Lecture Notes in Computer Science 5213

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
Massachusetts Institute of Technology, 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

Michael Kaminski Simone Martini (Eds.)

Computer Science Logic

22nd International Workshop, CSL 2008
17th Annual Conference of the EACSL
Bertinoro, Italy, September 16-19, 2008
Proceedings

Springer

Volume Editors

Michael Kaminski
Technion - Israel Institute of Technology
Department of Computer Science
Haifa 32000, Israel
E-mail: kaminski@cs.technion.ac.il

Simone Martini
Università di Bologna
Dip. di Scienze dell'Informazione
Mura Anteo Zamboni 7
40127 Bologna, Italy
E-mail: martini@cs.unibo.it

Library of Congress Control Number: 2008934680

CR Subject Classification (1998): F.4.1, F.4, I.2.3-4, F.3

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

ISSN 0302-9743
ISBN-10 3-540-87530-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-87530-7 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 is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2008
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12514106 06/3180 5 4 3 2 1 0

Preface

The annual conference of the European Association for Computer Science Logic (EACSL), CSL 2008, was held in Bertinoro, near Bologna (Italy), September 16–19, 2008. The conference series started as a program of International Workshops on Computer Science Logic, and then at its sixth meeting became the Annual Conference of the EACSL. This conference was the 22nd meeting and 17th EACSL conference; it was organized by the Department of Computer Science of *Alma Mater Studiorum* – Università di Bologna.

CSL 2008 was preceded on Monday, September 15th by the symposium *Bridging Logic and Computer Science* on the occasion of the 60th birthday of Johann A. Makowsky.

In response to the call for papers, a total of 102 abstracts were submitted to CSL 2008 of which 87 were followed by a full paper. The Program Committee selected 31 papers for presentation at the conference and publication in these proceedings, during a one-week electronic discussion on the EasyChair platform; each paper was refereed by three to five reviewers.

The Program Committee invited lectures from Luca Cardelli, Pierre-Louis Curien, Jean-Pierre Jouannaud, and Wolfgang Thomas. The papers provided by the invited speakers appear at the beginning of this volume.

Created in 2005, the Ackermann Award is the EACSL Outstanding Dissertation Award for Logic in Computer Science, sponsored for the years 2007-2009 by Logitech S.A. The award recipient for 2008 was Krishnendu Chatterjee, who was invited to present his work at the conference. Citation for the award, abstract of the thesis, and a biographical sketch of the recipient may be found at the end of the proceedings.

We sincerely thank the Program Committee and all of the referees for their generous work in reviewing the papers, as well as Ugo Dal Lago, the main local organizer. We also thank the *Alma Mater Studiorum* – Università di Bologna, the Istituto Nazionale di Alta Matematica – GNSAGA, and the Associazione Italiana di Logica e Applicazioni (AILA) for their financial support.

June 2008

Michael Kaminski
Simone Martini

Conference Organization

Program Committee

Zena Ariola, *Eugene*
Patrick Baillot, *Paris*
Patrick Cegielski, *Paris*
Gilles Dowek, *Palaiseau*
Amy Felty, *Ottawa*
Marcelo Fiore, *Cambridge*
Alan Jeffrey, *Lisle*
Michael Kaminski, *Haifa*, Co-chair
Leonid Libkin, *Edinburgh*
Zoran Majkic, *Beograd*
Simone Martini, *Bologna*, Co-chair
Dale Miller, *Palaiseau*
Luke Ong, *Oxford*
David Pym, *Bristol and Bath*
Alexander Rabinovich, *Tel Aviv*
Antonino Salibra, *Venezia*
Thomas Schwentick, *Dortmund*
Valentin Shehtman, *London*
Alex Simpson, *Edinburgh*
Gert Smolka, *Saarbrücken*
Kazushige Terui, *Kyoto*
Thomas Wilke, *Kiel*

Additional Referees

Andreas Abel
Shunichi Amano
Marcelo Arenas
Maria Artishchev-Zapolotsky
Eugene Asarin
Jeremy Avigad
David Baelde
Pablo Barcelo
Stefano Berardi
Ulrich Berger
Stefan Berghofer
Leopoldo Bertossi
Dietmar Berwanger
Stefano Bistarelli
Henrik Björklund
Frédéric Blanqui
Alexander Bochman
Manuel Bodirsky
Bernard Boigelot
Paola Bonacina
Richard Bonichon
Pierre Boudes
Patricia Bouyer
Julian Bradfield
James Brotherston
Chad E. Brown
Antonio Bucciarelli
Thierry Cachat
Venanzio Capretta
Arnaud Carayol
Kaustuv Chaudhuri
Robin Cockett
Matthew Collinson
Stephen Cook
Charalampos Cornaros
Ugo Dal Lago
Vincent Danos
Stéphane Demri
Michel de Rougemont
Mariangiola Dezani
Paolo Di Giamberardino
Pietro Di Gianantonio
Alexander Dikovsky
Manfred Droste
Arnaud Durand
Thomas Ehrhrard
José Espírito Santo
Kousha Etessami
François Fages
John Fearnley
Alain Finkel
Christophe Fouqueré
Goran Frehse
Fabio Gadducci
Didier Galmiche
Philipp Gerhardy
Silvia Ghilezan
Hugo Gimbert
Healfdene Goguen
Alexey Gotsman
Martin Grohe
Stefano Guerrini
Matthew Hague
Masahiro Hamano
Masahito Hasegawa
Hugo Herbelin
Frédéric Herbreteau
Miki Hermann
Olivier Hermant
Robin Hirsch
Robin Houston

Ullrich Hustadt
Paulin Jacobé de Naurois
Florent Jacquemard
Emil Jeřábek
Jean-Pierre Jouannaud
Marcin Jurdziński
Mark Kaminski
Daisuke Kimura
Daniel Kirsten
Jan Krajíček
Pavel Krcal
Stephan Kreutzer
Andrei Krokhin
Antonín Kučera
Viktor Kuncak
Salvatore La Torre
Robby Lampert
Cosimo Laneve
Martin Lange
Stéphane Lengrand
Kamal Lodaya
Christof Löding
William Lovas
Yoad Lusting
Ian Mackie
Giulio Manzonetto
Marcel Marquardt
Olga Marroquín Alonso
Andrea Masini
Ralph Matthes
Richard Mayr
Damiano Mazza
Antoine Meyer
Alexandre Miquel
Bernard Moeller
Brian Monahan
Angelo Montanari
Malika More
Larry Moss
Filip Murlak
Andrzej Murawski
Gopalan Nadathur
Frank Neven
Robert Nieuwenhuis
Aleksey Nogin
Karim Nour
Rotem Oshman
Luca Paolini
Michel Parigot
Grant Passmore
Mati Pentus
Benjamin Pierce
Lucia Pomello
Franck Pommereau
Ian Pratt-Hartmann
Silvio Ranise
Jason Reed
Laurent Regnier
Simona Ronchi
della Rocca
Luca Roversi
Paul Rozière
Michal Rutkowski
Jan Rutten
Andrey Rybalchenko
Sylvain Salvati
Alexis Saurin
Henning Schnoor
Jan Schwinghammer
Robert Seely
Damien Sereni
Olivier Serre
Marco Servetto
Anil Seth
Chung-chieh Shan
Ilya Shapirovsky
Cristina Sirangelo
Graham Steel
Christopher Stone
Lutz Strassburger
Aaron Stump
Guido Tack
Ryo Takemura
Tony Tan
Makoto Tatsuta
Simon J. Thompson
Michael Tiomkin
Lorenzo Tortora de Falco
Iddo Tzameret
Helmut Veith
Yde Venema
Nikolay Vereshchagin
Benjamin Werner
Anthony Widjaja To
Christopher Wilson
Greta Yorsh
Bruno Zanuttini
Noam Zeilberger
Elena Zucca
Uri Zwick

Local Organization

Ugo Dal Lago
Simone Martini

Table of Contents

Invited Talks

The Computability Path Ordering: The End of a Quest 1
Frédéric Blanqui, Jean-Pierre Jouannaud, and Albert Rubio

The Joy of String Diagrams 15
Pierre-Louis Curien

Model Transformations in Decidability Proofs for Monadic Theories 23
Wolfgang Thomas

Molecules as Automata 32
Luca Cardelli

Contributed Papers

An Infinite Automaton Characterization of Double Exponential Time . . . 33
Salvatore La Torre, P. Madhusudan, and Gennaro Parlato

Recursion Schemata for NC^k 49
Guillaume Bonfante, Reinhard Kahle, Jean-Yves Marion, and Isabel Oitavem

Extensional Uniformity for Boolean Circuits 64
Pierre McKenzie, Michael Thomas, and Heribert Vollmer

Pure Pointer Programs with Iteration 79
Martin Hofmann and Ulrich Schöpp

Quantified Positive Temporal Constraints 94
Witold Charatonik and Michał Wrona

Non-uniform Boolean Constraint Satisfaction Problems with Cardinality Constraint 109
Nadia Creignou, Henning Schnoor, and Ilka Schnoor

Fractional Collections with Cardinality Bounds, and Mixed Linear Arithmetic with Stars 124
Ruzica Piskac and Viktor Kuncak

Continuous Fragment of the mu-Calculus 139
Gaëlle Fontaine

On the Relations between the Syntactic Theories of $\lambda\mu$-Calculi 154
Alexis Saurin

A Constructive Semantic Approach to Cut Elimination in Type Theories with Axioms 169
Olivier Hermant and James Lipton

Proving Infinitude of Prime Numbers Using Binomial Coefficients 184
Phuong Nguyen

A Tight Karp-Lipton Collapse Result in Bounded Arithmetic 199
Olaf Beyersdorff and Sebastian Müller

A Calculus of Realizers for EM_1 Arithmetic 215
Stefano Berardi and Ugo de'Liguoro

Quantitative Game Semantics for Linear Logic 230
Ugo Dal Lago and Olivier Laurent

A Characterization of Hypercoherent Semantic Correctness in Multiplicative Additive Linear Logic 246
Paolo Tranquilli

An Indexed System for Multiplicative Additive Polarized Linear Logic 262
Masahiro Hamano and Ryo Takemura

A Characterisation of Lambda Definability with Sums Via $\top\top$-Closure Operators 278
Shin-ya Katsumata

Superposition for Fixed Domains 293
Matthias Horbach and Christoph Weidenbach

Non-finite Axiomatizability and Undecidability of Interval Temporal Logics with C, D, and T 308
Ian Hodkinson, Angelo Montanari, and Guido Sciavicco

On the Almighty Wand 323
Rémi Brochenin, Stéphane Demri, and Etienne Lozes

On Counting Generalized Colorings 339
T. Kotek, J.A. Makowsky, and B. Zilber

The Descriptive Complexity of Parity Games 354
Anuj Dawar and Erich Grädel

An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games 369
Sven Schewe

Quantitative Languages 385
Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger

Characterization of Logics over Ranked Tree Languages 401
Thomas Place

The Nesting-Depth of Disjunctive μ-Calculus for Tree Languages and the Limitedness Problem 416
Thomas Colcombet and Christof Löding

Upper Bounds on the Automata Size for Integer and Mixed Real and Integer Linear Arithmetic 431
Jochen Eisinger

Syntactic Metatheory of Higher-Order Subtyping 446
Andreas Abel and Dulma Rodriguez

On Isomorphisms of Intersection Types 461
Mariangiola Dezani-Ciancaglini, Roberto Di Cosmo, Elio Giovannetti, and Makoto Tatsuta

Undecidability of Type-Checking in Domain-Free Typed Lambda-Calculi with Existence 478
Koji Nakazawa, Makoto Tatsuta, Yukiyoshi Kameyama, and Hiroshi Nakano

Type-Based Termination with Sized Products 493
Gilles Barthe, Benjamin Grégoire, and Colin Riba

The Ackermann Session

The Ackermann Award 2008 508
J.A. Makowsky and D. Niwinski

Author Index 513

The Computability Path Ordering: The End of a Quest

Frédéric Blanqui[1], Jean-Pierre Jouannaud[2], and Albert Rubio[3]

[1] INRIA, Campus Scientifique, BP 239, 54506 Vandœuvre-lès-Nancy Cedex, France
[2] LIX, Projet INRIA TypiCal, École Polytechnique and CNRS, 91400 Palaiseau, France
[3] Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain

Abstract. In this paper, we first briefly survey automated termination proof methods for higher-order calculi. We then concentrate on the higher-order recursive path ordering, for which we provide an improved definition, the Computability Path Ordering. This new definition appears indeed to capture the essence of computability arguments *à la Tait and Girard*, therefore explaining the name of the improved ordering.

1 Introduction

This paper addresses the problem of automating termination proofs for typed higher-order calculi.

The first attempt we know of goes back to Breazu-Tannen and Gallier [24] and Okada [44]. Following up a pioneering work of BreazuTannen who considered the confluence of such calculi [23], both groups of authors showed independently that proving strong normalization of a polymorphic lambda-calculus with first-order constants defined by first-order rewrite rules was reducible to the termination proof of the set of rewrite rules: beta-reduction need not be considered. Both works used Girard's method based on *reducibility candidates* -also called sometimes *computability predicates*. They then gave rise to a whole new area, by extending the type discipline, and by extending the kind of rules that could be taken care of.

The type discipline was extended soon later independently by Barbanera and Dougerthy in order to cover the whole calculus of constructions [3,28].

Higher-order rewrite rules satisfying the *general schema*, a generalization of Gödel's primitive recursion rules for higher types, were then introduced by Jouannaud and Okada [34,35] in the case of a polymorphic type discipline. The latter work was then extended first by Barbanera and Fernandez [4,5] and finally by Barbanera, Fernandez and Geuvers to cover the whole calculus of constructions [6].

It turned out that recursors for *simple* inductive types could be taken care of by the general schema, but arbitrary strict inductive types could not, prompting for an extension of the schema, which was reformulated for that purpose by Blanqui, Jouannaud and Okada [16]. This new formulation was based on the notion of *computability closure* of a term $f(\overline{s})$ headed by a higher-order constant f, defined as a set containing the immediate subterms $\overline{s}$ of $f(\overline{s})$ and closed under computability preserving operations in the sense of Tait and Girard. Membership to the general schema was then defined for an

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 1–14, 2008.
© Springer-Verlag Berlin Heidelberg 2008

arbitrary rewrite rule as membership of its right-hand side to the computability closure of its left-hand side.

Besides being elegant, this formulation was indeed much more flexible and powerful. By allowing for more expressive rules *at the object level* of the calculus of constructions, it could handle many more inductive types than originally. The general schema was finally extended by Blanqui in a series of papers by allowing for *recursive rules on types*, in order to cover the entire calculus of inductive constructions including strong elimination rules [13,14].

The definition of the general schema used a precedence on higher-order constants, as does Dershowitz recursive path ordering for first-order terms [26]. This suggested generalizing this ordering to the higher-order case, a work done by Jouannaud and Rubio in the case of a simple type discipline under the name of HORPO [37]. Comparing two terms with HORPO starts by comparing their types under a given well-founded quasi-ordering on types before to proceed recursively on the structure of the compared terms, comparing first in the precedence the higher-order constants heading both terms. Following the recursive path ordering tradition, a subterm of the left-hand side could also be compared with the whole right-hand side, regardless of the precedence on their heads.

HORPO was then extended to cover the case of the calculus of constructions by Walukiewicz [51], and to use semantic interpretations of terms instead of a precedence on function symbols by Borralleras and Rubio [21]. HORPO was also improved by the two original authors in two different ways: by comparing in the so-called subterm case an arbitrary term belonging to the computability closure of the left-hand side term with the right-hand side term, therefore generalizing both HORPO and the general schema; and by allowing for a restricted polymorphic discipline [40]. An axiomatic presentation of the rules underlying HORPO can be found in [31]. A more recent work in the same direction is [27].

The ordering and the computability closure definitions turn out to share many similar constructs, raising expectations for a simpler and yet more expressive definition, instead of a pair of mutually inductive definitions for the computability closure and the ordering itself, as advocated in [17]. These expectations were partly met, on the one hand in [15] with a single computability oriented definition, and on the other hand in [18] where a new, syntax oriented recursive definition was given for HORPO. In contrast with the previous definitions, bound variables were handled explicitly by the ordering, allowing for arbitrary abstractions in the right-hand sides.

A third, different line of work was started by van de Pol and Schwichtenberg, who aimed at (semi)-automating termination proofs of higher-order rewrite rules based on higher-order pattern matching, a problem generally considered as harder as the previous one [47,49,48]. Related attempts with more automation appear in [43,38], but were rather unconclusive for practical applications. The general schema was then adapted by Blanqui to cover the case of higher-order pattern matching [11]. Finally, Jouannaud and Rubio showed how to turn any well-founded ordering on higher-order terms including beta and eta, into a well-founded ordering for proving termination of such higher-order rules, and introduced a very simple modification of HORPO as an application of this result [36].

A fourth line of work was started by Borralleras and Rubio. Among other material, Borralleras thesis [20] contained a constraint-based approach to the semantic path ordering [41] which was shown to encompass the dependency pairs method of Arts and Giesl [2,30] in all its various aspects. Besides the thesis itself, the principles underlying this work are also described in [21] and [22]. An interesting aspect is that they lift to the higher-order case. Extending the dependency pairs method to the higher-order case was also considered independently by Sakai *et al* [46,45] and Blanqui [10].

Finally, a last line of work addresses the question of proving termination of higher-order programs. This is of course a slightly different question, usually addressed by using abstract interpretations. These interpretations may indeed use the general schema or HORPO as a basic ingredient for comparing inputs of a recursive call to those of the call they originate from. This line of work includes [32,25,8,52,1,7,12,29]. An important related work, considering pure lambda terms, is [19].

We believe that our quest shall be shown useful for all these lines of work, either as a building block, or as a guiding principle.

In this paper, we first slightly improve the definition of HORPO in the very basic case of a simple type discipline, and rename it as the Computability Path Ordering. We then address the treatment of inductive types which remained *ad hoc* so far, therefore concluding our quest thanks to the use of accessibility, a relationship which was shown to generalize the notion of inductive type by Blanqui [13,14]. We finally list which are the most important question to be addressed for those who would like to start a new quest.

2 Higher-Order Algebras

Polymorphic higher-order algebras are introduced in [40]. Their purpose is twofold: to define a simple framework in which many-sorted algebra and typed lambda-calculus coexist; to allow for polymorphic types for both algebraic constants and lambda-calculus expressions. For the sake of simplicity, we will restrict ourselves to monomorphic types in this presentation, but allow us for polymorphic examples. Carrying out the polymorphic case is no more difficult, but surely more painful.

We give here the minimal set of notions to be reasonably self-contained.

Given a set $\mathcal{S}$ of *sort symbols* of a fixed arity, denoted by $s : *^n \to *$, the set of *types* is generated by the constructor $\to$ for *functional types*:

$$\mathcal{T}_\mathcal{S} := s(\mathcal{T}_\mathcal{S}^n) \mid (\mathcal{T}_\mathcal{S} \to \mathcal{T}_\mathcal{S}) \quad \text{for } s : *^n \to * \in \mathcal{S}$$

Function symbols are meant to be algebraic operators equiped with a fixed number n of arguments (called the *arity*) of respective types $\sigma_1, \ldots, \sigma_n$, and an *output type* σ. Let $\mathcal{F} = \biguplus_{\sigma_1,\ldots,\sigma_n,\sigma} \mathcal{F}_{\sigma_1\times\ldots\times\sigma_n\to\sigma}$. The membership of a given function symbol f to $\mathcal{F}_{\sigma_1\times\ldots\times\sigma_n\to\sigma}$ is called a *type declaration* and written $f : \sigma_1 \times \ldots \times \sigma_n \to \sigma$.

The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of *raw algebraic λ-terms* is generated from the signature $\mathcal{F}$ and a denumerable set $\mathcal{X}$ of variables according to the grammar:

$$\mathcal{T} := \mathcal{X} \mid (\lambda\mathcal{X} : \mathcal{T}_\mathcal{S}.\mathcal{T}) \mid @(\mathcal{T}, \mathcal{T}) \mid \mathcal{F}(\mathcal{T}, \ldots, \mathcal{T}).$$

The raw term $\lambda x : \sigma.u$ is an *abstraction* and $@(u, v)$ is an application. We may omit σ in $\lambda x : \sigma.u$ and write $@(u, v_1, \ldots, v_n)$ or $u(v_1, \ldots, v_n)$, $n > 0$, omitting applications. $\mathcal{V}ar(t)$ is the set of free variables of t. A raw term t is *ground* if $\mathcal{V}ar(t) = \emptyset$. The notation $\overline{s}$ shall be ambiguously used for a list, a multiset, or a set of raw terms $s_1, \ldots, s_n$.

Raw terms are identified with finite labeled trees by considering $\lambda x : \sigma.u$, for each variable x and type σ, as a unary function symbol taking u as argument to construct the raw term $\lambda x : \sigma.u$. *Positions* are strings of positive integers. $t|_p$ denotes the *subterm* of t at position p. We use $t \trianglerighteq t|_p$ for the subterm relationship. The result of replacing $t|_p$ at position p in t by u is written $t[u]_p$.

Typable raw terms are called *terms*. The typing judgements are standard. We categorize terms into three disjoint classes:

1. *Abstractions* headed by λ;
2. *Prealgebraic* terms headed by a function symbol, assuming (for the moment) that the output type of $f \in \mathcal{F}$ is a base type;
3. *Neutral* terms are variables or headed by an application.

Substitutions, rewrite rules and higher-order reduction orderings are as expected, see [40].

3 The Computability Path Ordering

CPO is generated from three basic ingredients: a *type ordering*; a *precedence* on functions symbols; and a *status* for the function symbols. Accessibility is an additionnal ingredient originating in inductive types, while the other three were already needed for defining HORPO. We describe these ingredients before defining the computability path ordering. We define the ordering in two steps, accessibility being used in the second step only. The first ordering is therefore simpler, while the second is more expressive.

