C$ ): the set of all  $\alpha \in C^\omega$  such that there exists  $F \in \mathcal{F}$  with  $\text{Inf}(\alpha) = F$ .

Note that any Büchi condition is a parity condition with two priorities, that any parity condition is both a Streett and a Rabin condition, and that any Streett or Rabin condition is a Muller condition. (However, the translation from a set of Streett/Rabin pairs to an equivalent family of accepting sets is, in general, exponential.) In fact, the intersection (union) of any two parity conditions is a Streett (Rabin) condition. Moreover, the complement of a Büchi (Streett) condition is a co-Büchi (Rabin) condition and vice versa, whereas the class of parity conditions and the class of Muller conditions are closed under complementation. Finally, note that any of the above condition is *prefix-independent*: for every  $\alpha \in C^\omega$  and  $x \in C^*$ ,  $\alpha$  satisfies the condition iff  $x\alpha$  does.

Theoretically, parity and Rabin conditions provide the best balance of expressiveness and simplicity: On the one hand, any SMG where player  $i$  has a Rabin objective admits a globally optimal positional strategy for this player [4]. On the other hand, any SMG with  $\omega$ -regular objectives can be reduced to an SMG with parity objectives using *finite memory* (see [24]). An important consequence of this reduction is that there exist globally optimal finite-state strategies in every SMG with  $\omega$ -regular objectives. In fact, there exist globally optimal pure strategies in every SMG with prefix-independent objectives [17].

In the following, for the sake of simplicity, we will only consider games where each vertex is coloured by itself, i.e.  $C = V$  and  $\chi = \text{id}$ . We would like to point out, however, that all our results remain valid for games with other colourings. For the same reason, we will usually not distinguish between a condition and its finite representation.

**Decision problems for two-player zero-sum games.** The main computational problem for two-player zero-sum games is computing the value (and optimal strategies for either player, if they exist). Rephrased as a decision problem, the problem looks as follows:

Given a 2SG  $\mathcal{G}$ , an initial vertex  $v_0$  and a rational probability  $p$ , decide whether  $\text{val}^{\mathcal{G}}(v_0) \geq p$ .

A special case of this problem arises for  $p = 1$ . Here, we only want to know whether player 0 can win the game almost surely (in the limit). Let us call the former problem the *quantitative* and the latter problem the *qualitative decision problem for 2SGs*.

Table 1 summarises the results about the complexity of the quantitative and the qualitative decision problem for two-player zero-sum stochastic games depending on the type of player 0's objective. For MDPs, both problems are decidable in polynomial time for all of the aforementioned objectives (i.e. up to Muller conditions) [3,13].

**Table 1.** The complexity of deciding the value in 2SGs

	Quantitative	Qualitative
(co-)Büchi	NP $\cap$ co-NP [6]	P-complete [14]
Parity	NP $\cap$ co-NP [6]	NP $\cap$ co-NP [6]
Streett	co-NP-complete [4,15]	co-NP-complete [4,15]
Rabin	NP-complete [4,15]	NP-complete [4,15]
Muller	PSPACE-complete [3,18]	PSPACE-complete [3,18]

### 3 Nash Equilibria and Their Decision Problems

To capture rational behaviour of (selfish) players, John Nash [20] introduced the notion of, what is now called, a *Nash equilibrium*. Formally, given a strategy profile  $\bar{\sigma}$  in an SMG  $(\mathcal{G}, v_0)$ , a strategy  $\tau$  of player  $i$  is called a *best response to  $\bar{\sigma}$*  if  $\tau$  maximises the expected payoff of player  $i$ :  $\Pr_{v_0}^{\bar{\sigma}-i, \tau'}(\text{Win}_i) \leq \Pr_{v_0}^{\bar{\sigma}-i, \tau}(\text{Win}_i)$  for all strategies  $\tau'$  of player  $i$ . A Nash equilibrium is a strategy profile  $\bar{\sigma} = (\sigma_i)_{i \in \Pi}$  such that each  $\sigma_i$  is a best response to  $\bar{\sigma}$ . Hence, in a Nash equilibrium no player can improve her payoff by (unilaterally) switching to a different strategy. For two-player zero-sum games, a Nash equilibrium is nothing else than a pair of optimal strategies.

**Proposition 1.** *Let  $(\mathcal{G}, v_0)$  be a two-player zero-sum game. A strategy profile  $(\sigma, \tau)$  of  $(\mathcal{G}, v_0)$  is a Nash equilibrium iff both  $\sigma$  and  $\tau$  are optimal. In particular, every Nash equilibrium of  $(\mathcal{G}, v_0)$  has payoff  $(\text{val}^{\mathcal{G}}(v_0), 1 - \text{val}^{\mathcal{G}}(v_0))$ .*

So far, most research on finding Nash equilibria in infinite games has focused on computing *some* Nash equilibrium [7]. However, a game may have several Nash equilibria with different payoffs, and one might not be interested in *any* Nash



equilibrium but in one whose payoff fulfils certain requirements. For example, one might look for a Nash equilibrium where certain players win almost surely while certain others lose almost surely. This idea leads to the following decision problem, which we call NE<sup>1</sup>

Given an SMG  $(\mathcal{G}, v_0)$  and thresholds  $\bar{x}, \bar{y} \in [0, 1]^I$ , decide whether there exists a Nash equilibrium of  $(\mathcal{G}, v_0)$  with payoff  $\geq \bar{x}$  and  $\leq \bar{y}$ .

Of course, as a decision problem the problem only makes sense if the game and the thresholds  $\bar{x}$  and  $\bar{y}$  are represented in a finite way. In the following, we will therefore assume that the thresholds and all transition probabilities are rational, and that all objectives are  $\omega$ -regular.

Note that NE puts no restriction on the type of strategies that realise the equilibrium. It is natural to restrict the search space to equilibria that are realised in pure, finite-state, stationary, or even positional strategies. Let us call the corresponding decision problems PureNE, FinNE, StatNE and PosNE, respectively.

In a recent paper [26], we studied NE and its variants in the context of *simple* stochastic multiplayer games (SSMGs). These are SMGs where each player's objective is to reach a certain set  $T$  of *terminal vertices*:  $v\Delta = \{v\}$  for each  $v \in T$ . In particular, such objectives are both Büchi and co-Büchi conditions. Our main results on SSMGs can be summarised as follows:

- PureNE and FinNE are undecidable;
- StatNE is contained in PSPACE, but NP- and SqrtSum-hard;
- PosNE is NP-complete.

In fact, PureNE and FinNE are undecidable even if one restricts to instances where the thresholds are binary, but distinct, or if one restricts to instances where the thresholds coincide (but are not binary). Hence, the question arises what happens if the thresholds are binary and coincide. This question motivates the following *qualitative* version of NE, a problem which we call QualNE:

Given an SMG  $(\mathcal{G}, v_0)$  and  $\bar{x} \in \{0, 1\}^I$ , decide whether  $(\mathcal{G}, v_0)$  has a Nash equilibrium with payoff  $\bar{x}$ .

In this paper, we show that QualNE, StatNE and PosNE are decidable for games with arbitrary  $\omega$ -regular objectives, and analyse the complexities of these problems depending on the type of the objectives.

## 4 Stationary Equilibria

In this section, we analyse the complexity of the problems PosNE and StatNE. Lower bounds for these problems follow from our results on SSMGs [26].

**Theorem 1.** *PosNE is NP-complete for SMGs with Büchi, co-Büchi, parity, Rabin, Streett, or Muller objectives.*

<sup>1</sup> In the definition of NE, the ordering  $\leq$  is applied componentwise.

*Proof.* Hardness was already proven in [26]. To prove membership in NP, we give a nondeterministic polynomial-time algorithm for deciding PosNE. On input  $\mathcal{G}, v_0, \bar{x}, \bar{y}$ , the algorithm simply guesses a positional strategy profile  $\bar{\sigma}$  (which is basically a mapping  $\bigcup_{i \in \Pi} V_i \rightarrow V$ ). Next, the algorithm computes the payoff  $z_i$  of  $\bar{\sigma}$  for each player  $i$  by computing the probability of the event  $\text{Win}_i$  in the Markov chain  $(\mathcal{G}^{\bar{\sigma}}, v_0)$ , which arises from  $\mathcal{G}$  by fixing all transitions according to  $\bar{\sigma}$ . Once each  $z_i$  is computed, the algorithm can easily check whether  $x_i \leq z_i \leq y_i$ . To check whether  $\bar{\sigma}$  is a Nash equilibrium, the algorithm needs to compute, for each player  $i$ , the value  $r_i$  of the MDP  $(\mathcal{G}^{\bar{\sigma}^{-i}}, v_0)$ , which arises from  $\mathcal{G}$  by fixing all transitions but the ones leaving vertices controlled by player  $i$  according to  $\bar{\sigma}$  (and imposing the objective  $\text{Win}_i$ ). Clearly,  $\bar{\sigma}$  is a Nash equilibrium iff  $r_i \leq z_i$  for each player  $i$ . Since we can compute the value of any MDP (and thus any Markov chain) with one of the above objectives in polynomial time [3,13], all these checks can be carried out in polynomial time.  $\square$

To prove the decidability of StatNE, we appeal to results established for the *Existential Theory of the Reals*,  $\text{ExTh}(\mathfrak{R})$ , the set of all existential first-order sentences (over the appropriate signature) that hold in  $\mathfrak{R} := (\mathbb{R}, +, \cdot, 0, 1, \leq)$ . The best known upper bound for the complexity of the associated decision problem is PSPACE [2], which leads to the following theorem.

**Theorem 2.** *StatNE is in PSPACE for SMGs with Büchi, co-Büchi, parity, Rabin, Streett, or Muller objective.*

*Proof (Sketch).* Since  $\text{PSPACE} = \text{NPSpace}$ , it suffices to provide a nondeterministic algorithm with polynomial space requirements for deciding StatNE. On input  $\mathcal{G}, v_0, \bar{x}, \bar{y}$ , where w.l.o.g.  $\mathcal{G}$  is an SMG with Muller objectives  $\mathcal{F}_i \in 2^V$ , the algorithm starts by guessing the support  $S \subseteq V \times V$  of a stationary strategy profile  $\bar{\sigma}$  of  $\mathcal{G}$ , i.e.  $S = \{(v, w) \in V \times V : \bar{\sigma}(w | v) > 0\}$ . From the set  $S$  alone, by standard graph algorithms (see [3,13]), one can compute (in polynomial time) for each player  $i$  the following sets:

1. the union  $F_i$  of all end components (i.e. bottom SCCs)  $C$  of the Markov chain  $\mathcal{G}^{\bar{\sigma}}$  that are winning for player  $i$ , i.e.  $C \in \mathcal{F}_i$ ;
2. the set  $R_i$  of vertices  $v$  such that  $\Pr_v^{\bar{\sigma}}(\text{Reach}(F_i)) > 0$ ;
3. the union  $T_i$  of all end components of the MDP  $\mathcal{G}^{\bar{\sigma}^{-i}}$  that are winning for player  $i$ .

After computing all these sets, the algorithm evaluates a suitable existential first-order sentence  $\psi$ , which can be computed in polynomial time from  $\mathcal{G}, v_0, \bar{x}, \bar{y}, (R_i)_{i \in \Pi}, (F_i)_{i \in \Pi}$  and  $(T_i)_{i \in \Pi}$  over  $\mathfrak{R}$  and returns the answer to this query. The sentence  $\psi$  states that there exists a stationary Nash equilibrium of  $(\mathcal{G}, v_0)$  with payoff  $\geq \bar{x}$  and  $\leq \bar{y}$  whose support is  $S$ .  $\square$

## 5 Equilibria with a Binary Payoff

In this section, we prove that QualNE is decidable. We start by characterising the existence of a Nash equilibrium with a binary payoff in any game with prefix-independent objectives.

### 5.1 Characterisation of Existence

For a subset  $U \subseteq V$ , we denote by  $\text{Reach}(U)$  the set  $V^* \cdot U \cdot V^\omega$ ; if  $U = \{v\}$ , we just write  $\text{Reach}(v)$  for  $\text{Reach}(U)$ . Finally, given an SMG  $\mathcal{G}$  and a player  $i$ , we denote by  $V_i^{>0}$  the set of all vertices  $v \in V$  such that  $\text{val}_i^{\mathcal{G}}(v) > 0$ . The following lemma allows to infer the existence of a Nash equilibrium from the existence of a certain strategy profile. The proof uses so-called *threat strategies* (also known as *trigger strategies*), which are the basis of the *folk theorems* in the theory of repeated games (cf. [22, Chapter 8]).

**Lemma 3.** *Let  $\bar{\sigma}$  be a pure strategy profile of  $\mathcal{G}$  such that, for each player  $i$ ,  $\Pr_{v_0}^{\bar{\sigma}}(\text{Win}_i) = 1$  or  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$ . Then there exists a pure Nash equilibrium  $\bar{\sigma}^*$  with  $\Pr_{v_0}^{\bar{\sigma}^*} = \Pr_{v_0}^{\bar{\sigma}}$ . If, additionally, all winning conditions are  $\omega$ -regular and  $\bar{\sigma}$  is finite-state, then there exists a finite-state Nash equilibrium  $\bar{\sigma}^*$  with  $\Pr_{v_0}^{\bar{\sigma}^*} = \Pr_{v_0}^{\bar{\sigma}}$ .*

*Proof (Sketch).* Let  $\mathcal{G}_i = (\{i, \Pi \setminus \{i\}\}, V, V_i, \bigcup_{j \neq i} V_j, \Delta, \text{Win}_i, V^\omega \setminus \text{Win}_i)$  be the 2SG where player  $i$  plays against the coalition  $\Pi \setminus \{i\}$  of all other players. Since the set  $\text{Win}_i$  is prefix-independent, there exists a globally optimal pure strategy  $\tau_i$  for the coalition in this game [17]. For each player  $j \neq i$ , this strategy induces a pure strategy  $\tau_{j,i}$  in  $\mathcal{G}$ . To simplify notation, we also define  $\tau_{i,i}$  to be an arbitrary finite-state strategy of player  $i$  in  $\mathcal{G}$ . Player  $i$ 's equilibrium strategy  $\sigma_i^*$  is defined as follows:

$$\sigma_i^*(xv) = \begin{cases} \sigma_i(xv) & \text{if } \Pr_{v_0}^{\bar{\sigma}}(xv \cdot V^\omega) > 0, \\ \tau_{i,j}(x_2v) & \text{otherwise,} \end{cases}$$

where, in the latter case,  $x = x_1x_2$  with  $x_1$  being the longest prefix of  $xv$  such that  $\Pr_{v_0}^{\bar{\sigma}}(x_1 \cdot V^\omega) > 0$  and  $j \in \Pi$  being the player that has deviated from  $\bar{\sigma}$ , i.e.  $x_1$  ends in  $V_j$ ; if  $x_1$  is empty or ends in a stochastic vertex, we set  $j = i$ . Intuitively,  $\sigma_i^*$  behaves like  $\sigma_i$  as long as no other player  $j$  deviates from playing  $\sigma_j$ , in which case  $\sigma_i^*$  starts to behave like  $\tau_{i,j}$ .

If each  $\text{Win}_i$  is  $\omega$ -regular, then each of the strategies  $\tau_i$  can be chosen to be a finite-state strategy. Consequently, each  $\tau_{j,i}$  can be assumed to be finite-state. If additionally  $\bar{\sigma}$  is finite-state, it is easy to see that the strategy profile  $\bar{\sigma}^*$ , as defined above, is also finite-state.

Note that  $\Pr_{v_0}^{\bar{\sigma}^*} = \Pr_{v_0}^{\bar{\sigma}}$ . We claim that  $\bar{\sigma}^*$  is a Nash equilibrium of  $(\mathcal{G}, v_0)$ .  $\square$

Finally, we can state the main result of this section.

**Proposition 2.** *Let  $(\mathcal{G}, v_0)$  be any SMG with prefix-independent winning conditions, and let  $\bar{x} \in \{0, 1\}^\Pi$ . Then the following statements are equivalent:*

1. *There exists a Nash equilibrium with payoff  $\bar{x}$ ;*
2. *There exists a strategy profile  $\bar{\sigma}$  with payoff  $\bar{x}$  such that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ ;*
3. *There exists a pure strategy profile  $\bar{\sigma}$  with payoff  $\bar{x}$  such that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ ;*
4. *There exists a pure Nash equilibrium with payoff  $\bar{x}$ .*

If additionally all winning conditions are  $\omega$ -regular, then any of the above statements is equivalent to each of the following statements:

- 5. There exists a finite-state strategy profile  $\bar{\sigma}$  with payoff  $\bar{x}$  such that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ ;
- 6. There exists a finite-state Nash equilibrium with payoff  $\bar{x}$ .

*Proof.* (1.  $\Rightarrow$  2.) Let  $\bar{\sigma}$  be a Nash equilibrium with payoff  $\bar{x}$ . We claim that  $\bar{\sigma}$  is already the strategy profile we are looking for:  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ . Towards a contradiction, assume that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) > 0$  for some player  $i$  with  $x_i = 0$ . Since  $V$  is finite, there exists a vertex  $v \in V_i^{>0}$  and a history  $x$  such that  $\Pr_{v_0}^{\bar{\sigma}}(xv \cdot V^\omega) > 0$ . Let  $\tau$  be an optimal strategy for player  $i$  in the game  $(\mathcal{G}, v)$ , and consider her strategy  $\sigma'$  defined by

$$\sigma'(yw) = \begin{cases} \sigma(yw) & \text{if } xv \not\preceq yw, \\ \tau(y'w) & \text{otherwise,} \end{cases}$$

where, in the latter case,  $y = xy'$ . A straightforward calculation yields that  $\Pr_{v_0}^{(\bar{\sigma}-i, \sigma')}(\text{Win}_i) > 0$ . Hence, player  $i$  can improve her payoff by playing  $\sigma'$  instead of  $\sigma_i$ , a contradiction to the fact that  $\bar{\sigma}$  is a Nash equilibrium.

(2.  $\Rightarrow$  3.) Let  $\bar{\sigma}$  be a strategy profile of  $(\mathcal{G}, v_0)$  with payoff  $\bar{x}$  such that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ . Consider the MDP  $\mathcal{M}$  that is obtained from  $\mathcal{G}$  by removing all vertices  $v \in V$  such that  $v \in V_i^{>0}$  for some player  $i$  with  $x_i = 0$ , merging all players into one, and imposing the objective

$$\text{Win} = \bigcap_{\substack{i \in \Pi \\ x_i = 1}} \text{Win}_i \cap \bigcap_{\substack{i \in \Pi \\ x_i = 0}} V^\omega \setminus \text{Win}_i.$$

The MDP  $\mathcal{M}$  is well-defined since its domain is a subarena of  $\mathcal{G}$ . Moreover, the value  $\text{val}^{\mathcal{M}}(v_0)$  of  $\mathcal{M}$  is equal to 1 because the strategy profile  $\bar{\sigma}$  induces a strategy  $\sigma$  in  $\mathcal{M}$  satisfying  $\Pr_{v_0}^\sigma(\text{Win}) = 1$ . Since each  $\text{Win}_i$  is prefix-independent, so is the set  $\text{Win}$ . Hence, there exists a pure, optimal strategy  $\tau$  in  $(\mathcal{M}, v_0)$ . Since the value is 1, we have  $\Pr_{v_0}^\tau(\text{Win}) = 1$ , and  $\tau$  induces a pure strategy profile of  $\mathcal{G}$  with the desired properties.

(3.  $\Rightarrow$  4.) Let  $\bar{\sigma}$  be a pure strategy profile of  $(\mathcal{G}, v_0)$  with payoff  $\bar{x}$  such that  $\Pr_{v_0}^{\bar{\sigma}}(\text{Reach}(V_i^{>0})) = 0$  for each player  $i$  with  $x_i = 0$ . By Lemma 3, there exists a pure Nash equilibrium  $\bar{\sigma}^*$  of  $(\mathcal{G}, v_0)$  with  $\Pr_{v_0}^{\bar{\sigma}} = \Pr_{v_0}^{\bar{\sigma}^*}$ . In particular,  $\bar{\sigma}^*$  has payoff  $\bar{x}$ .

(4.  $\Rightarrow$  1.) Trivial.

Under the additional assumption that all winning conditions are  $\omega$ -regular, the implications (2.  $\Rightarrow$  5.) and (5.  $\Rightarrow$  6.) are proven analogously; the implication (6.  $\Rightarrow$  1.) is trivial. □

As an immediate consequence of Proposition 2, we can conclude that finite-state strategies are as powerful as arbitrary mixed strategies as far as the existence of a Nash equilibrium with a binary payoff in SMGs with  $\omega$ -regular objectives is concerned. (This is not true for Nash equilibria with a non-binary payoff [25].)

**Corollary 1.** *Let  $(\mathcal{G}, v_0)$  be any SMG with  $\omega$ -regular objectives, and let  $x \in \{0, 1\}^{\Pi}$ . There exists a Nash equilibrium of  $(\mathcal{G}, v_0)$  with payoff  $\bar{x}$  iff there exists a finite-state Nash equilibrium of  $(\mathcal{G}, v_0)$  with payoff  $\bar{x}$ .*

*Proof.* The claim follows from Proposition 2 and the fact that every SMG with  $\omega$ -regular objectives can be reduced to one with prefix-independent  $\omega$ -regular (e.g. parity) objectives.  $\square$

### 5.2 Computational Complexity

We can now describe an algorithm for deciding QualNE for games with Muller objectives. The algorithm relies on the characterisation we gave in Proposition 2, which allows to reduce the problem to a problem about a certain MDP.

Formally, given an SMG  $\mathcal{G} = (\Pi, V, (V_i)_{i \in \Pi}, \Delta, (\mathcal{F}_i)_{i \in \Pi})$  with Muller objectives  $\mathcal{F}_i \subseteq 2^V$ , and a binary payoff  $\bar{x} \in \{0, 1\}^{\Pi}$ , we define the Markov decision process  $\mathcal{G}(\bar{x})$  as follows: Let  $Z \subseteq V$  be the set of all  $v$  such that  $\text{val}_i^{\mathcal{G}}(v) = 0$  for each player  $i$  with  $x_i = 0$ ; the set of vertices of  $\mathcal{G}(\bar{x})$  is precisely the set  $Z$ , with the set of vertices controlled by player 0 being  $Z_0 := Z \cap \bigcup_{i \in \Pi} V_i$ . (If  $Z = \emptyset$ , we define  $\mathcal{G}(\bar{x})$  to be a trivial MDP with the empty set as its objective.) The transition relation of  $\mathcal{G}(\bar{x})$  is the restriction of  $\Delta$  to transitions between  $Z$ -states. Note that the transition relation of  $\mathcal{G}(\bar{x})$  is well-defined since  $Z$  is a subarena of  $\mathcal{G}$ . We say that a subset  $U \subseteq V$  has payoff  $\bar{x}$  if  $U \in \mathcal{F}_i$  for each player  $i$  with  $x_i = 1$  and  $U \notin \mathcal{F}_i$  for each player  $i$  with  $x_i = 0$ . The objective of  $\mathcal{G}(\bar{x})$  is  $\text{Reach}(T)$  where  $T \subseteq Z$  is the union of all end components  $U \subseteq Z$  that have payoff  $\bar{x}$ .

**Lemma 4.** *Let  $(\mathcal{G}, v_0)$  be any SMG with Muller objectives, and let  $\bar{x} \in \{0, 1\}^{\Pi}$ . Then  $(\mathcal{G}, v_0)$  has a Nash equilibrium with payoff  $\bar{x}$  iff  $\text{val}^{\mathcal{G}(\bar{x})}(v_0) = 1$ .*

*Proof.* ( $\Rightarrow$ ) Assume that  $(\mathcal{G}, v_0)$  has a Nash equilibrium with payoff  $\bar{x}$ . By Proposition 2, this implies that there exists a strategy profile  $\bar{\sigma}$  of  $(\mathcal{G}, v_0)$  with payoff  $\bar{x}$  such that  $\text{Pr}_{v_0}^{\bar{\sigma}}(\text{Reach}(V \setminus Z)) = 0$ . We claim that  $\text{Pr}_{v_0}^{\bar{\sigma}}(\text{Reach}(T)) = 1$ . Otherwise, by Lemma 1, there would exist an end component  $C \subseteq Z$  such that  $C \notin \mathcal{F}_i$  for some player  $i$  with  $x_i = 1$  or  $C \in \mathcal{F}_i$  for some player  $i$  with  $x_i = 0$ , and  $\text{Pr}_{v_0}^{\bar{\sigma}}(\{\alpha \in V^\omega : \text{Inf}(\alpha) = C\}) > 0$ . But then,  $\bar{\sigma}$  cannot have payoff  $\bar{x}$ , a contradiction. Now, since  $\text{Pr}_{v_0}^{\bar{\sigma}}(\text{Reach}(V \setminus Z)) = 0$ ,  $\bar{\sigma}$  induces a strategy  $\sigma$  in  $\mathcal{G}(\bar{x})$  such that  $\text{Pr}_{v_0}^{\sigma}(B) = \text{Pr}_{v_0}^{\bar{\sigma}}(B)$  for every Borel set  $B \subseteq Z^\omega$ . In particular,  $\text{Pr}_{v_0}^{\sigma}(\text{Reach}(T)) = 1$  and hence  $\text{val}^{\mathcal{G}(\bar{x})}(v_0) = 1$ .

( $\Leftarrow$ ) Assume that  $\text{val}^{\mathcal{G}(\bar{x})}(v_0) = 1$  (in particular,  $v_0 \in Z$ ), and let  $\sigma$  be an optimal strategy in  $(\mathcal{G}(\bar{x}), v_0)$ . From  $\sigma$ , using Lemma 2, we can devise a strategy  $\sigma'$  such that  $\text{Pr}_{v_0}^{\sigma'}(\{\alpha \in V^\omega : \text{Inf}(\alpha) \text{ has payoff } \bar{x}\}) = 1$ . Finally,  $\sigma'$  can be extended to a strategy profile  $\bar{\sigma}$  of  $\mathcal{G}$  with payoff  $\bar{x}$  such that  $\text{Pr}_{v_0}^{\bar{\sigma}}(\text{Reach}(V \setminus Z)) = 0$ . By Proposition 2, this implies that  $(\mathcal{G}, v_0)$  has a Nash equilibrium with payoff  $\bar{x}$ .  $\square$

Since the value of an MDP with reachability objectives can be computed in polynomial time (via linear programming, cf. [23]), the difficult part lies in computing the MDP  $\mathcal{G}(\bar{x})$  from  $\mathcal{G}$  and  $\bar{x}$  (i.e. its domain  $Z$  and the target set  $T$ ).

**Theorem 3.** *QualNE is in PSPACE for games with Muller objectives.*

*Proof.* Since  $\text{PSPACE} = \text{NPSpace}$ , it suffices to give a nondeterministic algorithm with polynomial space requirements. On input  $\mathcal{G}, v_0, \bar{x}$ , the algorithm starts by computing for each player  $i$  with  $x_i = 0$  the set of vertices  $v$  with  $\text{val}_i^{\mathcal{G}}(v) = 0$ , which can be done in polynomial space (see table 1). The intersection of these sets is the domain  $Z$  of the Markov decision process  $\mathcal{G}(\bar{x})$ . If  $v_0$  is not contained in this intersection, the algorithm immediately rejects. Otherwise, the algorithm proceeds by guessing a set  $T' \subseteq Z$  and for each  $v \in T'$  a set  $U_v \subseteq Z$  with  $v \in U_v$ . If, for each  $v \in T'$ , the set  $U_v$  is an end component with payoff  $\bar{x}$ , the algorithm proceeds by computing (in polynomial time) the value  $\text{val}^{\mathcal{G}(\bar{x})}(v_0)$  of the MDP  $\mathcal{G}(\bar{x})$  with  $T'$  substituted for  $T$  and accepts if the value is 1. In all other cases, the algorithm rejects.

The correctness of the algorithm follows from Lemma 4 and the fact that  $\Pr_{v_0}^{\sigma}(\text{Reach}(T')) \leq \Pr_{v_0}^{\sigma}(\text{Reach}(T))$  for any strategy  $\sigma$  in  $\mathcal{G}(\bar{x})$  and any subset  $T' \subseteq T$ .  $\square$

Since any SMG with  $\omega$ -regular can effectively be reduced to one with Muller objectives, Theorem 3 implies the decidability of QualNE for games with arbitrary  $\omega$ -regular objectives (e.g. given by S1S formulae). Regarding games with Muller objectives, a matching PSPACE-hardness result appeared in [18], where it was shown that the qualitative decision problem for 2SGs with Muller objectives is PSPACE-hard, even for games without stochastic vertices. However, this result relies on the use of arbitrary colourings.

With similar arguments as for games with Muller objectives, we can show that QualNE is in NP for games with Streett objectives, and in co-NP for games with Rabin objectives. A matching NP-hardness result for games with Streett objectives was proven in [25], and the proof of this result can easily be modified to prove co-NP-hardness for games with Rabin objectives; both hardness results hold for games with only two players and without stochastic vertices.

**Theorem 4.** *QualNE is NP-complete for games with Streett objectives, and co-NP-complete for games with Rabin objectives.*

Since any parity condition can be turned into both a Streett and a Rabin condition where the number of pairs is linear in the number of priorities, we can immediately infer from Theorem 4 that QualNE is in  $\text{NP} \cap \text{co-NP}$  for games with parity objectives.

**Corollary 2.** *QualNE is in  $\text{NP} \cap \text{co-NP}$  for games with parity objectives.*

It is a major open problem whether the qualitative decision problem for 2SGs with parity objectives is in P. This would imply that QualNE is decidable in polynomial time for games with parity objectives since this would allow us to compute the domain of the MDP  $\mathcal{G}(\bar{x})$  in polynomial time. For each  $d \in \mathbb{N}$ , a class of games where the qualitative decision problem is provably in P is the class of all 2SGs with parity objectives that uses at most  $d$  priorities [5]. For  $d = 2$ , this class includes all 2SGs with a Büchi or a co-Büchi objective (for player 0). Hence, we have the following theorem.

**Theorem 5.** *For each  $d \in \mathbb{N}$ , QualNE is in P for games with parity winning conditions that use at most  $d$  priorities. In particular, QualNE is in P for games with (co-)Büchi objectives.*

## 6 Conclusion

We have analysed the complexity of deciding whether a stochastic multiplayer game with  $\omega$ -regular objectives has a Nash equilibrium whose payoff falls into a certain interval. Specifically, we have isolated several decidable restrictions of the general problem that have a manageable complexity (PSPACE at most). For instance, the complexity of the qualitative variant of NE is usually not higher than for the corresponding problem for two-player zero-sum games.

Apart from settling the complexity of NE (where arbitrary mixed strategies are allowed), two directions for future work come to mind: First, one could study other restrictions of NE that might be decidable. For example, it seems plausible that the restriction of NE to games with two players is decidable. Second, it seems interesting to see whether our decidability results can be extended to more general models of games, e.g. *concurrent games* or games with infinitely many states like *pushdown games*.

## References

1. Allender, E., Bürgisser, P., Kjeldgaard-Pedersen, J., Miltersen, P.B.: On the complexity of numerical analysis. In: CCC 2006, pp. 331–339. IEEE Computer Society Press, Los Alamitos (2006)
2. Canny, J.: Some algebraic and geometric computations in PSPACE. In: STOC 1988, pp. 460–469. ACM Press, New York (1988)
3. Chatterjee, K.: Stochastic  $\omega$ -regular games. PhD thesis, U.C. Berkeley (2007)
4. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: The complexity of stochastic Rabin and Streett games. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 878–890. Springer, Heidelberg (2005)
5. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
6. Chatterjee, K., Jurdziński, M., Henzinger, T.A.: Quantitative stochastic parity games. In: SODA 2004, pp. 121–130. ACM Press, New York (2004)
7. Chatterjee, K., Majumdar, R., Jurdziński, M.: On Nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)
8. Chen, X., Deng, X.: Settling the complexity of two-player Nash equilibrium. In: FOCS 2006, pp. 261–272. IEEE Computer Society Press, Los Alamitos (2006)
9. Condon, A.: The complexity of stochastic games. *Information and Computation* 96(2), 203–224 (1992)
10. Courcoubetis, C.A., Yannakakis, M.: The complexity of probabilistic verification. *Journal of the ACM* 42(4), 857–907 (1995)

11. Courcoubetis, C.A., Yannakakis, M.: Markov decision processes and regular events. *IEEE Transactions on Automatic Control* 43(10), 1399–1418 (1998)
12. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. In: *STOC 2006*, pp. 71–78. ACM Press, New York (2006)
13. de Alfaro, L.: *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University (1997)
14. de Alfaro, L., Henzinger, T.A.: Concurrent omega-regular games. In: *LICS 2000*, pp. 141–154. IEEE Computer Society Press, Los Alamitos (2000)
15. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs (extended abstract). In: *FoCS 1988*, pp. 328–337. IEEE Computer Society Press, Los Alamitos (1988)
16. Etesami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science* 4(4) (2008)
17. Horn, F., Gimbert, H.: Optimal strategies in perfect-information stochastic games with tail winning conditions. *CoRR*, abs/0811.3978 (2008)
18. Hunter, P., Dawar, A.: Complexity bounds for regular games. In: Jedrzejowicz, J., Szepietowski, A. (eds.) *MFCS 2005*. LNCS, vol. 3618, pp. 495–506. Springer, Heidelberg (2005)
19. Martin, D.A.: The determinacy of Blackwell games. *Journal of Symbolic Logic* 63(4), 1565–1581 (1998)
20. Nash Jr., J.F.: Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the USA* 36, 48–49 (1950)
21. Neyman, A., Sorin, S. (eds.): *Stochastic Games and Applications*. NATO Science Series C, vol. 570. Springer, Heidelberg (2003)
22. Osborne, M.J., Rubinstein, A.: *A Course in Game Theory*. MIT Press, Cambridge (1994)
23. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Chichester (1994)
24. Thomas, W.: On the synthesis of strategies in infinite games. In: Mayr, E.W., Puech, C. (eds.) *STACS 1995*. LNCS, vol. 900, pp. 1–13. Springer, Heidelberg (1995)
25. Ummels, M.: The complexity of Nash equilibria in infinite multiplayer games. In: Amadio, R.M. (ed.) *FOSSACS 2008*. LNCS, vol. 4962, pp. 20–34. Springer, Heidelberg (2008)
26. Ummels, M., Wojtczak, D.: The complexity of Nash equilibria in simple stochastic multiplayer games. In: *ICALP 2009 (II)*. LNCS, vol. 5556, pp. 297–308. Springer, Heidelberg (2009)
27. Ummels, M., Wojtczak, D.: Decision problems for Nash equilibria in stochastic games. *CoRR*, abs/0904.3325 (2009)



# On the Complexity of Branching-Time Logics\*

Volker Weber\*\*

Fakultät für Informatik, Technische Universität Dortmund  
44221 Dortmund, Germany  
volker.weber@tu-dortmund.de

**Abstract.** We classify the complexity of the satisfiability problem for extensions of CTL and UB. The extensions we consider are Boolean combinations of path formulas, fairness properties, past modalities, and forgettable past. Our main result shows that satisfiability for CTL with all these extensions is still in **2EXPTIME**, which strongly contrasts with the nonelementary complexity of CTL\* with forgettable past. We give a complete classification of combinations of these extensions, yielding a dichotomy between extensions with **2EXPTIME**-complete and those with **EXPTIME**-complete complexity. In particular, we show that satisfiability for the extension of UB with forgettable past is complete for **2EXPTIME**, contradicting a claim for a stronger logic in the literature. The upper bounds are established with the help of a new kind of pebble automata.

**Keywords:** branching-time logic, CTL, complexity of satisfiability, pebble automata, alternating tree automata, forgettable past.

## 1 Introduction

Branching-time logics like CTL are an important framework for the specification and verification of concurrent and reactive systems [6,13,11]. Their history reaches almost thirty years back, when Lamport discussed the differences between linear-time and branching-time semantics of temporal logics in 1980 [19]. The first branching-time logic, called UB, was proposed the year after by Ben-Ari, Pnueli, and Manna, introducing the concept of existential and universal path quantification [2]. By extending UB with the “until” modality, Clarke and Emerson obtained the computational tree logic CTL [5], the up to date predominant branching-time logic.

Since then, many extensions of these logics have been considered. Some of these extensions aimed at more expressive power, others were introduced with the intention to make specification easier. In this paper, we consider four of these extensions that have been discussed at length in the literature, namely Boolean combinations of path formulas, fairness, past modalities, and forgettable past.

Combining these extensions, we obtain a wealth of branching-time logics. Many of the logics have been studied for their expressive power, the complexity of

---

\* Supported by DFG grant SCHW 678/4-1.

\*\* Deceased.

their satisfiability and model checking problems, and for optimal model checking algorithms. Nevertheless, for most of these logics the picture is still incomplete.

In this work, we complete the picture for the complexity of the satisfiability problem. Concretely, we completely classify the complexity of satisfiability for all branching-time logics obtained from UB and CTL by any combination of the extensions listed above.

Let us take a look at those parts of the picture that are already there. The classical results in the area are the proofs of **EXPTIME**-completeness for satisfiability of UB [2] and CTL [8]. In the following paragraphs, we review known results for the extensions we consider.

*Boolean Combinations of Path Formulas.* Both, UB and CTL, require that every temporal operator is immediately preceded by a path quantifier. Emerson and Halpern were the first to study a logic that also allows Boolean combinations of temporal operators, i.e., of path formulas, as in  $\mathbf{E}(\mathbf{F}p \wedge \neg \mathbf{F}q)$  [8]. They called these logics  $\text{UB}^+$  and  $\text{CTL}^+$  and obtained the following hierarchy on their expressive power:  $\text{UB} \prec \text{UB}^+ \prec \text{CTL} \equiv \text{CTL}^+$ . Concerning complexity<sup>1</sup>,  $\text{CTL}^+$  has been shown to be complete for **2EXPTIME** by Johannsen and Lange [15]. The precise complexity of  $\text{UB}^+$  is unknown.

*Fairness.* CTL cannot express fairness properties, e.g., that there exists a path on which a proposition  $p$  holds infinitely often. Therefore, Emerson et al. introduced ECTL by extending CTL with a new temporal operator  $\mathbf{F}^\infty$ , such that  $\mathbf{E}\mathbf{F}^\infty p$  expresses the property above. The logic combining ECTL with the extension discussed before,  $\text{ECTL}^+$ , roughly corresponds to the logic CTF of [7].

The logic  $\text{CTL}^*$  of Emerson and Halpern extends  $\text{ECTL}^+$  with nesting of temporal operators as in  $\mathbf{E}\mathbf{G}(p \vee \mathbf{X}p)$  [9]. Satisfiability for  $\text{CTL}^*$ , and therefore for  $\text{ECTL}^+$ , is **2EXPTIME**-complete [26,11].

*Past Modalities.* While being common in linguistics and philosophy, past modalities are mostly viewed only as means to make specification more intuitive in computer science. For a discussion of this issue and of the possible different semantics of past modalities, we refer to [16,21]. We adopt the view of a linear, finite, and cumulative past, which is reflected in our definition of semantics of branching-time logics based on computation trees.

We use PCTL to refer to the extension of CTL with the past counterparts of the CTL temporal operators, and likewise for other logics. While PCTL is strictly more expressive than CTL [16], this is not the case for  $\text{PCTL}^*$  and  $\text{CTL}^*$  [14,20]. In both cases, past modalities do not increase the complexity: PCTL is **EXPTIME**-complete [16] and  $\text{PCTL}^*$  has recently been shown to be **2EXPTIME**-complete by Bozzelli [3].

*Forgettable Past.* Once past modalities are available, restricting their scope is a natural way to facilitate their use in specification. To this end, Laroussinie and Schnoebelen introduced a new operator **N** for “from now on” to forget about

---

<sup>1</sup> The *complexity of a logic* always refers to the complexity of its satisfiability problem.

the past [20]. I.e., past modalities in the scope of a **N**-operator do not reach further back than the point where the **N**-operator was applied. For results on the expressive power of this operator, see [20].

Satisfiability for the extension of PCTL with the **N**-operator, PCTL+**N**, was claimed to be in **EXPTIME** by Laroussinie and Schnoebelen [21]. In contrast to this, a nonelementary lower bound for PCTL\*+**N** was shown in [28]. Nevertheless, the latter logic is known to be no more expressive than CTL\* [20].

The logic PECTL<sup>+</sup>+**N**, i.e., CTL with all the extensions considered here, also has the same expressive power as CTL\* [20]. But the proof uses the separation result of Gabbay for linear temporal logic [12], causing a nonelementary blow-up. No elementary upper bound for the complexity of satisfiability for PECTL<sup>+</sup>+**N** is known so far.

In this paper, we completely classify the complexity of satisfiability for all branching-time logics obtained from UB and CTL by combination of the extensions discussed above. In detail, we obtain the following results:

- We show that satisfiability for all of these logics that allow Boolean combinations of path formulas is **2EXPTIME**-complete, improving the known lower bound for CTL<sup>+</sup> to UB<sup>+</sup>.
- Likewise, we show that all logics with forgettable past are **2EXPTIME**-hard, even if only the past modality **P** for “somewhere in the past” is allowed. This contradicts the claim of Laroussinie and Schnoebelen of **EXPTIME** membership for PCTL+**N** in [21].
- We show that all logics that include neither Boolean combinations of path formulas nor forgettable past are in **EXPTIME**.
- Finally, we show a **2EXPTIME** upper bound for PECTL<sup>+</sup>+**N**, i.e., for CTL with all the considered extensions. This strongly contrasts but does not contradict the nonelementary complexity of PCTL\*+**N**, although both logics are equally expressive.

The upper bounds are obtained by translation into alternating tree automata [25]. To prove the upper bound for PECTL<sup>+</sup>+**N**, we introduce the model of *k*-weak-pebble hesitant alternating tree automata and show that their nonemptiness problem is in **2EXPTIME**. These automata differ from the one-pebble alternating Büchi tree automata of [28] in two respects. First, they use a different acceptance condition to handle fairness, as proposed by Kupferman, Vardi, and Wolper for an automata model for CTL\* model checking [18]. Second, the model allows more than one pebble, but only of a *weak* kind. These pebbles are used to handle forgettable past and Boolean combinations of path formulas.

Due to lack of space all proofs are omitted, except for some proof sketches. They are available in the full version of the paper [27].