3.1 Basic Ingredients

- a *precedence* $\geq_{\mathcal{F}}$ on symbols in $\mathcal{F} \cup \{@\}$, with $f >_{\mathcal{F}} @$ for all $f \in \mathcal{F}$.
- a status for symbols in $\mathcal{F} \cup \{@\}$ with $@ \in Mul$.
- and a quasi-ordering on types $\geq_{\mathcal{TS}}$ called *the type ordering* satisfying the following properties, where $=_{\mathcal{TS}}$ denotes its associated equivalence relation $\geq_{\mathcal{TS}} \cap \leq_{\mathcal{TS}}$ and $>_{\mathcal{TS}}$ its strict part $\geq_{\mathcal{TS}} \setminus \leq_{\mathcal{TS}}$:
  1. *Well-foundedness*: $>_{\mathcal{TS}}^{\rightarrow} = >_{\mathcal{TS}} \cup \rhd_{\rightarrow}$ is well-founded, where $\sigma \rightarrow \tau \rhd_{\rightarrow} \sigma$;
  2. *Right arrow subterm*: $\sigma \rightarrow \tau >_{\mathcal{TS}} \tau$;
  3. *Arrow preservation*: $\tau \rightarrow \sigma =_{\mathcal{TS}} \alpha$ iff $\alpha = \tau' \rightarrow \sigma'$, $\tau' =_{\mathcal{TS}} \tau$ and $\sigma =_{\mathcal{TS}} \sigma'$;
  4. *Arrow decreasingness*: $\tau \rightarrow \sigma >_{\mathcal{TS}} \alpha$ implies $\sigma \geq_{\mathcal{TS}} \alpha$ or else $\alpha = \tau' \rightarrow \sigma'$, $\tau' =_{\mathcal{TS}} \tau$ and $\sigma >_{\mathcal{TS}} \sigma'$;

Arrow preservation and decreasingness imply that the type ordering does *not*, in general, have the left arrow subterm property: $\sigma \rightarrow \tau \not\geq_{\mathcal{TS}} \sigma$. A first axiomatic definition

of the type ordering was given in [39], which did not need right arrow subterm. A new one, expected to be easier to understand, was given in [40] based solely on $\geq_{\mathcal{T}_S}$, which uses another axiom, *arrow monotonicity*, to force the right arrow subterm property. As pointed out to us recently, this set of axioms is unfortunately inconsistent [50]. However, the restriction of the recursive path ordering proposed there for a type ordering does not satisfy arrow monotonicity, but does satisfy instead the corrected set of axioms given here.

We now give two important properties of the type ordering:

Lemma 1. *[40] Assuming $\sigma =_{\mathcal{T}_S} \tau$, σ is a data type iff τ is a data type.*

Lemma 2. *If $\alpha \to \sigma \geq_{\mathcal{T}_S} \beta \to \tau$ then $\sigma \geq_{\mathcal{T}_S} \tau$.*

Proof. If $\alpha \to \sigma =_{\mathcal{T}_S} \beta \to \tau$ then, by arrow preservation, $\alpha =_{\mathcal{T}_S} \beta$ and $\sigma =_{\mathcal{T}_S} \tau$. If $\alpha \to \sigma >_{\mathcal{T}_S} \beta \to \tau$, then, by arrow decreasingness, either $\alpha =_{\mathcal{T}_S} \beta$ and $\sigma >_{\mathcal{T}_S} \tau$, or else $\sigma >_{\mathcal{T}_S} \beta \to \tau$. In the latter case, $\beta \to \tau >_{\mathcal{T}_S} \tau$ by right arrow subterm and we conclude by transitivity. □

3.2 Notations

Our ordering notations are as follows:

- $s \succ^X t$ for the main ordering, with a finite set of variables $X \subset \mathcal{X}$ and the convention that X is omitted when empty;
- $s : \sigma \succ^X_{\mathcal{T}_S} t : \tau$ for $s \succ^X t$ and $\sigma \geq_{\mathcal{T}_S} \tau$;
- $l : \sigma \succ_{\mathcal{T}_S} r : \tau$ as initial call for each $l \to r \in R$;
- $s \succ \overline{t}$ is a shorthand for $s \succ u$ for all $u \in \overline{t}$;
- $\succeq$ is the reflexive closure of $\succ$.

We can now introduce the definition of CPO.

3.3 Ordering Definition

Definition 1. *$s : \sigma \succ^X t : \tau$ iff either:*

1. *$s = f(\overline{s})$ with $f \in \mathcal{F}$ and either of*
   (a) $t \in X$
   (b) $t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in \mathcal{F}$, $s \succ^X \overline{t}$ and $\overline{s}(\succ_{\mathcal{T}_S})_{stat_f} \overline{t}$
   (c) $t = g(\overline{t})$ with $f >_{\mathcal{F}} g \in \mathcal{F} \cup \{@\}$ and $s \succ^X \overline{t}$
   (d) $t = \lambda y : \beta.w$ and $s \succ^{X \cup \{z\}} w\{y \mapsto z\}$ for $z : \beta$ fresh
   (e) $u \succeq_{\mathcal{T}_S} t$ for some $u \in \overline{s}$
2. *$s = @(u, v)$ and either of*
   (a) $t \in X$
   (b) $t = @(u', v')$ and $\{u, v\}(\succ_{\mathcal{T}_S})_{mul}\{u', v'\}$
   (c) $t = \lambda y : \beta.w$ and $s \succ^X w\{y \mapsto z\}$ for $z : \beta$ fresh
   (d) $u \succeq^X_{\mathcal{T}_S} t$ or $v \succeq^X_{\mathcal{T}_S} t$
   (e) $u = \lambda x : \alpha.w$ and $w\{x \mapsto v\} \succeq^X t$

3. $s = \lambda x : \alpha.u$ *and either of*
   (a) $t \in X$
   (b) $t = \lambda y : \beta.w$, $\alpha =_{\mathcal{T}_S} \beta$ *and* $u\{x \mapsto z\} \succ^X w\{y \mapsto z\}$ *for* $z : \beta$ *fresh*
   (c) $t = \lambda y : \beta.w$, $\alpha \neq_{\mathcal{T}_S} \beta$ *and* $s \succ^X w\{y \mapsto z\}$ *for* $z : \beta$ *fresh*
   (d) $u\{x \mapsto z\} \succeq^X_{\mathcal{T}_S} t$ *for* $z : \alpha$ *fresh*
   (e) $u = @(v, x)$, $x \notin \mathcal{V}ar(v)$ *and* $v \succeq^X t$

Because function symbols, applications and abstractions do not behave exactly the same, we chosed to organize the definition according to the left-hand side head symbol: a function symbol, an application, or an abstraction successively. In all three cases, we first take care of the case where the right-hand side is a bound variable -case named *variable*-, then headed by a symbol which is the same as (or equivalent to) the left-hand side head symbol -case *status*-, or headed by a symbol which is strictly smaller in the precedence than the left-hand side head symbol -case *precedence*-, before to go with the -case *subterm*. The precedence case breaks into two sub-cases when the left-hand side is a function symbol, because abstractions, which can be seen as smaller than other symbols, need renaming of their bound variable when pulled out, which makes their treatment a little bit different formally from the standard precedence case. There are two specific cases for *application* and *abstraction*: one for beta-reduction, and one for eta-reduction, which are both built in the definition.

This new definition schema appeared first in [18] in a slightly different format. It incorporates two major innovations with respect to the version of HORPO defined in [40]. The first is that terms can be ordered without requiring that their types are ordered accordingly. This will be the case whenever we can conclude that some recursive call is terminating by using computability arguments rather than an induction on types. Doing so, the ordering inherits directly much of the expressivity of the computability closure schema used in [40]. The second is the annotation of the ordering by the set of variables X that were originally bound in the right-hand side term, but have become free when taking some subterm. This allows rules 1d, 2c and 3c to pull out abstractions from the right-hand side regardless of the left-hand side term, meaning that abstractions are smallest in the precedence. Among the innovations with respect to [18] are rules 3c, which compares abstractions whose bound variables have non-equivalent types, and rule 2d, whose formulation is now stronger.

This definition suffers some subtle limitations:

1. Case 1d uses recursively the comparison $s \succ^{X \cup \{z\}} w\{y \mapsto z\}$ for z fresh, implying that the occurrences of z in w can be later taken care of by Case 1a, 2a or 3a. This is no limitation.

   Cases 2c and 3c use instead the recursive comparison $s \succ^X w\{y \mapsto z\}$, with z fresh, hence $z \notin X$. As a consequence, these recursive calls cannot succeed if $z \in \mathcal{V}ar(w)$. We could have added this redundant condition for sake of clarity. We prefered to privilege uniformity and locality of tests.

   As a consequence, Cases 1d, 2c and 3c cannot be packed together as it was unfortunately done in [18], where correct proofs were however given which did of course not correspond to the given definition.

2. The subterm case 1e uses recursively the comparison $u \succ_{\mathcal{T}_\mathcal{S}} t$ instead of the expected comparison $u \succ^X_{\mathcal{T}_\mathcal{S}} t$.

   On the other hand, the other subterm definitions, Cases 2d and 3d use the expected comparisons $u \succeq^X_{\mathcal{T}_\mathcal{S}} t$ or $v \succeq^X_{\mathcal{T}_\mathcal{S}} t$ in the first case, and $u\{x \mapsto z\} \succeq^X_{\mathcal{T}_\mathcal{S}} t$ in the second. This implies again that the various subterm cases cannot be packed together.
3. Case 1b uses recursively the comparison $\overline{s}(\succ_{\mathcal{T}_\mathcal{S}})_{stat_f}\overline{t}$ instead of the stronger comparison $\overline{s}(\succ^X_{\mathcal{T}_\mathcal{S}})_{stat_f}\overline{t}$.

All our restrictions are justified by their use in the well-foundedness proof of $\succ_{\mathcal{T}_\mathcal{S}}$. There is an even better argument: the ordering would not be well-founded otherwise, as can be shown by means of counter-examples. We give two below.

We start with an example of non-termination obtained when replacing the recursive call $\overline{s}(\succ_{\mathcal{T}_\mathcal{S}})_{stat_f}\overline{t}$ by $\overline{s}(\succ^X_{\mathcal{T}_\mathcal{S}})_{stat_f}\overline{t}$ in Case 1b.

Example 1. Let a be a type, and $\{f : a \times a \to a, g : (a \to a) \to a\}$ be the signature. Let us consider the following non-terminating rule (its right-hand side beta-reduces to its left-hand side in one beta-step):

$$f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \to @(\lambda x.f(x,x), g(\lambda x.f(x,x)))$$

Let us assume that $f >_\mathcal{F} g$ and that f has a multiset status. We now show that the ordering modified as suggested above succeeds with the goal

1. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ_{\mathcal{T}_\mathcal{S}} @(\lambda x.f(x,x), g(\lambda x.f(x,x)))$.

Since type checks are trivial, we will omit them, although the reader will note that there are very few of them indeed. Our goal yields two sub-goals by Case 1c:

2. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ \lambda x.f(x,x)$ and
3. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ g(\lambda x.f(x,x))$.

Sub-goal 2 yields by Case 1d

4. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ^{\{z\}} f(z,z)$ which yields by Case 1b
5. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ^{\{z\}} z$ twice, solved by Case 1a and
6. $\{g(\lambda x.f(x,x)), g(\lambda x.f(x,x))\}(\succeq^{\{z\}}_{\mathcal{T}_\mathcal{S}})\{z,z\}$ solved by Case 1a applied twice.

We are left with sub-goal 3 which yields by Case 1c

7. $f(g(\lambda x.f(x,x)), g(\lambda x.f(x,x))) \succ \lambda x.f(x,x)$, which happens to be the already solved sub-goal 2, and we are done.

With the definition we gave, sub-goal 6 becomes:
$\{g(\lambda x.f(x,x)), g(\lambda x.f(x,x))\}(\succ_{\mathcal{T}_\mathcal{S}})_{mul}\{z,z\}$ and does not succeed since the set of previously bound variables has been made empty.

The reader can check that chosing the precedence $g >_\mathcal{F} f$ yields exactly the same result in both cases. □

Next is an example of non-termination due to Cynthia Kop and Femke van Raamsdong [50], obtained when replacing the recursive call $s \succ^X w\{y \mapsto z\}$ by $s \succ^{X \cup \{z\}} w\{y \mapsto z\}$ in Case 2c.

Example 2. Let o be a type, and $\{f : o \to o, A : o, B : o \to o \to o\}$ be the signature. Let us consider the following non-terminating set of rules:

$$\begin{aligned} @(@(B, A), A) &\to @(\lambda z : o.f(z), A) && (1)\\ f(A) &\to @(@(B, A), A) && (2) \end{aligned}$$

since

$$@(@(B, A), A) \underset{1}{\longrightarrow} @(\lambda z : o.f(z), A) \underset{\beta}{\longrightarrow} f(A) \underset{2}{\longrightarrow} @(@(B, A), A)$$

Let us assume that $A >_{\mathcal{F}} f >_{\mathcal{F}} B$ and consider the goals:

1. $@(@(B, A), A) : o \succ_{\mathcal{T}_S} @(\lambda z : o.f(z), A) : o$, and
2. $f(A) : o \succ_{\mathcal{T}_S} @(@(B, A), A) : o$.

Goal 1 yields two sub-goals by Case 2b:

3. $A : o \succeq_{\mathcal{T}_S} A : o$, which succeeds trivially, and
4. $@(B, A) : o \to o \succ_{\mathcal{T}_S} \lambda z : o.f(z) : o \to o$ which yields by modified Case 2c:
5. $@(B, A) \succ^{\{z\}} f(z)$, which yields in turn by Case 2d
6. $A : o \succ^{\{z\}}_{\mathcal{T}_S} f(z) : o$ which yields by Case 1c
7. $A : o \succ^{\{z\}}_{\mathcal{T}_S} z : o$ which succeeds by Case 1a.

Note that we have used B for its large type, and A for eliminating $f(z)$, exploiting a kind of divide and conquer ability of the ordering. We are left with goal 2 which yields two subgoals by Case 1d

8. $f(A) \succ A$ which succeeds by Case 1e, and
9. $f(A) \succ @(B, A)$, which yields by Case 1c:
10. $f(A) \succ A$, which succeeds by Case 1e, and
11. $f(A) \succ B$, which succeeds by Case 1c, therefore ending the computation. □

More examples justifying our claim that the quest has come to en end are given in the full version of this paper.

We give now an example of use of the computability path ordering with the inductive type of Brouwer's ordinals, whose constructor lim takes an infinite sequence of ordinals to build a new, limit ordinal, hence admits a functional argument of type $\mathbb{N} \to O$, in which O occurs positively. As a consequence, the recursor admits a more complex structure than that of natural numbers, with an explicit abstraction in the right-hand side of the rule for lim. The strong normalization proof of such recursors is known to be hard.

Example 3. Brouwer's ordinals.

$0 : O \qquad S : O \to O \qquad lim : (\mathbb{N} \to O) \to O$
$rec : O \times \alpha \times (O \to \alpha \to \alpha) \times ((\mathbb{N} \to O) \to (\mathbb{N} \to \alpha) \to \alpha) \to \alpha$

The rules defining the recursor on Brouwer's ordinals are:

$rec(0, U, X, W) \to U$
$rec(S(n), U, X, W) \to @(X, n, rec(n, U, X, W))$
$rec(lim(F), U, X, W) \to @(W, F, \lambda n.rec(@(F, n), U, X, W))$

Let us try to prove that the third rule is in $\succ_{\mathcal{T}_S}$.

1. $s = rec(lim(F), U, X, W) \succ_{\mathcal{T}_S} @(W, F, \lambda n.rec(@(F, n), U, X, W))$ yields 4 subgoals according to Case 1c:
2. $\alpha \geq_{\mathcal{T}_S} \alpha$ which is trivially satisfied, and
3. $s \succ \{W, F, \lambda n.rec(@(F, n), U, X, W)\}$ which simplifies to:
4. $s \succ W$ which succeeds by Case 1e,
5. $s \succ F$, which generates by Case 1e the comparison $lim(F) \succ_{\mathcal{T}_S} F$ which fails since $lim(F)$ has a type which is strictly smaller than the type of F.
6. $s \succ \lambda n.rec(@(F, n), U, X, W)$ which yields by Case 1d
7. $s \succ^{\{n\}} rec(@(F, n), U, X, W)$ which yields by Case 1b
8. $\{lim(F), U, X, W\}(\succ_{\mathcal{T}_S})_{mul}\{@(F, n), U, X, W\}$, which reduces to
9. $lim(F) \succ_{\mathcal{T}_S} @(F, n)$, whose type comparison succeeds, yielding by Case 1c
10. $lim(F) \succ F$ which succeeds by Case 1e, and
11. $lim(F) \succ n$ which fails because track of n has been lost!

Solving this example requires therefore: first, to access directly the subterm F of s in order to avoid the type comparison for $lim(F)$ and F when checking recursively whether the comparison $s \succ \lambda n.rec(@(F, n), U, X, W)$ holds; and second, to keep track of n when comparing $lim(F)$ and n.

3.4 Accessibility

While keeping the same type structure, we make use here of a fourth ingredient, the *accessibility* relationship for data types introduced in [11]. This will allow us to solve Brouwer's example, as well as other examples of non-simple inductive types.

We say that a data type is *simple* is it is a type constant. We restrict here our definition of accessibility to simple data types. To this end, we assume that all type constructors are constants, that is, have arity zero. We can actually do a little bit more, assuming that simple data types are not greater or equal (in $\geq_{\mathcal{T}_S}$) to non-constant data types, allowing the simple data types to live in a separate world.

The sets of *positive and negative positions* in a type σ are inductively defined as follows:

- $Pos^+(\sigma) = \{\varepsilon\}$ if σ is a simple data type
- $Pos^-(\sigma) = \emptyset$ if σ is a simple data type
- $Pos^\delta(\sigma \to \tau) = 1 \cdot Pos^{-\delta}(\sigma) \cup 2 \cdot Pos^\delta(\tau)$
  where $\delta \in \{+, -\}$, $-+ = -$ and $-- = +$ (usual rules of signs)

Then we say that a simple data type σ occurs (only) *positively* in a type τ if it occurs only at positive positions: $Pos(\sigma, \tau) \subseteq Pos^+(\tau)$, where $Pos(\sigma, \tau)$ is the set of positions of the occurrences of σ in τ.

The set $Acc(f)$ of *accessible argument positions* of a function symbol $f : \sigma_1 \dots \sigma_n \to \sigma$, where σ is a simple data type, is the set of integers $i \in \{1, \dots, n\}$ such that:

- no simple data type greater than σ occurs in σ_i,
- simple data types equivalent to σ occurs only positively in σ_i.

Then a term u is *accessible* in a term v, written $v \rhd_{acc} u$, iff v is a pre-algebraic term $f(\overline{s})$ and there exists $i \in Acc(f)$ such that either $u = s_i$ or u is accessible in s_i ($\rhd_{acc}$ is transitive).

A term u is accessible in a sequence of terms $\overline{v}$ iff it is accessible in some $v \in \overline{v}$, in which case we write $\overline{s} \rhd_{acc} u$. Note that the terms accessible in a term v are strict subterms of v.

We can now obtain a more elaborated ordering as follows:

Definition 2. *$s : \sigma \succ^X t : \tau$ iff either:*

1. *$s = f(\overline{s})$ with $f \in \mathcal{F}$ and either of*
   (a) *$t \in X$*
   (b) *$t = g(\overline{t})$ with $f =_{\mathcal{F}} g \in \mathcal{F}$, $s \succ^X \overline{t}$ and $\overline{s}(\succ_{\mathcal{I}_S} \cup \succ^{X,s}_{acc})_{stat_f} \overline{t}$*
   (c) *$t = g(\overline{t})$ with $f >_{\mathcal{F}} g \in \mathcal{F} \cup \{@\}$ and $s \succ^X \overline{t}$*
   (d) *$t = \lambda y : \beta.w$ and $s \succ^{X \cup \{z\}} w\{y \mapsto z\}$ for $z : \beta$ fresh*
   (e) *$u \succeq_{\mathcal{I}_S} t$ for some $u \in \overline{s}$*
   (f) *$u \succeq_{\mathcal{I}_S} t$ for some u such that $\overline{s} \rhd_{acc} u$*
2. *$s = @(u, v)$ and either of*
   (a) *$t \in X$*
   (b) *$t = @(u', v')$ and $\{u, v\}(\succ_{\mathcal{I}_S})_{mul}\{u', v'\}$*
   (c) *$t = \lambda y : \beta.w$ and $s \succ^X w\{y \mapsto z\}$ for $z : \beta$ fresh*
   (d) *$u \succeq^X_{\mathcal{I}_S} t$ or $v \succeq^X_{\mathcal{I}_S} t$*
   (e) *$u = \lambda x : \alpha.w$ and $w\{x \mapsto v\} \succeq^X t$*
3. *$s = \lambda x : \alpha.u$ and either of*
   (a) *$t \in X$*
   (b) *$t = \lambda y : \beta.w$, $\alpha =_{\mathcal{I}_S} \beta$ and $u\{x \mapsto z\} \succ^X w\{y \mapsto z\}$ for $z : \beta$ fresh*
   (c) *$t = \lambda y : \beta.w$, $\alpha \neq_{\mathcal{I}_S} \beta$ and $s \succ^X w\{y \mapsto z\}$ for $z : \beta$ fresh*
   (d) *$u\{x \mapsto z\} \succeq^X_{\mathcal{I}_S} t$ for $z : \alpha$ fresh*
   (e) *$u = @(v, x)$, $x \notin \mathcal{V}ar(v)$ and $v \succeq^X t$*

where $u : \sigma \succ^{X,s}_{acc} t : \tau$ iff $\sigma \geq_{\mathcal{I}_S} \tau$, $t = @(v, \overline{w})$, $u \rhd_{acc} v$ and $s \succ^X \overline{w}$.

The only differences with the previous definition are in Case 1b of the main definition which uses an additional ordering $\succ^{X,s}_{acc}$ based on the accessibility relationship $\rhd_{acc}$ to compare subterms headed by equivalent function symbols, and in Case 1f which uses the same relationship $\rhd_{acc}$ to reach deep subterms that could not be reached otherwise. Following up a previous discussion, notice that we have kept the same formulation in Cases 2c and 3c, rather than use the easier condition $y \notin \mathcal{V}ar(w)$.