## Note

Tragically, Volker Weber died in the night of April 6-7, 2009, right after submitting this paper to CSL 2009. An obituary is printed in the proceedings. Most of the remarks by the reviewers were incorporated by his advisor, Thomas

Schwentick. However, major changes were avoided (and had not been requested by the reviewers). The help of the reviewers is gratefully acknowledged.

## 2 Preliminaries

This section contains the definitions of branching-time logics and tree automata. Both are with respect to infinite trees, which we are going to define first.

A *tree* is a set  $T \subseteq \mathbb{N}^*$  such that if  $x \cdot c \in T$  with  $x \in \mathbb{N}^*$  and  $c \in \mathbb{N}$ , then  $x \in T$  and  $x \cdot c' \in T$  for all  $0 < c' < c$ . The empty string  $\varepsilon$  is the *root* of  $T$  and for all  $c \in \mathbb{N}$ ,  $x \cdot c \in T$  is called a *child* of  $x$ . The parent of a node  $x$  is sometimes denoted by  $x \cdot -1$ . We use  $T_x := \{y \in \mathbb{N}^* \mid x \cdot y \in T\}$  to denote the *subtree* rooted at the node  $x \in T$ . The branching degree  $\text{deg}((x))$  is the number of children of a node  $x$ . Given a set  $D \subseteq \mathbb{N}$ , a  $D$ -tree is a computation tree such that  $\text{deg}((x)) \in D$  for all nodes  $x$ .

A *path*  $\pi$  in  $T$  is a prefix-closed minimal set  $\pi \subseteq T$ , such that for every  $x \in \pi$ , either  $x$  has no child or there is a unique  $c \in \mathbb{N}$  with  $x \cdot c \in \pi$ . We use “ $\leq$ ” (“ $<$ ”) to denote the (strict) ancestor-relation on  $T$ .

A *labeled tree*  $(T, V)$  over a finite alphabet  $\Sigma$  consists of a tree  $T$  and a labeling function  $V : T \rightarrow \Sigma$ , assigning a symbol from  $\Sigma$  to every node of  $T$ . We are mainly interested in the case where  $\Sigma = 2^{\text{PROP}}$  for some set PROP of propositions. Such *computation trees* result from the unfolding of Kripke structures. In the following, we consider only computation trees and refer to them as trees. We identify  $(T, V)$  with  $T$ .

### 2.1 Branching-Time Logics

We shortly define the branching-time logics we are going to study. These definitions are mainly standard.

We start by defining the logic incorporating all the extensions discussed in the introduction. The *state formulas*  $\varphi$  and *path formulas*  $\psi$  of  $\text{PECTL}^+ + \mathbf{N}$  are given by the following rules:

$$\begin{aligned} \varphi &::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{E}\psi \mid \mathbf{N}\varphi \\ \psi &::= \varphi \mid \psi \wedge \psi \mid \neg\psi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi \mid \mathbf{F}^\infty\varphi \mid \mathbf{Y}\varphi \mid \varphi\mathbf{S}\varphi \end{aligned}$$

where  $p \in \text{PROP}$  for some set of propositional symbols PROP.  $\text{PECTL}^+ + \mathbf{N}$  is the set of all state formulas generated by these rules.

We use the usual abbreviations **true**, **false**,  $\varphi \vee \varphi$ ,  $\varphi \rightarrow \varphi$ ,  $\varphi \leftrightarrow \varphi$ , and

$$\begin{aligned} \mathbf{A}\psi &::= \neg\mathbf{E}\neg\psi & \mathbf{F}\varphi &::= \mathbf{true}\mathbf{U}\varphi & \mathbf{G}\varphi &::= \neg\mathbf{F}\neg\varphi \\ \mathbf{G}^\infty\varphi &::= \neg\mathbf{F}^\infty\neg\varphi & \mathbf{P}\varphi &::= \mathbf{true}\mathbf{S}\varphi & \mathbf{H}\varphi &::= \neg\mathbf{P}\neg\varphi \end{aligned}$$

The semantics of  $\text{PECTL}^+ + \mathbf{N}$  is defined with respect to a computation tree  $T$ , a node  $x \in T$ , and, in case of a path formula, a path  $\pi$  in  $T$  starting at the root of  $T$ . We omit the rules for propositions and Boolean connectives.

$$\begin{array}{ll}
T, x \models \mathbf{E}\psi & \text{iff there exists a path } \pi \text{ in } T, \text{ such that } x \in \pi \text{ and } T, \pi, x \models \psi \\
T, x \models \mathbf{N}\varphi & \text{iff } T_x, \varepsilon \models \varphi \\
T, \pi, x \models \varphi & \text{for a state formula } \varphi, \text{ iff } T, x \models \varphi \\
T, \pi, x \models \mathbf{X}\varphi & \text{iff } T, \pi, x \cdot c \models \varphi, \text{ where } c \in D \text{ and } x \cdot c \in \pi \\
T, \pi, x \models \varphi_1 \mathbf{U}\varphi_2 & \text{iff there is a node } y \geq x \text{ in } \pi, \text{ such that } T, \pi, y \models \varphi_2 \\
& \text{and for all } x \leq z < y \text{ we have } T, \pi, z \models \varphi_1 \\
T, \pi, x \models \mathbf{F}^\infty \varphi & \text{iff there are infinitely many nodes } y \in \pi \text{ such that } T, \pi, y \models \varphi \\
T, \pi, x \models \mathbf{Y}\varphi & \text{iff } x \neq \varepsilon \text{ and } T, \pi, x \cdot -1 \models \varphi \\
T, \pi, x \models \varphi_1 \mathbf{S}\varphi_2 & \text{iff there is a node } y \leq x \text{ in } \pi, \text{ such that } T, \pi, y \models \varphi_2 \\
& \text{and for all } y < z \leq x \text{ we have } T, \pi, z \models \varphi_1
\end{array}$$

A formula  $\varphi$  is called *satisfiable* if there is a tree  $T$  such that  $T, x \models \varphi$ .

All other logics we consider are syntactical fragments of  $\text{PECTL}^+ + \mathbf{N}$ .

$$\begin{array}{ll}
\text{UB}^+ & \varphi := p \mid \varphi \wedge \varphi \mid \neg \varphi \mid \mathbf{E}\psi \\
& \psi := \varphi \mid \psi \wedge \psi \mid \neg \psi \mid \mathbf{X}\varphi \mid \mathbf{F}\varphi \\
\text{UB} + \mathbf{P} + \mathbf{N} & \varphi := p \mid \varphi \wedge \varphi \mid \neg \varphi \mid \mathbf{E}\mathbf{X}\varphi \mid \mathbf{E}\mathbf{F}\varphi \mid \mathbf{A}\mathbf{F}\varphi \mid \mathbf{P}\varphi \mid \mathbf{N}\varphi \\
\text{PECTL} & \varphi := p \mid \varphi \wedge \varphi \mid \neg \varphi \mid \mathbf{E}\mathbf{X}\varphi \mid \mathbf{E}(\varphi \mathbf{U}\varphi) \mid \mathbf{A}(\varphi \mathbf{U}\varphi) \mid \mathbf{E}\mathbf{F}^\infty \varphi \mid \mathbf{Y}\varphi \mid \varphi \mathbf{S}\varphi
\end{array}$$

## 2.2 Weak-Pebble Automata

We introduce alternating tree automata equipped with a weak kind of pebbles. We call these pebbles *weak* as they can only be used to mark a node while the automaton inspects the subtree below<sup>2</sup>. In particular, a weak-pebble automaton can only see the last pebble it dropped.

For a given set  $X$ , we use  $\mathcal{B}^+(X)$  to denote the set of *positive Boolean formulas over  $X$* , i.e., formulas built from **true**, **false** and the elements of  $X$  by  $\wedge$  and  $\vee$ . A subset  $Y \subseteq X$  *satisfies* a boolean formula  $\alpha \in \mathcal{B}^+(X)$  if and only if assigning **true** to the elements in  $Y$  and **false** to the elements in  $X \setminus Y$  makes  $\alpha$  true.

**Definition 2.1.** A *k-weak-pebble alternating tree automaton (k-WPAA)* is a tuple  $A = (Q, \Sigma, D, q^0, \delta, F)$ , such that  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet,  $D$  a finite set of arities,  $q^0 \in Q$  is the initial state,  $F$  is an acceptance condition, and  $\delta$  is a transition function

$$\delta : Q \times \Sigma \times D \times \mathbb{B} \rightarrow (\{\text{drop, lift}\} \times Q) \cup \mathcal{B}^+((D \cup \{-1, 0, \text{root}\}) \times Q)$$

such that  $\delta(q, \sigma, d, \text{false}) \neq (\text{lift}, p)$ , no Boolean combination  $\delta(q, \sigma, d, b)$  contains any  $(d', p)$  with  $d' \in D$  and  $d' > d$ , and no Boolean combination  $\delta(q, \sigma, d, \text{true})$  contains any  $(-1, p)$ .

<sup>2</sup> A similar restriction on pebbles was considered in [23].

In the following definition of the semantics of a  $k$ -WPAA, we will use tuples  $\bar{y} = (y_1, \dots, y_k) \in (D^* \cup \{\perp\})^k$ , called pebble placements, to denote the positions of the pebbles, where “ $\perp$ ” means that the pebble is not placed. As  $k$ -WPAA will be restricted to use their pebbles in a stack-wise fashion, pebble 1 being the first pebble to be placed, there will always be an  $i \in [1, k]$ , such that  $y_j \neq \perp$  for all  $j \leq i$  and  $y_j = \perp$  for all  $j > i$ . I.e.,  $i$  is the maximal index of a placed pebble and we will refer to it by  $\text{mpp}(\bar{y})$ . Note that  $\text{mpp}(\bar{y}) = 0$  if and only if no pebble is placed and that  $y_{\text{mpp}(\bar{y})}$  is the position of the last placed pebble otherwise.

**Definition 2.2.** A configuration  $(q, x, \bar{y}) \in Q \times D^* \times (D^* \cup \{\perp\})^k$  of a  $k$ -WPAA  $A = (Q, \Sigma, D, q^0, \delta, F)$  consists of a state, the current position in the tree, and the positions of the pebbles.

A run  $r$  of  $A$  on a  $\Sigma$ -labeled  $D$ -tree  $(T, V)$  is a  $\mathbb{N}$ -tree  $(T', V')$ , whose nodes are labeled by configurations of  $A$  and that is compatible with the transition function. More precisely, the root of  $T'$  must be labeled by  $(q^0, \varepsilon, \bar{y})$  with  $\text{mpp}(\bar{y}) = 0$ , and for every node  $v \in T'$  labeled by  $(q, x, \bar{y})$  the following conditions depending on  $\delta$  hold, where  $d := \text{deg}(x)$  and  $b = \text{true}$  if and only if  $y_{\text{mpp}(\bar{y})} = x$ .

$\delta(q, V(x), d, b) = (\text{drop}, p)$ : If  $\text{mpp}(\bar{y}) < k$ , then  $v$  has a child labeled with  $(q, x, \bar{y}')$ , where  $y'_{\text{mpp}(\bar{y})+1} = x$  and  $y'_j = y_j$  for all  $j \neq \text{mpp}(\bar{y})+1$ . Otherwise, i.e., if all pebbles are already placed, the transition cannot be applied.

$\delta(q, V(x), d, b) = (\text{lift}, p)$ : By Definition 2.1,  $b = \text{true}$  and therefore  $y_{\text{mpp}(\bar{y})} = x$ . Then  $v$  has a child labeled with  $(q, x, \bar{y}')$ , where  $y'_{\text{mpp}(\bar{y})} = \perp$  and  $y'_j = y_j$  for all  $j \neq \text{mpp}(\bar{y})$ .

$\delta(q, V(x), d, b) = \alpha$  for a boolean combination  $\alpha \in \mathcal{B}^+((D \cup \{-1, 0, \text{root}\}) \times Q)$ : There has to be a set  $Y \subseteq (D \cup \{-1, 0, \text{root}\}) \times Q$ , such that  $Y$  satisfies  $\alpha$ , and, for every  $(c, p) \in Y$ , there is a child  $v \cdot c$  of  $v$  in  $T'$  such that  $v \cdot c$  is labeled by  $(p, x \cdot c, \bar{y})$ , where  $x \cdot 0$  and  $x \cdot \text{root}$  denote the node  $x$  itself. Additionally, we require that  $Y$  does not contain a tuple  $(-1, q)$  if  $x = \varepsilon$  and that  $Y$  contains a tuple  $(\text{root}, q)$  only if  $x = \varepsilon$ .

We call a run  $r$  *accepting* if every infinite path of  $r$  satisfies the acceptance condition and every finite path ends in a configuration where a transition to the Boolean combination  $\text{true}$  applies. A labeled tree  $(T, V)$  is *accepted* by  $A$  if and only if there is an accepting run of  $A$  on  $(T, V)$ . The *language* of  $A$  is the set of trees accepted by  $A$  and denoted  $L(A)$ .

**Definition 2.3.** A *symmetric  $k$ -weak-pebble alternating tree automaton* is a tuple  $A = (Q, \Sigma, q^0, \delta, F)$ , such that  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet,  $q^0 \in Q$  is the initial state,  $F$  is an acceptance condition, and

$$\delta : Q \times \Sigma \times \mathbb{B} \rightarrow (\{\text{drop}, \text{lift}\} \times Q) \cup \mathcal{B}^+((\{\square, \diamond, -1, 0, \text{root}\}) \times Q)$$

is a transition function such that  $\delta(q, \sigma, \text{false}) \neq (\text{lift}, p)$  and  $\delta(q, \sigma, \text{true})$  does not contain any  $(-1, p)$ .

The semantics of a symmetric  $k$ -WPAA are defined as for (nonsymmetric)  $k$ -WPAA, except for the last case,  $\delta(q, V(x), d, b) = \alpha$ , where we require that for

every tuple  $(\diamond, p) \in Y$  there is child of  $v$  in  $T'$  labeled by  $(p, x \cdot c)$  for some child  $x \cdot c$  of  $x$  in  $T$ , and for every tuple  $(\square, p) \in Y$  and every child  $x \cdot c$  of  $x$  in  $T$ , there is child of  $v$  in  $T'$  labeled by  $(p, x \cdot c)$ . The conditions on tuples  $(-1, p)$ ,  $(0, p)$ , and  $(\text{root}, p)$  remain unchanged.

So far, we have not specified the acceptance conditions for our automata. For our purposes, hesitant alternating tree automata as introduced by Kupferman, Vardi, and Wolper are a good choice [18]. They allow for an easy translation from CTL\* [18] and have proved useful in studies of CTL\* with past [17,3].

A *k-weak-pebble hesitant alternating tree automaton* (*k-WPHAA* for short)  $A = (Q, \Sigma, D, q^0, \delta, F)$  is a *k-WPAA* with  $F = \langle G, B \rangle$ ,  $G, B \subseteq Q$ , that satisfies the following conditions:

- There exists a partition of  $Q$  into disjoint sets  $Q_1, \dots, Q_m$  and every set  $Q_i$  is classified as either *existential*, *universal*, or *transient*.
- There exists a partial order  $\leq$  between the sets  $Q_i$ , such that every transition from a state in  $Q_i$  leads to states contained either in the same set  $Q_i$  or in a set  $Q_j$  with  $Q_j < Q_i$ .
- If  $q \in Q_i$  for a transient set  $Q_i$ , then  $\delta(q, \sigma, d, b)$  contains no state from  $Q_i$ .
- If  $Q_i$  is an existential set,  $q \in Q_i$  and  $\delta(q, \sigma, d, b) = \alpha$ , then  $\alpha$  contains only disjunctively related tuples with states from  $Q_i$ .
- If  $Q_i$  is a universal set,  $q \in Q_i$  and  $\delta(q, \sigma, d, b) = \alpha$ , then  $\alpha$  contains only conjunctively related tuples with states from  $Q_i$ .

Every infinite path  $\pi$  in a run of a *k-WPHAA* gets trapped in an existential or a universal set  $Q_i$ . The acceptance condition  $\langle G, B \rangle$  is satisfied by  $\pi$ , if either  $Q_i$  is existential and  $\text{inf}(\pi) \cap G \neq \emptyset$ , or  $Q_i$  is universal and  $\text{inf}(\pi) \cap B = \emptyset$ , where  $\text{inf}(\pi)$  denotes the set of states that occur infinitely often on  $\pi$ .

We also consider *symmetric k-WPHAA*. Here, we additionally require that for every existential (resp. universal) set  $Q_i$  and every state  $q \in Q_i$ ,  $\delta(q, \sigma, d, b)$  does not contain a tuple  $(\square, p)$  (resp.  $(\diamond, p)$ ) with  $p \in Q_i$ .

The size of an automaton is defined as the sum of the sizes of its components. Note that this includes the size of  $D$  in the case of nonsymmetric automata.

If we remove the pebbles from our automata, we obtain (*symmetric*) *two-way hesitant alternating tree automata*. Such an automaton has a transitions function of the form  $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\{\square, \diamond, -1, 0, \text{root}\} \times Q)$  in the symmetric case and is obtained from the above definitions in a straightforward way.

Symmetric two-way HAA have been used by Bozzelli to prove membership in **2EXPTIME** for CTL\* with past [3]. Opposed to the definition given there, we do not enforce that infinite paths in a run move only downward in the tree from a certain point on. Therefore, our results on symmetric two-way HAA are not implied by [3]. Nevertheless, the restricted version of [3] would suffice to prove our results on the complexity of branching-time logics.

### 3 Complexity of Satisfiability

We give a complete classification of the complexity of the satisfiability problem for all branching-time logics obtained from UB and CTL by any combination of

the extensions discussed above. As our main theorem, we prove **2EXPTIME**-completeness for all logics including forgettable past or Boolean combinations of path formulas.

**Theorem 3.1.** *The satisfiability problems for all branching-time logics that are a syntactical fragment of  $\text{PECTL}^+ + \text{N}$  and that syntactically contain  $\text{UB}^+$  or  $\text{UB} + \text{P} + \text{N}$  are complete for **2EXPTIME**.*

The upper bound is proved in Section 3.2 with the help of weak-pebble alternating tree automata. There, we also show that satisfiability for all remaining logics is in **EXPTIME**.

The lower bounds on  $\text{UB} + \text{P} + \text{N}$  and  $\text{UB}^+$  are proved next.

### 3.1 Lower Bounds

We obtain both results by reduction from the  $2^n$ -corridor tiling game, which is based on the  *$2^n$ -corridor tiling problem*. An instance  $I = (T, H, V, F, L, n)$  of the latter problem consists of a finite set  $T$  of tile types, horizontal and vertical constraints  $H, V \subseteq T \times T$ , constraints  $F, L \subseteq T$  on the first and the last row, and a number  $n$  given in unary. The task is to decide, whether  $T$  tiles the  $2^n \times m$ -corridor for some number  $m$  of rows, respecting the constraints. We assume w.l.o.g. that there is always a possible next move for both players.

The game version of this problem corresponds to alternating Turing Machines: The  *$2^n$ -corridor tiling game* is played by two players  $E$  and  $A$  on an instance  $I$  of the  $2^n$ -corridor tiling problem. The players alternately place tiles row by row starting with player  $E$  and following the constraints, as the opponent wins otherwise.  $E$  wins the game if a row consisting of tiles from  $L$  is reached. To decide whether  $E$  has a winning strategy in such a game is complete for **AEXPSPACE** [4], which is the same as **2EXPTIME**.

**Proposition 3.2.** *Satisfiability for  $\text{UB} + \text{P} + \text{N}$  is **2EXPTIME**-hard.*

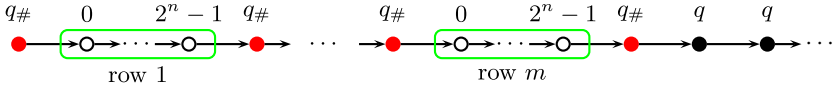
*Proof.* Given an instance  $I = (T, H, V, F, L, n)$  of the  $2^n$ -corridor tiling game, we build a  $\text{UB} + \text{P} + \text{N}$ -formulas  $\varphi_I$  that is satisfiable if and only if player  $E$  has a winning strategy on  $I$ .

We encode such a winning strategy as a finite  $T$ -labeled tree as described in [28]: The levels of the tree alternately correspond to moves of  $E$  and  $A$ , and every node representing a move of  $E$  has one child for every next move of  $A$ . As each player always has a possible move, the only way to win for  $E$  is to reach a line with tiles from  $L$ . Therefore, every path in the encoding has to represent a tiling respecting all constraints.

The formula  $\varphi_I = \varphi_s \wedge \varphi_n \wedge \varphi_t$  consists of three parts: The formula  $\varphi_s$  describes the structure of the encoding and  $\varphi_n$  introduces a numbering of the nodes representing one line of the tiling using the propositions  $q_0, \dots, q_{n-1}$  as shown in Figure 1. Both are  $\text{UB}$ -formulas and can be taken from [28].

The formula  $\varphi_t$  is used to describe the actual tiling. It states that every node corresponding to a position in the tiling is labeled with exactly one proposition





**Fig. 1.** A path in the encoding of a winning strategy for the  $2^n$ -corridor tiling game with  $m$  rows [28]

$p_t$ , representing the tile  $t \in T$ , and that all constraints are respected. We use  $\vartheta$  as an abbreviation for  $\neg q \wedge \neg q_{\#} \wedge \neg \mathbf{P}(q_{\#} \wedge \mathbf{P}(\neg q_{\#} \wedge \mathbf{P}q_{\#}))$ , expressing that the current node represents a position in the tiling and that there is at most one row above. This, together with the  $\mathbf{N}$ -operator, will allow us to check the vertical constraints.

$$\begin{aligned} \varphi_t &= \mathbf{AG}([\neg q \wedge \neg q_{\#}] \rightarrow [\bigvee_{t \in T} (p_t \wedge \bigwedge_{t' \neq t' \in T} \neg p_{t'}) \wedge \theta_H \wedge \theta_V \wedge \theta_A]) \wedge \theta_{A'} \wedge \theta_F \wedge \theta_L \\ \theta_H &= \neg \bigwedge_{i=0}^{n-1} q_i \rightarrow \bigwedge_{t \in T} (p_t \rightarrow \mathbf{AX} \bigvee_{(t,t') \in H} p_{t'}) \\ \theta_V &= \mathbf{NAXAG}([\vartheta \wedge \bigwedge_{i=0}^{n-1} (q_i \leftrightarrow \mathbf{PH}q_i)] \rightarrow \bigwedge_{t \in T} (p_t \rightarrow \bigvee_{(t,t') \in V} \mathbf{PH}p_{t'})) \end{aligned}$$

We omit the formulas corresponding to the constraints  $F$  and  $L$ .

The subformulas  $\theta_A$  and  $\theta_{A'}$  enforce that every possible move of  $A$  is represented, where  $\theta_{A'}$  treats the special case of the first row of the tiling.

$$\begin{aligned} \theta_A &= q_0 \rightarrow \mathbf{NAXAG}([\vartheta \wedge \neg q_0 \wedge \bigwedge_{i=1}^{n-1} (q_i \leftrightarrow \mathbf{PH}q_i)] \\ &\quad \rightarrow \bigwedge_{t \in T} (p_t \rightarrow \bigwedge_{(t,t') \in H} [\mathbf{EX}p_{t'} \vee \mathbf{PH} \bigvee_{(t'',t') \notin V} p_{t''}])) \\ \theta_{A'} &= \mathbf{AG}([\neg q_0 \wedge \neg \mathbf{P}(q_{\#} \wedge \mathbf{P}\neg q_{\#})] \rightarrow \bigwedge_{t \in T} (p_t \rightarrow \bigwedge_{(t,t') \in H} \mathbf{EX}p_{t'})) \end{aligned}$$

Note that a model for  $\varphi_I$  might encode several possible moves for  $E$  and moves of  $E$  and  $A$  might be represented more than once. But by removing duplicates and restricting to one arbitrary move for  $E$  at every position where  $E$  has to move, we obtain a winning strategy for  $E$  on  $I$  from a model for  $\varphi_I$ . For the reverse direction, a winning strategy for  $E$  can be directly turned into a model for  $\varphi_I$ . □

Concerning Boolean combinations of path formulas, we can refine the proof of **2EXPTIME**-hardness for  $\text{CTL}^+$  by Johannsen and Lange [15] to show the following theorem.

**Proposition 3.3.** *Satisfiability for  $\text{UB}^+$  is **2EXPTIME**-hard.*

The proof is by reduction from the  $2^n$ -corridor tiling game. The main idea is to use a numbering of the rows modulo three to able to express that a path reaches up to the next row, but not beyond, without the **U**-operator. The details can be found in the full version.

### 3.2 Upper Bound

We prove the upper bound of Theorem 3.1 in two steps. First, we show how to translate a PECTL-formula into a two-way hesitant alternating tree automaton, thereby giving an exponential time algorithm for PECTL-satisfiability. Afterwards, we extend this construction to  $\text{PECTL}^+ + \mathbf{N}$  and weak-pebble automata.

**Theorem 3.4.** *The satisfiability problem for PECTL is EXPTIME-complete.*

*Proof.* The lower bound follows from the EXPTIME-hardness of CTL. To prove the upper bound, we extend the construction from [25] that translates a given CTL-formulas into an alternating Büchi tree automaton.

Let  $\varphi$  be a PECTL-formula and  $\text{PROP}_\varphi$  the set of proposition symbols occurring in  $\varphi$ . We construct a symmetric two-way hesitant alternating tree automata  $\mathfrak{A}_\varphi = (Q, \Sigma, q^0, \delta, \langle G, B \rangle)$  with  $\Sigma = 2^{\text{PROP}_\varphi}$ , such that  $\varphi$  holds at the root of some  $\Sigma$ -labeled tree  $(T, V)$  if and only if  $\mathfrak{A}_\varphi$  accepts this tree. The result follows since nonemptiness for these automata is in EXPTIME (Theorem 4.6).

Let  $\bar{\psi}$  denote the dual of a formula  $\psi$ . The dual of a formula is obtained by switching  $\wedge$  and  $\vee$ , and by negating all other maximal subformulas, identifying  $\neg\neg\psi$  and  $\psi$ . E.g., the dual of  $p \vee (\neg q \wedge \mathbf{EG}p)$  is  $\neg p \wedge (q \vee \neg \mathbf{EG}p)$ .

The set  $Q$  of states of  $\mathfrak{A}_\varphi$  is based on the Fisher-Ladner-closure of  $\varphi$ . It contains  $\varphi$ ,  $(\mathbf{EXEF}^\infty \psi) \wedge \psi$  for every subformula  $\mathbf{EF}^\infty \psi$  of  $\varphi$ , and is closed under subformulas and negation. The initial state is  $\varphi$ . The set  $G$  contains all formulas of the form  $\neg \mathbf{E}(\chi \mathbf{U} \psi)$ ,  $\neg \mathbf{A}(\chi \mathbf{U} \psi)$ , and  $(\mathbf{EXEF}^\infty \psi) \wedge \psi$ , the set  $B$  all formulas of the form  $\neg \mathbf{EXEF}^\infty \psi$ . The transition function is defined as follows,

$$\begin{aligned}
 \delta(\text{true}, \sigma) &= \text{true} & \delta(p, \sigma) &= \text{true} & \text{if } p \in \sigma \\
 \delta(\text{false}, \sigma) &= \text{false} & \delta(p, \sigma) &= \text{false} & \text{if } p \notin \sigma \\
 \delta(\psi \wedge \xi, \sigma) &= (0, \psi) \wedge (0, \xi) & \delta(\neg\psi, \sigma) &= \overline{\delta(\psi, \sigma, b)} \\
 \delta(\mathbf{EX}\psi, \sigma) &= (\diamond, \psi) & \delta(\mathbf{Y}\psi, \sigma) &= (-1, \psi) \\
 \delta(\mathbf{EF}^\infty \psi, \sigma) &= (\diamond, \mathbf{EF}^\infty \psi) \vee (0, (\mathbf{EXEF}^\infty \psi) \wedge \psi) \\
 \delta(\mathbf{E}(\chi \mathbf{U} \psi), \sigma) &= (0, \psi) \vee ((0, \chi) \wedge (\diamond, \mathbf{E}(\chi \mathbf{U} \psi))) \\
 \delta(\mathbf{A}(\chi \mathbf{U} \psi), \sigma) &= (0, \psi) \vee ((0, \chi) \wedge (\square, \mathbf{A}(\chi \mathbf{U} \psi))) \\
 \delta(\chi \mathbf{S} \psi, \sigma) &= (0, \psi) \vee ((0, \chi) \wedge (-1, \chi \mathbf{S} \psi))
 \end{aligned}$$

where the notion of dual is extended to the transition function  $\delta$  in the obvious way, e.g.,  $\overline{\delta(\mathbf{E}(\chi \mathbf{U} \psi), \sigma)} = (0, \bar{\psi}) \wedge ((0, \bar{\chi}) \vee (\square, \neg \mathbf{E}(\chi \mathbf{U} \psi)))$ .

To show that  $\mathfrak{A}_\varphi$  is a hesitant automaton, we have to define the partition of  $Q$ . The formulas  $(\mathbf{EXEF}^\infty \psi) \wedge \psi$ ,  $\mathbf{EXEF}^\infty \psi$ , and  $\mathbf{EF}^\infty \psi$  form an existential set. Likewise,  $(\neg \mathbf{EXEF}^\infty \psi) \vee \neg\psi$ ,  $\neg \mathbf{EXEF}^\infty \psi$ , and  $\neg \mathbf{EF}^\infty \psi$  form a universal set. Every other formula  $\psi \in Q$  constitutes a singleton set  $\{\psi\}$ . These sets are all transient, except for the sets  $\{\mathbf{E}(\chi \mathbf{U} \psi)\}$  and  $\{\neg \mathbf{A}(\chi \mathbf{U} \psi)\}$ , which are existential,

and the sets  $\{\neg\mathbf{E}(\chi\mathbf{U}\psi)\}$  and  $\{\mathbf{A}(\chi\mathbf{U}\psi)\}$ , which are universal. The partial order on these sets is induced by the subformula relation.  $\square$

We extend this construction to prove our result on  $\text{PECTL}^+ + \mathbf{N}$ . To this end, we have to find a way to deal with the  $\mathbf{N}$ -operator and Boolean combinations of path formulas. As we will see, pebbles can be used to handle both.

To obtain the desired result, it is important<sup>3</sup> that the number of pebbles an automaton uses does not depend on the formula from which it is constructed. But if we translate a  $\text{PECTL}^+ + \mathbf{N}$ -formula into an equivalent pebble automaton along the lines of the above construction, the number of pebbles depends on the nesting depth of  $\mathbf{N}$ -operators and Boolean combinations of path formulas. To avoid this, we show that we can restrict to formulas with limited nesting. But there is a price we have to pay: We will only obtain an equisatisfiable formula/automaton but not an equivalent one as in the proof of Theorem 3.4. Nevertheless, this will suffice to obtain our complexity result.

We say that a  $\text{PECTL}^+ + \mathbf{N}$ -formula is in *normal form*, if it does not contain any nesting of  $\mathbf{N}$ -operators, all path quantifiers that are followed by a Boolean combination of path formulas are not nested and occur only in the scope of an  $\mathbf{N}$ -operator, and finally all Boolean combinations of path formulas are in negation normal form.

**Lemma 3.5.** *Every  $\text{PECTL}^+ + \mathbf{N}$ -formula  $\varphi$  can be efficiently transformed into a  $\text{PECTL}^+ + \mathbf{N}$ -formula  $\psi$  in normal form with  $|\psi| = O(|\varphi|)$ , such that  $\psi$  is satisfiable if and only if  $\varphi$  is satisfiable.*

We show that any  $\text{PECTL}^+$ -formula in normal form can be translated into a  $k$ -WPHAA with only two pebbles. This completes the proof of Theorem 3.1 as nonemptiness for these automata is in  $\mathbf{2EXPTIME}$  by Theorem 4.1.

**Lemma 3.6.** *Given a  $\text{PECTL}^+ + \mathbf{N}$ -formula  $\varphi$  in normal form, we can construct a symmetric 2-WPHAA  $\mathfrak{A}_\varphi$  of size  $O(|\varphi|)$ , such that  $L(\mathfrak{A}_\varphi) \neq \emptyset$  if and only if  $\varphi$  is satisfiable.*

*Proof.* We extend the proof of Theorem 3.4, showing how to use pebbles to handle the additional features of  $\text{PECTL}^+ + \mathbf{N}$ . Since  $\varphi$  is given in normal form, two pebbles will suffice.

The handling of the  $\mathbf{N}$ -operator is straightforward: We only have to drop the pebble and never lift it again. As  $k$ -WPHAA are not allowed to move above a pebble in a tree, the dropping of the pebble corresponds accurately to the meaning of the  $\mathbf{N}$ -operator.

<sup>3</sup> Comment by Thomas Schwentick: this remark might puzzle the reader in the light of the results of Section 4. In an earlier version of the paper, the upper bound on the branching width in Proposition 4.5 was  $2^{O(n^k)}$ , hence the need to bound the number  $k$  of pebbles. However, shortly before submission time, Volker discovered that this upper bound can be improved to  $2^{n^{2 \cdot (k+1)}}$ , thus resolving the need to bound the number of pebbles. Seemingly, he did not find time to fully adapt (and simplify) the paper accordingly.

The handling of Boolean combinations of path formulas is more involved and mainly a matter of synchronization. An automaton corresponding to the formula  $\mathbf{E}(\mathbf{F}^\infty p \wedge \mathbf{F}^\infty q)$  cannot simply split into two automata corresponding to  $\mathbf{F}^\infty p$  and  $\mathbf{F}^\infty q$ , respectively, as these two automata need to run on the same path in the tree. I.e., they have to be synchronized.

We will achieve synchronization using two different techniques. To this end, let  $\mathbf{E}\psi$  be a subformula of  $\varphi$  such that  $\psi$  is a Boolean combination of path formulas  $\rho_1, \dots, \rho_l$  and  $\pi$  a path on which we want to evaluate  $\psi$ . For some of the path formulas  $\rho_i$ , it is the case that if they hold on  $\pi$ , then a finite prefix suffices to witness this, e.g., if  $\rho_i = p\mathbf{U}q$ . For other path formulas we have to consider the whole, probably infinite path  $\pi$ . What thereof is the case depends on the temporal operator and whether it appears negated or not.

To evaluate  $\mathbf{E}\psi$  at a node  $u$ ,  $\mathfrak{A}_\varphi$  will first guess a finite prefix of a path. The intention is that this prefix can serve as a witness for all path formulas  $\rho_i$  that allow for a finite witness, either to show that they hold or that they do not hold on this path.

Those parts of  $\psi$  that refer to the finite prefix are now checked by  $\mathfrak{A}_\varphi$  while moving up the tree again. E.g., if  $\rho_i = \mathbf{F}p$ , then the subautomaton  $\mathfrak{A}_{\rho_i}$  corresponding to  $\rho_i$  will run upwards looking for a node  $v$  labeled by  $p$ . As there is only one path going upward in a tree, we get synchronization for free. But we have to make sure that  $v$  is a descendant of  $u$ . Therefore, the second pebble has to be dropped at  $u$  before the automaton starts to guess the finite prefix. This allows  $\mathfrak{A}_{\rho_i}$  to reject when it reaches the pebble position without having seen a state labeled by  $p$ . On the other hand, if  $\rho_i = \mathbf{F}\mathbf{F}p$ , it is important that the second pebble can be lifted again as the witness might be above  $u$  in this case.

We still have to synchronize those subautomata that correspond to path formulas talking about the infinite suffix of the path. But there are only two types of conditions left, namely those of the form  $\mathbf{G}\chi$  and those of the form  $\mathbf{F}^\infty\chi$ . We can easily see that if a path satisfies a positive Boolean combination of such conditions, then every suffix of this path does so as well. This allows us to deploy the following technique.

Roughly speaking, we want to reduce the satisfiability problem for  $\varphi$  to satisfiability over a restricted class of models, where the suffixes of witnessing paths for Boolean combinations of path formulas are labeled by additional propositions. More precisely, for a subformula  $\mathbf{E}\psi$  of  $\varphi$  we introduce a new propositional symbol  $p_\psi$  and add to  $\varphi$  the new conjunct  $\mathbf{A}\mathbf{G}(\neg p_\psi \vee \mathbf{E}\mathbf{G}p_\psi)$ . The automaton we are going to construct for this extended formula will, when evaluating  $\mathbf{E}\psi$ , work as follows: It will drop the pebble and guess a prefix of a path on which  $\psi$  is supposed to hold. But this prefix has to end in a node labeled by  $p_\psi$ . The conditions on this finite prefix can be checked as described above. For the conditions on the suffix, we use the labeling to synchronize the independent subautomata corresponding to conditions to be checked.

Note that we cannot guaranty that there is only one path labeled by  $p_\psi$ . But we can simply check that the conditions hold on all paths labeled by  $p_\psi$  as there is at least one such path. Please also note that this technique cannot be used for the conditions on the finite prefix of the path as the labeling is not allowed to depend on the node at which  $\mathbf{E}\psi$  is evaluated.  $\square$

## 4 Nonemptiness of Weak-Pebble Automata

The complexity of the nonemptiness problem for weak-pebble alternating tree automata is analyzed in this section.

**Theorem 4.1.** *The nonemptiness problem for (symmetric)  $k$ -weak-pebble hesitant alternating tree automata is complete for **2EXPTIME**.*

We prove this theorem for the case of nonsymmetric automata by reduction to the nonemptiness problem for Rabin tree automata [22,24]. Afterwards, we generalize the result to symmetric automata by observing that every symmetric  $k$ -WPHAA accepts a tree of bounded branching degree.

**Definition 4.2.** A *nondeterministic Rabin tree automaton* (NRA)  $\mathfrak{A}$  is a tuple  $(Q, \Sigma, D, q^0, \delta, F)$ , such that  $Q$  is a finite set of states,  $\Sigma$  is a finite alphabet,  $D$  a finite set of arities,  $q^0 \in Q$  is the initial state,  $F$  is a set  $\{\langle G_1, B_1 \rangle, \dots, \langle G_m, B_m \rangle\}$  with  $G_i, B_i \subseteq Q$ , and  $\delta : Q \times \Sigma \times D \rightarrow 2^{Q^*}$  is a transition function, such that  $\delta(q, \sigma, d) \subseteq Q^d$  for all  $q \in Q, \sigma \in \Sigma$ , and  $d \in D$ .

We omit the (straightforward) definition of a run. An infinite path  $\pi$  in a run of  $\mathfrak{A}$  satisfies the acceptance condition  $F = \{\langle G_1, B_1 \rangle, \dots, \langle G_m, B_m \rangle\}$  if and only if there exists a pair  $\langle G_i, B_i \rangle \in F$  such that  $\text{inf}(\pi) \cap G_i \neq \emptyset$  and  $\text{inf}(\pi) \cap B_i = \emptyset$ .

A run  $r$  on a tree  $T$  is accepting iff every infinite path of  $r$  satisfies the acceptance condition and we have  $\delta(q, V(x), 0) = \{\varepsilon\}$  for every finite path ending in a state  $q$  at a leaf  $x$  of  $T$ , where  $\varepsilon$  denotes the sequence of states of length 0.

The last part of the definition is nonstandard and used to avoid the requirement that every path of  $T$  is infinite. Note that  $\mathfrak{A}$  rejects a finite path if  $\delta(q, V(x), 0) = \emptyset$ .

The nonemptiness problem for these automata is **NP**-complete in general, but it can be solved in polynomial time if the number of tuples in the acceptance condition is bounded by a constant [11]. For our purposes, one tuple suffices.

**Proposition 4.3** ([11]). *The nonemptiness problem for nondeterministic Rabin tree automata whose acceptance condition contains only one tuple can be decided in polynomial time.*

Together with the following translation, this yields Theorem 4.1 for the case of nonsymmetric  $k$ -WPHAA.

**Lemma 4.4.** *For every  $k$ -WPHAA  $\mathfrak{A} = (Q_{\mathfrak{A}}, \Sigma, D, q_{\mathfrak{A}}^0, \delta_{\mathfrak{A}}, \langle G_{\mathfrak{A}}, B_{\mathfrak{A}} \rangle)$ , there is a nondeterministic Rabin tree automaton  $\mathfrak{B} = (Q_{\mathfrak{B}}, \Sigma, D, q_{\mathfrak{B}}^0, \delta_{\mathfrak{B}}, \{\langle G_{\mathfrak{B}}, B_{\mathfrak{B}} \rangle\})$ , such that  $L(\mathfrak{A}) = L(\mathfrak{B})$  and the number of states of  $\mathfrak{B}$  is at most doubly exponential in  $|Q_{\mathfrak{A}}|$ . Moreover,  $\mathfrak{B}$  can be constructed from  $\mathfrak{A}$  in exponential space.*

*Proof.* We start with the observation that we can restrict to homogeneous runs of  $\mathfrak{A}$ , i.e., to runs where  $\mathfrak{A}$  always behaves in the same way when being in the same configuration. Formally, we call a run  $r$  *homogeneous*, if whenever two nodes of  $r$  are labeled by the same configuration, then the set of labels occurring at their children is also the same. If  $\mathfrak{A}$  has an accepting run, it also has an accepting

homogeneous run. This follows immediately from the existence of memoryless winning strategies for two-player parity games on infinite graphs [10,29].

Next, we describe how to construct the automaton  $\mathfrak{B}$  from  $\mathfrak{A}$ . When running on a tree  $T$ ,  $\mathfrak{B}$  will guess a homogeneous run  $r$  of  $\mathfrak{A}$  and accept if and only if the guessed run  $r$  is accepting. Of course,  $\mathfrak{B}$  cannot guess  $r$  at once. Instead,  $\mathfrak{B}$  will guess at every node  $x \in T$  how  $\mathfrak{A}$  behaved at  $x$  during  $r$ . The consistency of these guesses has to be checked by  $\mathfrak{B}$ . Additionally,  $\mathfrak{B}$  has to check whether the guessed run is accepting.