We could of course strengthen $\succ_{acc}^{X,s}$ by giving additional cases, for handling abstractions and function symbols on the right [11,15]. We could also think of improving Case 1e by replacing $\overline{s} \rhd_{acc} u$ by the stronger condition $\overline{s} \succ_{acc}^{X,s} u$. We have not tried these improvements yet.

We now revisit Brouwer's example, whose strong normalization proof is checked automatically by this new version of the ordering:

Example 4. Brouwer's ordinals.

We skip goals 2,3,4 which do not differ from the previous attempt.

1. $s = rec(lim(F), U, X, W) \succ_{\mathcal{T}_S} @(W, F, \lambda n.rec(@(F, n), U, X, W))$ yields 4 subgoals according to Case 1c:
5. $s \succ F$, which succeeds now by Case 1f,
6. $s \succ \lambda n.rec(@(F, n), U, X, W)$ which yields by Case 1d
7. $s \succ^{\{n\}} rec(@(F, n), U, X, W)$ which yields goals 8 and 12 by Case 1b
8. $\{lim(F), U, X, W\}(\succ_{\mathcal{T}_S} \cup \succ_{acc}^{\{n\},s})_{mul}\{@(F, n), U, X, W\}$, which reduces to
9. $lim(F) \succ_{acc}^{\{n\},s} @(F, n)$ which succeeds since $O =_{\mathcal{T}_S} O$, F is accessible in $lim(F)$ and $s \succ^{\{n\}} n$ by case Case 1a. Our remaining goal
10. $s \succ^{\{n\}} \{@(F, n), U, X, W\}$
    decomposes into three goals trivially solved by Case 1e, that is
11. $s \succ^{\{n\}} \{U, X, W\}$, and one additional goal
12. $s \succ^{\{n\}} @(F, n)$ which yields two goals by Case 1c
13. $s \succ^{\{n\}} F$, which succeeds by Case 1f, and
14. $s \succ^{\{n\}} n$ which succeeds by Case 1a, thus ending the computation.

4 Conclusion

The full version including all proofs as well as an implementation of CPO with examples is available from the web page of the authors.

We want to stress that the basic version of CPO has reached a point where we cannot expect any major improvement, as indicated by the counter-examples found to our own attempts to improve the ordering. Perhaps, one last question left open is the possibility of ordering $\mathcal{F} \cup \{@\}$ arbitrarily -this would be useful for some examples, e.g., some versions of Jay's pattern calculus [33].

On the other hand, there is room left for improvement of the accessibility relationship, which is restricted so far to terms headed by function symbols having a basic output type.

A more challenging problem to be investigated then is the generalization of this new definition to the calculus of constructions along the lines of [51] and the suggestions made in [40], where an RPO-like ordering on types was proposed which allowed to give a single definition for terms and types. Starting this work with definition 1 is of course desirable.

Finally, it appears that the recursive path ordering and the computability closure are kind of dual of each other: the definitions are quite similar, the closure constructing a set of terms while the ordering deconstructs terms to be compared, the basic case being

the same: bound variables and subterms. Besides, the properties to be satisfied by the type ordering, infered from the proof of the computability predicates, almost characterize a recursive path ordering on the first-order type structure. An intriguing, challenging question is therefore to understand the precise relationship between computability predicates and path orderings.

Acknowledgements. The second author wishes to point out the crucial participation of Mitsuhiro Okada to the very beginning of this quest, and to thank Makoto Tatsuta for inviting him in december 2007 at the National Institute for Informatics in Tokyo, whose support provided him with the ressources, peace and impetus to conclude this quest with his coauthors. We are also in debt with Cynthia Kop and Femke van Raamsdonk for pointing out to us a (hopefully minor) mistake in published versions of our work on HORPO.

References

1. Abel, A.: Termination checking with types. Theoretical Informatics and Applications 38(4), 277–319 (2004)
2. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236, 133–178 (2000)
3. Barbanera, F.: Adding algebraic rewriting to the Calculus of Constructions: strong normalization preserved. In: Okada, M., Kaplan, S. (eds.) CTRS 1990. LNCS, vol. 516. Springer, Heidelberg (1991)
4. Barbanera, F., Fernández, M.: Combining first and higher order rewrite systems with type assignment systems. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664. Springer, Heidelberg (1993)
5. Barbanera, F., Fernández, M.: Modularity of termination and confluence in combinations of rewrite systems with λ_ω. In: Lingas, A., Carlsson, S., Karlsson, R. (eds.) ICALP 1993. LNCS, vol. 700. Springer, Heidelberg (1993)
6. Barbanera, F., Fernández, M., Geuvers, H.: Modularity of strong normalization and confluence in the algebraic-λ-cube. In: Proceedings of the 9th IEEE Symposium on Logic in Computer Science (1994)
7. 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)
8. Ben-Amram, A.M., Jones, N.D., Lee, C.S.: The size-change principle for program termination. In: Proceedings of the 28th ACM Symposium on Principles of Programming Languages (2001)
9. Blanqui, F.: Definitions by rewriting in the Calculus of Constructions (extended abstract). In: Proceedings of the 16th IEEE Symposium on Logic in Computer Science (2001)
10. Blanqui, F.: Higher-order dependency pairs. In: Proceedings of the 8th International Workshop on Termination (2006)
11. Blanqui, F.: Termination and confluence of higher-order rewrite systems. In: Bachmair, L. (ed.) RTA 2000. LNCS, vol. 1833. Springer, Heidelberg (2000)
12. 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)
13. Blanqui, F.: Definitions by rewriting in the Calculus of Constructions. Mathematical Structures in Computer Science 15(1), 37–92 (2005)

14. Blanqui, F.: Inductive types in the Calculus of Algebraic Constructions. Fundamenta Informaticae 65(1-2), 61–86 (2005)
15. Blanqui, F.: Computability closure: Ten years later. In: Comon-Lundh, H., Kirchner, C., Kirchner, H. (eds.) Jouannaud Festschrift. LNCS, vol. 4600, pp. 68–88. Springer, Heidelberg (2007)
16. Blanqui, F., Jouannaud, J.-P., Okada, M.: Inductive-data-type Systems. Theoretical Computer Science 272, 41–68 (2002)
17. Blanqui, F., Jouannaud, J.-P., Rubio, A.: Higher-order termination: from Kruskal to computability. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 1–14. Springer, Heidelberg (2006)
18. Blanqui, F., Jouannaud, J.-P., Rubio, A.: HORPO with computability closure: A reconstruction. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 138–150. Springer, Heidelberg (2007)
19. Bohr, N., Jones, N.: Termination Analysis of the untyped lambda-calculus. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 1–23. Springer, Heidelberg (2004)
20. Borralleras, C.: Ordering-based methods for proving termination automatically. PhD thesis, Universitat Politècnica de Catalunya, Spain (2003)
21. Borralleras, C., Rubio, A.: A monotonic higher-order semantic path ordering. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250. Springer, Heidelberg (2001)
22. Borralleras, C., Rubio, A.: Orderings and constraints: Theory and practice of proving termination. In: Comon-Lundh, H., Kirchner, C., Kirchner, H. (eds.) Jouannaud Festschrift. LNCS, vol. 4600, pp. 28–43. Springer, Heidelberg (2007)
23. Breazu-Tannen, V.: Combining algebra and higher-order types. In: Proceedings of the 3rd IEEE Symposium on Logic in Computer Science (1988)
24. Breazu-Tannen, V., Gallier, J.: Polymorphic rewriting conserves algebraic strong normalization. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372. Springer, Heidelberg (1989)
25. Chin, W.N., Khoo, S.C.: Calculating sized types. Journal of Higher-Order and Symbolic Computation 14(2-3), 261–300 (2001)
26. Dershowitz, N.: Orderings for term rewriting systems. Theoretical Computer Science 17, 279–301 (1982)
27. Dershowitz, N.: Personal Communication (2008)
28. Dougherty, D.: Adding algebraic rewriting to the untyped lambda calculus. Information and Computation 101(2), 251–267 (1992)
29. Giesl, J., Swiderski, S., Schneider-Kamp, P., Thiemann, R.: Automated termination analysis for haskell: From term rewriting to programming languages. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 297–312. Springer, Heidelberg (2006)
30. 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)
31. Goubault-Larrecq, J.: Well-founded recursive relations. In: Fribourg, L. (ed.) CSL 2001 and EACSL 2001. LNCS, vol. 2142. Springer, Heidelberg (2001)
32. Hughes, J., Pareto, L., Sabry, A.: Proving the correctness of reactive systems using sized types. In: Proceedings of the 23th ACM Symposium on Principles of Programming Languages (1996)
33. Jay, C.B.: The pattern calculus. ACM Transactions on Programming Languages and Systems 26(6), 911–937 (2004)
34. Jouannaud, J.-P., Okada, M.: A computation model for executable higher-order algebraic specification languages. In: Proceedings of the 6th IEEE Symposium on Logic in Computer Science (1991)

35. Jouannaud, J.-P., Okada, M.: Abstract Data Type Systems. Theoretical Computer Science 173(2), 349–391 (1997)
36. Jouannaud, J.-P., Rubio, A.: Higher-order orderings for normal rewriting. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 387–399. Springer, Heidelberg (2006)
37. Jouannaud, J.-P., Rubio, A.: The Higher-Order Recursive Path Ordering. In: Proceedings of the 14th IEEE Symposium on Logic in Computer Science (1999)
38. Jouannaud, J.-P., Rubio, A.: Rewrite orderings for higher-order terms in eta-long beta-normal form and the recursive path ordering. Theoretical Computer Science 208, 33–58 (1998)
39. Jouannaud, J.-P., Rubio, A.: Higher-order recursive path orderings "à la carte", Draft (2001)
40. Jouannaud, J.-P., Rubio, A.: Polymorphic higher-order recursive path orderings. Journal of the ACM 54(1), 1–48 (2007)
41. Kamin, S., Lévy, J.-J.: Two generalizations of the Recursive Path Ordering (unpublished, 1980)
42. Krishnamoorthy, M.S., Narendran, P.: On recursive path ordering. Theoretical Computer Science 40(2-3), 323–328 (1985)
43. Loria-Saenz, C., Steinbach, J.: Termination of combined (rewrite and λ-calculus) systems. In: Rusinowitch, M., Remy, J.-L. (eds.) CTRS 1992. LNCS, vol. 656. Springer, Heidelberg (1993)
44. Okada, M.: Strong normalizability for the combined system of the typed lambda calculus and an arbitrary convergent term rewrite system. In: Proceedings of the 1989 International Symposium on Symbolic and Algebraic Computation. ACM Press, New York (1989)
45. Sakai, M., Kusakari, K.: On dependency pair method for proving termination of higher-order rewrite systems. IEICE Transactions on Information and Systems E88-D(3), 583–593 (2005)
46. Sakai, M., Watanabe, Y., Sakabe, T.: An extension of dependency pair method for proving termination of higher-order rewrite systems. IEICE Transactions on Information and Systems E84-D(8), 1025–1032 (2001)
47. van de Pol, J.: Termination proofs for higher-order rewrite systems. In: Heering, J., Meinke, K., Möller, B., Nipkow, T. (eds.) HOA 1993. LNCS, vol. 816. Springer, Heidelberg (1994)
48. van de Pol, J.: Termination of higher-order rewrite systems. PhD thesis, Utrecht Universiteit, Nederlands (1996)
49. van de Pol, J., Schwichtenberg, H.: Strict functionals for termination proofs. In: Dezani-Ciancaglini, M., Plotkin, G. (eds.) TLCA 1995. LNCS, vol. 902. Springer, Heidelberg (1995)
50. van Raamsdong, F., Kop, C.: Personal Communication (2008)
51. Walukiewicz-Chrzaszcz, D.: Termination of rewriting in the Calculus of Constructions. Journal of Functional Programming 13(2), 339–414 (2003)
52. Xi, H.: Dependent types for program termination verification. Journal of Higher-Order and Symbolic Computation 15(1), 91–131 (2002)

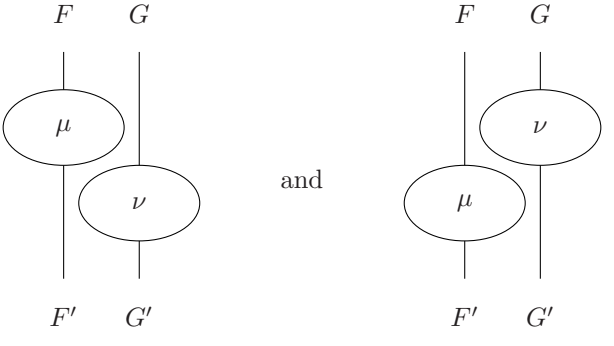
The Joy of String Diagrams

Pierre-Louis Curien

Preuves, Programmes et Systèmes, CNRS and University Paris 7

Abstract. In the past recent years, I have been using string diagrams to teach basic category theory (adjunctions, Kan extensions, but also limits and Yoneda embedding). Using graphical notations is undoubtedly joyful, and brings us close to other graphical syntaxes of circuits, interaction nets, etc... It saves us from laborious verifications of naturality, which is built-in in string diagrams. On the other hand, the language of string diagrams is more demanding in terms of typing: one may need to introduce explicit coercions for equalities of functors, or for distinguishing a morphism from a point in the corresponding internal homset. So that in some sense, string diagrams look more like a language "à la Church", while the usual mathematics of, say, Mac Lane's "Categories for the working mathematician" are more "à la Curry".

Natural transformations are traditionally represented as pasting diagrams, where natural transformations $\mu : F \to F'$ appear as surfaces between an upper border F and a lower border F'. But the dual notation of string diagrams turns out to be more adapted to formal manipulations. In this notation, the Godement's rule, which says that the pasting diagram obtained by putting aside $\mu : F \to F'$ and $\nu : G \to G'$ makes sense, i.e., can be parsed indifferently as the vertical composition $(\nu F') \circ (G\mu)$ or the vertical composition $(G'\mu) \circ (\nu F)$ – has a "physical" translation in terms of "moving elevators" up and down. The respective parsings are, indeed, represented as

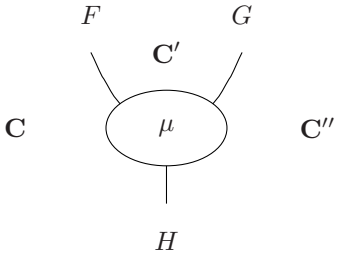


Hence, the underlying naturality equations remain explicit, in the form of suitable deformations of diagrams.

In string diagrams, functors are 1-dimensional (like in the pasting diagrams), natural transformations are 0-dimensional (think of the circle around μ, ν as just

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 15–22, 2008.
© Springer-Verlag Berlin Heidelberg 2008

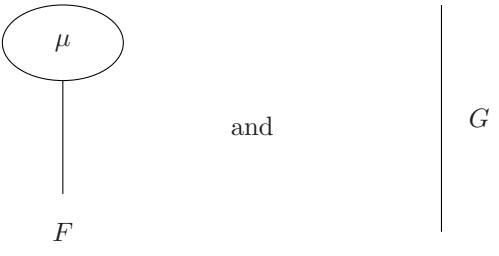
a node in a graph). As for the categories, if $F : \mathbf{C} \to \mathbf{C}'$, $G : \mathbf{C}' \to \mathbf{C}''$, and $H : \mathbf{C} \to \mathbf{C}''$, then, seeing the edges of the graph as half-lines, the diagram below representing a natural transformation $\mu : GF \to H$ (we write freely GF for $G \circ F$) delineates three regions, corresponding to the three categories. In other words, in this representation, categories are 2-dimensional.



The situation is thus Poincaré dual to that of pasting diagrams:

	categories	fonctors	natural transformations
pasting diagrams	0	1	2
string diagrams	2	1	0

Another strong point of string diagrams is that they allow us to deal with identity functors and natural transformations implicity. We represent, say, $\mu : id \to F$ (with $F : \mathbf{C} \to \mathbf{C}$), and $id : G \to G$ (with $G : \mathbf{C} \to \mathbf{C}'$) as

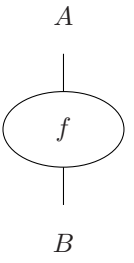


respectively.

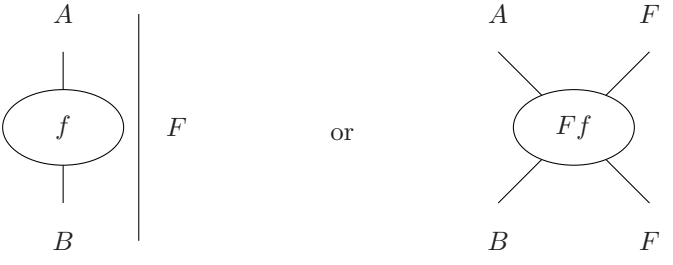
String diagrams are related to boolean circuits, interaction nets, etc... (see e.g. `http://iml.univ-mrs.fr/~lafont`). We use string diagrams (originally designed and used in the setting of monoidal categories, Hopf algebras, quantum groups, etc... , see e.g. [2]) not only for the 2-categorical machinery of adjunctions and monads, but also for recasting other basic material of category theory [3]. In this extended abstract, we content ourselves with pointing out the underlying coercions that we have to make explicit in order to treat this material graphically (see [1] for more joy!).

1 Hom-Functors

We first notice that we can also use string diagrams to describe morphisms $f : A \to B$ in a category $\mathbf{C}$. It suffices to see A and B as functors from the terminal category $\mathbf{1}$ to $\mathbf{C}$, yielding

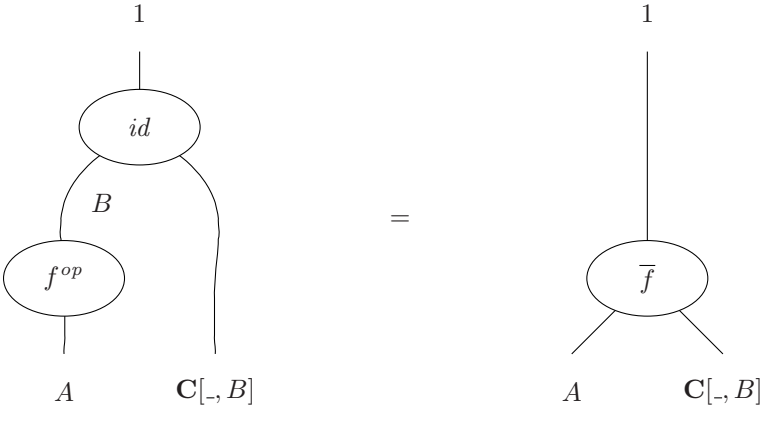


with the left and right half plane corresponding to **1** and **C**. For Ff, by definition of the horizontal composition of natural transformations, we can write indifferently (i.e., we can use the following as a valid transformation of string diagrams):

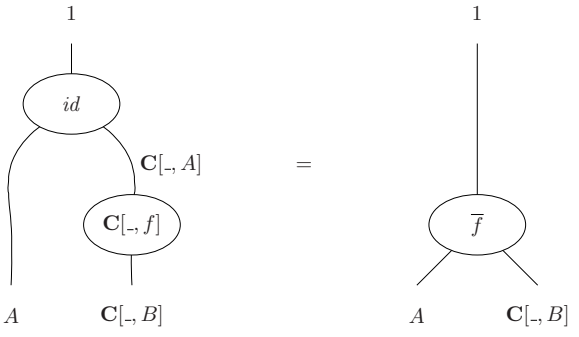


But we can also view a morphism $f : A \to B$ as a morphism $\overline{f} : 1 \to \mathbf{C}[A, B]$ in **Set** (the category of sets and functions). We use overlining to make the coercion explicit between the two representations. Then it turns out that the action of the hom functors can be described through the following equations:

EQUATION *Homleft*:



EQUATION *Homright*:



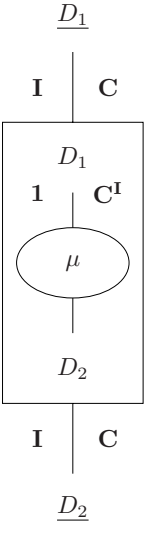
One can give graphical proofs of Yoneda lemma, and of the density of representable presheaves: every functor $F : \mathbf{C}^{op} \to \mathbf{Set}$ is the colimit of functors of the form $\mathbf{C}[\_, C]$.

2 Limits

Recall that, given a diagram $D : \mathbf{I} \to \mathbf{C}$, a cone from an object C can be described as a natural transformation from ΔC to D, where $\Delta : \mathbf{C} \to \mathbf{C}^{\mathbf{I}}$ is the curried form of the first projection functor from $\mathbf{C} \times \mathbf{I}$ to $\mathbf{C}$. This indicates that we should draw D as a functor from $\mathbf{1}$ to $\mathbf{C}^{\mathbf{I}}$. On the other hand, if we want to talk e.g. of preservation of limits, we need to deal with FD, for some functor $F : \mathbf{C} \to \mathbf{C}'$, and then we will have to view D as a functor from I to $\mathbf{C}$. Under this guise, we denote it as $\underline{D}$.

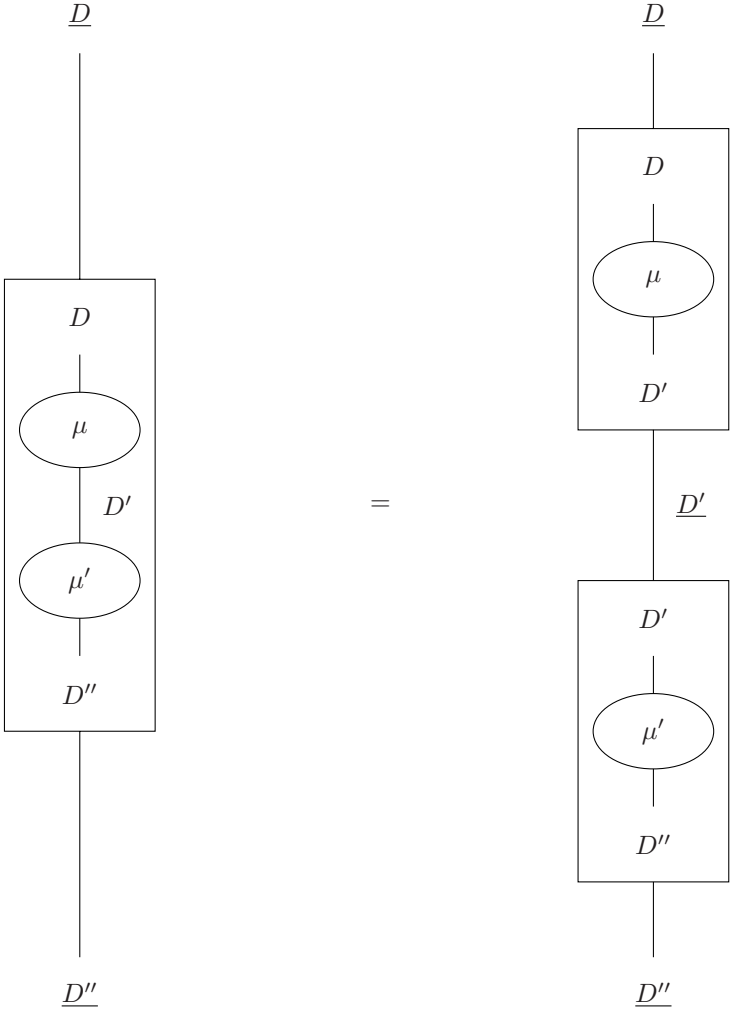
Note that in any cartesian closed category, there is a bijective correspondence betwen the morphisms from A to B and the points of B^A, i.e., the morphisms from 1 to B^A. We use here underlining as an *explicit coercion* from the latter to the former.

Graphically, we introduce boxes of the following kind:



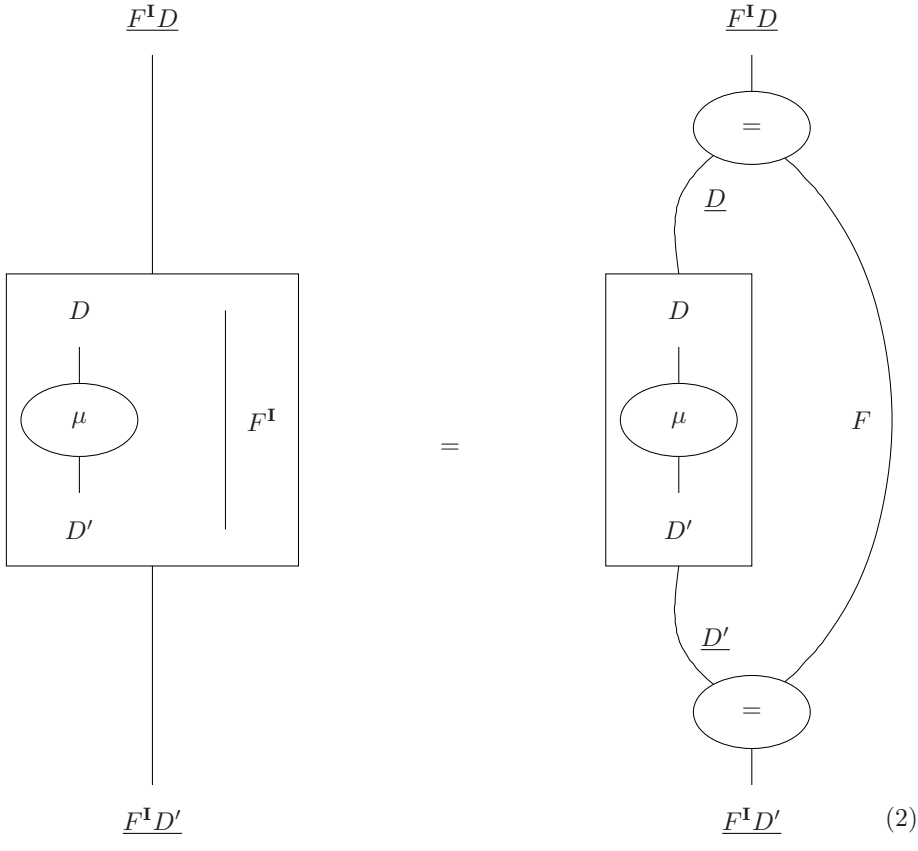
where the contents of the box is a string diagram living in $\mathbf{Cat}[\mathbf{1}, \mathbf{C}^{\mathbf{I}}]$ (where **Cat** is the category of categories) while the whole diagram, once coerced, lives in $\mathbf{Cat}[\mathbf{I}, \mathbf{C}]$, and can be inserted in a larger diagram (e.g. by placing a wire $F : \mathbf{C} \to \mathbf{C}'$ on the right).

We have the following law of commutation between coercion and composition:



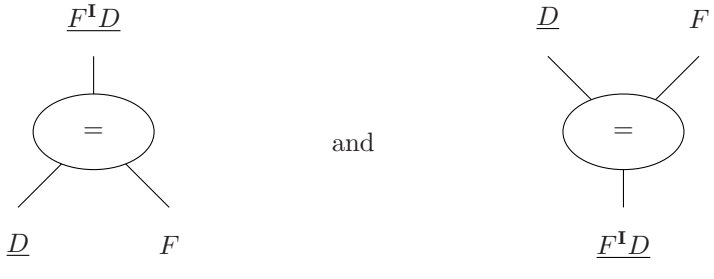
(1)

As an illustration, given a functor $F : \mathbf{C} \to \mathbf{C}'$, we show how to describe the action of the functor $F^{\mathbf{I}} : \mathbf{C}^{\mathbf{I}} \to \mathbf{C}'^{\mathbf{I}}$ on morphisms:



(2)

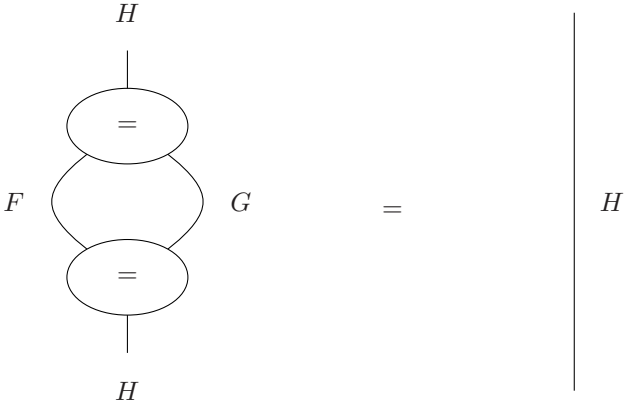
Notice the introduction of explicit equality nodes on the right hand side, which in fact describe the action of $F^{\mathbf{I}}$ on objects:



One can give graphical proofs of facts and results such as: if $F \dashv G$ (i.e.. F is left adjoint to G), then $F^{\mathbf{I}} \dashv G^{\mathbf{I}}$, or: right adjoints preserve limits.

3 Explicit Equalities

In the previous section, we have introduced explicit equality nodes, that allowed us to give the same interface to both sides of the equation describing the behaviour of $F^{\mathbf{I}}$ (respecting the interfaces is a key matter in 2-dimensional proofs). In this (final) section, we state a "coherence" result for string diagrams written only using equality nodes, which we call equality diagrams. We impose, besides associativity, the following three axioms:



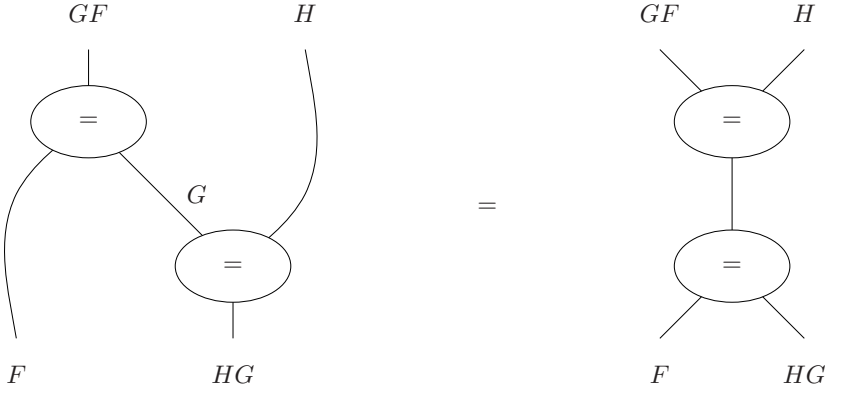
(3)

We do not require the converse, i.e.

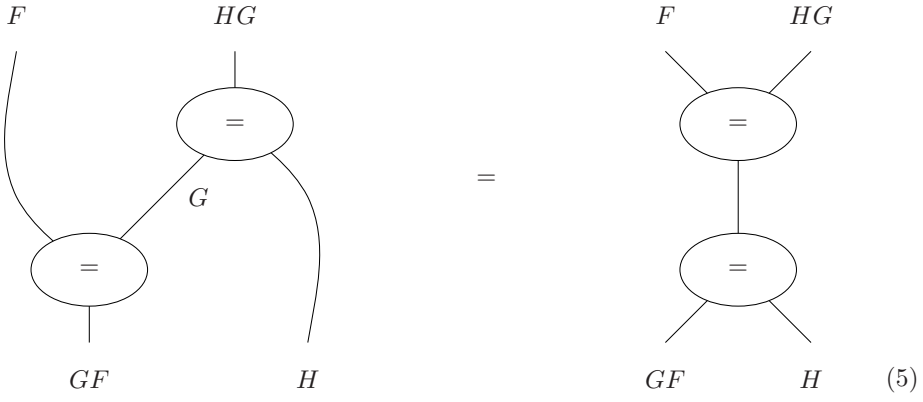
$$(=: H \to GF) \circ (=: GF \to H) = (id : G \to G) \cdot (id : F \to F)$$

for two reasons:

1. The most general type for the left hand side is $(=: H \to G_1F_1) \circ (=: G_2F_2 \to H)$, with no other requirement than $G_1F_1 = H = G_2F_2$. This contrasts with the situation above, where the plugging of $(=: H \to G_1F_1)$ *above* $(=: G_2F_2 \to H)$ forces $F_1 = F_2$ and $G_1 = G_2$.
2. We can have the effect of this equation by inserting it in a context (plugging $(=: H \to GF)$ above and $(=: GF \to H)$ below).



(4)



(5)

These equations suffice to prove that all equality diagrams with the same interface (given by the wires coming in and the wires coming out of the diagram) are provably equal.

References

[1] Curien, P.-L.: Category theory: a programming language-oriented introduction (forthcoming)
[2] Kassel, C.: Quantum groups. Springer, Heidelberg (1995)
[3] Mac Lane, S.: Categories for the working mathematician. Springer, Heidelberg (1971)

Model Transformations in Decidability Proofs for Monadic Theories

Wolfgang Thomas

RWTH Aachen University, Lehrstuhl Informatik 7, 52056 Aachen, Germany
thomas@informatik.rwth-aachen.de

Abstract. We survey two basic techniques for showing that the monadic second-order theory of a structure is decidable. In the first approach, one deals with finite fragments of the theory (given for example by the restriction to formulas of a certain quantifier rank) and – depending on the fragment – reduces the model under consideration to a simpler one. In the second approach, one applies a global transformation of models while preserving decidability of the theory. We suggest a combination of these two methods.

1 Introduction

Half a century ago, the first papers appeared on decidability of monadic second-order theories using concepts of automata theory. In 1958, Büchi, Elgot, and independently Trakhtenbrot announced the first results on the "logic-automata connection", showing that the weak monadic second-order theory of the successor structure $(\mathbb{N}, +1)$ of the natural numbers is decidable. Results on the unrestricted monadic second-order theory (short: MSO-theory) were then established by Büchi [4] (decidability of S1S, the monadic theory of $\mathcal{N} = (\mathbb{N}, +1)$) and by Rabin [13] (decidability of S2S, the monadic theory of the infinite binary tree $\mathcal{T}_2 = (\{0,1\}^*, \mathrm{Succ}_0, \mathrm{Succ}_1)$). All these results were shown by the transformation of formulas into finite automata (over infinite words, respectively over infinite trees).

The present note deals with the ongoing research in establishing larger and larger classes of infinite structures for which the MSO-theory is decidable (or in other words: the model-checking problem with respect to MSO-logic is decidable). We recall two methods that have been introduced for this purpose. The first is a transformation of the structure $\mathcal{S}$ under consideration into a simpler structure using a pumping argument. This method involves a finite equivalence that allows to compress certain parts of $\mathcal{S}$ to smaller ones. The finite equivalence takes into account only a finite fragment of the MSO-theory of $\mathcal{S}$; thus the transformation has to be done separately for any such fragment. The second method shows decidability of the entire MSO-theory of a structure $\mathcal{S}$ in one step, in which $\mathcal{S}$ is obtained by a transformation of another structure whose MSO-theory is known to be decidable. The purpose of this paper is to describe both methods and to suggest that a combination of them may be useful.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 23–31, 2008.
© Springer-Verlag Berlin Heidelberg 2008

As a prerequisite we discuss possibilities to define equivalences that determine "finite fragments" of a theory. These equivalences come in two forms: referring to automata with a certain number of states, and referring to formulas of a certain quantifier rank. We recall these equivalences and their use in the next section.

The paper is a discussion of methods rather than an exposition of results, and thus adheres to an informal style and assumes knowledge of the basics on monadic theories (as found e.g. in [11]).

The MSO-theory of a structure $\mathcal{S}$ is denoted $\mathrm{MTh}(\mathcal{S})$. In this paper we focus on the case of labelled transition systems, i.e. vertex- and edge-labelled graphs. We use the format $G = (V, (E_a)_{a\in A}, (P_b)_{b\in B})$ with finite label alphabets A, B, where E_a is the set of edges labelled a and $P_b \subseteq V$ the set of vertices labelled b.

2 Equivalences

A natural approach for showing decidability of the (say monadic) theory of a structure is to settle the problem for any given finite fragment of the theory, and for this to apply a composition of submodels. A standard option is to restrict to sentences up to some given quantifier rank. Another approach refers to automata with a given number of states (when formulas are known to be equivalent to automata). An extreme case is that one only considers a single automaton (corresponding to a single formula). In each of these cases one derives a corresponding equivalence between structures, and we call the equivalence classes "types". One now tries to compose a model from "simple" parts that has the same type as the original model, and at the same time to compose its type from the types of the components.

This approach has been most successful over (labelled) linear orderings, but it can also be applied with more technical work over more complex structures like trees and certain graphs. For the theory S1S the method involves a composition of an ω-word α (identified here with a labelled ω-ordering) from finite segment orderings (words). It turns out possible to represent α as an "infinite sum" of summand models such that the types of all finite summands are the same (except for the first summand). This allows to deduce the corresponding type of α from the two constituent types (the initial, respectively the repeated type). A composition of this simple form is guaranteed by Ramsey's Infinity Lemma [14]. The types of the segments define a finite coloring of the set of pairs (i, j) (with (i, j) we associate the type of the segment $\alpha[i, j)$). By Ramsey's Lemma there is an infinite "homogeneous" set $H = \{h_0 < h_1 < \ldots\}$: All segments $\alpha[h_i, h_j)$ with $i < j$, and in particular all segments $\alpha[h_i, h_{i+1})$ have the same type. In addition to Ramsey's Lemma one needs also a summation result for the types: First, the type of a concatenation (sum) of two words is determined by (and computable from) their types; so type equivalence is a congruence with respect to concatenation. Second, the type of an infinite sum of words of the same type is determined by (and computable from) this type.

This composition occurs in two versions in the literature. In the first version, one refers to a given Büchi automaton $\mathcal{A}$ and defines theory-fragments via the

transition structure of $\mathcal{A}$: One declares two segments (i.e., finite words) u, v equivalent (written $u \sim_{\mathcal{A}} v$) if the following holds: $\mathcal{A}$ can proceed from state p to state q via u iff this is possible via v, and this is possible with a visit to a final state via u iff it is possible via v. It is easy to show that this equivalence relation is a congruence of finite index and that the $\sim_{\mathcal{A}}$-type of a finite word w determines whether $\mathcal{A}$ accepts the infinite word $www\ldots$.

In the world of logic, one cannot achieve the congruence property on the level of a single formula. One obtains it when passing to the level of formula sets classified by the measure of quantifier rank: Call u, v n-equivalent (short $u \equiv_n v$) if u and v satisfy the same sentences of quantifier rank $\leq n$. Then again we obtain an equivalence relation of finite index. The fact that $\equiv_n$ is a congruence is not as immediate as for $\sim_{\mathcal{A}}$ but can be established by the standard method of Ehrenfeucht-Fraïssé games. An analogous congruence can also be introduced in the domain of automata: Define $u \approx_n v$ if $u \sim_{\mathcal{A}} v$ holds for each automaton with $\leq n$ states. It is then clear that the sequences $\equiv_n$ and $\approx_n$ are compatible in the sense that they mutually refine each other and hence that their intersections coincide.

When monadic formulas can be transformed into automata, it is often convenient to work with the relations $\sim_{\mathcal{A}}$ or $\approx_n$. This connection to automata exists over words and trees. Over generalized domains, such as dense labelled orderings, it is hard and maybe unnatural to try to invent suitable "automata"; here the logical equivalence has the advantage to be applicable directly. This is a key aspect in the "composition method" as developed by Shelah [17].

3 Reduction to Periodic Structures

In a pioneering paper, Elgot and Rabin [10] studied structures $(\mathbb{N}, \mathrm{Succ}, P)$ with a designated set $P \subseteq \mathbb{N}$ and showed for certain P that $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is decidable. Note that there are examples of recursive predicates P such that $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is undecidable. (Consider a recursively enumerable nonrecursive set S with enumeration $s_0, s_1, \ldots$, and introduce P by its characteristic sequence $\chi_P := 10^{s_0}10^{s_1}1\ldots$. Then P is recursive, and we have $n \in S$ iff there is a number in P such that its $(n+1)$-st successor is the next P-number; thus S is reducible to $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$.) There are also predicates P where the decidability of $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is open. The most prominent example is the prime number predicate $\mathbb{P}$.

The examples P given in [10] such that $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is decidable are the predicate of the factorial numbers, the predicate of powers of k (for fixed k), and the predicate of k-th powers (for fixed k). Another predicate to which the method can be applied is the set $\{2 \uparrow n \mid n \geq 0\}$ of "hyperpowers of 2", inductively defined by $2 \uparrow 0 = 1$ and $2 \uparrow (n+1) = 2^{2 \uparrow n}$. Further examples are given in [6].

The starting point for the decidability proof is the transformation of an S1S-formula $\varphi(X)$ into a Büchi automaton $\mathcal{A}_\varphi$ that accepts a 0-1-sequence iff it is

the characteristic sequence χ_P of a predicate P satisfying $\varphi(X)$. This allows to restate the decision problem for $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ as follows:

$(*)$ *Decide for any Büchi automaton whether it accepts the fixed ω-word χ_P.*

As an example consider the predicate $P = \mathrm{Fac}$ of factorial numbers. Given a Büchi automaton $\mathcal{A}$ over the characteristic sequence χ_{Fac} of the factorial predicate, one can shorten (by a pumping argument) the segments of successive zeros between any two letters 1 in χ_{Fac} in such a way that (1) the distance between two successive letters 1 in the new sequence χ' is bounded, and (2) $\mathcal{A}$ accepts χ_{Fac} iff $\mathcal{A}$ accepts χ'. More precisely, one replaces each segment 10^m1 by the shortest segment $10^{m'}1$ such that $0^m \sim_{\mathcal{A}} 0^{m'}$. It turns out that regardless of the choice of $\mathcal{A}$ the resulting "compressed" sequence χ' is *ultimately periodic*. Since in this case the acceptance problem can be decided, the problem $(*)$ is decidable.

This compression is done for the equivalence $\sim_{\mathcal{A}}$ associated with a given automaton $\mathcal{A}$; similarly, one can also use the relation $\equiv_n$ or $\approx_n$ in place of $\sim_{\mathcal{A}}$ and thus capture all formulas of quantifier rank $\leq n$, respectively all automata with $\leq n$ states, in one step. In all three cases we deal with a "finite fragment" of the theory. The essence thus is the transformation of the given structure to a simpler one (namely, an ultimately periodic one) that is equivalent with respect to a finite fragment of the MSO-theory.

It is important to note that the transformation into a periodic model is *computable* in the parameter $\mathcal{A}$, respectively n. Even without insisting on this computability requirement, for each ω-word χ such a transformation into an ultimately periodic structure *exists* (given $\mathcal{A}$, respectively n) — again by applying Ramsey's Lemma.

Recently, it was shown that the Elgot-Rabin method can be "uniformized" in the following sense ([15]): One considers the logical equivalence $\equiv_n$ (or its automata theoretic analogue $\approx_n$) and observes that $\equiv_{n+1}$ is a refinement of $\equiv_n$. An iterative application of Ramsey's Lemma yields for any χ_P a "uniformly homogeneous" set $H_P = \{h_0 < h_1 < \ldots\}$ which supplies periodic decompositions for all values of n simultaneously: For each n, all segments $\chi_P[h_i, h_{i+1})$ with $i \geq n$ are $\equiv_n$-equivalent. As a consequence, the truth of a sentence of quantifier-depth n is determined by the $\equiv_n$-types of $\chi_P[0, h_n)$ and $\chi_P[h_n, h_{n+1})$; in fact we have $\chi_P \equiv_n \chi_P[0, h_n) + (\chi_P[h_n, h_{n+1}))^\omega$.

Again, this decomposition is possible for each χ_P. One can show that a uniformly homogeneous set H_P exists which is recursive in P'' (the second recursion theoretic jump of P). A *recursive* uniformly homogeneous set H_P exists iff $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is decidable. As an illustration consider a predicate P where the decidability of $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is unsettled, namely the prime number predicate $\mathbb{P}$. Let $H_{\mathbb{P}} = \{h_0 < h_1 < \ldots\}$ be a corresponding — currently unknown — uniformly homogeneous set. We may be interested in the truth of the sentence TPH (twin prime hypothesis) saying "there are infinitely many twin primes", i.e. pairs $(m, m+2)$ with $m, m+2 \in \mathbb{P}$. The truth of TPH is open. Since TPH can be written as a monadic sentence of quantifier-depth 5, it suffices to inspect the segment $\chi_{\mathbb{P}}[h_5, h_6)$ of $\chi_{\mathbb{P}}$ for an occurrence of twin primes, to check whether TPH

holds. If TPH fails then the last twin prime pair would appear up to number h_5 and none in $\chi_{\mathbb{P}}[h_5, h_6)$.