To perform these tasks,  $\mathfrak{B}$  needs to maintain some information about the guessed run  $r$  of  $\mathfrak{A}$ . More precisely, the state taken by  $\mathfrak{B}$  at a node  $x \in T$  will contain several sets of states of  $\mathfrak{A}$  that describe  $r$  at  $x$ . Additionally, there will be some information that will be used to verify that all infinite paths in  $r$  going through  $x$  satisfy the acceptance condition of  $\mathfrak{A}$ . This information will not uniquely determine  $r$ , but it will be sufficient to ensure the existence of an accepting run. A description of the information  $\mathfrak{B}$  stores in its states along with a formal definition of  $\mathfrak{B}$  can be found in the full version.  $\square$

To transfer this result to the case of symmetric automata, we have to deal with the fact that these automata accept trees of arbitrary, even infinite branching degree. But we can show that a symmetric  $k$ -WPHAA always accepts a tree whose branching degree is at most exponential in the size of the automaton.

**Proposition 4.5.** *For every symmetric  $k$ -WPHAA  $\mathfrak{A}$  with  $n$  states: If  $L(\mathfrak{A}) \neq \emptyset$ , then  $\mathfrak{A}$  accepts a tree whose branching degree is at most  $2^{n^2 \cdot (k+1)}$ .*

This can be proved similarly to a corresponding result for symmetric alternating one-pebble Büchi automata in [28]. See the full version for details.

Now, we can adapt Lemma 4.4 to symmetric  $k$ -WPHAA simply by considering every possible branching degree smaller than the bound provided by Proposition 4.5. This yields the upper bound of Theorem 4.1 for symmetric  $k$ -WPHAA. The matching lower bound follows from the **2EXPTIME**-hardness of  $\text{PECTL}^+ + \text{N}$  via Lemma 3.6 and Proposition 4.5.

Reviewing the proof of Lemma 4.4, we observe that the resulting NRA  $\mathfrak{B}$  is only of exponential size if  $\mathfrak{A}$  does not use any pebbles, i.e., if  $\mathfrak{A}$  is a (symmetric) two-way HAA. This yields the following theorem, which has been proved before by Bozzelli for a slightly more restricted model [3].

**Theorem 4.6.** *The nonemptiness problem for (symmetric) two-way hesitant alternating tree automata is complete for **EXPTIME**.*

## 5 Conclusions

In this paper, we considered the branching-time logics UB and CTL and their extensions by Boolean combinations of path formulas, fairness, past modalities, and forgettable past. While we think that this set of extensions is a reasonable choice, there are certainly other extensions or restrictions, such as existential or universal fragments, that deserve attention.

We gave a complete classification of the complexity of the satisfiability problem for these logics, obtaining a dichotomy between **EXPTIME**-complete and **2EXPTIME**-complete logics. There are many open questions concerning the expressive power of these logics and the complexity of their model checking problems that should be addressed in future work.

## References

1. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
2. Ben-Ari, M., Pnueli, A., Manna, Z.: The temporal logic of branching time. *Acta Informatica* 20, 207–226 (1983)
3. Bozzelli, L.: The complexity of CTL\* + linear past. In: Amadio, R.M. (ed.) FOS-SACS 2008. LNCS, vol. 4962, pp. 186–200. Springer, Heidelberg (2008)
4. Chlebus, B.S.: Domino-tiling games. *J. Comput. Syst. Sci.* 32(3), 374–392 (1986)
5. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
6. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 995–1072. Elsevier, MIT Press (1990)
7. Emerson, E.A., Clarke, E.M.: Characterizing correctness properties of parallel programs using fixpoints. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 169–181. Springer, Heidelberg (1980)
8. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. Syst. Sci.* 30(1), 1–24 (1985)
9. Emerson, E.A., Halpern, J.Y.: “Sometimes” and “not never” revisited: on branching versus linear time temporal logic. *J. ACM* 33(1), 151–178 (1986)
10. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. In: Proc. of the 32nd FOCS 1991, pp. 368–377. IEEE, Los Alamitos (1991)
11. Emerson, E.A., Jutla, C.S.: The complexity of tree automata and logics of programs. *SIAM J. Comput.* 29(1), 132–158 (1999)
12. Gabbay, D.M.: The declarative past and imperative future: Executable temporal logic for interactive systems. In: Banieqbal, B., Pnueli, A., Barringer, H. (eds.) Temporal Logic in Specification. LNCS, vol. 398, pp. 409–448. Springer, Heidelberg (1989)
13. Grumberg, O., Veith, H. (eds.): 25 Years of Model Checking. LNCS, vol. 5000. Springer, Heidelberg (2008)
14. Hafer, T., Thomas, W.: Computation tree logic CTL\* and path quantifiers in the monadic theory of the binary tree. In: Ottmann, T. (ed.) ICALP 1987. LNCS, vol. 267, pp. 269–279. Springer, Heidelberg (1987)
15. Johannsen, J., Lange, M.: CTL<sup>+</sup> is complete for double exponential time. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 767–775. Springer, Heidelberg (2003)
16. Kupferman, O., Pnueli, A.: Once and for all. In: Proc. of the 10th LICS 1995, pp. 25–35. IEEE, Los Alamitos (1995)
17. Kupferman, O., Vardi, M.Y.: Memoryful branching-time logic. In: Proc. of the 21st LICS 2006, pp. 265–274. IEEE, Los Alamitos (2006)
18. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. *J. ACM* 47(2), 312–360 (2000)

19. Lamport, L.: “Sometimes” is sometimes “not never”. In: Proc. of the 7th POPL 1980, pp. 174–185. ACM Press, New York (1980)
20. Laroussinie, F., Schnoebelen, P.: A hierarchy of temporal logics with past. *Theor. Comput. Sci.* 148(2), 303–324 (1995)
21. Laroussinie, F., Schnoebelen, P.: Specification in CTL+Past for verification in CTL. *Inf. Comput.* 156(1-2), 236–263 (2000)
22. Rabin, M.O.: Decidability of second order theories and automata on infinite trees. *Trans. AMS* 141, 1–35 (1969)
23. ten Cate, B., Segoufin, L.: XPath, transitive closure logic, and nested tree walking automata. In: Proc. of the 27th PODS 2008, pp. 251–260. ACM Press, New York (2008)
24. Thomas, W.: Automata on infinite objects. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, vol. B, pp. 133–192. Elsevier, MIT Press (1990)
25. Vardi, M.Y.: Alternating automata and program verification. In: van Leeuwen, J. (ed.) *Computer Science Today*. LNCS, vol. 1000, pp. 471–485. Springer, Heidelberg (1995)
26. Vardi, M.Y., Stockmeyer, L.J.: Improved upper and lower bounds for modal logics of programs. In: Proc. of the 17th STOC 1985, pp. 240–251. ACM Press, New York (1985)
27. Weber, V.: On the complexity of branching-time logics. Available from arXiv:0906.2521 [cs.LO]
28. Weber, V.: Branching-time logics repeatedly referring to states. Accepted to JoLLI (2009); An extended abstract appeared in the proceedings of HyLo (2007)
29. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata and infinite trees. *Theoretical Computer Science* 200, 135–183 (1998)

## Obituary Notice by Thomas Schwentick

We mourn the loss of Volker Weber, who died suddenly and unexpectedly on the 7th of April 2009. He was 30 years old. Volker began his scientific work at the faculty of Mathematics and Computer Science of the Philipps-Universität in Marburg. The main results of his masters thesis were published in a leading theoretical Computer Science conference.

Since October 2005, Volker worked as a researcher and PhD student at the Technische Universität Dortmund. His dissertation dealt with the complexity and expressiveness of hybrid logics, and was at the time of his death close to completion. Volker had fast become one of the leading experts in the field of hybrid logics. The scientific community knew him as a talented but modest researcher. Interacting with him was both personally and scientifically enriching.

Furthermore, Volker was actively involved in the self-administration of the faculty. In particular, he always had an open ear for the opinions of students and worked hard in various committees to improve the teaching offered by the faculty. Not least, Volker was an inspiring and committed teacher of Computer Science.

We will miss him deeply, as a friend, as a researcher and as a colleague.



# Nominal Domain Theory for Concurrency

David Turner and Glynn Winskel

University of Cambridge Computer Laboratory

**Abstract.** This paper investigates a methodology of using FM (Fraenkel-Mostowski) sets, and the ideas of nominal set theory, to adjoin name generation to a semantic theory. By developing a domain theory for concurrency within FM sets the domain theory inherits types and operations for name generation, essentially without disturbing its original higher-order features. The original domain theory had a metalanguage HOPLA (Higher Order Process Language) and accordingly this expands to a metalanguage, Nominal HOPLA, with name generation (closely related to an earlier language new-HOPLA). Nominal HOPLA possesses an operational and denotational semantics which are related via soundness and adequacy results, again carried out within FM sets.

## Introduction

Fraenkel-Mostowski (FM) set theory provided an early example of a set theory violating the Axiom of Choice (AC). It did this by building a set theory around a basic set of finitely permutable atoms  $\mathbb{A}$ . Functions had to respect the permutability of atoms, which was sufficient to disallow functions required to fulfill AC. Atoms share the same properties as names in computer science. Most often the precise nature of names is unimportant; what matters is their ability to identify and their distinctness. For this reason FM set theory has begun to play a foundational role in computer science, especially in syntax, making formal previously informal and often inaccurate assumptions about, for example, the freshness of variables in substitution [2,3]. This paper turns FM set theory to the problem of adjoining names and name generation to a semantic theory, a domain theory for concurrency.

At heart what makes FM set theory important for treating names are adjunctions associated with new-name abstraction. The simplest and best-known adjunction, implicit in [3], is for the category of nominal sets (those FM sets which remain invariant under all finite permutations of names). Its right adjoint  $\delta$  constructs a form of function space consisting of ‘new-name abstractions’. Closely related though less well-known are the adjunctions in FM sets on which this paper hinges. Here the associated functors can only be defined locally w.r.t. the sets of names involved.

Importantly, aside from these name features, FM set theory behaves much like more familiar set theories such as ZF, which is invaluable in transferring developments in a name-free setting into FM sets. For us it will mean that a path-based domain theory for concurrency can be *systematically* extended with name generation by working within FM set theory.

In the domain theory a process denotes a set of paths in a path order, specifying the type of computations it can do. Path sets provide a fully-abstract denotational semantics for the higher order process language HOPLA [4]. HOPLA was extended with name generation to a language new-HOPLA, able to express for example the pi-Calculus, Higher-Order pi-Calculus and mobile ambients [8]. But providing a denotational semantics was problematic. With the then standard way to adjoin name generation to a category of domains, by moving to a functor category, indexing both processes and their types by the current set of names, it became difficult to show that enough function spaces existed (there is an error in [6]). These problems are obviated by working within FM set theory. The way is open to developing more complicated semantics, such as that based on presheaves over path categories, within FM sets [9].

## 1 FM Sets

We provide a brief introduction to Fraenkel-Mostowski (FM) sets [2,3].

Fix an infinite set of names (or ‘atoms’) written  $\mathbb{A}$ . A *finite permutation* of  $\mathbb{A}$  is a permutation  $\sigma$  of  $\mathbb{A}$  such that  $\sigma a \neq a$  for only finitely many  $a \in \mathbb{A}$ . The collection of all finite permutations of  $\mathbb{A}$  forms a group. The group is generated by all transpositions  $(ab)$  which swap a name  $a$  and a name  $b$ .

Imagine building a hierarchy of sets as in ZF, but starting from  $\mathbb{A}$  rather than the empty set. The permutation action on the collection of atoms induces a permutation action  $\cdot$  on the hierarchy of elements by  $\in$ -recursion, giving rise to a notion of support. A set  $s \subseteq \mathbb{A}$  *supports* the element  $x$  if for any finite permutation  $\sigma$  such that  $\sigma a = a$  for all  $a \in s$  it is also the case that  $\sigma \cdot x = x$ . If  $x$  has a finite support then it has a smallest finite support, written  $\text{supp}(x)$ . The FM sets are defined to be those elements with hereditarily finite support.

The collection of all FM sets and finitely-supported functions forms a category **FMSet** which has subcategories **FMSet<sub>s</sub>** comprising sets and functions all of whose supports are contained in the finite set of names  $s$ . The subcategory **NSet** ( $= \mathbf{FMSet}_\emptyset$ ) of *nominal* sets consists of those FM sets and functions with empty support.

FM sets allow the usual operations of set theory, though with the proviso that elements must always have finite support. In addition there are important operations associated with names. The binary predicate  $x \# y$  expresses that two FM sets  $x$  and  $y$  have disjoint supports. If  $f : \mathbb{A} \rightarrow X$  is a finitely-supported function and  $X$  is a FM set then **fresh  $a$  in  $f$**  denotes the unique  $x \in X$  such that  $f a = x$  for any  $a \in \mathbb{A}$  such that  $a \# \langle f, f a \rangle$  as long as such an  $a \in \mathbb{A}$  exists. When  $X = \{\top, \perp\}$  then  $f : \mathbb{A} \rightarrow X$  is a predicate on  $\mathbb{A}$  and **fresh  $a$  in  $f$**  coincides with  $\mathbb{N}a.f a$  where  $\mathbb{N}$  is the ‘new’ quantifier of Pitts and Gabbay. This permits the definition of the  $\alpha$ -equivalence relation  $\sim_\alpha$  between pairs  $\langle x, a \rangle$  where  $x$  is an FM set and  $a$  is a name, by setting

$$\langle x_1, a_1 \rangle \sim_\alpha \langle x_2, a_2 \rangle \text{ iff } \mathbb{N}b. (a_1 b) \cdot x_1 = (a_2 b) \cdot x_2.$$

---

<sup>1</sup> This paper summarises Turner’s PhD thesis [5], where all proofs and a fuller set of references can be found; we apologise for the paucity of references forced here.

The  $\alpha$ -equivalence class  $\{\langle a, x \rangle\}_{\sim_\alpha}$  is an FM set written  $[a].x$ . Note that  $\text{supp}([a].x) = \text{supp}(x) \setminus \{a\}$  so that  $a$  is ‘bound’ in  $[a].x$ . The operation of *concretion* acts so  $([a].x)@b =_{\text{def}} (ab) \cdot x$  provided  $[a].x \# b$ . We write  $\mathcal{A}bs(\mathbb{A})$  for the class of  $\alpha$ -equivalence classes.

### 1.1 Name Generation in Nominal Sets

Defining

$$X \otimes Y =_{\text{def}} \{\langle x, y \rangle \mid x \# y\}$$

gives a tensor  $\otimes$  on **NSet**. Provided  $X$  is a nominal set,  $\alpha$ -equivalence  $\sim_\alpha$  restricts to an equivalence relation on  $X \times \mathbb{A}$ . The quotient  $(X \times \mathbb{A})/\sim_\alpha$  is written  $\delta X$ . For example,

$$\delta \mathbb{A} = \{\text{fresh } b \text{ in } [b].a \mid a \in \mathbb{A}\} \dot{\cup} \{\text{fresh } a \text{ in } [a].a\} \cong \mathbb{A} \dot{\cup} \{*\}.$$

The operation  $\delta$  is the object part of a right adjoint to  $(-) \otimes \mathbb{A}$ ; the counit is given by concretion  $@$ . The right adjoint  $\delta$  constructs a form of function space: for a nominal set  $X$ , the nominal set  $\delta X$  consists of ‘new-name abstractions’  $x'$  which applied to a fresh name  $a$  yield  $x'@a$  in  $X$ . New-name abstractions in  $\delta X$  capture the effect of new-name generation, albeit in a rather subtle way.

### 1.2 Name Generation in FM Sets

Unfortunately  $(-) \otimes \mathbb{A}$  is no longer a functor on the larger category **FMSet**. If names are to appear explicitly in our syntax, in operations and types (the case for Nominal HOPLA—though not new-HOPLA<sup>2</sup>) we are led outside **NSet**, and name generation requires an alternative to the adjunction  $(-) \otimes \mathbb{A} \dashv \delta$ .

Turner<sup>[5]</sup> exhibits a suitable adjunction in FM sets given by the situation

$$(-)^{\#a} : \mathbf{FMSet}_s \rightleftarrows \mathbf{FMSet}_{s \dot{\cup} \{a\}} : \delta_a$$

now local to a finite set of names  $s$  with  $a \in \mathbb{A} \setminus s$ . The left adjoint  $(-)^{\#a}$  is defined on objects by  $X^{\#a} =_{\text{def}} \{x \in X \mid a \# x\}$  and on arrows by restriction. The right adjoint  $\delta_a$  can be described as a subset of  $\alpha$ -equivalence classes  $x'$ : on objects

$$\delta_a X =_{\text{def}} \{x' \in \mathcal{A}bs(\mathbb{A}) \mid \forall b. x'@b \in (ab) \cdot X\},$$

and if  $f : X \rightarrow Y$  is an arrow of  $\mathbf{FMSet}_{s \dot{\cup} \{a\}}$  and  $x' \in \delta_a X$  then  $\delta_a f(x') =_{\text{def}} \text{fresh } b \text{ in } [b].(((ab) \cdot f)(x'@b))$ . The unit has components  $\xi_X : x \mapsto \text{fresh } b \text{ in } [b].x$  and the counit,  $\zeta_X : x' \mapsto x'@a$ . Notice that if  $X$  has empty support then  $X$  is a nominal set and  $\delta_a X = \delta X$ . In particular  $\delta_a \mathbb{A} = \delta \mathbb{A} \cong \mathbb{A} \dot{\cup} \{*\}$ . Also if  $s' \subseteq s$  it follows that  $s'$  and  $\mathbb{A} \setminus s'$  are both objects of  $\mathbf{FMSet}_s$ . In this case  $\delta_a s' = \{\text{fresh } b \text{ in } [b].c \mid c \in s'\} \cong s'$  via the isomorphism above, and  $\delta_a(\mathbb{A} \setminus s') = \{\text{fresh } b \text{ in } [b].c \mid c \in \mathbb{A} \setminus s'\} \dot{\cup} \{\text{fresh } b \text{ in } [b].b\} \cong (\mathbb{A} \setminus s') \dot{\cup} \{*\}$ .

<sup>2</sup> A parallel to this paper showing how nominal sets **NSet** are sufficient to produce an adequate denotational semantics for new-HOPLA is underway.

### 1.3 Name Generation in FM Preorders

We will see that the adjunction for name generation can be imported into other structures, of which preorders are the simplest. An **FM-preorder** is defined, as usual, to comprise  $\langle \mathbb{P}, \leq_{\mathbb{P}} \rangle$  where  $\mathbb{P}$  and  $\leq_{\mathbb{P}}$  are both FM sets such that  $\leq_{\mathbb{P}}$  is a reflexive and transitive binary relation on  $\mathbb{P}$ . The  $\in$ -recursive nature of the permutation action on FM sets gives rise to a permutation action on FM-preorders, where  $\sigma \cdot \mathbb{P} = \{\sigma \cdot p \mid p \in \mathbb{P}\}$  and  $p \leq_{\mathbb{P}} p'$  if and only if  $\sigma \cdot p \leq_{\sigma \cdot \mathbb{P}} \sigma \cdot p'$ . Functions in **FMSet** must be finitely-supported so we define the category **FMPre** to consist of FM-preorders and finitely-supported monotone functions (again the standard definition). For  $s$  a finite set of names, **FMPre<sub>s</sub>** is the subcategory of **FMPre** consisting of only those objects and arrows which are supported by  $s$ .

FM-preorders inherit mechanisms for name generation directly from those in FM sets. Let  $s$  be a finite set of names and  $a \notin s$ . For  $\langle \mathbb{P}, \leq_{\mathbb{P}} \rangle$  an object of **FMPre<sub>s</sub>**, define

$$\langle \mathbb{P}, \leq_{\mathbb{P}} \rangle^{\#a} = \langle \mathbb{P}^{\#a}, \leq_{\mathbb{P}^{\#a}} \rangle$$

ordered by  $\leq_{\mathbb{P}^{\#a}}$  the restriction of  $\leq_{\mathbb{P}}$ . For  $\langle \mathbb{P}, \leq_{\mathbb{P}} \rangle$  an object of **FMPre<sub>s ∪ {a}</sub>**, define  $\delta_a \langle \mathbb{P}, \leq_{\mathbb{P}} \rangle = \langle \delta_a \mathbb{P}, \leq_{\delta_a \mathbb{P}} \rangle$  where for  $p'_1$  and  $p'_2$  elements of  $\delta_a \mathbb{P}$

$$p'_1 \leq_{\delta_a \mathbb{P}} p'_2 \Leftrightarrow_{\text{def}} \forall b. p'_1 @ b \leq_{(ab) \cdot \mathbb{P}} p'_2 @ b.$$

Taking their action on maps to be that of the corresponding functors on **FMSet**, we obtain a functor  $(-)^{\#a} : \mathbf{FMPre}_s \rightarrow \mathbf{FMPre}_{s \cup \{a\}}$  and its right adjoint  $\delta_a$ . The adjunction shares the same unit  $\xi$  and counit  $\zeta$  as those for FM-sets.

## 2 A Linear Category of FM Domains

The development of the domain theory in FM-sets here is substantially the same as an earlier domain theory developed in traditional set theory [4]. The one extra constraint here is that all sets (so subsets and functions) must be finitely-supported.