A similar theory of model transformation can be developed for expansions of the binary tree by a unary predicate P, i.e. for models $(\{0,1\}^*, \mathrm{Succ}_0, \mathrm{Succ}_1, P)$. The desired "compression" of the structure is then a regular tree. The situation is much more complicated than it is over $\mathcal{N}$. First, the composition technique is technically more involved. Second, one does not know (as yet) a decomposition that corresponds to Ramsey's Lemma over ω-words. For recent work in this direction see [12].

The case of labelled tree structures is also interesting for its connection with Seese's conjecture of decidability of monadic theories. The conjecture can be stated as follows (see [2]): A structure has a decidable MSO-theory iff it can be MSO-interpreted in an expansion $\mathcal{T} = (\{0,1\}^*, \mathrm{Succ}_0, \mathrm{Succ}_1, \overline{P})$ of the binary tree by a finite tuple $\overline{P}$ of unary predicates such that $\mathrm{MTh}(\mathcal{T})$ is decidable.

4 Transformations Preserving Decidability

In his celebrated paper [13], Rabin starts with many applications of his main result, the decidability of $\mathrm{MTh}(\mathcal{T}_2)$, before entering the tedious proof. Starting from $\mathrm{MTh}(\mathcal{T}_2)$, several other theories are shown to be decidable by the method of interpretation. An MSO-interpretation of a relational structure $\mathcal{S}$ in $\mathcal{T}_2$ is an MSO-description of the universe and the relations of a copy of $\mathcal{S}$ in $\mathcal{T}_2$. Given such a description, the decidability of $\mathrm{MTh}(\mathcal{S})$ can be derived from the decidability of $\mathrm{MTh}(\mathcal{T}_2)$.

Another very powerful transformation that preserves the decidability of the MSO-theory is the "iteration" of a given structure in the form of a tree-like model. We use here a simple form of iteration which is appropriate for transition graphs as considered in this paper: the unfolding $U(G)$ of a graph G (from a definable vertex v_0). The structure $U(G)$ is a tree whose vertices are the finite paths $\pi = v_0 a_1 v_1 \dots a_m v_m$ in G where $(v_i, v_{i+1}) \in E_{a_{i+1}}$, and the pair $(\pi, (\pi a_{m+1} v_{m+1}))$ belongs then to the edge relation $E_{a_{m+1}}$ of $U(G)$. A fundamental result of Muchnik (announced in [16]) says that $\mathrm{MTh}(U(G))$ is decidable if $\mathrm{MTh}(G)$ is. Proofs are given in [7,9,18] and the expository paper [1].

Caucal observed in [5] that a large class of infinite graphs arises if MSO-interpretation and unfolding are applied in alternation. The *Caucal hierarchy* is a sequence $\mathcal{C}_0, \mathcal{C}_1, \dots$ of classes of graphs where

- $\mathcal{C}_0$ consists of the finite graphs,
- $\mathcal{C}_{n+1}$ consists of the graphs obtained from the graphs in $\mathcal{C}_n$ by an unfolding and a subsequent MSO-interpretation.

The original definition referred to a different transformation (inverse rational mappings rather than MSO-interpretations); for the equivalence between the two see [8]. The hierarchy is strict; a method to separate the levels is presented in [3].

Let $\mathcal{C} = \bigcup_i \mathcal{C}_i$. Each structure in $\mathcal{C}$ has a decidable MSO-theory. The class $\mathcal{C}$ contains an abundance of structures, and the extension of the higher levels is not

well understood. Many interesting examples occur on the first three levels. $\mathcal{C}_1$ is the class of "prefix-recognizable graphs" (encompassing the transition graphs of pushdown automata). Moreover, the expansions of $\mathcal{N}$ by the predicate of the factorial numbers, by the powers of (some fixed) k, and by the k-th powers (k fixed), respectively, are all in $\mathcal{C}_3$. This provides a more uniform proof of decidability than by the Elgot-Rabin method: It is no more necessary to provide a structure decomposition for each finite theory fragment; rather the membership of the structure in $\mathcal{C}$ suffices as the decidability proof for the full MSO-theory.

Only very few structures are known that have a decidable MSO-theory but are located outside $\mathcal{C}$. An example noted in the literature (see [8,2]) is the structure $(\mathbb{N}, \mathrm{Succ}, P)$ where P is the set of 2-hyperpowers $2 \uparrow n$. We shall generate this structure in an extension of the Caucal hierarchy.

5 Limit Models

By an iterated application of interpretations and unfoldings one can generate finite trees t_i $(i = 0, 1, 2, \ldots)$ where t_i has height $2 \uparrow i$ and $2 \uparrow (i+1)$ leaves. For $i = 0$ we take the binary tree consisting of the root and two sons. In order to construct t_{i+1}, consider t_i with $2 \uparrow i$ leaves. Along the frontier we introduce two identical edge relations S_1, S_2 (and as universe we take their common domain): For both S_1 and S_2 start from the rightmost leaf, proceed leaf by leaf towards the left to the leftmost leaf (which yields $2 \uparrow (i-1)$ edges), and continue with one more step to the parent of the leftmost leaf. Clearly S_1, S_2 are MSO-definable in t_i. The unfolding of this successor structure from its root, which is the rightmost leaf of t_i, gives the desired tree t_{i+1} of height $2 \uparrow (i+1)$ with $2 \uparrow (i+2)$ leaves.

Let $\prod_i t_i$ be the "limit model" of the t_i $(i \geq 0)$ where the rightmost leaf of t_i coincides with the root of t_{i+1}. An interpretation in the limit model will generate a structure $(\mathbb{N}, \mathrm{Succ}, P)$ which does not belong to the Caucal hierarchy: As copy of $(\mathbb{N}, \mathrm{Succ})$ one uses the infinite sequence of leaves, and one declares as P-elements the "first leaves" of the t_i. The difference between successive P-elements is then $(2 \uparrow (i+1)) - 1$ (for $i = 0, 1, \ldots$). By a refinement of the construction one can also generate a copy of $(\mathbb{N}, \mathrm{Succ}, H)$ where H is the set of hyperpowers of 2. For this, we use a structure $\prod_i t'_i$ where each t'_i contributes only $2 \uparrow (i+1) - 2 \uparrow i$ rather than $2 \uparrow (i+1) - 1$ leaves. Technically we work with the t_i as above but expanded by a singleton predicate Q that marks the $((2 \uparrow i)+1)$-st of its $2 \uparrow (i+1)$ leaves. To construct the t'_i inductively, one starts with t_1 and fixes Q_{t_1} as the set containing the second leaf. For the step from t'_i to t'_{i+1} we have to proceed (in the definition of Q) from a number of the form $2^k - k$ to $2^{2^k} - 2^k$; this is possible (using a little technical work) by observing that $2^{2^k} - 2^k = 2^k(2^{2^k - k} - 1)$.

The model $\prod_i t_i$ (and similarly $\prod_i t'_i$) is generated by an infinite sequence of interpretations and unfoldings. However, each of the interpretations is based on the same formulas defining the universe and the relations, and for each unfolding one uses the same formula for defining the root vertex. So we speak of an *interpretation-unfolding scheme* that generates $\prod_i t_i$ ($\prod_i t'_i$, respectively). In our

example we referred to a single definable vertex for the "next unfolding"; in this case we speak of a *linear* interpretation-unfolding scheme.

The limit models $\prod_i t_i$ and $\prod_i t'_i$ have decidable MSO-theories. To show this, we have (presently) to resort to the "non-uniform" method of model reduction (see Section 3). This requires to invoke the equivalences $\equiv_n$ and the associated n-types. We observe that the n-type of t_{i+1} is computable from the n-type of t_i (similarly for t'_i and t'_{i+1}). Then — by finiteness of the set of n-types — the generated sequence of n-types is ultimately periodic, which allows to compute the n-type of the limit model.

We do not know whether by a linear interpretation-unfolding scheme (and an extra interpretation in the limit model) one can generate a structure $(\mathbb{N}, \mathrm{Succ}, P)$ whose monadic theory is undecidable. This is connected with the old problem to find (in any way) a non-artificial — i.e. number theoretically meaningful — recursive predicate P such that $\mathrm{MTh}(\mathbb{N}, \mathrm{Succ}, P)$ is undecidable.

It seems interesting to analyze also non-linear schemes. In this case, one would allow to expand a given model at several vertices, which leads to a tree-like construction. The formula that defines the set of vertices where the unfolding takes place is then satisfied by several vertices. We show that such a scheme can lead to a structure with an undecidable MSO-theory. Consider a Turing machine M (say with set Q of states and tape alphabet Σ) that accepts a non-recursive language. As initial model we use the tree $\mathcal{S}_0$ of all initial configurations $q_0 a_1 \dots a_n$ with initial state q_0 and input word $a_1 \dots a_n$ (coded by paths with successive edge labels $q_0, a_1, \dots, a_n, \$$ where $\$$ is an endmarker). This is an infinite tree with a decidable MSO-theory. By an interpretation-unfolding scheme we generate, level by level, a tree model $\mathcal{S}_M$ in which all M-computations can be traced as paths. A word w will be accepted by M iff a sentence φ_w is true in $\mathcal{S}_M$ that expresses the following: From the vertex after the initial path $q_0 w\$$ there is a path to a configuration with an accepting state of M.

We describe the interpretation-unfolding scheme. Consider a tree $\mathcal{S}_k$ that is generated at level k of the construction. We first treat the case $k > 0$ and later $k = 0$. Let r be the root of $\mathcal{S}_k$. The next unfolding will take place at any vertex v which is the source of an $\$$-labelled edge and such that between r and v there is no other $\$$-labelled edge. Vertex v is the end of a path from r labelled by an M-configuration, say $w_1 bqa w_2$. We define a structure $\mathcal{S}_v$ as follows: The universe is given by the path from r to v, the sequence of edge labels along the path gives the next M-configuration after $w_1 bqa w_2$, and there is a new $\$$-labelled edge from v back to r. So we obtain $\mathcal{S}_v$ from $\mathcal{S}_k$ by adding the $\$$-edge and by changing the bqa-labelled path segment to a new one according to the table of M. (In the case the length of the configuration increases by one, the original $\$$-labelled edge (v, v') gets a letter from Σ, and the $\$$-labelled back-edge to r starts at v'.) It is clear that for each pair $(q, a) \in Q \times \Sigma$ (which fixes an M-transition) the respective structure $\mathcal{S}_v$ can be defined; a disjunction over all (q, a) gives the desired interpretation. The unfolding of $\mathcal{S}_v$ at v will produce an infinite sequence of finite paths labelled with the new configuration. In the subsequent step, only the first such path will stay (by the definition of the new v and the new $\mathcal{S}_v$).

In the initial step (where $k = 0$) the initial configuration is simply copied; this ensures that the first copy stays unmodified by the construction.

Clearly one can express in the limit model $\mathcal{S}_M$ by an MSO-sentence φ_w that for input w there is an accepting computation of M. So $\mathrm{MTh}(\mathcal{S}_M)$ is undecidable.

Interpretation-unfolding schemes are thus a powerful tool to generate models (and in general too powerful to obtain only structures with a decidable monadic theory). By a linear interpretation scheme it was possible to synthesize a structure $(\mathbb{N}, \mathrm{Succ}, P)$ (namely, where P is the set of hyperpowers of 2) which was previously just given a priori. The decidability of its monadic theory was shown using the "non-uniform" method of model reduction. An open issue is to find easily verified conditions that ensure decidability of the MSO-theory of a limit model. Also one can study schemes that involve transfinite stages of construction.

6 Conclusion

We surveyed two techniques for proving that the MSO-theory of an infinite labelled graph is decidable: the "non-uniform" method of model reduction à la Elgot and Rabin and two "uniform" types of model transformation, namely MSO-interpretations and unfoldings. We proposed to study models that are generated by an infinite number of steps involving the latter two operations. It was illustrated that in this context a combination of the uniform and the non-uniform approach gives a further small step in building up more infinite graphs that have a decidable MSO-theory.

We have not touched the rich landscape of recent studies on other types of interpetations and of model composition. A good survey on the state-of-the-art is given in [2]. On the side of logics, one should note that for applications in infinite-state verification weaker logics than MSO-logic are of interest, for which the class of structures with a decidable model-checking problem can then be expanded.

Acknowledgment

I thank Christof Löding and Basile Morcrette for helpful discussions.

References

1. Berwanger, D., Blumensath, A.: The monadic theory of tree-like structures. In: [11], pp. 285–302
2. Blumensath, A., Colcombet, T., Löding, C.: Logical theories and compatible operations. In: Flum, J., et al. (eds.) Logic and Automata, pp. 73–106. Amsterdam Univ. Press, Amsterdam (2007)
3. Blumensath, A.: On the structure of graphs in the Caucal hierarchy. Theor. Comput. Sci. 400, 19–45 (2008)
4. Büchi, J.R.: On a decision method in restricted second-order arithmetic. In: Nagel, E., et al. (eds.) Logic, Methodology, and Philosophy of Science: Proceedings of the 1960 International Congress, pp. 1–11. Stanford Univ. Press (1962)

5. Caucal, D.: On infinite graphs having a decidable monadic theory. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 165–176. Springer, Heidelberg (2002)
6. Carton, O., Thomas, W.: The monadic theory of morphic infinite words and generalizations. In: Nielsen, M., Rovan, B. (eds.) MFCS 2000. LNCS, vol. 1893, pp. 275–284. Springer, Heidelberg (2000)
7. Courcelle, B.: The monadic second-order logic of graphs IX: machines and their behaviours. Theor. Comput. Sci. 151, 125–162 (1995)
8. Carayol, A., Wöhrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)
9. Courcelle, B., Walukiewicz, I.: Monadic second-order logic, graph coverings and unfoldings of transition systems. Ann. Pure Appl. Logic 92, 35–62 (1998)
10. Elgot, C.C., Rabin, M.O.: Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. J. Symb. Logic 31, 169–181 (1966)
11. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002)
12. Montanari, A., Puppis, G.: A contraction method to decide MSO theories of deterministic trees. In: Proc. 22nd IEEE Symposium on Logic in Computer Science (LICS), pp. 141–150
13. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. 141, 1–35 (1969)
14. Ramsey, F.P.: On a problem of formal logic. Proc. London Math. Soc. s2-30, 264–286 (1930)
15. 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)
16. Semenov, A.: Decidability of monadic theories. In: Chytil, M.P., Koubek, V. (eds.) MFCS 1984. LNCS, vol. 176, pp. 162–175. Springer, Heidelberg (1984)
17. Shelah, S.: The monadic theory of order. Ann. Math. 102, 379–419 (1975)
18. Walukiewicz, I.: Monadic second-order logic on tree-like structures. Theor. Comput. Sci. 275, 311–346 (2002)

Molecules as Automata

Luca Cardelli

Microsoft Research

Molecular biology investigates the structure and function of biochemical systems starting from their basic building blocks: macromolecules. A macromolecule is a large, complex molecule (a protein or a nucleic acid) that usually has inner mutable state and external activity. Informal explanations of biochemical events trace individual macromolecules through their state changes and their interaction histories: a macromolecule is endowed with an identity that is retained through its transformations, even through changes in molecular energy and mass. A macromolecule, therefore, is qualitatively different from the small molecules of inorganic chemistry. Such molecules are stateless: in the standard notation for chemical reactions they are seemingly created and destroyed, and their atomic structure is used mainly for the bookkeeping required by the conservation of mass.

Attributing identity and state transitions to molecules provides more than just a different way of looking at a chemical event: it solves a fundamental difficulty with chemical-style descriptions. Each macromolecule can have a huge number of internal states, exponentially with respect to its size, and can join with other macromolecules to from even larger state configurations, corresponding to the product of their states. If each molecular state is to be represented as a stateless chemical species, transformed by chemical reactions, then we have a huge explosion in the number of species and reactions with respect to the number of different macromolecules that actually, physically, exist. Moreover, macromolecules can join to each other indefinitely, resulting in situations corresponding to infinite sets of chemical reactions among infinite sets of different chemical species. In contrast, the description of a biochemical system at the level of macromolecular states and transitions remains finite: the unbounded complexity of the system is implicit in the potential molecular interactions, but does not have to be written down explicitly. Molecular biology textbooks widely adopt this finite description style, at least for the purpose of illustration.

At the core, we can therefore regard a macromolecule as some kind of automaton, characterized by a set of internal states and a set of discrete transitions between states driven by external interactions. We can thus try to handle molecular automata by some branch of automata theory and its outgrowths: cellular automata, Petri nets, and process algebra. The peculiarities of biochemistry, however, are such that until recently one could not easily pick a suitable piece of automata theory off the shelf. Many sophisticated approaches have now been developed, and we are particularly fond of stochastic process algebra. In this talk, however, we do our outmost to remain within the bounds of a much simpler theory. We go back, in a sense, to a time before cellular automata, Petri nets and process algebra, which all arose from the basic intuition that automata should interact with each other. Our main criterion is that, as in finite-state automata, we should be able to easily and separately draw the individual automata, both as a visual aid to design and analysis, and to emulate the illustration-based approach found in molecular biology textbooks.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, p. 32, 2008.
© Springer-Verlag Berlin Heidelberg 2008

An Infinite Automaton Characterization of Double Exponential Time*

Salvatore La Torre[1], P. Madhusudan[2], and Gennaro Parlato[1,2]

[1] Università di Salerno, Italy
[2] University of Illinois, Urbana-Champaign, USA

Abstract. Infinite-state automata are a new invention: they are automata that have an infinite number of states represented by words, transitions defined using rewriting, and with sets of initial and final states. Infinite-state automata have gained recent interest due to a remarkable result by Morvan and Stirling, which shows that automata with transitions defined using rational rewriting precisely capture context-sensitive (NLINSPACE) languages. In this paper, we show that infinite automata defined using a form of multi-stack rewriting precisely defines double exponential time (more precisely, 2ETIME, the class of problems solvable in $2^{2^{O(n)}}$ time). The salient aspect of this characterization is that the automata have no ostensible limits on time nor space, and neither direction of containment with respect to 2ETIME is obvious. In this sense, the result captures the complexity class qualitatively, by restricting the power of rewriting.

1 Introduction

The theory of infinite-state automata is a new area of research (see [21] for a recent survey). Infinite-state automata (not to be confused with *finite state automata on infinite words*) are automata with *infinitely* many states that can read *finite* words and accept or reject them, in much the same way as finite-state automata would. In order to represent infinite-state automata using finite means, the states, the transitions, and the initial and final state sets are represented *symbolically*.

The infinite-state automata we study in this paper are defined by using *words* to represent the states of the automaton. Let us fix a finite alphabet Σ as the input alphabet for the infinite-state automata. The set of states of an infinite-state automaton over Σ are words over a finite alphabet Π (which does not need to be related to Σ in any way). The initial and final sets of states of this automaton are defined using word-languages over Π accepted by finitely presented devices (e.g. finite-state automata over Π). Transitions between states are defined using *rewriting* rules that rewrite words to other words: for each $a \in \Sigma$, we have a rewrite rule that rewrites words over Π. A state $u \in \Pi^*$ leads to state $u' \in \Pi^*$ on $a \in \Sigma$ iff the rewrite rule for a can rewrite u to u'. There is a variety of choices for the power of rewriting, but in any case the rewriting rules are presented finitely (e.g. using finite transducers). The language accepted by an infinite-state automaton is defined in the natural way: a word $w \in \Sigma^*$ is accepted if there is a path from *some* initial state to *some* final state tracing w in the automaton.

* The first and third authors were partially supported by the MIUR grants ex-60% 2006 and 2007 Università degli Studi di Salerno.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 33–48, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Infinite-state automata are naturally motivated in *formal verification*, where, intuitively, a *state* of the model can be represented using a word, and the system's evolution can be described using rewriting rules. Classic examples include boolean abstraction of recursive programs [2] (where a system is described using a state and a stack encoded into words, and the rewriting corresponds to *prefix rewriting*) and *regular model-checking*, where parameterized finite-state systems are represented with finite words and transitions defined using *synchronous rational rewriting* [3].

Infinite-state automata are radically different computation models than Turing machines especially when computational complexity issues are at hand. The notion that rewriting words (or terms) can be a basis for defining computability goes back to the works of Axel Thue [22] (Thue systems) and Emil Post [17] (Post's tag systems). Formal languages defined using grammars (the Chomsky hierarchy) are also in the spirit of rewriting, with semi-Thue systems corresponding to unrestricted grammars and hence Turing machines. While Turing machines can be viewed as rewrite systems (rewriting one configuration to another), the study of computational complexity is often based on time and space constraints on the Turing machine model, and natural counterparts to complexity classes in terms of rewrite systems don't currently exist.

Given a word $w \in \Sigma^*$, note that infinite automata have possibly an *infinite* number of paths on w. Hence, deciding whether w is accepted by the infinite-state automaton is in no way trivial. However, if rewriting rules can be simulated by Turing machines (which will usually be the case), the language accepted by the infinite-state automaton is recursively enumerable.

Recently, Morvan and Stirling showed the following remarkable result: infinite state automata where states are finite words, initial and final sets are defined using regular languages, and transitions are defined using *rational relations*, accept precisely the class of *context-sensitive languages* (nondeterministic linear-space languages) ([15]; see also [5]). Rational relations are relations $R \subseteq \Pi^* \times \Pi^*$ that can be effected by finite-state automata: $(u, u') \in R$ iff the automaton can read u on an input tape and write u' on the output tape, where the two tape heads are only allowed to move right (but can move independent of each other).

Note that the only constraint placed in the above result is the power of rewriting (rational relations) and there is no ostensible limit on space or time. In other words, the constraint on rewriting is a *qualitative constraint* with no apparent restriction on complexity. Indeed, even establishing the upper bound (the easier direction), namely that these automata define languages that are accepted by linear-bounded Turing machines is non-trivial. A naive *simulation* of the infinite-state automaton will not work as the words that represent states on the run can be unboundedly large even for a fixed word w. Notice that we do *not* allow ϵ-transitions in infinite automata, as allowing that would make infinite automata with even regular rewriting accept the class of all recursively enumerable languages.

Our main contribution here is an infinite automaton characterization for the class 2ETIME, the class of languages accepted by Turing machines in $exp(exp(O(n)))$[1] time, using a qualitative constraint on rewriting, which is a restricted form of multi-stack pushdown rewriting.

[1] $exp(x)$ denotes 2^x.

A simple generalization of regular rewriting is *pushdown rewriting*, where we allow the rewriting automata the use of a work stack which can be used for intermediate storage when rewriting a word to another. For example the relation $\{(w, ww^r)|w \in \Pi^*\}$ (where w^r denotes the reverse of w) is not regular but can be effected by a pushdown rewrite system. However, defining infinite automata with the power of pushdown rewriting quickly leads to *undecidability* of the membership problem, and these automata can accept non-recursive languages.

We hence place a restriction on pushdown rewriting. We demand that the rewriting device takes its input in a read-only tape and writes it to a write-only tape, and has access to some stacks, but it can switch only a *bounded* number of times the source from which it is reading symbols (i.e., the input tape and the stacks). In other words, the pushdown rewriting can be split into k phases, where in each phase, it either reads from the input tape and does not pop any stack, or pops from just one stack but doesn't read from the input tape. This restriction puts the problem of checking membership just within the boundary of decidability, and results in an automaton model that defines a class of recursive languages.

We show that infinite automata restricted to bounded-phase pushdown rewriting precisely defines the class 2ETIME.

The upper bound, showing the membership problem for any such infinite automaton is decidable in 2ETIME, is established by reducing it to the *emptiness* problem for *finite-phased multi-stack visibly pushdown automata*, which we have shown recently to be decidable [12]. Note that (non-deterministic) Turing machines that directly and naively simulate the infinite automaton could take unbounded space and time. Visibly pushdown automata [1] are pushdown automata where the input symbols determine the operation on the stack, and multi-stack visibly pushdown automata generalize them to multiple stacks [12]. Intuitively, the *accepting runs* that are followed by an n-stack pushdown rewriting system when it transforms a word u to u' can be seen as a multi-stack ($(n+2)$-stack) *visibly* pushdown automaton. Hence membership of w for an infinite automaton reduces to checking *emptiness* of the language of accepting runs over w. Moreover, if each rewriting step in the infinite automaton is bounded phase, then the number of phases in the multi-stack automata is $O(|w|)$. In [12], we show that the k-phase reachability for multi-stack automata is solvable in $exp(exp(O(poly(k))))$ time using (monadic second-order) logic interpretations on finite trees. We sharpen the above result in this paper to obtain $exp(exp(O(k)))$ time decision procedure for emptiness by implementing two crucial subprocedures that correspond to capturing the linear ordering and the successor relation from the tree directly using nondeterministic tree automata and two-way alternating tree automata, respectively.

Turning to the lower bound, we establish that all 2ETIME languages are accepted by infinite automata defined using bounded-phase pushdown rewriting. We show that for every alternating ESPACE Turing machine (i.e. working in space $2^{O(n)}$, which is equivalent to 2ETIME [9]), there is an infinite automaton with bounded-phase rewriting accepting the same language.

Related Work: A recent result by Rispal [18] shows that infinite automata defined using *synchronous rational relations*, which are strictly less powerful than rational relations, also define exactly the class of context-sensitive languages (see also [5]). Meyer [14]

has characterized the class ETIME (the class of languages accepted by Turing machines in time $exp(O(n))$) with infinite automata defined via automatic term transducers.

Bounded-phase visibly multi-stack pushdown automata have been introduced and studied by us in [12]. These automata capture a robust class of context-sensitive languages that is closed under all the boolean operations and has decidable decision problems. Also, they turned out to be useful to show decidability results for concurrent systems communicating via unbounded FIFO queues [13].

Capturing complexity classes using logics on graphs in descriptive complexity theory [10], which was spurred by Fagin's seminal result capturing NP using $\exists SO$, also has the feature that the characterizations capture complexity classes without any apparent restriction of time or space.

Finally, there's a multitude of work on characterizing the infinite graphs that correspond to restricted classes of machines (pushdown systems [16], prefix-recognizable graphs [7], higher-order pushdown automata [4], linear-bounded automata [6], and the entire Chomsky hierarchy [8]).

2 Multi-stack Pushdown Rewriting

A multi-stack pushdown transducer is a transducer from words to words that has access to one or more pushdown stacks.

For any set X, let X_ϵ denote $X \cup \{\epsilon\}$, and let X^* denote the set of finite words over X. Also, for any $i, j \in \mathbb{N}$, let $[i, j]$ denote the set $\{i, i+1, \ldots, j\}$.

Fix finite alphabets Π and Γ. An n-stack pushdown transducer over Π is a tuple $\mathcal{T} = (Q, q_0, \delta, \Gamma, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Γ is the stack alphabet, and $F \subseteq Q$ is the set of final states. The transition relation is $\delta \subseteq (Q \times Q \times \Pi_\epsilon \times \Pi_\epsilon \times [0, n] \times \Gamma_\epsilon \times \Gamma_\epsilon)$, with the restriction that if $(q, q', a, b, i, \gamma, \gamma') \in \delta$, then $\gamma = \gamma' = \epsilon$ iff $i = 0$.

A transition of the form $(q, q', a, b, i, \gamma, \gamma')$, with $a, b \in \Pi_\epsilon$ and $\gamma, \gamma' \in \Gamma_\epsilon$, intuitively means that the pushdown transducer, when in state q with γ on the top of its i'th stack (provided $i > 0$) can read a from the input tape, write b onto the output tape, replace γ with γ' onto the i'th stack, and transition to state q'. When $i = 0$, $\gamma = \gamma' = \epsilon$ and hence no stack is touched when changing state though input symbols can be read.

Note that $\gamma = \epsilon$ and $\gamma' \neq \epsilon$ corresponds to a push transition, $\gamma \neq \epsilon$ and $\gamma = \epsilon$ corresponds to a pop transition. Without loss of generality, let us assume that in every transition, $\gamma = \epsilon$ or $\gamma' = \epsilon$ holds, and if $a \neq \epsilon$ then $\gamma = \epsilon$ (i.e., when reading a symbol from the input tape, none of the stacks can be popped).

A configuration of the pushdown transducer $\mathcal{T}$ is a tuple $(w_1 q w_2, \{s_i\}_{i=1}^n, w')$ where $w_1, w_2, w' \in \Pi^*$, $q \in Q$ and $s_i \in \Gamma^*$ for each $i \in [1, n]$. Such a configuration means that the input head is positioned just after w_1 on the input tape that has $w_1 w_2$ written on it, q is the current state, s_i is the current content of the i'th stack, and w' is the output written thus far onto the output tape (with the head positioned at the end of w').

Transitions between configurations are defined by moves in δ as follows:

$$(w_1 q a w_2, \{s_i\}_{i=1}^n, w') \xrightarrow{(q,q',a,b,j,\gamma,\gamma')} (w_1 a q' w_2, \{s'_i\}_{i=1}^n, w' b),$$

where $(q, q', a, b, j, \gamma, \gamma') \in \delta$, if $j \neq 0$ then $s_j = \widehat{s}\gamma$ and $s'_j = \widehat{s}\gamma'$, and $s'_i = s_i$ for each $i \neq j$.

Let us define the configuration graph of the transducer $\mathcal{T}$ as the graph whose vertices are the configurations and whose edges are the transitions between configurations as defined above. A multi-stack pushdown transducer $\mathcal{T}$ rewrites w to w', if there is a path in the configuration graph from configuration $(q_0 w, \{\epsilon\}_{i=1}^n, \epsilon)$ to configuration $(wq_f, \{s_i\}_{i=1}^n, w')$, with $q_f \in F$.

Pushdown rewriting is powerful, and the problem of deciding whether w can be rewritten to w' even in two steps by even a one-stack transducer is undecidable (see Appendix for a proof):

Lemma 1. *The problem of checking if a word w can be rewritten to a word w' in two steps by a 1-stack pushdown transducer is undecidable.*

We want a tractable notion of transducers in order to define infinite automata that accept recursive languages. We hence introduce a bounded version of pushdown transducers.

We say that a pushdown transducer is *k-phase* ($k \in \mathbb{N}$), if, when transforming any w_1 to w_2, it switches at most k times between reading the input and popping either one of the stacks, and between popping different stacks. More formally, a transition of the form $(q, q', a, b, i, \gamma, \gamma')$ is a *not-pop transition* if it's not a transition that pops any stack, i.e. if $\gamma' \neq \epsilon$ or $i = 0$. Let $NotPop$ denote the set of not-pop transitions. Let Pop_i ($i \neq 0$) denote the set of all transitions except those that read from the input tape or pop from a stack j different from i, i.e. Pop_i is the set of transitions of the form $(q, q', a, b, j, \gamma, \gamma')$ where $a = \epsilon$ and if $j \neq i$ then $\gamma = \epsilon$.

A k-phase transducer is one which on any run $c_0 \xrightarrow{m_1} c_1 \xrightarrow{m_2} c_2 \dots \xrightarrow{m_i} c_i$ the sequence $m_1 m_2 \dots m_i$ can be split as $w_1 w_2 \dots w_k$ where for every $h \in [1, k]$, $w_h \in NotPop^* \cup \bigcup_{i=1}^n (Pop_i^*)$.

A bounded-phase pushdown transducer is a pushdown transducer which is k-phase for some $k \in \mathbb{N}$.

Infinite Automata Defined by Multi-stack Pushdown Transducers

We define now *infinite-state* automata over an alphabet Σ. The states in this automaton will correspond to words over an alphabet Π, the set of states one can transition to from a state on a letter d in Σ will be defined using a multi-stack pushdown transducer corresponding to d, and initial and final state sets will be identified using regular sets of words over Π.

Fix a finite alphabet Σ. An infinite-state pushdown transducer automaton (PTA) over Σ is a tuple $\mathcal{A} = (\Pi, \{\mathcal{T}_d\}_{d \in \Sigma}, Init, Final)$, where Π is a finite alphabet, for each $d \in \Sigma$, $\mathcal{T}_d$ is a pushdown transducer over Π, and $Init$ and $Final$ are finite-state automata (NFAs) over Π.

A PTA $\mathcal{A} = (\Pi, \{\mathcal{T}_d\}_{d \in \Sigma}, Init, Final)$ defines an infinite graph $G = (V, E)$ where the set of vertices V is set of words over Π and E is the set of all edges $v \xrightarrow{d} v'$ such that the pushdown transducer $\mathcal{T}_d$ can rewrite v to v'.

A bounded-phase PTA (BPTA) is a PTA in which every transducer is of bounded-phase.

A run of the PTA $\mathcal{A}$ on a word over $d_1 \dots d_n \in \Sigma^*$ is a sequence $v_0, v_1, \dots v_n$, where v_0 is accepted by the automaton $Init$, and for each $i \in [1,n]$, $v_{i-1} \xrightarrow{d_i} v_i$ is in G. Such a run is accepting if the final vertex is accepted by $Final$, i.e. $v_n \in L(Final)$.

A word w is accepted by a PTA $\mathcal{A}$ if there is some accepting run of $\mathcal{A}$ on w. The language accepted by $\mathcal{A}$, denoted $\mathcal{L}(A)$ is the set of all words it accepts.

In the rest of the paper we often write $exp(x)$ for 2^x. Let 2ETIME(Σ) denote the class of all languages over Σ that can be accepted by Turing machines working in time $exp(exp(O(n)))$.

We can now state our main theorem:

Theorem 1
*A language L over Σ is accepted by a bounded-phase PTA iff $L \in$*2ETIME(Σ).

3 The Upper Bound

In this section, we show that bounded-phase pushdown transducer automata define a class of languages contained in 2ETIME.

Let us fix a BPTA $\mathcal{A} = (\Pi, \{\mathcal{T}_d\}_{d\in\Sigma}, Init, Final)$. The proof that $L(\mathcal{A})$ is contained in 2ETIME is structured as follows:

(a) First, we show that the problem of checking if a word w is accepted by a BPTA can be reduced to the *emptiness* problem for k-phase multi-stack visibly pushdown automata (defined below) of state-space $O(|w|)$ and such that $k = O(|w|)$.

(b) In [12], we have shown that the emptiness problem for k-phase multi-stack pushdown automata with state-space Q can be decided in $exp(|Q| \cdot exp(O(poly(k))))$ time. Applying this would give a 2EXPTIME procedure and not a 2ETIME procedure for our problem (2EXPTIME is the class of problems that can be solved by a Turing machine using $exp(exp(O(poly(n))))$ time). Consequently, we sharpen the result above, and show that emptiness can be indeed decided in time $exp(|Q| \cdot exp(O(k)))$, which establishes our theorem.

Bounded Phase Multi-stack Pushdown Automata

Multi-stack visibly pushdown automata (MVPA) are automata with a finite number of stacks, where the input letter determines which stack the automaton touches and whether it pushes or pops from that stack. We refer to actions that push onto a stack as calls and actions that pop a stack as returns.

An n-stack call-return alphabet is a tuple $\widetilde{\Sigma}_n = \langle \{(\Sigma_c^i, \Sigma_r^i)\}_{i\in[1,n]}, \Sigma_{int} \rangle$ of pairwise disjoint finite alphabets. For any $i \in [1,n]$, Σ_c^i is a finite set of *calls of the stack i*, Σ_r^i is a finite set of *returns of stack i*, and Σ_{int} is a finite set of *internal actions*. Let $\widetilde{\Sigma}$ denote the union of all the alphabets in $\widetilde{\Sigma}_n$.

An n-stack visibly pushdown automaton $M = (Q, Q_I, \Gamma, \delta, Q_F)$ (where Q is a finite set of states, $Q_I \subseteq Q$ and $Q_F \subseteq Q$ are initial and final sets of states, Γ is the stack alphabet and δ is the transition relation) over such an alphabet can push on the i'th stack exactly one symbol when it reads a call of the i'th call alphabet, and pop exactly

one symbol from the i'th stack when it reads a return of the i'th return alphabet. Also, it cannot touch any stack when reading an internal letter. The semantics of MVPAs is defined in the obvious way, and we refer the reader to [12] for details.

A k-phase MVPA (k-MVPA) is intuitively an MVPA which works in (at most) k phases, where in each phase it can push onto *any* stack, but pop at most from one stack. Formally, given a word $w \in \widetilde{\Sigma}^*$, we denote with $Ret(w)$ the set of all returns in w. A word w is a *phase* if $Ret(w) \subseteq \Sigma_r^i$, for some $i \in [1, n]$, and we say that w is a *phase of stack* i. A word $w \in \widetilde{\Sigma}^+$, is a k-phase word if k is the minimal number such that w can be factorized as $w = w_1 w_2 \dots w_k$, where w_h is a phase for each $h \in [1, k]$. Let $Phases(\widetilde{\Sigma}_n, k)$ denote the set of all k-phase words over $\widetilde{\Sigma}_n$.

For any k, a *k-phase multi-stack visibly pushdown automaton* (k-MVPA) A over $\widetilde{\Sigma}_n$ is an MVPA M parameterized with a number k; the language accepted by A is $L(A) = L(M) \cap Phases(\widetilde{\Sigma}_n, k)$.

Reduction to k-MVPA Emptiness

Consider a BPTA $\mathcal{A} = (\Pi, \{\mathcal{T}_d\}_{d \in \Sigma}, Init, Final)$. Recall that given a word $w = d_1 \dots d_m \in \Sigma^*$, the automaton $\mathcal{A}$ accepts w iff there is a sequence of words $u_0, \dots, u_m$ such that $u_0 \in L(Init)$, $u_m \in L(Final)$, and for each $i \in [1, m]$, u_{i-1} can be rewritten to u_i by the transducer $\mathcal{T}_{d_i}$.

Suppose that the transducers of $\mathcal{A}$ have at most n stacks. We consider the $(n+2)$-stack call-return alphabet $\widetilde{\Sigma}_{n+2} = \langle \{(\Sigma_c^i, \Sigma_r^i)\}_{i \in [1,n+2]}, \{int\} \rangle$ where each $\Sigma_c^i = \{c_i\}$ and $\Sigma_r^i = \{r_i\}$. I.e., we have exactly one call and one return for each stack, and exactly one internal letter.

Assume that an $(n+2)$-stack MVPA starts with u_{i-1}^r on stack 1. Using stacks $2, \dots, n+1$ as the intermediate stacks, it can generate u_i on stack $n+2$ by simulating the transducer $\mathcal{T}_{d_i}$ (the word it reads is dictated by the actions performed on the stack). Then, it can replace stack 1's content with the reverse of stack $(n+2)$'s content to get u_i^r on the stack 1, and empty stacks $2, \dots, n+1$. Since the pushdown rewrite system is bounded phase, it follows that the above rewriting takes only a bounded number of phases. Simulating the rewrites for the entire word w (i.e. $u_0 \rightarrow u_1 \rightarrow \dots u_m$), and checking the initial words and final words belong to $Init$ and $Final$, respectively, takes at most $O(m)$ phases. Moreover, we can build this MVPA to have $O(m)$ states (for a fixed BPTA $\mathcal{A}$). We hence have:

Lemma 2. *The problem of checking whether a word w is accepted by a fixed PTA is polynomial-time reducible to the emptiness problem of an $O(|w|)$-phase* MVPA *with $O(|w|)$ states.*

Solving k-MVPA Emptiness

In [12], the decidability of emptiness of k-MVPA proceeds by first defining a map from words over $\widetilde{\Sigma}$ to trees, called *stack trees*, by showing that the set of stack trees that correspond to words forms a *regular* set of trees, and reducing k-MVPA emptiness to emptiness of tree automata working on the corresponding stack trees.

The map from words to trees rearranges the positions of the word into a binary tree by encoding a matching return of a call as its right child. This mapping hence easily captures the matching relation between calls and returns, but loses sight of the linear order in w. Recovering the linear order is technically hard, and is captured using monadic second-order logic (MSO) on trees.

Fix a k-phase word w of length m. We say that a factorization $w_1, \ldots, w_k$ of w is *tight* if: (1) the first symbol of w_h is a return for every $h \in [2, k]$, (2) if $k > 1$ then $Ret(w_1) \neq \emptyset$, and (3) w_h and w_{h+1} are phases of different stacks for every $h \in [1, k-1]$. It is easy to see that, for every k-phase word w there is a unique tight factorization, and thus we can uniquely assign a phase number to each letter occurrence within w as follows: for $w = w'dw''$, $d \in \widetilde{\Sigma}$, the phase of d is h iff $w_1, \ldots, w_k$ is the tight factorization of w and d is within w_h.

A *stack tree* is defined as follows:

Definition 1. *Let w be a k-phase word over $\widetilde{\Sigma}_n$ with $|w| = m$, and $w_1, \ldots, w_k$ be the tight factorization of w. The word-to-tree map of w, $wt(w)$, which is a $(\widetilde{\Sigma} \times [1, k])$-labeled tree (V, λ), and the bijection $pos : V \to [1, m]$ are inductively defined (on $|w|$) as follows:*

- *If $m = 1$, then $V = \{root\}$, $\lambda(root) = (w, 1)$, and $pos(root) = 1$.*
- *Otherwise, let $w = w'd$, $d \in \widetilde{\Sigma}$, and $wt(w') = (V', \lambda')$. Then:*
  - *$V = V' \cup \{v\}$ with $v \notin V'$.*
  - *$\lambda(v) = (d, k)$ and $\lambda(v') = \lambda'(v')$, for every $v' \in V'$.*
  - *If there is a $j < m$ such that d is a return and the j'th letter of w is its matching call (of the same stack), then v is the right-child of $pos^{-1}(j)$. Otherwise v is the left-child of $pos^{-1}(m-1)$.*
  - *$pos(v) = m$.*

The tree $wt(w)$ is called the stack tree *of w. A k-stack tree is the stack tree of a k-phase word.*

The proof that the set of stack trees that correspond to words accepted by a k-MVPA forms a regular set of trees requires showing that: (a) the set of all stack trees is regular and (b) given a stack tree, checking whether a k-MVPA has an accepting run over the corresponding word can be done by a tree automaton.

Part (a) involves the definition of a linear order $\prec$ on tree nodes which corresponds the linear order on the word from the stack tree, and [12] shows that given a tree automaton of size r accepting the $\prec$ relation (formally, accepting trees with two nodes marked x and y such that $x \prec y$), we can build an automaton of size exponential in r to accept all stack trees. It is further shown in [12] that the $\prec$ relation can be captured by an automaton of size $r = exp(poly(k))$. In order to get a $exp(exp(O(k)))$ automaton for accepting stack trees, we show now that the $\prec$ relation can be defined using automata of size $r = exp(O(k))$ (Lemma 4 below).

Part (b) requires traversing the stack tree according to the linear order on w using a *two-way alternating automaton*. We show below that there is a two-way alternating tree automaton of size $2^{O(k)}$ that traverses the tree consecutively from one node to its successor. More precisely, we show that given a tree where the first and last events of each

phase are marked, there is a 2-way alternating automaton that, when placed at a node x in the tree, will navigate to the successor of x (reaching a final state) (Lemma 5 below). It follows from [12] that using this automaton, we can check whether the word corresponding to the stack tree is accepted by a k-MVPA using a nondeterministic automaton of size $exp(exp(O(k)))$. This primarily involves an exponential conversion of alternating tree automata to nondeterministic automata [19,23], followed by other checks that can be effected by nondeterministic automata of similar size.

We present the above two results in two technical lemmas below.

Tree Automata Accepting Stack Trees

Here we give a characterization of $\prec$ which leads to a direct construction of a tree automaton of size $exp(O(k))$ that captures it.

For a $(\widetilde{\Sigma} \times [1,k])$-labeled tree $T = (V, \lambda)$, we define a map $phase_T : V \to [1,k]$ as $phase_T(x) = h$ iff $\lambda(x) = (d,h)$ for some $d \in \widetilde{\Sigma}$.