The objects of the *linear* category **FMLin** are FM-preorders  $\mathbb{P}$ , thought of as consisting of computation paths with the preorder  $p \leq p'$  expressing how a path  $p$  extends to a path  $p'$ . A path order  $\mathbb{P}$  determines a domain  $\widehat{\mathbb{P}}$ , that of its *path sets*, finitely-supported down-closed sets w.r.t.  $\leq_{\mathbb{P}}$ , ordered by inclusion. The arrows of **FMLin**, *linear* maps, from  $\mathbb{P}$  to  $\mathbb{Q}$  are finitely-supported functions from  $\widehat{\mathbb{P}}$  to  $\widehat{\mathbb{Q}}$  which preserve joins of finitely-supported subsets. The category **FMLin** is monoidal-closed with a tensor given by the product  $\mathbb{P} \times \mathbb{Q}$  of FM-preorders and a corresponding function space by  $\mathbb{P}^{op} \times \mathbb{Q}$ . The category has all biproducts (where the objects are given by disjoint juxtaposition of preorders) which serve as both products and coproducts.

In fact, the category **FMLin** will have enough structure to form a model of Girard's (classical) linear logic [1]. As usual, one can move to more liberal maps through the use of a suitable comonad (an exponential of linear logic often

written !). Here,  $!\mathbb{P}$ , for an FM-preorder  $\mathbb{P}$ , will (essentially) consist of the *isolated* elements of the domain  $\widehat{\mathbb{P}}$  under inclusion— $!\mathbb{P}$  can be thought of as consisting of compound paths, associated with several runs. The  $\mathbf{coKleisli}$  category of  $!$  consists of FM-preorders which consist of *continuous* functions between the domains of path sets. However, in the regime of FM sets, we will have to exercise some care in choosing what ‘continuous’, and also ‘isolated’, are to mean if fundamental operations of name generation are to be continuous.

### 2.1 Name Generation in $\mathbf{FMLin}$

$\mathbf{FMLin}$  inherits name generation from  $\mathbf{FMPre}$ . Let  $s \subseteq_{\text{fin}} \mathbb{A}$  and  $a \in \mathbb{A} \setminus s$ . There is a name-generation adjunction

$$(-)^{\#a+} \dashv \delta_a^+ : \mathbf{FMLin}_s \rightleftarrows \mathbf{FMLin}_{s \dot{\cup} \{a\}}.$$

Here  $\mathbf{FMLin}_s$  is the subcategory of  $\mathbf{FMLin}$  whose objects and arrows are all supported by  $s$ . The key laws are isomorphisms

$$\phi_{\mathbb{P}} : \widehat{\mathbb{P}^{\#a}} \cong \widehat{\mathbb{P}^{\#a}} \quad \text{and} \quad \theta_{\mathbb{Q}} : \delta_a \widehat{\mathbb{Q}} \cong \widehat{\delta_a \mathbb{Q}}$$

natural in  $\mathbb{P}$  in  $\mathbf{FMPre}_s$  and  $\mathbb{Q}$  in  $\mathbf{FMPre}_{s \dot{\cup} \{a\}}$ . The isomorphisms and inverses are given concretely as follows:

$$\begin{aligned} \phi_{\mathbb{P}}(x) &=_{\text{def}} \{p \in x \mid a \# p\} & \text{and} & \quad \phi_{\mathbb{P}}^{-1}(x) =_{\text{def}} x \cup \bigcup_{b \# x, \mathbb{P}} (ab) \cdot x \\ \theta_{\mathbb{Q}}^{(a)}(y') &=_{\text{def}} \{q' \mid \exists b. q' @ b \in y' @ b\} & \text{and} & \quad \theta_{\mathbb{Q}}^{-1}(y) =_{\text{def}} \mathbf{fresh} \ b \ \mathbf{in} \ [b]. \{q \mid [b].q \in y\}. \end{aligned}$$

Define the functor  $(-)^{\#a+} : \mathbf{FMLin}_s \rightarrow \mathbf{FMLin}_{s \dot{\cup} \{a\}}$  to act as  $(-)^{\#a}$  on objects and take  $f : \mathbb{P} \rightarrow_{\mathbb{L}} \mathbb{Q}$  to  $\phi_{\mathbb{Q}} \circ f^{\#a} \circ \phi_{\mathbb{P}}^{-1} : \mathbb{P}^{\#a} \rightarrow \mathbb{Q}^{\#a}$ , and  $\delta_a^+$  similarly.

In [5] it is shown that these functors are well-defined, and that the composite bijection

$$\begin{aligned} \mathbf{FMLin}_{s \dot{\cup} \{a\}}(\mathbb{P}^{\#a}, \mathbb{Q}) &\cong \mathbf{FMPre}_{s \dot{\cup} \{a\}}(\mathbb{P}^{\#a}, \widehat{\mathbb{Q}}) \cong \mathbf{FMPre}_s(\mathbb{P}, \delta_a \widehat{\mathbb{Q}}) \cong \\ &\mathbf{FMPre}_s(\mathbb{P}, \widehat{\delta_a \mathbb{Q}}) \cong \mathbf{FMLin}_s(\mathbb{P}, \delta_a \mathbb{Q}), \end{aligned}$$

got via the isomorphism  $\theta_{\mathbb{Q}}$ , extends to an adjunction with unit  $\widehat{\xi}$  and counit  $\widehat{\zeta}$ .

## 3 Continuity in FM Domains

Linear maps are too restrictive to give a semantics for concurrent processes. In [4] the solution was to turn from linear to continuous maps, which preserve only directed joins, via a suitable comonad on  $\mathbf{FMLin}$ . But this is not appropriate in the FM setting: the desired semantics for name generation is not directed-join continuous!

### 3.1 Continuity and Name Generation

To see this, we consider a term construction  $\mathbf{new} a.t$  inspired by new-HOPLA [8]. Imagine that  $t$  denotes a process whose actions lie within the set of names  $\mathbb{A}$ ; so its denotation  $\llbracket t \rrbracket$  is an element of  $\widehat{\mathbb{A}}$ . By definition the term  $\mathbf{new} a.t$  denotes an element of  $\widehat{\delta_a \mathbb{A}}$ ; its denotation  $\llbracket \mathbf{new} a.t \rrbracket$  is given as  $\theta_{\mathbb{A}}([a].\llbracket t \rrbracket)$ , where  $\theta_{\mathbb{A}} : \delta_a(\widehat{\mathbb{A}}) \cong \widehat{\delta_a \mathbb{A}}$  is the isomorphism described in the previous section. The term  $\mathbf{new} a.t$  denotes a process with actions of the form  $[b].c$  and  $[c].c$  from  $\delta_a \mathbb{A}$ .

Consider now an open term  $\mathbf{new} a.(-)$ . Substitution into  $\mathbf{new} a.(-)$  replaces  $a$  with a name  $a'$  fresh w.r.t. the argument being substituted, if necessary. Consequently, the substitution of  $\mathbb{A}$ , with empty support, results in denotation  $\theta_{\mathbb{A}}([a].\mathbb{A})$  which can be shown to contain  $[a].a$ . Whereas, the substitution of  $s \subseteq_{\text{fin}} \mathbb{A}$ , results in denotation  $\theta_{\mathbb{A}}([a'].s)$ , with  $a' \notin s$ , a denotation which cannot contain  $[a].a$ . As  $\mathbb{A} = \bigcup_{s \subseteq_{\text{fin}} \mathbb{A}} s$  is a directed join, this shows that  $\mathbf{new} a.(-)$  does not yield a directed-join continuous function.

### 3.2 FM-Continuity

It makes little difference to classical domain theory whether one uses increasing (ordinal-indexed) sequences or directed sets, because the Axiom of Choice (AC) can be used to move between the two. However, AC does not hold in the theory of FM sets, and this equivalence breaks down. A particular difference is that in any sequence in FM set theory with support  $s$  each element of the sequence must also have support  $s$ ; this ‘uniformity’ of support does not hold for directed sets in general.

**Definition 1.** *An FM set  $X$  has **uniform support**  $s$  if every element  $x \in X$  is supported by  $s$ . An **FM-directed** set is a directed set with uniform support. If  $\mathbb{P}, \mathbb{Q}$  are FM-preorders, say that a function  $f : \widehat{\mathbb{P}} \rightarrow \widehat{\mathbb{Q}}$  is **FM-continuous** if it is finitely-supported and preserves joins of FM-directed sets. (Note FM-linear maps are FM-continuous.)*

If  $X$  has uniform support then it can be wellordered within FM set theory: AC gives an (external) wellordering and the uniformity ensures that this wellordering is itself finitely-supported. Approximation by FM-directed sets and approximation by (ordinal-indexed) sequences are equivalent in FM set theory.

Returning to the example of  $\mathbf{new} a.(-)$ , notice that the directed set  $\{s \mid s \subseteq_{\text{fin}} \mathbb{A}\}$  does not have a uniform support. Let  $X \subseteq \widehat{\mathbb{A}}$  be directed with uniform support  $s$ . Then every  $x \in X$  is either a subset of  $s$  or a superset of  $\mathbb{A} \setminus s$ , so  $X$  is finite. Since  $X$  is also directed it contains a maximum element. As a direct consequence,  $\mathbf{new} a.(-)$  is FM-continuous.

### 3.3 FM-Isolated Elements

We investigate the structure of isolated elements of domains  $\widehat{\mathbb{P}}$ , for  $\mathbb{P}$  an FM-preorder, with respect to FM-directed sets.

**Definition 2.** An element  $P \in \widehat{\mathbb{P}}$  is **FM-isolated** (or simply **isolated**) iff for all FM-directed sets  $X \subseteq \widehat{\mathbb{P}}$ , if  $P \subseteq \bigcup X$  then there exists  $x \in X$  such that  $P \subseteq x$ .

For example, every element of  $\widehat{\mathbb{A}}$  is isolated, because any FM-directed subset of  $\widehat{\mathbb{A}}$  contains a maximum element (see above). More generally,

**Definition 3.** For  $\mathbb{P}$  a FM-preorder,  $F$  a finite subset of  $\mathbb{P}$  and  $s$  a finite set of names containing  $\text{supp}(\mathbb{P})$ , define  $\langle F \rangle_s =_{\text{def}} \bigcup_{\sigma \#_s} \sigma \cdot F$ ; write  $\langle F \rangle_{s\downarrow}$  for the down-closure of  $\langle F \rangle_s$ .

Every  $x \in \widehat{\mathbb{A}}$  is of this form: either  $x$  is finite and hence  $x = \langle x \rangle_{\text{supp}(x)}$  or else  $x$  is cofinite and hence  $x = \langle \{a\} \rangle_{\text{supp}(x)}$  for any  $a \in x$ . In general:

**Lemma 1.** If  $F \subseteq_{\text{fin}} \mathbb{P}$  and  $s$  is a finite set of names that supports  $\mathbb{P}$  then  $\langle F \rangle_{s\downarrow}$  is isolated in  $\widehat{\mathbb{P}}$ . Conversely, if  $P \in \widehat{\mathbb{P}}$  is isolated and  $\text{supp}(P, \mathbb{P}) \subseteq s$  then there exists  $F \subseteq_{\text{fin}} \mathbb{P}$  such that  $P = \langle F \rangle_{s\downarrow}$ .

### 3.4 The Category FMCTs

Let **FMCTs** be the category with objects FM-preorders and arrows from  $\mathbb{P}$  to  $\mathbb{Q}$  the FM-continuous functions from  $\widehat{\mathbb{P}}$  to  $\widehat{\mathbb{Q}}$ .

We can characterise FM-continuous maps in terms of FM-linear maps whose source is under an exponential  $!$ . It is sensible to define  $!\mathbb{P}$  as comprising the FM-isolated elements of  $\widehat{\mathbb{P}}$  ordered by inclusion. However, with an eye to defining recursive types, we instead define  $!\mathbb{P}$  to be the equivalent FM-preorder with elements  $\langle F \rangle_s$  where  $F \subseteq_{\text{fin}} \mathbb{P}$  and  $s$  supports  $\mathbb{P}$ ; its order is given by taking  $P \leq_{!\mathbb{P}} P'$  whenever  $\forall p \in P \exists p' \in P'. p \leq_{\mathbb{P}} p'$ .

Each  $\widehat{\mathbb{P}}$  is the free FM-directed-join completion of  $!\mathbb{P}$ . (The order  $\widehat{\mathbb{P}}$  is algebraic with respect to approximation by FM-directed sets.) It follows that  $!$  extends to functor making an adjunction  $\mathbf{FMLin}(!\mathbb{P}, \mathbb{Q}) \cong \mathbf{FMCTs}(\mathbb{P}, \mathbb{Q})$ , where the inclusion is right adjoint to the  $!$ . Its unit  $\eta_{\mathbb{P}} : \mathbb{P} \xrightarrow{\subset} !\mathbb{P}$  is given concretely by  $\eta_{\mathbb{P}} X = \{P \in !\mathbb{P} \mid P \subseteq X\}$ . The adjunction satisfies the conditions Benton *et al* proposed for a model of linear logic [1].

### 3.5 Name Generation in FMCTs

We inherit adjunctions

$$(-)^{\#a^{++}} \dashv \delta_a^{++} : \mathbf{FMCTs}_s \rightleftarrows \mathbf{FMCTs}_{s \cup \{a\}}$$

supporting name generation in **FMCTs** from the adjunctions  $(-)^{\#a^+} \dashv \delta_a^+$  on the linear categories. Here  $s \subseteq_{\text{fin}} \mathbb{A}$  and  $a \in \mathbb{A} \setminus s$  and  $\mathbf{FMCTs}_s$  is the subcategory of **FMCTs** supported by  $s$ . In detail,  $(-)^{\#a^{++}}$  and  $\delta_a^{++}$  act respectively as  $(-)^{\#a}$  and  $\delta_a$  on objects. The arrow  $f : \mathbb{P} \xrightarrow{\subset} \mathbb{Q}$  of  $\mathbf{FMCTs}_s$  is taken to the composite  $f^{\#a^{++}} =_{\text{def}} \phi_{\mathbb{Q}} \circ f^{\#a} \circ \phi_{\mathbb{P}}^{-1}$  and the arrow  $g : \mathbb{P} \xrightarrow{\subset} \mathbb{Q}$  of  $\mathbf{FMCTs}_{s \cup \{a\}}$  is taken to

$\delta_a^{++}g =_{\text{def}} \theta_{\mathbb{Q}} \circ \delta_a g \circ \theta_{\mathbb{P}}^{-1}$ . These definitions coincide with those of  $(-)^{\#a+}$  and  $\delta_a^+$  on linear arrows.

Via an isomorphism  $!((-)^{\#a}) \cong !( -)^{\#a}$ , analogous to  $\phi^{-1}$  of section 2.1, we obtain as a composite the bijection

$$\begin{aligned} \mathbf{FMCT}s_{s \dot{\cup} \{a\}}(\mathbb{P}^{\#a}, \mathbb{Q}) &\cong \mathbf{FMLin}_{s \dot{\cup} \{a\}}(!(\mathbb{P}^{\#a}), \mathbb{Q}) \cong \mathbf{FMLin}_{s \dot{\cup} \{a\}}((!\mathbb{P})^{\#a}, \mathbb{Q}) \\ &\cong \mathbf{FMLin}_s(!\mathbb{P}, \delta_a \mathbb{Q}) \cong \mathbf{FMCT}s_s(\mathbb{P}, \delta_a \mathbb{Q}), \end{aligned}$$

of the adjunction  $(-)^{\#a++} \dashv \delta_a^{++}$ , with unit  $\widehat{\xi}$  and counit  $\widehat{\zeta}$  —see [5].

The machinery of freshness, the functors  $(-)^{\#a}$  and the isomorphisms  $\phi_{\mathbb{P}} : \widehat{\mathbb{P}}^{\#a} \rightarrow \widehat{\mathbb{P}}^{\#a}$ , can be extended to model freshness with respect to a finite set of names  $s$ . This is used to capture ‘freshness assumptions’ in the type system: a variable of type  $\mathbb{P}^{\#s}$  insists that it receives input that is fresh for  $s$ , and a term of type  $\mathbb{P}^{\#s}$  avoids the names in  $s$  in its evaluation. Concretely,  $\mathbb{P}^{\#s} = \{p \in \mathbb{P} \mid p \# s\}$  with order given by the restriction of the order on  $\mathbb{P}$ , while  $\phi_{\mathbb{P}}^{(s)}x = \{p \in x \mid p \# s\}$ , for  $x \in \widehat{\mathbb{P}}^{\#s}$ .

## 4 Nominal HOPLA

Nominal HOPLA is an expressive calculus for higher-order processes with non-determinism and name-binding. It can be seen as a straightforward extension of HOPLA with terms of the form  $\mathbf{new} a. t$  and  $t[a]$  which arise directly from the adjunction  $(-)^{\#a++} \dashv \delta_a^{++}$ . Its syntax is defined in FM sets.

### 4.1 Syntax

Fix a set of term variables  $x, y, \dots$  and a set of type variables  $P, \dots$ , each invariant under the permutation action. Types are given by the grammar

$$\mathbb{P}, \mathbb{Q} ::= P \mid !\mathbb{P} \mid \mathbb{Q} \rightarrow \mathbb{P} \mid \delta \mathbb{P} \mid \bigoplus_{\ell \in L} \mathbb{P}_{\ell} \mid \mu_j \mathbf{P}. \mathbb{P},$$

where  $P$  is a type variable,  $\mathbf{P}$  is a list of type variables, and  $\mu_j \mathbf{P}. \mathbb{P}$  binds  $\mathbf{P}$ , and a nominal set  $L$  is used to index components of a sum type (a biproduct in  $\mathbf{FMLin}$ ).

A closed type is a type with no free variables, and in the following, closed types are normally simply called ‘types’.

**Terms and actions** are given by mutually recursive grammars. **Terms** are given by the following grammar, where  $x$  ranges over variables,  $a$  ranges over names,  $s$  over finite sets of names,  $p$  over actions,  $\ell$  over labels and  $\mathbb{P}$  over types.

$$t, u ::= x \mid \mathbf{rec} x. t \mid \sum_{i \in I} t_i \mid !t \mid [u > p(x : \mathbb{P} \# s) => t] \mid \lambda x. t \mid t(u : \mathbb{P}) \mid \mathbf{new} a. t \mid t[a] \mid \ell : t \mid \pi_{\ell} t \mid \mathbf{abst} \mid \mathbf{rept}$$

The forms  $\mathbf{rec} x. t$ ,  $[u > p(x : \mathbb{P} \# s) => t]$  and  $\lambda x. t$  all bind  $x$  in  $t$ , and the set of free variables of  $t$  is defined in the usual way. The form  $\mathbf{new} a. t$  binds the name  $a$  in  $t$ . In a nondeterministic sum the mapping  $i \mapsto t_i$  is a finitely supported function from a nominal set  $I$ . Write  $\mathbf{nil}$  for the empty sum.



**Actions.** play a central role in the operational semantics of Nominal HOPLA—section 4.3. The grammar of actions, labelling the transitions in the operational semantics, is given as follows where  $t$  ranges over closed terms,  $a$  ranges over names and  $\ell$  over labels.

$$p ::= ! \mid \ell : p \mid t \mapsto p \mid \mathbf{abs} \ p \mid \mathbf{new} \ a. p$$

The form  $\mathbf{new} \ a. p$  binds the name  $a$  in  $p$ .

Actions and terms form nominal sets where the permutation actions are given by the obvious structural recursion.

**Substitution.** Substitution  $t[v/y]$  of term  $v$  for variable  $y$  in a term  $t$  is defined as usual. The substitution is capture-avoiding in both variables and names, in the sense that for substitution into a term of the forms  $\mathbf{rec} \ \mathbf{x}. t$ ,  $[u > p(\mathbf{x} : \mathbb{P} \# s) \Rightarrow t]$  and  $\lambda \mathbf{x}. t$  the variable  $\mathbf{x}$  is assumed not to be free in  $v$ , and for substitution into a term of the form  $\mathbf{new} \ a. t$  the name  $a$  is chosen to be fresh for  $v$ .

## 4.2 Typing Rules

**For Terms.** An environment  $\Gamma = \mathbf{x}_1 : \mathbb{P}_1 \# s_1, \dots, \mathbf{x}_n : \mathbb{P}_n \# s_n$  where  $\mathbf{x}_1, \dots, \mathbf{x}_n$  are distinct variables,  $\mathbb{P}_1, \dots, \mathbb{P}_n$  are types and  $s_1, \dots, s_n$  are finite sets of names. The intended meaning of  $\mathbf{x} : \mathbb{P} \# s$  is that the variable  $\mathbf{x}$  takes values of type  $\mathbb{P}$  that are assumed to be fresh for  $s$ .

Terms of Nominal HOPLA are typed with judgements of the form  $\Gamma \vdash_s t : \mathbb{P}$ , where  $\Gamma$  is an environment,  $s$  is a finite set of names,  $t$  is a term and  $\mathbb{P}$  is a type. The type  $\mathbb{P}$  describes the actions that the term may perform. The environment  $\Gamma$  records types and freshness assumptions for the variables of  $t$ . The set  $s$  represents the ‘current’ set of names.