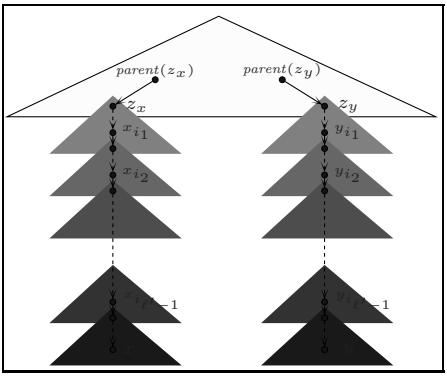
Stack trees must first satisfy some simple conditions. A tree is *well-formed* if (i) the phase numbers are monotonically increasing along any path in the tree, (ii) every right child is a return, with a call of the same stack as its parent, and (iii) the phase of the root is 1.

Let T be a well-formed tree, x be a node of T, x' be an ancestor of x, and $x_1 \ldots x_\ell$ be the path in T from x' to x. Let $I = \{i_1, i_2, \ldots, i_{\ell'-1}\}$ be the set of all indices $i \in [1, \ell - 1]$ such that $phase_T(x_i) \neq phase_T(x_{i+1})$. Assume that $i_1 < i_2 < \ldots < i_{\ell'-1}$. We denote by $PhasePath_T(x', x)$ the sequence $p_1, p_2, \ldots, p_{\ell'}$ such that $p_j = phase_T(x_{i_j})$ for every $j \in [1, \ell' - 1]$, and $p_{\ell'} = phase_T(x_\ell)$.

In the following, $<_{pre}$ is the linear order of nodes according to a preorder visit of the tree, and T_z denotes the largest subtree of T which contains z and whose nodes are labeled with the same phase number as z.

Definition 2. *Let $T = (V, \lambda)$ be a well-formed tree. For every $x, y \in V$, $x \prec_* y$ if one of the following holds:*

1. $phase_T(x) < phase_T(y)$;
2. $T_x = T_y$ *and* $x <_{pre} y$;
3. *There exists an ancestor z_x of x and an ancestor z_y of y such that*
   - $z_x \neq z_y$,
   - $phase_T(parent(z_x)) < phase_T(z_x)$,
   - $phase_T(parent(z_y)) < phase_T(z_y)$,
   - $PhasePath_T(z_x, x) = PhasePath_T(z_y, y) = p_1, \ldots, p_{\ell'}$



(see figure on the right, where similarly shaded regions belong to the same phase), and one of the following holds

(a) ℓ' is odd and $phase_T(parent(z_y)) < phase_T(parent(z_x))$, or ℓ' is even and $phase_T(parent(z_x)) < phase_T(parent(z_y))$.

(b) $T_{parent(z_x)} = T_{parent(z_y)}$, *and either* ℓ' *is odd and* $parent(z_y) <_{pre} parent(z_x)$, *or* ℓ' *is even and* $parent(z_x) <_{pre} parent(z_y)$.

It is not hard to see that there is a non-deterministic automaton that guesses the phase-path $p_1, \ldots, p_{\ell'}$ (since this sequence is always ordered in increasing order, we can represent it as the set $\{p_1, \ldots, p_{\ell'}\}$, and hence the number of guesses is $O(2^k)$) and checks whether $x \prec_* y$.

The following lemma states that $\prec_*$ and $\prec$ indeed coincide.

Lemma 3 (CHARACTERIZATION OF $\prec$)**.** *Let* $T = (V, \lambda)$ *be a* $(\widetilde{\Sigma} \times [k])$*-labeled tree that is well-formed. Then,* $x \prec_* y$ *if and only if* $x \prec y$ *for every* $x, y \in V$.

From the above argument and lemma, and the result shown in [12] we get:

Lemma 4. *For any* k*, there is a nondeterministic tree automaton of size* $exp(O(k))$ *that accepts a well-formed tree with two nodes labeled* x *and* y *iff* $x \prec y$.

Thus, we have the following theorem.

Theorem 2. *For any positive integer* k*, there is a nondeterministic tree automaton of size* $exp(exp(O(k)))$ *which accepts the set of all* k*-stack trees.*

Tree Automata Traversing Stack Trees

Given a k-stack tree T and two nodes x, y of T, we say that y is the *successor* of x if x corresponds to a position j of w and y to position $j + 1$ of w, where $wt(w) = T$.

In this section, we show that there is a two-way alternating tree automaton (see [19,23] for a definition), with $exp(O(k))$ states, that when started at a node x on a k-stack tree T, navigates to the successor of x. We will assume that we are given *markers* that mark the first letter (marked with s) and last letter (marked with e) of each phase. In fact, we can build conjunctively another automaton that checks using $exp(exp(O(k)))$ states that these markers are correct.

Formally, let $T = (V, \lambda)$ be a $(\widetilde{\Sigma} \times [1, k] \times \{s, e, \perp\})$-labeled tree and $T' = (V, \lambda')$ be the $(\widetilde{\Sigma} \times [1, k])$-labeled tree where $\lambda'(x) = (a, i)$ if $\lambda(x) = (a, i, d)$. We say that T is a *k-stack tree with markers*, if T' is a k-stack tree, and all the vertices corresponding to positions of $wt^{-1}(T')$ where a phase starts (resp. ends) are labeled in T with s (resp. e). For $x, y \in V$, we say that y is the successor of x if y is the successor of x in T'.

Lemma 5. *There exists a two-way alternating tree automaton, with* $exp(O(k))$ *states that given a* k*-stack tree* T*, when started at a node* x *of* T*, will navigate precisely to the successor of* x *(reaching a final state).*

Proof. The 2-way alternating automaton is best described *algorithmically*. It will be easy to see that this algorithm can be executed by a 2-way alternating automaton of the required size. The algorithm is shown in Fig. 1. With $\mathsf{EndPhase}(x)$ we denote a predicate that holds true whenever x is the last letter of a phase. With $\mathsf{NextPhase}(i)$, $i < k$, we denote the first letter of phase $i + 1$. With $\mathsf{PrefixSucc}(x)$, we denote the next letter in the preorder visit of T_x. With $\mathsf{ParentRoot}(x)$, we denote the parent of the root of T_x. $\mathsf{BeginPhase}(x)$, $\mathsf{PrevPhase}(i)$ and $\mathsf{PrefixPred}(x)$ are defined analogously.

```
Procedure Successor(x)
if EndPhase(x) then
        return (NextPhase(phase_T(x)));
elseif ( y ← PrefixSucc(x) exists ) then
        return (y) ;
else {z ← ParentRoot(x);
      z' ← Predecessor(z);
      while ( phase_T(rightChild(z'))
              ≠ phase_T(x) ) do
                z' ← Predecessor(z');
      return (rightChild(z')); }
```

```
Procedure Predecessor(x)
if BeginPhase(x) then
        return (PrevPhase(phase_T(x)));
elseif ( y ← PrefixPred(x) exists )
        then return (y) ;
else {z ← ParentRoot(x);
      z' ← Successor(z);
      while ( phase_T(rightChild(z'))
              ≠ phase_T(x) ) do
                z' ← Successor(z');
      return (rightChild(z')); }
```

Fig. 1. Successor and predecessor in stack trees

Intuitively, if x is the last letter of a phase, we navigate to the first letter of the next phase (effected by the first clause). Otherwise, we check whether we can find the successor locally, in the same subtree T_x; this corresponds to finding the next element in the preorder visit of T_x and is delegated to the second clause. If x is the last letter of T_x, then the successor is hard to find. Let z be the parent of the root of T_x and i be the phase number of x. Intuitively, the successor of x is obtained by taking the *last* node *before* z that has a matching return whose phase is i. We hence execute the function Predecessor iteratively till we reach a node that has a right-child of phase i.

Implementing the above requires a 2-way alternating automaton to keep a list of phase numbers. Such list can be maintained as a set (since the phase numbers on the list are ordered), and we can engineer the automaton to have $exp(O(k))$ states. Alternation is used to prove falsity of conditional clauses that are not pursued in the algorithm. □

From the above lemmas and the result from [12], we get:

Theorem 3. *The emptiness problem for k-*MVPA*s of state-space Q is decidable in time $exp(|Q| \cdot exp(O(k)))$.*

Combining Lemma 2 and the above theorem we get:

Theorem 4. *The membership problem for BPTAs is decidable in* 2ETIME.

4 The Lower Bound

In this section, we show that any language in 2ETIME is accepted by an infinite-state bounded-phase pushdown transducer automata, thereby completing the proof that such automata exactly characterize 2ETIME (Theorem 1).

We start giving a lemma which describes an interesting feature of the bounded-phase multi-stack pushdown rewriting. It states that if we have an unbounded number of pairs of bounded-length words, say bounded by N, then we can check whether every pair (w, w') is such that $|w| = |w'|$ and for each i the i'th symbol of w and w' belong to some relation over symbols, using at most $\lceil \log N \rceil / c$-steps of 2^c-phase multi-stack pushdown

rewriting. Consider a finite relation $R \subseteq \Pi \times \Pi$, and two words $w = a_1 \dots a_m$ and $w' = a'_1 \dots a'_{m'}$ over Π. We say that (w, w') satisfies R if and only if $m = m'$ and $(a_i, a'_i) \in R$ for $i = 1, \dots, m$.

Lemma 6. *Let Π be a finite alphabet, $\#$ be a symbol which is not in Π, $R \subseteq \Pi \times \Pi$, and w be any word of the form $u_1\#v_1\#u_2\#v_2\# \dots \#u_m\#v_m$, with $m > 0$ and $u_i, v_i \in \Pi^{2^{cn}}$ for $i = 1, \dots, m$ with $c, n > 0$.*

There exists a 2^c-phase 2-stack pushdown transducer $\mathcal{T}$ that rewrites within n steps each such word w to a symbol \$ if and only if (u_i, v_i) satisfies R for every $i = 1, \dots, m$.

Proof sketch. The transducer $\mathcal{T}$ splits each pair (u_i, v_i) into 2^c pairs of words, and writes them onto the output tape. This transducer can be implemented using two stacks and 2^c-phases. In n steps, the transducer hence reduces the problem of checking whether every (u_i, v_i) satisfies R to that of checking whether a large number of pairs of letters belongs to R, which can be effected by a regular automaton. □

A transducer, as stated in the above lemma, can be used to check for a Turing machine whether a configuration is a legal successor of another one. We apply this result as a crucial step in proving the following theorem which states the claimed lower bound.

Theorem 5. *For each language L in $2\text{ETIME}(\Sigma)$, there is a bounded-phase pushdown transducer automaton $\mathcal{A}$ such that $L = \mathcal{L}(\mathcal{A})$.*

Proof sketch. We reduce the membership problem for alternating Turing machines working in $2^{O(n)}$ space to the membership problem for BPTAs. The result then follows from [9].

We briefly sketch a BPTA $\mathcal{A}$ that accepts a words w if and only if w is accepted by a $2^{O(n)}$ space Turing machine $\mathcal{M}$. First $\mathcal{A}$ guesses a word w and a run t of $\mathcal{M}$ encoding them as a sequence of pairs of words (u_i, v_i) such that all the steps taken in t, and w along with the initial configuration, are all represented by at least one such pair. Then, it checks if the guessed sequence indeed encodes an accepting run of $\mathcal{M}$ on w.

In the first task we make use of a slight variation of a standard encoding of trees by words where each pair of consecutive configurations of $\mathcal{M}$ are written consecutively in the word. The second task is by Lemma 6. We observe that it suffices to have single initial and final states for $\mathcal{A}$. □

5 Discussion

We have shown an infinite-automata characterization of the class 2ETIME. This result was obtained independently of the work by Meyer showing that term-automatic infinite automata capture the class ETIME [14]. These two results, along with the characterization of NLINSPACE [15], are currently the only characterizations of complexity classes using infinite automata.

The power of multi-stack rewriting. While infinite automata capture fairly complex languages, there has been little study done on how simple infinite automata can be designed to solve natural algorithmic problems. In this section, we investigate the power

of our rewriting. We give infinite automata that solve SAT and QBF (crucially using Lemma 6), and explore connections to infinite automata based on term rewriting. While this of course follows from the lower bound shown in Section 4, the construction is instructive.

We start observing some interesting features of bounded-phase multi-stack pushdown rewriting. We can generate words corresponding to tree encodings, or, in general, belonging to a context free language. (Checking whether a word belongs to a context free language while rewriting can be a problem though: for example, it is not clear how to rewrite in 1-step a word w to a symbol 1 iff $w \in \{a^n b^n \mid n \geq 0\}$.) Also, in each rewriting we can duplicate a bounded number of times any portion of the read word. This can be useful to start many threads of computation on the same string thus speeding-up the total computation. Finally, words can be (evenly) split into a bounded number of sub-words. By iterating such splitting, we can check simple relations between an unbounded number of words, each of exponential length, as shown in Lemma 6.

SAT and QBF. Let us encode Boolean formulas in the standard way, by representing each quantifier, connective, constant and bracket with different symbols, and variables with unbounded length binary strings.

On the first step, $\mathcal{A}$ prepares the computation by rewriting its initial state with a triple (w_1, w_2, w_3) where w_1 is the encoding of a well-formed formula, w_2 is a copy of w_1 along with a valuation for each variable *occurrence*, and w_3 is the list of variable occurrences coupled with their valuation as annotated in w_2. The word w_1 is guessed nondeterministically using a stack to ensure it is well-formed, and is used by $\mathcal{A}$ to match the input formula. The word w_2 is obtained by copying w_1 and nondeterministically guessing on each variable occurrence a valuation (note that two occurrences of the same variable may be assigned with different values along some runs). Word w_2 is used to evaluate the formula in the guessed valuation. Word w_3 is extracted from w_2 and is later used to generate all pairs $(xb, x'b')$ where x, x' are variable occurrences and b, b' are respectively their assigned values. Such pairs are then checked to see if they define a consistent valuation.

Observe now that evaluating the formula requires a number of steps of rewriting bounded by its height. Also, the pairs of occurrences can be generated in $n - 1$ steps of rewriting where n is the number of variable occurrences in the formula: a sequence $x_1 \dots x_n$ is rewritten according to the recurrence $pairs(x_1 \dots x_n)$ is (x_1, x_2) along with $pairs(x_1 x_3 \dots x_n)$ and $pairs(x_2 x_3 \dots x_n)$. Finally, from Lemma 6 checking for pair consistency can be done in the length of the variable representation. Therefore, all tasks are accomplished by the time $\mathcal{A}$ terminates its input and therefore it can correctly accept or reject the input word.

This construction can be generalized to encode QBF. The main difference is that variables are assigned one at each step: when the corresponding quantifier is eliminated. The elimination of universal quantifiers requires duplication of the formula, which can be effected using a work stack.

Term-automatic rewriting. Another way to define infinite automata is to represent states using *terms* (or trees), and use term rewriting to define relations. In [14], term

automatic rewriting infinite automata are considered, and it is shown that they precisely capture ETIME (the class of languages accepted by Turing machines in time $2^{O(n)}$).

A binary relation R over terms is *automatic* if it is definable via a tree automaton which reads overlappings of the pair of terms, i.e., the terms are read synchronously on the parts where the corresponding domains intersect (see [14]).

Intuitively, a stack allows us to faithfully represent terms using a well-bracketed word. We now show how to directly translate a term-automatic infinite automaton $\mathcal{A}$ to a multi-stack rewriting infinite automaton $\mathcal{B}$ accepting the same language. Automaton $\mathcal{B}$ on the first step nondeterministically guesses the entire run of $\mathcal{A}$, i.e., a sequence of terms $t_1, \ldots, t_N$ where $N-1$ is the length of the word which will be read. Then, it checks if it is indeed an accepting run by generating all the pairs of consecutive terms in the sequence, and then checking them as in Lemma 6. To ensure that terms match when paired, we need to guess terms which all have the same shape (with dummy labels used to mark unused parts of the tree). Also, in order to have all tasks processed on time (i.e., before the input to the automaton is completely read), the guessed terms must be of size at most exponential in N. It is not hard to show by standard techniques that if a term-automatic infinite automaton has an accepting run over a word w, then it has also an accepting run on it which visits terms of size at most exponential in the length of w. Hence the infinite automaton $\mathcal{B}$ accepts the same language as $\mathcal{A}$.

Conclusions and future directions. We have defined (B)PTA with possible infinite initial and final states. Restricting the definition to single initial and final state does not alter the class of recognized languages. In fact, for each (B)PTA $\mathcal{A}$, we can easily construct a language equivalent (B)PTA $\mathcal{A}'$ which has only an initial and a final state.

We observe that, since the construction in Theorem 5 showing 2ETIME hardness uses transducers with only two stacks, the full power of BPTA can be achieved with just two stacks. If we allow transducers with only one stack we can show 2^{2^n} lower bound but it is left open whether we can capture all 2ETIME (i.e. time $2^{2^{O(n)}}$) using just one-stack transducers.

There are several choices for rewriting that can be studied. For example, prefix rewriting (where essentially the input word is treated as a stack, and an automaton works on it to produce a new stack) precisely defines context-free languages [21]. Regular and synchronized regular rewriting leads to automata that accept context-sensitive languages [15,18]. Reducing the power of rewriting to one that is weaker than synchronous regular relations seems hard (for e.g., consider relations $R \subseteq \Sigma^* \times \Sigma^*$ where the language $\{w\#w' \mid (w, w') \in R\}$ is regular; this leads to infinite automata that only capture *regular* languages).

We believe that our results may open a new technique to finding rewriting classes that capture complexity classes. Intuitively, a rewriting mechanisms for which checking whether any word in a regular language L can be rewritten in n steps to a word in a regular language L' can be solved in time (or space) $C(n)$ may be a good way to come up with conjectur rewriting schemes that define infinite automata for the class $C(n)$-time (or space).

Along this vein, consider bounded context-switching rewriting where the input word is rewritten to an output word using a finite number of stacks, but where there is only a *bounded* number of switches between the stacks (including the input tape). This is

weaker than the rewriting in this paper as the automaton is not allowed to push onto all stacks in one phase. The membership problem for bounded-context-switching automata can be seen to be NP-complete, and it will be interesting to see if this leads us to an infinite automaton characterization of NP.

The most interesting question would be to investigate if any complexity-theoretic result can be proved in a radically different fashion using infinite automata. As mentioned in [21], given that we have infinite automata for the class NL, showing that NL=CO-NL using infinite automata seems an excellent idea to pursue.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)
2. Ball, T., Rajamani, S.K.: Bebop: A symbolic model checker for boolean programs. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN 2000. LNCS, vol. 1885, pp. 113–130. Springer, Heidelberg (2000)
3. Bouajjani, A., Habermehl, P., Vojnar, T.: Abstract regular model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 372–386. Springer, Heidelberg (2004)
4. Carayol, A., Wöhrle, S.: The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 112–123. Springer, Heidelberg (2003)
5. Carayol, A., Meyer, A.: Context-sensitive languages, rational graphs and determinism. Logical Methods in Computer Science 2(2) (2006)
6. Carayol, A., Meyer, A.: Linearly bounded infinite graphs. Acta Inf. 43(4), 265–292 (2006)
7. Caucal, D.: On infinite transition graphs having a decidable monadic theory. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 194–205. Springer, Heidelberg (1996)
8. Caucal, D., Knapik, T.: A Chomsky-like hierarchy of infinite graphs. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 177–187. Springer, Heidelberg (2002)
9. Chandra, A.K., Kozen, D., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
10. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory. Springer, Heidelberg (1995)
11. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
12. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: LICS, pp. 161–170. IEEE Computer Society, Los Alamitos (2007)
13. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
14. Meyer, A.: Traces of term-automatic graphs. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 489–500. Springer, Heidelberg (2007)
15. Morvan, C., Stirling, C.: Rational graphs trace context-sensitive languages. In: Sgall, J., Pultr, A., Kolman, P. (eds.) MFCS 2001. LNCS, vol. 2136, pp. 548–559. Springer, Heidelberg (2001)
16. Muller, D.E., Schupp, P.E.: The theory of ends, pushdown automata, and second-order logic. Theor. Comput. Sci. 37, 51–75 (1985)
17. Post, E.L.: Formal reductions of the general combinatorial decision problem. American Journal of Mathematics 65(2), 197–215 (1943)
18. Rispal, C.: The synchronized graphs trace the context-sensitive languages. Electr. Notes Theor. Comput. Sci. 68(6) (2002)

19. Slutzki, G.: Alternating Tree Automata. Theor. Comput. Sci. 41, 305–318 (1985)
20. Thomas, W.: Languages, automata, and logic. Handbook of formal languages 3, 389–455 (1997)
21. Thomas, W.: A short introduction to infinite automata. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 130–144. Springer, Heidelberg (2002)
22. Thue, A.: Probleme über veränderungen von zeichenreihen nach gegebener regeln. Kra. Vidensk. Selsk. Skrifter. 1. Mat. Nat. Kl. 10 (1914)
23. Vardi, M.: Reasoning about The Past with Two-Way Automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)

Recursion Schemata for $NC^{k\star}$

Guillaume Bonfante[1], Reinhard Kahle[2], Jean-Yves Marion[1], and Isabel Oitavem[3]

[1] Loria - INPL, 615, rue du Jardin Botanique, BP-101, 54602 Villers-lès-Nancy, France
{Jean-Yves.Marion,Guillaume.Bonfante}@loria.fr
[2] CENTRIA and DM, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
kahle@mat.uc.pt
[3] UNL and CMAF, Universidade de Lisboa, Av. Prof. Gama Pinto, 2, 1649-003 Lisboa, Portugal
isarocha@ptmat.fc.ul.pt

Abstract. We give a recursion-theoretic characterization of the complexity classes NC^k for $k \geq 1$. In the spirit of implicit computational complexity, it uses no explicit bounds in the recursion and also no separation of variables is needed. It is based on three recursion schemes, one corresponds to time (time iteration), one to space allocation (explicit structural recursion) and one to internal computations (mutual in place recursion). This is, to our knowledge, the first exact characterization of NC^k by function algebra over infinite domains in implicit complexity.

1 Introduction

Since the seminal works of Simmons [19], of Leivant [11, 12], of Bellantoni and Cook [3], and of Girard [8], *implicit computational complexity* has provided models over infinite domains of major complexity classes which are independent from the notion of time or of space related to machines.

These studies have nowadays at least two twin directions. The first direction concerns the characterization of complexity classes by means of logics or of recursion schemes. A motivation is to have a mathematical model of resource-bounded computations. The second direction is more practical and aims to analyze and certify resources, which are necessary for a program execution. One of the major challenges here is to capture a broad class of useful programs whose complexity is bounded. There are several approaches [1, 6, 10, 16].