*Structural rules.* *Weakening:* the environment may be extended with extra variables. *Exchange:* two variables in the environment may be exchanged. *Contraction:* a pair of variables (with equal types) may be replaced by a single variable. In addition to these standard rules are two rules associated with names:

*Fresh-Weakening.* It is possible to impose extra freshness assumptions on a variable.

$$\frac{\Gamma, \mathbf{x} : \mathbb{Q} \# s'' \vdash_s t : \mathbb{P}}{\Gamma, \mathbf{x} : \mathbb{Q} \# s' \vdash_s t : \mathbb{P}} \quad (s'' \subseteq s' \subseteq s)$$

*Support-Weakening.* It is possible to extend the ‘current’ set  $s$  of names.

$$\frac{\Gamma \vdash_{s'} t : \mathbb{P}}{\Gamma \vdash_s t : \mathbb{P}} \quad (s' \subseteq s)$$

*Variable.* A bare variable is typed by the environment in the obvious fashion.

$$\frac{-}{\mathbf{x} : \mathbb{P} \# \emptyset \vdash_{\emptyset} \mathbf{x} : \mathbb{P}}$$

*Prefix.* The term constructor  $!$  takes a term  $t$  to a term  $!t$  that intuitively may perform an anonymous action  $!$  and resume as  $t$ . The possible action  $!$  is recorded in the type.

$$\frac{\Gamma \vdash_s t : \mathbb{P}}{\Gamma \vdash_s !t : !\mathbb{P}}$$

*Match.* A term of the form  $[u > q(\mathbf{x} : \mathbb{Q}' \# s') \Rightarrow t]$  intuitively matches the output of  $u$  against the action  $q$  and feeds the resumption of  $u$  into the variable  $\mathbf{x}$  in  $t$ . If  $\mathbf{x}$  has some freshness assumptions imposed on it then  $u$  and  $q$  must satisfy those assumptions. The side condition that  $s'' \subseteq s \setminus s'$  is assumed.

$$\frac{\Gamma, \mathbf{x} : \mathbb{Q}' \# s' \vdash_s t : \mathbb{P} \quad \Lambda \vdash_{s''} u : \mathbb{Q} \quad \vdash_{s''} \mathbb{Q} : q : \mathbb{Q}'}{\Gamma, \Lambda \# s' \vdash_s [u > q(\mathbf{x} : \mathbb{Q}' \# s') \Rightarrow t] : \mathbb{P}}$$

*Recursion.* A term of the form  $\mathbf{rec} \mathbf{x}.t$  intuitively acts as its unfolding  $t[\mathbf{rec} \mathbf{x}.t/\mathbf{x}]$ , so that  $\mathbf{x}$  must be of the same type as  $t$ .

$$\frac{\Gamma, \mathbf{x} : \mathbb{P}^{\#\emptyset} \vdash_s t : \mathbb{P}}{\Gamma \vdash_s \mathbf{rec} \mathbf{x}.t : \mathbb{P}}$$

*Function Abstraction and Application.* A term  $t$  of type  $\mathbb{P}$  may be abstracted with respect to the free variable  $\mathbf{x}$  of type  $\mathbb{Q}$  to leave a term  $\lambda \mathbf{x}.t$  of type  $\mathbb{Q} \rightarrow \mathbb{P}$  that can in turn be applied to a term of type  $\mathbb{Q}$  in the usual fashion.

$$\frac{\Gamma, \mathbf{x} : \mathbb{Q}^{\#\emptyset} \vdash_s t : \mathbb{P}}{\Gamma \vdash_s \lambda \mathbf{x}.t : \mathbb{Q} \rightarrow \mathbb{P}} \quad \frac{\Gamma \vdash_s t : \mathbb{Q} \rightarrow \mathbb{P} \quad \Lambda \vdash_s u : \mathbb{Q}}{\Gamma, \Lambda \vdash_s t(u : \mathbb{Q}) : \mathbb{P}}$$

*Labelling and Label Projection.* The actions of a term  $t$  may be ‘tagged’ with a label  $\ell_0$  by forming the term  $\ell_0 : t$ . The effect of the term former  $\pi_{\ell_0}$  is that terms of the form  $\pi_{\ell_0} t$  can perform only the actions of  $t$  that are tagged by the label  $\ell_0$ . In both of these rules the support of  $\ell_0$  must be contained in  $s$ .

$$\frac{\Gamma \vdash_s t : \mathbb{P}_{\ell_0}}{\Gamma \vdash_s \ell_0 : t : \bigoplus_{\ell \in L} \mathbb{P}_{\ell}} \quad \frac{\Gamma \vdash_s t : \bigoplus_{\ell \in L} \mathbb{P}_{\ell}}{\Gamma \vdash_s \pi_{\ell_0} t : \mathbb{P}_{\ell_0}}$$

*Nondeterministic Sum.* A term  $\sum_{i \in I} t_i$  makes a nondeterministic choice amongst its components and behaves as the chosen component. The mapping  $i \mapsto \Gamma \vdash_{s_i} t_i : \mathbb{P}$  must be supported by  $s$ .

$$\frac{\Gamma \vdash_{s_i} t_i : \mathbb{P} \quad \text{each } i \in I}{\Gamma \vdash_s \sum_{i \in I} t_i : \mathbb{P}}$$

*Recursive Type Folding and Unfolding.* As the recursively-defined type  $\mu_j \mathbf{P}. \mathbb{P}$  is isomorphic (and not equal) to its unfolding  $\mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}]$  it is necessary to record any uses of the isomorphism  $\mathbf{abs} = \mathbf{rep}^{-1}$  in the syntax of the term.

$$\frac{\Gamma \vdash_s t : \mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}]}{\Gamma \vdash_s \mathbf{abs} t : \mu_j \mathbf{P}. \mathbb{P}} \quad \frac{\Gamma \vdash_s t : \mu_j \mathbf{P}. \mathbb{P}}{\Gamma \vdash_s \mathbf{rep} t : \mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}]}$$

*Name Abstraction and Application.* The only alteration to the syntax of terms over that of conventional HOPLA is the following pair of term formers. Intuitively the term  $\mathbf{new} a.t$  can perform the same actions as  $t$  with the name  $a$  bound, whereas the term  $t[a]$  takes the outputs of  $t$ , which contain a bound name since  $t$  is of type  $\delta\mathbb{P}$ , and instantiates that name as  $a$ . In both cases the side-condition  $a \notin s$  is assumed.

$$\frac{\Gamma^{\#a} \vdash_{s \dot{\cup} \{a\}} t : \mathbb{P}}{\Gamma \vdash_s \mathbf{new} a.t : \delta\mathbb{P}} \quad \frac{\Gamma \vdash_s t : \delta\mathbb{P}}{\Gamma^{\#a} \vdash_{s \dot{\cup} \{a\}} t[a] : \mathbb{P}}$$

**For Actions.** Actions are typed by judgements of the form  $\vdash_s \mathbb{P} : p : \mathbb{P}'$  where  $s$  is a finite set of names and  $\mathbb{P}$  and  $\mathbb{P}'$  are types. Intuitively, a term of type  $\mathbb{P}$  may perform an action  $p$  and resume as a term of type  $\mathbb{P}'$ .

$$\frac{\vdash_{s'} \mathbb{P} : p : \mathbb{P}' \quad (s' \subseteq s)}{\vdash_s \mathbb{P} : p : \mathbb{P}'} \quad \frac{-}{\vdash_{\emptyset} !\mathbb{P} : ! : \mathbb{P}} \quad \frac{\vdash_s \mathbb{P} : p : \mathbb{P}' \quad \vdash_s u : \mathbb{Q}}{\vdash_s \mathbb{Q} \rightarrow \mathbb{P} : u \mapsto p : \mathbb{P}'}$$

$$\frac{\vdash_s \mathbb{P}_{\ell_0} : p : \mathbb{P}'}{\vdash_s \bigoplus_{\ell \in L} \mathbb{P}_{\ell} : \ell_0 : p : \mathbb{P}'} \quad \frac{\vdash_s \mathbb{P}_j[\mu \mathbf{P}, \mathbb{P}/\mathbf{P}] : p : \mathbb{P}'}{\vdash_s \mu_j \mathbf{P}. \mathbb{P} : \mathbf{abs} \ p : \mathbb{P}'} \quad \frac{\vdash_{s \dot{\cup} \{a\}} \mathbb{P} : p : \mathbb{P}'}{\vdash_s \delta\mathbb{P} : \mathbf{new} a.p : \delta\mathbb{P}'}$$

Substitution respects the type system of Nominal HOPLA, as long as freshness assumptions are themselves respected.

**Lemma 2 (Syntactic Substitution Lemma).** *Suppose that  $t$  and  $v$  satisfy  $\Gamma, \mathbf{y} : \mathbb{R}^{\#r} \vdash_s t : \mathbb{P}$  and  $\Delta \vdash_{s_1} v : \mathbb{R}$  where  $s_1 \cap r = \emptyset$  and the variables in  $\Gamma$  are distinct from those in  $\Delta$ . Then  $\Gamma, \Delta^{\#r} \vdash_{s \cup s_1} t[v/\mathbf{y}] : \mathbb{P}$ .*

### 4.3 Operational Semantics

Nominal HOPLA is given an operational semantics in the style of a labelled transition system. That a term  $t$  such that  $\vdash t : \mathbb{P}$  may perform an action  $p$  such that  $\vdash \mathbb{P} : p : \mathbb{P}'$  and resume as the term  $t'$  is written  $\mathbb{P} : t \xrightarrow{p} t'$ . The operational semantics of closed, well-typed terms are defined below.

$$\frac{\mathbb{P} : t[\mathbf{rec} \mathbf{x}.t/\mathbf{x}] \xrightarrow{p} t'}{\mathbb{P} : \mathbf{rec} \mathbf{x}.t \xrightarrow{p} t'} \quad \frac{\mathbb{P} : t_{i_0} \xrightarrow{p} t'}{\mathbb{P} : \sum_{i \in I} t_i \xrightarrow{p} t'}$$

$$\frac{-}{!\mathbb{P} : !t \xrightarrow{!} t} \quad \frac{\mathbb{P} : t[u'/\mathbf{x}] \xrightarrow{p} t' \quad \mathbb{Q} : u \xrightarrow{q} u' \quad \vdash \mathbb{Q} : q : \mathbb{Q}'}{\mathbb{P} : [u > q(\mathbf{x} : \mathbb{Q}' \# s')] \Rightarrow t] \xrightarrow{p} t'}$$

$$\frac{\mathbb{P} : t \xrightarrow{p} t'}{\delta\mathbb{P} : \mathbf{new} a.t \xrightarrow{\mathbf{new} a, p} \mathbf{new} a.t'} \quad \frac{\delta\mathbb{P} : t \xrightarrow{\mathbf{new} a, p} \mathbf{new} a.t'}{\mathbb{P} : t[a] \xrightarrow{p} t'}$$

$$\frac{\mathbb{P} : t[u/\mathbf{x}] \xrightarrow{p} t'}{\mathbb{Q} \rightarrow \mathbb{P} : \lambda \mathbf{x}.t \xrightarrow{u \mapsto p} t'} \quad \frac{\mathbb{Q} \rightarrow \mathbb{P} : t \xrightarrow{u \mapsto p} t'}{\mathbb{P} : t(u : \mathbb{Q}) \xrightarrow{p} t'}$$

$$\begin{array}{c}
 \frac{\mathbb{P}_{\ell_0} : t \xrightarrow{P} t'}{\bigoplus_{\ell \in L} \mathbb{P}_{\ell} : \ell_0 : t \xrightarrow{\ell_0 : P} t'} \\
 \frac{\mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}] : t \xrightarrow{P} t'}{\mu_j \mathbf{P}. \mathbb{P} : \mathbf{abs} t \xrightarrow{\mathbf{abs} P} t'} \\
 \frac{\bigoplus_{\ell \in L} \mathbb{P}_{\ell} : t \xrightarrow{\ell_0 : P} t'}{\mathbb{P}_{\ell_0} : \pi_{\ell_0} t \xrightarrow{P} t'} \\
 \frac{\mu_j \mathbf{P}. \mathbb{P} : t \xrightarrow{\mathbf{abs} P} t'}{\mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}] : \mathbf{rep} t \xrightarrow{P} t'}
 \end{array}$$

The following lemma demonstrates that the operational semantics given above interacts well with the type system described above.

**Lemma 3.** *If  $\mathbb{P} : t \xrightarrow{P} t'$  then  $\vdash t : \mathbb{P}$  and there exists a unique  $\mathbb{P}'$  such that the judgement  $\vdash \mathbb{P} : p : \mathbb{P}'$  holds; furthermore  $\vdash t' : \mathbb{P}'$ .*

#### 4.4 Denotational Semantics

**Types and Environments.** A closed type denotes a nominal preorder (an FM-preorder with empty support). We will specify the preorder inductively by rules saying what paths belong to types (judgements  $p : \mathbb{P}$ ) and what the preorder is on them (judgements  $p \leq_{\mathbb{P}} p'$ ). (The method is inspired by [7].) The language of paths is given by

$$p ::= Q \mid Q \mapsto p \mid \ell : p \mid \mathbf{abs} p \mid \mathbf{new} a. p,$$

where  $Q$  is a set of paths of the form  $\langle \{p_1, \dots, p_n\} \rangle_s$ ,  $\ell$  is a label and  $a$  is a name.

$$\begin{array}{c}
 \frac{p_1 : \mathbb{P} \quad \dots \quad p_n : \mathbb{P}}{\langle \{p_1, \dots, p_n\} \rangle_s : \mathbb{P}} \\
 \frac{p : \mathbb{P}_{\ell_0}}{\ell_0 : p : \bigoplus_{\ell \in L} \mathbb{P}_{\ell}} \quad (\ell_0 \in L) \\
 \frac{Q : !Q \quad p : \mathbb{P}}{Q \mapsto p : Q \rightarrow \mathbb{P}} \\
 \frac{p : \mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}]}{\mathbf{abs} p : \mu_j \mathbf{P}. \mathbb{P}} \quad \frac{p : \mathbb{P}}{\mathbf{new} a. p : \delta \mathbb{P}}
 \end{array}$$

where the ordering  $\leq_{\mathbb{P}}$  of paths of type  $\mathbb{P}$  is given recursively as follows.

$$\begin{array}{c}
 \frac{P \preceq_{\mathbb{P}} P'}{P \leq_{! \mathbb{P}} P'} \\
 \frac{p \leq_{\mathbb{P}_{\ell_0}} p'}{\ell_0 : p \leq \bigoplus_{\ell \in L} \mathbb{P}_{\ell} \ell_0 : p'} \\
 \frac{Q' \leq_{! Q} Q \quad p \leq_{\mathbb{P}} p'}{Q \mapsto p \leq_{Q \rightarrow \mathbb{P}} Q' \mapsto p'} \\
 \frac{p \leq_{\mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}]} p'}{\mathbf{abs} p \leq_{\mu_j \mathbf{P}. \mathbb{P}} \mathbf{abs} p'} \quad \frac{p \leq_{\mathbb{P}} p'}{\mathbf{new} a. p \leq_{\delta \mathbb{P}} \mathbf{new} a. p'}
 \end{array}$$

Here,  $P \preceq_{\mathbb{P}} P'$  means that for all  $p \in P$  there exists  $p' \in P'$  such that  $p \leq_{\mathbb{P}} p'$ . It is straightforward to show that these definitions construct path orders that are nominal preorders and hence objects of  $\mathbf{FMPre}_{\emptyset}$ . As in HOPLA, in a recursively-defined type  $\mu_j \mathbf{P}. \mathbb{P}$  each path is of the form  $\mathbf{abs} p$  which means there is an isomorphism  $\mathbf{rep} : \mu_j \mathbf{P}. \mathbb{P} \cong \mathbb{P}_j[\mu \mathbf{P}. \mathbb{P}/\mathbf{P}] : \mathbf{abs}$ , where  $\mathbf{abs}(p) =_{\text{def}} \mathbf{abs} p$  and  $\mathbf{rep}(\mathbf{abs} p) =_{\text{def}} p$ .

An environment  $\mathbf{x}_1 : \mathbb{P}_1 \#^{s_1}, \dots, \mathbf{x}_n : \mathbb{P}_n \#^{s_n}$  (with freshness constraints contained in  $s_0$ ) denotes an object  $\widehat{\mathbb{P}_1 \#^{s_1}} \& \dots \& \widehat{\mathbb{P}_n \#^{s_n}}$ . Notice that such an object is isomorphic to  $\widehat{\mathbb{P}_1 \#^{s_1}} \times \dots \times \widehat{\mathbb{P}_n \#^{s_n}}$  via the isomorphisms  $\phi^{(s)}$  and  $m$ , and it will be convenient to use a ‘tuple’ notation for environments in the following.

**Terms and Actions.** Typing judgements  $\Gamma \vdash_s t : \mathbb{P}$  denote arrows

$$[\Gamma \vdash_s t : \mathbb{P}] : [\Gamma] \xrightarrow{\mathbb{C}} \mathbb{P}$$

in  $\mathbf{FMCT}_s$ . The denotation of a typing judgement is built by recursion on the derivation of the typing judgement, making use of the various universal constructions available in FM domains. Typing judgements  $\vdash_s \mathbb{P} : p : \mathbb{P}'$  denote arrows

$$[\vdash_s \mathbb{P} : p : \mathbb{P}'] : \mathbb{P} \xrightarrow{\mathbb{C}} !\mathbb{P}'$$

in  $\mathbf{FMCT}_s$  by recursion on the structure of  $p$  as shown below. Intuitively the arrow  $[\vdash_s \mathbb{P} : p : \mathbb{P}']$  matches its input against the action  $p$  and returns a collection of possible resumptions after performing  $p$ . When the types are clear we abbreviate  $[\Gamma \vdash_s t : \mathbb{P}]$  and  $[\vdash_s \mathbb{P} : p : \mathbb{P}']$  to  $[t]$  and  $[p]$ .

*Prefixing and Matching.* The adjunction  $\mathbf{FMLin}(!\mathbb{P}, \mathbb{Q}) \cong \mathbf{FMCT}_s(\mathbb{P}, \mathbb{Q})$  gives the semantics for an anonymous prefix action, written  $!$ . The unit  $\eta$  acts as a constructor for this action, taking a term  $t$  to the prefixed term  $!t$  as follows.

**Definition 4.** *Suppose that  $\Gamma \vdash_s !t : !\mathbb{P}$  is derived from  $\Gamma \vdash_s t : \mathbb{P}$ . Let  $\gamma \in \widehat{[\Gamma]}$  and  $P \in !\mathbb{P}$ . Then  $P \in [\Gamma \vdash_s !t : !\mathbb{P}]\langle\gamma\rangle$  iff  $P \subseteq [\Gamma \vdash_s t : \mathbb{P}]\langle\gamma\rangle$ .*

The denotation of the judgement  $\vdash_{\emptyset} !\mathbb{P} : ! : \mathbb{P}$  is simply the identity map. The counit  $\epsilon$  acts as a destructor for the  $!$  action, intuitively ‘matching’ a  $!$  action in the output of a term  $u$  and passing the resumption after performing the  $!$  to a term  $t$ .

**Definition 5.** *Suppose that  $\gamma \in \widehat{[\Gamma]}$  and  $\lambda \in \widehat{[\Lambda^{\#s'}]}$  and  $p \in \mathbb{P}$ . Then  $p \in [\Gamma, \Lambda^{\#s'} \vdash_s [u > q(\mathbf{x} : \mathbb{Q}' \# s') \Rightarrow t] : \mathbb{P}]\langle\gamma, \lambda\rangle$  iff there exists  $Q \in !\mathbb{Q}'$  such that  $p \in [t]\langle\gamma, Q\rangle$ ,  $Q \in ([q] \circ [u])\langle\lambda\rangle$  and  $Q \# s'$ .*

*Names and Binding* The adjunction  $(-)\#^{a++} \dashv \delta_a^{++}$  gives rise to the denotational semantics for terms of the form  $\mathbf{new} a.t$  and  $t[a]$ . Concrete definitions of these operations are given here.

**Definition 6.** *Suppose  $\Gamma \vdash_s \mathbf{new} a.t : \delta\mathbb{P}$  is derived from  $\Gamma^{\#a} \vdash_{s \cup \{a\}} t : \mathbb{P}$  where  $a \notin s$ . Let  $\gamma \in \widehat{[\Gamma]}$ , let  $b$  be a fresh name and let  $p \in \mathbb{P}$ . Then,  $\mathbf{new} b.p \in [\Gamma \vdash_s \mathbf{new} a.t : \delta\mathbb{P}]\langle\gamma\rangle$  iff  $(ab) \cdot p \in [\Gamma^{\#a} \vdash_{s \cup \{a\}} t : \mathbb{P}]\langle\gamma\rangle$ .*

**Definition 7.** *Suppose  $\Gamma^{\#a} \vdash_{s \cup \{a\}} t[a] : \mathbb{P}$  derives from  $\Gamma \vdash_s t : \delta\mathbb{P}$  where  $a \notin s$ . Let  $\gamma \in \widehat{[\Gamma^{\#a}]}$  and let  $p \in \mathbb{P}$ . Then,  $\mathbf{new} a.p \in [\Gamma \vdash_s t : \delta\mathbb{P}]\langle\gamma\rangle$  iff  $p \in [\Gamma^{\#a} \vdash_{s \cup \{a\}} t[a] : \mathbb{P}]\langle\gamma\rangle$ .*

The *structural rules* simply adjust the types of the denotations without substantially altering their semantics. They make use of the cartesian structure of each  $\mathbf{FMCT}_s$ ; weakening corresponds to projection, for example. The semantics of the first new structural rule (fresh-weakening) comes from the obvious inclusion

$(-)^{\#a} \Rightarrow (-)$  combined with the isomorphism  $\phi$ , and the second new structural rule (support-weakening) from the inclusion  $\mathbf{FMCT}_{s'} \hookrightarrow \mathbf{FMCT}_s$ . The denotational semantics of the remaining constructs follows that of HOPLA in [4] very closely. The semantics of *higher-order* processes arises from the cartesian-closed structure of  $\mathbf{FMCT}_s$ . The semantics of *labelled* processes is based on the biproducts in the linear category; injection into the biproduct corresponds to tagging the outputs of a process with a particular label, and projection to matching against a label. Via cartesian closure a hom-set of  $\mathbf{FMCT}_s$  inherits a partial order by inclusion, which in particular has all joins of  $\omega$ -chains. This provides a standard semantics to *recursion* in the language. The semantics of *nondeterministic sums* is given by union.

Substitution amounts to composition of denotations. However, care must be taken to ensure that all the relevant freshness assumptions are satisfied.

**Lemma 4 (Semantic Substitution Lemma).** *Suppose that  $\Gamma, y : \mathbb{R}^{\#r} \vdash_s t : \mathbb{P}$  and  $\Delta \vdash_{s_1} v : \mathbb{R}$  where  $s_1 \cap r = \emptyset$  and the variables in  $\Gamma$  are distinct from those in  $\Delta$ . Then*

$$\llbracket \Gamma, \Delta^{\#r} \vdash_{s \cup s_1} t[v/y] : \mathbb{P} \rrbracket = \llbracket \Gamma, y : \mathbb{R}^{\#r} \vdash_s t : \mathbb{P} \rrbracket \circ (\mathbf{1}_\Gamma \& \llbracket \Delta \vdash_{s_1} v : \mathbb{R} \rrbracket^{\#r++}).$$

## 5 Soundness and Adequacy

The possibility of observing an action  $p$  of a process  $f$  is caught by a judgement  $\mathbb{P} : t \xrightarrow{p} t'$ . In fact the match operator reduces these general observations to observations of just  $!$  actions, because to observe the action  $p$  in the term  $t$  is the same as to observe a  $!$  action in the term  $[t > p(x:\mathbb{P} \# s) \Rightarrow !\mathbf{nil}]$ .

**Lemma 5 (Soundness).** *If  $!\mathbb{P} : t \xrightarrow{!} t'$  and  $s$  is a finite set of names such that  $\text{supp}(t, t') \subseteq s$  then  $\llbracket \vdash_s !t' : !\mathbb{P} \rrbracket \subseteq \llbracket \vdash_s t : !\mathbb{P} \rrbracket$ .*

Define a logical relation  $X \trianglelefteq_{\mathbb{P}} t$  between subsets  $X \subseteq \mathbb{P}$  and terms such that  $\vdash t : \mathbb{P}$  by way of an auxiliary relation  $p \in_{\mathbb{P}} t$  between paths  $p \in \mathbb{P}$  and terms such that  $\vdash t : \mathbb{P}$  as shown in [5]. The intuition behind the statement that  $p \in_{\mathbb{P}} t$  is that  $p$  is a computation path of type  $\mathbb{P}$  that the process  $t$  may perform. Its definition is by recursion on the structure of paths.

$$\begin{aligned} X \trianglelefteq_{\mathbb{P}} t &\iff \forall p \in X. p \in_{\mathbb{P}} t \\ P \in_{!\mathbb{P}} t &\iff \exists t'. !\mathbb{P} : t \xrightarrow{!} t' \text{ and } P \trianglelefteq_{\mathbb{P}} t' \\ Q \mapsto p \in_{\mathbb{Q}} \rightarrow_{\mathbb{P}} t &\iff \forall u. (Q \trianglelefteq_{\mathbb{Q}} u \Rightarrow p \in_{\mathbb{P}} t(u:\mathbb{Q})) \\ \mathbf{new} a. p \in_{\delta\mathbb{P}} t &\iff \forall a. p \in_{\mathbb{P}} t[a] \\ \ell_0 : p \in_{\bigoplus_{\ell \in L} \mathbb{P}_{\ell}} t &\iff p \in_{\mathbb{P}_{\ell_0}} \pi_{\ell_0} t \\ \mathbf{abs} p \in_{\mu_j \mathbb{P}. \mathbb{P}} t &\iff p \in_{\mathbb{P}_j[\mu \mathbb{P}. \mathbb{P}/\mathbb{P}]} \mathbf{rep} t \end{aligned}$$

This relation can be used to demonstrate that if a path  $p$  appears — semantically — in the denotation  $\llbracket t \rrbracket$  then the term  $t$  can — operationally — perform the path  $p$ .

**Lemma 6.** *Suppose  $\Gamma \vdash_s t : \mathbb{P}$  where  $\Gamma = \mathbf{x}_1 : \mathbb{P}_1^{\#s_1}, \dots, \mathbf{x}_n : \mathbb{P}_n^{\#s_n}$ . For each  $i \in \{1, \dots, n\}$  let  $\gamma_i \in \widehat{\mathbb{P}}_i^{\#s_i}$  and let  $v_i$  be a closed term such that  $\vdash_{s \setminus s_i} v_i : \mathbb{P}_i$  and  $\gamma_i \trianglelefteq_{\mathbb{P}_i} v_i$ . Then*

$$\llbracket \Gamma \vdash_s t : \mathbb{P} \rrbracket \langle \gamma_1, \dots, \gamma_n \rangle_{\Gamma} \trianglelefteq_{\mathbb{P}} t[v]$$

where  $t[v]$  is the term obtained by simultaneously substituting each  $\mathbf{x}_i$  with  $v_i$ .

**Lemma 7.** *If  $\vdash_s \mathbb{P} : p : \mathbb{P}'$  and  $X \trianglelefteq_{\mathbb{P}} t$  and  $P \in \llbracket p \rrbracket X$  then there exists  $t'$  such that  $\mathbb{P} : t \xrightarrow{p} t'$  and  $P \trianglelefteq_{\mathbb{P}'} t'$ .*

We obtain the main theorem of this paper, namely the adequacy of the denotational semantics of Nominal HOPLA with respect to observations of  $!$  actions.

**Theorem 1 (Adequacy).**  $\llbracket \vdash t : !\mathbb{P} \rrbracket \neq \emptyset$  if and only if there exists  $t'$  such that  $!\mathbb{P} : t \xrightarrow{!} t'$ .

Nominal HOPLA subsumes new-HOPLA and inherits its expressiveness. What of full abstraction? Names introduce new subtleties. The obstacle to full abstraction and a tentative proposal to achieve it are described in [5].

## References

1. Benton, N., Bierman, G., de Paiva, V., Hyland, M.: Linear lambda-calculus and categorical models revisited. In: Martini, S., Börger, E., Kleine Büning, H., Jäger, G., Richter, M.M. (eds.) CSL 1992. LNCS, vol. 702, pp. 61–84. Springer, Heidelberg (1993)
2. Gabbay, M.J.: A Theory of Inductive Definitions with Alpha-Equivalence. PhD thesis, Cambridge University (2001)
3. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2001)
4. Nygaard, M., Winskel, G.: Domain theory for concurrency. Theor. Comput. Sci. 316(1-3), 153–190 (2004)
5. Turner, D.C.: Nominal Domain Theory for Concurrency. PhD thesis, Cambridge University (submitted) (2008), <http://www.cl.cam.ac.uk/~dct25/>
6. Winskel, G.: Name generation and linearity. In: LICS 2005: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science, Washington, DC, USA, pp. 301–310. IEEE Computer Society Press, Los Alamitos (2005)
7. Winskel, G., Larsen, K.G.: Using information systems to solve recursive domain equations effectively. Extended abstract: Springer Lecture Notes in Computer Science, vol. 173. Full version: Technical Report UCAM-CL-TR-51, University of Cambridge, Computer Laboratory (1984)
8. Winskel, G., Zappa Nardelli, F.: new-HOPLA: a higher-order process language with name generation. In: Proc. of 3rd IFIP TCS, pp. 521–534 (2004)

# The Ackermann Award 2009

Johann A. Makowsky and Alexander Razborov

Members of EACSL Jury for the Ackermann Award

The fifth **Ackermann Award** is presented at this CSL'09, held in Coimbra, Portugal. This is the third year in which the EACSL Ackermann Award is generously sponsored. Our sponsor is the world's leading provider of personal peripherals, Logitech S.A., situated in Romanel, Switzerland<sup>1</sup>.

Eligible for the 2009 **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. 2007 and 31.12. 2008. The Jury received 12 nominations for the **Ackermann Award 2009**. The candidates came from 10 different nationalities from Europe, North America and Asia and received their PhDs in 9 different countries in Europe and North America.

The topics covered the full range of Logic and Computer Science as represented by the LICS and CSL Conferences. All the submissions were of very high standard and contained outstanding results in their particular domain. In the past the Jury reached a consensus to give more than one award. This time, in spite of the extreme high quality of the nominated theses, the Jury decided finally, to give for the year 2009 **only one** award. The 2009 **Ackermann Award** winner is

Jakob Nordström

for his thesis *Short Proofs May Be Spacious: Understanding Space in Resolution* issued by the Royal Institute of Technology, Stockholm, Sweden, May 2008, supervised by Prof. Johan Håstad.

The Jury wishes to congratulate the recipient of the Ackermann Award for his outstanding work and wishes him a successful continuation of his career.

The Jury wishes also to congratulate all the remaining 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.

## Jakob Nordström

**Citation.** Jakob Nordström receives the *2009 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

*Short Proofs May Be Spacious: Understanding Space in Resolution.*

---

<sup>1</sup> We would like to thank Daniel Borel, Co-founder and Chairman of the Board of Logitech S.A, for his generous support of the Ackermann Award for the years 2007-2009. For a history of the company, founded in 1981 in Switzerland, consult <http://www.logitech.com>



The thesis greatly advances our understanding of space-related measures in proof complexity. It completely fills the last remaining gap in the picture of most basic relations between space and other fundamental complexity measures like length or width for the system of propositional resolution. The result confirms that space is indeed an inherently independent measure by exhibiting natural contradictions possessing very short and narrow refutations but no low-space refutation. This solution of a well-known open problem complements and contrasts with previously known simulations in the opposite direction.

**Background of the thesis.** There are many good reasons that make the system of propositional resolution (formally introduced by Robinson in 1965) one of the most popular and widely studied propositional proof systems, both by theoreticians and practitioners.

On the automated theorem proving side, propositional resolution makes the core of one of the most known technique in the field, first-order resolution. As exemplified by the work of several previous Ackermann Award winners, the presence of unification does bring about a new level of interesting and difficult complexity questions. But all the same, the apparent simplicity of the propositional case is very deluding, and many fundamental problems here still remain unresolved.

It is worth noting that resolution often appears in automated theorem proving without invitation. For example, the famous DPLL algorithm developed by Davis, Putnam (1960) and Davis, Putnam, Logemann, Loveland (1962) *before* Robinson's work, still forms the basis for many efficient SAT solvers, as well as for many implementations of first-order theorem provers. The proof-search heuristic in this algorithm is represented by the choice of the branching literal, and, abstracting from the issue of *finding* the best heuristics, we arrive at the proof system that is nowadays called *tree-like resolution* (more modern versions of this algorithm result in unrestricted resolution). Any lower bounds proved for this system automatically translate into ultimate (i.e., regardless of the effort invested into making good branching choices) limitations of any DPLL-like algorithm.

On the theoretical side, propositional resolution had been considered in Proof Complexity (Tseitin 1968) well before the formal inauguration of the subject (Cook 1975). Tseitin worked with (and proved conclusive lower bounds for) its weaker version that has become known as regular resolution. The most natural problem of extending his result by removing the assumption of regularity had to wait for yet another 10 years to be solved (Haken 1985). Since the work of Haken, various methods of analyzing minimal refutation size in resolution (that essentially amount to inherent limitations on the running time of DPLL-like procedures) have been developed (Ben-Sasson and Wigderson 2000; Raz 2001; Razborov 2001-2004), and by now the program of understanding resolution proof complexity for the most frequently used "benchmark" contradictions (pigeon-hole-principle, Tseitin tautologies, random 3-CNFs) is more or less complete.

This reasonably satisfactory situation pertains only to the size (aka length) of resolution proofs. From both practical and theoretical perspective, in many situations more elaborate complexity measures capture the essence of the problem better.

The first “non-standard” measure for resolution was considered by Ben-Sasson and Wigderson in 2000: this is refutation *width* defined as the maximal number of literals in a clause appearing in the given refutation. They proved a surprising simulation stating that any short proof can be made (somewhat) narrow.

While running-time is the ultimate measure of interest, running-space is arguably more restrictive constraint in practice with direct fatal effects on time. Theoretically, running-space is captured by the model of *space complexity* (Esteban and Torán 2001; Alekhovich, Ben-Sasson, Razborov, Wigderson 2001). Assume that we download our axioms to RAM only as we need them, and, in order to free memory, we may also discard those intermediate theorems that are no longer needed. In the space model we measure the maximal amount of “information” (details vary in different versions) we should keep in memory at any particular moment.

Atserias and Dalmau proved in 2003 another surprising simulation: every low-space proof can be made narrow and hence (for certain trivial reasons) short. An intriguing problem left open by this research was whether any form of the converse is true, and if good upper bounds on length or width imply at least some non-trivial information about minimal space. This question was discussed in many previous papers, but there was no consensus on what the right answer should be. These papers, however, identified a prominent family of candidate contradictions for separating space from length, so-called *pebbling contradictions*. It is worth noting that although space lower bounds were proved already in the original papers by Esteban et. al, Alekhovich et. al, all of them dealt with tautologies that were known to be complex also in terms of resolution length. In particular, previously known methods did not seem to work for the pebbling contradictions.

**Nordström’s thesis.** The main contribution of the thesis consists in the ultimate solution of this open problem. But before supplying a few more details, let us make one technical remark.

As the author himself admits, the thesis represents work in progress, and it was written in the midst of intensive research. This fact is reflected by the ascending stricture of the text: different chapters prove increasingly strong versions of the main result. In particular, its final version in Chapter 11 was found roughly at the time the thesis went to press, and the proofs are rather sketchy. But since it is very plausible that it is this version that will make its way to textbooks, we will focuss on it nonetheless.

The main result was already informally formulated in the previous section. Numerically, there exist explicit families of 3-CNF contradictions with  $O(n)$  clauses that possess a resolution refutation of constant width and linear length  $O(n)$  (both are optimal), but such that the clause space of any refutation of this contradiction is  $\Omega(n/\log n)$  (which is also best possible). It rules out any hope to gain information about space from the existence of short and/or narrow resolution refutations. In a sense, it is only this result that firmly established our belief that space complexity is really an independent complexity measure that can not be derived from the others. This fills the last remaining gap in the picture of basic relations between length, width and space.

In his proof Jakob Nordström significantly develops and enhances techniques based upon an important concept of pebbling. These techniques go back to the 1970s, and it was very insightful and surprising to see that their modification can be helpful for solving this prominent open problem. The proof, however, contains many elements that are entirely new and are likely to be used elsewhere. In particular, the concept of an *XOR-pebbling contradiction* introduced in the thesis readily generalizes to arbitrary  $k$ -CNFs, thereby yielding a useful structural result whose potential importance stretches well beyond pebbling-related contradictions.

Another contribution of the thesis establishes a trade-off between the complexity measures in question. Nordström exhibits explicit contradictions that possess *either* length-efficient *or* space-efficient refutations, but these two requirements can not be fulfilled simultaneously.

The thesis is based upon papers published in STOC06 (Best Student Paper Award, submitted to *SIAM Journal on Computing*), STOC08 and FOCS08. Parts of this research are co-authored with his advisor Johan Håstad (Royal Institute of Technology, Stockholm), as well as Eli Ben-Sasson (Technion–Israel Institute of Technology, Haifa).

**Biographic Sketch.** Karl Jakob Nordström was born April 11, 1972, in Sweden. He currently is a postdoctoral fellow at the Massachusetts Institute of Technology (MIT). He received his Master of Science in Computer Science and Mathematics at Stockholm University in 2001, and his PhD in Computer Science at the Royal Institute of Technology (KTH) in 2008, while being a research assistant sponsored by the President of KTH. In 2006 he received the best student paper award at 38th ACM Symposium on Theory of Computing (STOC'06).

1997-1998 he served as a military interpreter at the Swedish Armed Forces Language Institute, where he graduated as the best student of the 1998 class. He still works as an interpreter and translator between Russian and Swedish/English, and was engaged as interpreter for among others His Majesty the King of Sweden, the Prime Minister of Sweden, the Speaker of the Swedish Parliament and the Supreme Commander of the Swedish Armed Forces. 2001-2002 he was the Secretary of the Swedish Association of Military Interpreters, and 2002-2005 served as its President.

In 1992 he received his Diploma in Choir Conducting with extended Music Theory from Tallinn Music Upper Secondary School, Estonia, and in 1994 he founded the vocal ensemble *Collegium Vocale Stockholm*, which he led till 1999. Throughout the 90s, the ensemble gave a number of concerts presenting mainly Renaissance and Baroque music.

## The Ackermann Award

The EACSL Board decided in November 2004 to launch the EACSL Outstanding Dissertation Award for Logic in Computer Science, the **Ackermann Award**,

The award<sup>2</sup> is named after the eminent logician Wilhelm Ackermann (1896-1962), mostly known for the Ackermann function, a landmark contribution in early complexity theory and the study of the rate of growth of recursive functions, and for his coauthorship with D. Hilbert of the classic *Grundzüge der Theoretischen Logik*, first published in 1928. Translated early into several languages, this monograph was the most influential book in the formative years of mathematical logic. In fact, Gödel's completeness theorem proves the completeness of the system presented and proved sound by Hilbert and Ackermann. As one of the pioneers of logic, W. Ackermann left his mark in shaping logic and the theory of computation.

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

The Jury for the **Ackermann Award** consists 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 current jury consists of R. Alur (Philadelphia, USA) J. van Benthem (Amsterdam, The Netherlands), P.-L. Curien (Paris, France) A. Dawar (Cambridge, U.K., Member of the EACSL Board) A. Durand (Paris, France) M. Grohe (Berlin, Germany), M. Hyland (Cambridge, U.K.), J.A. Makowsky (Haifa, Israel, President of EACSL), G. Plotkin (Edinburgh, U.K., LICS Organizing Committee) and A. Razborov (Chicago, USA).

Previous winners of the Ackermann Award were

**2005, Oxford:**

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

**2006, Szeged:**

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

**2007, Lausanne**

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

**2008, Bertinoro:**

Krishnendu Chatterjee from India.

Detailed reports on their work appeared in the CSL'05, CSL'06, CSL'07 and CSL'08 proceedings, and are also available via the EACSL homepage.

---

<sup>2</sup> Details concerning the Ackermann Award and a biographic sketch of W. Ackermann was published in the CSL'05 proceedings and can also be found at <http://www.eacsl.org/award.html>

# Author Index

- Abel, Andreas 40  
Accattoli, Beniamino 55  
Adler, Isolde 71  
Alur, Rajeev 86  
Atserias, Albert 102
- Bárány, Vince 117  
Berger, Ulrich 132  
Birkedal, Lars 440  
Blanqui, Frédéric 147  
Bojańczyk, Mikołaj 1
- Cao, Yongzhi 302  
Cate, Balder ten 287  
Černý, Pavol 86  
Ciabattoni, Agata 163  
Cirstea, Corina 179  
Cockett, Robin 194  
Coquand, Thierry 2
- Dezani-Ciancaglini, Mariangiola 209  
Duparc, Jacques 225
- Egger, Jeff 240  
Endrullis, Jörg 255
- Facchini, Alessandro 225  
Ferreira, Gilda 3
- Gaboardi, Marco 271  
Geuvers, Herman 255  
Gheerbrant, Amélie 287  
Giannini, Paola 209  
Grohe, Martin 20  
Guerrini, Stefano 55  
Guo, Heng 302
- Hofmann, Martin 317  
Horbach, Matthias 332
- Kaiser, Lukasz 117  
Kreutzer, Stephan 348  
Kupke, Clemens 179  
Kuske, Dietrich 364
- Laird, James 379  
Lohrey, Markus 364
- Makowsky, Johann A. 561  
Møgelberg, Rasmus Ejlers 240  
Montanari, Angelo 394  
Moschovakis, Yiannis N. 24  
Munch-Maccagnoni, Guillaume 409  
Murlak, Filip 225
- Oliva, Paulo 3
- Pattinson, Dirk 179  
Péchoux, Romain 271  
Puppis, Gabriele 394
- Rabinovich, Alexander 117, 424  
Razborov, Alexander 561  
Reus, Bernhard 440  
Rodriguez, Dulma 317  
Ronchi Della Rocca, Simona 209  
Roux, Cody 147
- Sala, Pietro 394  
Santocanale, Luigi 194  
Schwinghammer, Jan 440  
Simpson, Alex 240  
Straßburger, Lutz 163  
Sumii, Eijiro 455
- Tatsuta, Makoto 470  
Terui, Kazushige 163  
To, Anthony Widjaja 485  
Tranquilli, Paolo 500  
Turner, David 546
- Ummels, Michael 515
- Wang, Hanpin 302  
Weber, Volker 530  
Weidenbach, Christoph 332  
Weinstein, Scott 86  
Weyer, Mark 71, 102  
Winkel, Glynn 546  
Wojtczak, Dominik 515
- Xu, Zhongyuan 302
- Yang, Hongseok 440
- Zantema, Hans 255