This paper falls in the first direction which can be seen as a guideline for the second approach. We give a recursion-theoretic characterization of each class NC^k by means of a function algebra INC^k based on tree recursion. We demonstrate that $\mathsf{INC}^k = NC^k$ for $k \geq 1$.

⋆ Research supported by the project *Teorias e linguagens de programação para computações com recursos limitados* within the *Programa PESSOA 2005/2006* of *GRICES - EGIDE* and partly by the FCT project POCI/MAT/61720/2004 and by DM, FCT-UNL.

This work was complete while the first author visited CMAF, Universidade de Lisboa, and the support of the visit is gratefully acknowledged.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 49–63, 2008.
© Springer-Verlag Berlin Heidelberg 2008

The classes NC^k were firstly described based on circuits. NC^k is the class of functions accepted by uniform boolean circuit families of depth $O(\log^k n)$ and polynomial size with bounded fan-in gates, where n is the length of the input—see [2] or [9]. In [18], Ruzzo identifies NC^k with the classes of languages recognized by alternating Turing machines (in short ATMs) in time $O(\log^k n)$ and space $O(\log n)$.

In fact, the main difficulty in this characterization of NC^k relies on the double constraint about time and space. Other previous characterizations based on tree recursion fail to exactly capture for this reason. In 1998, Leivant [13] characterized NC using a hierarchy of classes RSR, such that $\mathsf{RSR}_k \subseteq \mathrm{NC}^k \subseteq \mathsf{RSR}_{k+2}$ for $k \geq 2$. In the sequence of [4] and [17], this result was refined in [5] by defining term systems T^k such that $T^k \subseteq \mathrm{NC}^k \subseteq T^{k+1}$ for $k \geq 2$. Both approaches are defined in a sorted context, either with safe/normal arguments or with tiered recursion.

We define INC^k as classes of functions, over the tree algebra $\mathbb{T}$, closed under composition and three recursion schemes over $\mathbb{T}$: time iteration, explicit structural recursion and mutual in place recursion. No explicit bounds are used in the schemes and also no separation of variables is needed. The mutual in place recursion scheme, one main point of our contribution, is related to previous work of Leivant and Marion, see [14]. The absence of tiering mechanism is related to [15], so that similar diagonalization argument should be possible.

2 Preliminaries

Let $\mathbb{W}$ be the set of words over $\{0,1\}$. We denote by ϵ the empty word and by $\mathbb{W}_i$ the subset of $\mathbb{W}$ of words of length exactly i. We consider the tree algebra $\mathbb{T}$, generated by three 0-ary constructors $\mathbf{0}, \mathbf{1}, \bot$ and a binary constructor $\star$, in other words, binary trees with leaves are labeled by $\{\mathbf{0}, \mathbf{1}, \bot\}$. $\mathsf{S}(t)$ denotes the size of a tree, $\mathsf{H}(t)$ corresponds to the usual notion of height. We say that a tree t is *perfectly balanced* if it has $2^{\mathsf{H}(t)}$ leaves. All along, 0 serves as false, 1 as true and $\bot$ as the undefined.

Given a non-empty (enumerable) set of variables $\mathcal{X}$, we denote by $\mathbb{T}(\mathcal{X})$ the term-algebra of binary trees whose leaves are labeled by $\mathbf{0}, \mathbf{1}, \bot$ or variables from $\mathcal{X}$. If t, u denote some terms and x is a variable, the term $t[x \leftarrow u]$ denotes the substitution of x by u in t. Then, $t[x \leftarrow u, y \leftarrow v] = t[x \leftarrow u][y \leftarrow v]$. All along, we take care to avoid clashes of variables. When we have a collection I of variable substitutions, we use the notation $t[(x_w \leftarrow u_w)_{w \in I}]$. Again, we will avoid conflicts of variables.

We now introduce some convenient notations, used extensively all along the paper. Given a set of variables $\mathcal{X} = (x_w)_{w \in \mathbb{W}}$, we define a family of perfectly balanced trees that we call tree patterns $(\mathbf{t}_i)_{i \in \mathbb{N}}$ in $\mathbb{T}(\mathcal{X})$ where each leaf is labeled by a distinct variable:

$$\mathbf{t}_0 = x_\epsilon$$
$$\mathbf{t}_{i+1} = \mathbf{t}_i[(x_w \leftarrow x_{0w})_{w \in \mathbb{W}_i}] \star \mathbf{t}_i[(x_w \leftarrow x_{1w})_{w \in \mathbb{W}_i}]$$

Observe that the index in a variable of some tree pattern indicates the path from the root to it. For example, $\mathfrak{t}_2 = (x_{00} \star x_{01}) \star (x_{10} \star x_{11})$. The use of the $\mathfrak{t}$'s and substitutions makes notations very short. For instance, $\mathfrak{t}_2[(x_w \leftarrow f_w(x_w))_{w\in\mathbb{W}_2}] = (f_{00}(x_{00}) \star f_{01}(x_{01})) \star (f_{10}(x_{10}) \star f_{11}(x_{11}))$. This notation is particularly useful to define "big-step" recursion equations as in:

$$f((x_{00} \star x_{01}) \star (x_{10} \star x_{11})) = (f(x_{00}) \star f(x_{01})) \star (f(x_{10}) \star f(x_{11}))$$

which we shall note: $f(\mathfrak{t}_2) = \mathfrak{t}_2[(x_w \leftarrow f(x_w))_{w\in\mathbb{W}_2}]$

3 The Classes INC^k

Definition 1. *The set of* basic functions *is* $\mathcal{B} = \{\mathbf{0}, \mathbf{1}, \bot, \star, (\pi_i^j)_{i\le j}, \mathsf{cond}, \mathsf{d}_0, \mathsf{d}_1\}$ *where* $\mathbf{0}, \mathbf{1}, \bot$ *and* $\star$ *are the constructors of the algebra* $\mathbb{T}$, d_0 *and* d_1 *are the destructors of* $\mathbb{T}$, cond *is a conditional and* π_i^j *are the projections. Destructors and conditional are defined as follows:*

$$\begin{aligned}
\mathsf{d}_0(c) = \mathsf{d}_1(c) = c, \quad & c \in \{\mathbf{0}, \mathbf{1}, \bot\} \\
\mathsf{d}_0(t_0 \star t_1) = t_0, \quad & \mathsf{d}_1(t_0 \star t_1) = t_1, \\
\mathsf{cond}(\mathbf{0}, x_0, x_1, x_\bot, x_\star) = x_0, \quad & \mathsf{cond}(\mathbf{1}, x_0, x_1, x_\bot, x_\star) = x_1, \\
\mathsf{cond}(\bot, x_0, x_1, x_\bot, x_\star) = x_\bot, \quad & \mathsf{cond}(t_0 \star t_1, x_0, x_1, x_\bot, x_\star) = x_\star.
\end{aligned}$$

The set of basic functions closed by composition is called the set of explicitly defined functions*. If the output of a function is* $\mathbf{0}, \mathbf{1}$ *or* $\bot$*, then we say that the function is* boolean*. If the definition of a function does not use* $\star$*, the function is said to be* $\star$*-free. As a shorthand notation, we use* $\mathsf{d}_{b_1\cdots b_l}$ *for the function* $\mathsf{d}_{b_l} \circ \cdots \circ \mathsf{d}_{b_1}$.

Definition 2. INC^k *is the closure of the set* $\mathcal{B}$ *under composition, mutual in place recursion (*MIP*), explicit structural recursion (*ESR*), and time iteration (*TI*) for* k.

The mentioned schemes are described below.

To relate functions over words to functions over trees, we encode words of $\mathbb{W}$ by perfectly balanced trees of $\mathbb{T}$. For this, we define $\mathtt{tr}(w)$ as the perfectly balanced tree of height $\lceil \log(|w|) \rceil$ whose leaves read from left to right are the letters of w padded by $\bot$ on the right if necessary.

A function $\phi : \mathbb{W}^n \to \mathbb{W}$ is represented by a function $f \in \mathbb{T}^n \to \mathbb{T}$ iff for all words $w_1, \ldots, w_n$, $f(\mathtt{tr}(w_1), \ldots, \mathtt{tr}(w_n)) = \mathtt{tr}(\phi(w_1, \ldots, w_n))$. Actually, the representation of $\phi(w_1, \ldots, w_n)$ does not need to be canonical, that is the height of the output tree may be greater than $\lceil \log(|\phi(w_1, \ldots, w_n)|) \rceil$.

Theorem 3. *For* $k \ge 1$*, the set of functions over words represented in* INC^k *is exactly the set of functions computed by circuits in* NC^k.

The proof of the theorem is a direct consequence of Proposition 13 and Proposition 15 coming in Section 4 and 5.

3.1 Mutual in Place Recursion

As a shorthand for finite sequences, we use $(\bar{\cdot})$. The notation can be nested such as in $\bar{\sigma}(\bar{u})$ which denotes a sequence $\sigma_1(u_1, \ldots, u_{k_1}), \ldots, \sigma_n(u_1, \ldots, u_{k_n})$.

The first recursion scheme, *mutual in place recursion*, is the key element of our characterization.

Definition 4. *The functions $(f_i)_{i\in I}$ (with the set I finite) are defined by* mutual in place recursion *(*MIP*) if they are defined by a set of equations, with $i, j, l \in I$ and $c \in \{\mathbf{0}, \mathbf{1}, \bot\}$, of the form*

$$f_i(t_0 \star t_1, \bar{u}) = f_j(t_0, \bar{\sigma}_{i,0}(t_0 \star t_1, \bar{u})) \star f_l(t_1, \bar{\sigma}_{i,1}(t_0 \star t_1, \bar{u})) \quad (1)$$

$$f_i(c, \bar{u}) = g_{i,c}(\bar{u}) \quad (2)$$

where $\bar{\sigma}_{i,0}$ and $\bar{\sigma}_{i,1}$ are sequences of $\star$-free explicitely defined functions and the functions $g_{i,c}$ are explicitely defined boolean functions.

Notice that the first argument is shared by the entire set of mutually defined functions as recursion argument. While for the others, copies, switch and visit can be performed freely. As a consequence, for any such function f, one may observe that $f(t, \bar{x})$ is a tree with the exact shape of t but, possibly, with different leaves. This results from the constraints on $\bar{\sigma}_{i,0}$, $\bar{\sigma}_{i,1}$, and $g_{i,c}$. Actually, informally, to compute the value corresponding to each leaf, one first runs a transducer using the path to that leaf as input. At the end, one computes the bit by a conditional using the outputs of the transducer as pointers to some bits in the input tree.

Example 5. The following function turns the leaves of its argument to some fixed constant $c \in \{\mathbf{0}, \mathbf{1}, \bot\}$:

$$\mathsf{const}_c(t_0 \star t_1) = \mathsf{const}_c(t_0) \star \mathsf{const}_c(t_1)$$

$$\mathsf{const}_c(c') = c \qquad c' \in \{\mathbf{0}, \mathbf{1}, \bot\}$$

Taking the convention that $b \vee \bot = \bot \vee b = \bot$, one may compute (with MIP-recursion) the bitwise-or of two perfectly balanced trees of common size.

$$\mathsf{or}(t_0 \star t_1, u) = \mathsf{or}(t_0, \mathsf{d}_0(u)) \star \mathsf{or}(t_1, \mathsf{d}_1(u))$$

$$\mathsf{or}(\mathbf{0}, u) = \mathsf{cond}(u, \mathbf{0}, \mathbf{1}, \bot, \bot)$$

$$\mathsf{or}(\mathbf{1}, u) = \mathsf{cond}(u, \mathbf{1}, \mathbf{1}, \bot, \bot)$$

$$\mathsf{or}(\bot, u) = \bot$$

Actually, all "bitwise boolean formula" of several balanced trees of the same size can be written in a similar manner.

We now give some closure properties of MIP-definable functions, the first one allows us to define a family of MIP-definable functions in terms of the shorthand notation introduced above.

Lemma 6. *We suppose given a (finite) family $(n_i)_{i\in I}$ of integers, and a family $(f_i)_{i\in I}$ of functions satisfying equations of the form:*

$$f_i(\mathbf{t}_{n_i}, \bar{u}) = \mathbf{t}_{n_i}[(x_w \leftarrow f_{\mathbf{p}(i,w)}(x_w, \bar{\sigma}_{i,w}(\bar{u})))_{w\in\mathbb{W}_{n_i}}], \quad (3)$$

$$f_i(\mathbf{t}_m[(x_w \leftarrow c_w)_{w\in\mathbb{W}_m}], \bar{u}) = \mathbf{t}_m[(x_w \leftarrow g_{i,w,c_w}(\bar{u}))_{w\in\mathbb{W}_m}], 0 \le m < n_i, \quad (4)$$

where $\mathbf{p}$ is a finite mapping from $I \times \mathbb{W}$ to I, $c_w \in \{\mathbf{0}, \mathbf{1}, \bot\}$, $\bar{\sigma}_{i,w}$ are vectors of $\star$-free explicitly defined functions, and $(g_{i,w,c_w})_{i\in I, w\in\mathbb{W}, c_w\in\{\mathbf{0},\mathbf{1},\bot\}}$ are explicitely defined boolean functions. Then, the functions $(f_i)_{i\in I}$ are MIP*-definable.*

One may note that the equations above specify the functions only for well balanced trees. Since we use this Lemma only for such trees, we do not care with the values for other inputs given by the proof below.

Proof. In an equation such as Equation (3), we call n_i the level of the definition of f_i. The proof is by induction on the maximal level of the functions $N = \max_{i\in I} n_i$. If $N = 1$, then the equations correspond to usual MIP-equations.

Suppose now $N > 1$. For all the indices i such that f_i has level N, we replace its definitional equations by:

$$\begin{aligned}
f_i(t_0 \star t_1, \bar{u}) &= f_{i\bullet 0}(t_0, \bar{u}) \star f_{i\bullet 1}(t_1, \bar{u}) \\
f_{i\bullet w}(t_0 \star t_1, \bar{u}) &= f_{i\bullet w0}(t_0, \bar{u}) \star f_{i\bullet w1}(t_1, \bar{u}), && (1 < |w| < N-1) \\
f_{i\bullet w}(t_0 \star t_1, \bar{u}) &= f_{\mathbf{p}(i,w0)}(t_0, \bar{\sigma}_{i,w0}(\bar{u})) \star f_{\mathbf{p}(i,w1)}(t_1, \bar{\sigma}_{i,w1}(\bar{u})), && (|w| = N-1) \\
f_{i\bullet w}(c, \bar{u}) &= g_{i,w,c}(\bar{u}), && (1 \le |w| < N) \\
f_i(c, \bar{u}) &= g_{i,\epsilon,c}(\bar{u})
\end{aligned}$$

where the indices $i \bullet w$ are fresh. One may observe that the level of each of these functions is 1. We end by induction.

The following Lemma is easy to verify:

Lemma 7. *Suppose that $f \in (f_i)_{i\in I}$ is defined by* MIP*-recursion. Then, any function $g(t, \bar{u}) = f(t, \bar{\sigma}(t, \bar{u}))$ where the $\bar{\sigma}$ are $\star$-free explicitly defined functions can be defined by* MIP*-recursion.*

3.2 Explicit Structural Recursion

The recursion scheme defined here corresponds to the space aspect of functions definable in INC^k. It will be used to construct trees of height $O(\log n)$, see the following Lemma.

Definition 8. Explicit structural recursion *(*ESR*) is the following scheme:*

$$\begin{aligned}
f(t_0 \star t_1, \bar{u}) &= h(f(t_0, \bar{u}), f(t_1, \bar{u})) \\
f(c, \bar{u}) &= g(c, \bar{u}) && c \in \{\mathbf{0}, \mathbf{1}, \bot\}
\end{aligned}$$

where h and g are explicitely defined.

As usual, when characterizing implicitely small classes of complexity, one prevents the step function, h, to be itself defined by recursion of its critical arguments. This is often achieved by imposing some tiering discipline. Here, we just ask that the functions involved in the recursion are not themselves defined by any recursion scheme, i.e., that they are explicitely defined.

Lemma 9. *Given two natural numbers α_0 and α_1, there is a function f defined by* ESR *such that for any tree t, $\mathsf{H}(f(t)) = \alpha_1\mathsf{H}(t) + \alpha_0$.*

Proof. The proof is immediate, taking f defined by explicit structural recursion with $h = h_{\alpha_1}$ and $g = h_{\alpha_0}(\mathbf{1}, \mathbf{1})$ where $h_1(w_0, w_1) = w_0 \star w_1$ and $h_i(w_0, w_1) = h_{i-1}(w_0, w_1) \star h_{i-1}(w_0, w_1)$ for $i > 1$.

3.3 Time Iteration

The following scheme allows us to iterate MIP-definable functions. It serves to capture the time aspect of functions definable in NC^k. The scheme depends on the parameter k used for the stratification.

Definition 10. *Given $k \geq 1$, a function f is defined by k-*time iteration *(k-*TI*) from the function h which is* MIP*-definable and the function g if:*

$$\begin{aligned}
f(t_1' \star t_1'', t_2, \ldots, t_k, s, \bar{u}) &= h(f(t_1', t_2, \ldots, t_k, s, \bar{u}), \bar{u}) \\
f(c_1, t_2' \star t_2'', t_3, \ldots, t_k, s, \bar{u}) &= f(s, t_2', t_3 \ldots, t_k, s, \bar{u}) \\
&\vdots \\
f(c_1, \ldots, c_{i-1}, t_i' \star t_i'', t_{i+1}, \ldots, t_k, s, \bar{u}) &= f(c_1, \ldots, c_{i-2}, s, t_i', t_{i+1}, \ldots, t_k, s, \bar{u}) \\
&\vdots \\
f(c_1, \ldots, c_k, s, \bar{u}) &= g(s, \bar{u})
\end{aligned}$$

where $c_1, \ldots, c_k \in \{\mathbf{0}, \mathbf{1}, \bot\}$.

Notice that if (k-TI) would allow the function h to be, for instance, $\star$ then, by the following lemma, we would obviously violate the space constraint of the classes NC^k. Informally, (k-TI) enables us to iterate $O(\log^k n)$ times functions which do not increase the space needs; as remarked above, MIP-definable functions are such ones.

Lemma 11. *Given a* MIP*-definable function h, a function g and constants β_1 and β_0, there is a function f defined by k-*TI *such that for all perfectly balanced trees t*

$$f(t, \bar{u}) = \underbrace{h(\ldots h}_{\beta_1(\mathsf{H}(t))^k + \beta_0 \ \mathit{times}} (g(t, \bar{u}), \bar{u}) \ldots).$$

Proof. The proof follows the lines of Lemma 9.

4 Simulation of Alternating Turing Machines

We introduce alternating random access Turing machines (ARMs) as described in [14] by Leivant, see also [7, 18]. An ARM $M = (Q, q_0, \delta)$ consists of a (finite) set of states Q, one of these, q_0, being the initial state and actions δ to be described now. States are classified as disjunctive or conjunctive, those are called action states, or as accepting, rejecting and reading states. The operational semantics of an ARM, M, is a two stage process: firstly, generating a computation tree; secondly, evaluating that computation tree for the given input. A configuration $K = (q, w_1, w_2)$ consists of a state q and two work-stacks $w_i \in \mathbb{W}$, $i \in \{1, 2\}$. The initial configuration is given by the initial state q_0 of the machine and two empty stacks.

First, one builds a computation tree, a tree whose nodes are configurations. The root of a computation tree is the initial configuration. Then, if the state of a node is an action state, depending on the state and on the bits at the top of the work-stacks, one spawns a pair of successor configurations obtained by pushing/popping letters on the work-stacks. The t-time computation tree is the tree obtained by this process until height t.

Wlog, we assume that for each action state q, one of the two successor configurations, let us say the first one, lets the stacks unchanged. And for the second successor configuration, either the first stack or the second one is modified, but not both simultaneously. We write accordingly the transition function δ for action states: $\delta(q, a, b) = (q', q'', \mathsf{pop}_i)$ with $i \in \{1, 2\}$ means that being in state q with top bits being a and b, the first successor configuration has state q' and stacks unchanged, and the second successor has state q'' and pops one letter on stack i. When we write $\delta(q, a, b) = (q', q'', \mathsf{push}_i(c))$, with $i \in \{1, 2\}$ and $c \in \mathbb{W}_1$, it is like above but we push the letter c on the top of the stack i.

The evaluation of a finite computation tree T is done as follows. Beginning from the leaves of T until its root, one labels each node (q, w_1, w_2) according to:

- if q is a rejecting (resp. accepting) state, then it is labeled by 0 (resp. 1);
- if q is a c, j-reading state ($c = 0, 1, j = 1, 2$), then it is labeled by 0 or 1 according to whether the n'th bit of the input is c, where n is the content read on the j'th stack. If n is too large, the label is $\perp$;
- if q is an action state,
  - if it has zero or one child, it is labeled $\perp$;
  - if it has two children, take the labels of its two children and compute the current label following the convention that $c = (c \vee \perp) = (\perp \vee c) = (c \wedge \perp) = (\perp \wedge c)$ with $c \in \{0, 1, \perp\}$.

The label of a computation tree is the label of the root of the computation tree thus obtained.

We say that the machine works in time $f(n)$ if, for all inputs, the $f(n)$-time tree evaluates to 0 or 1 where n is the size of the input. It works in space $s(n)$ if the size of the stacks are bounded by $s(n)$.

Actually, to relate our function algebra to the NC^k, we say that a function is in $\mathrm{ATM}(O(\log^k n, O(\log n))$, for $k \geq 1$ if it is polynomially bounded and bitwise computed by an ATM working in time $O(\log^k n)$ and space $O(\log n)$.

Theorem 12 (Ruzzo [18]). NC^k *is exactly the set of languages recognized by ARM working in time* $O(\log(n)^k)$ *and space* $O(\log(n))$.

From that, one inclusion (from the right to the left) of our main theorem is a corollary of:

Proposition 13. *Given* $k \geq 1$ *and constants* $\alpha_1, \alpha_0, \beta_1, \beta_0$, *any* ARM *working in space* $\alpha_1 \log(n) + \alpha_0$ *and time* $\beta_1 \log^k(n) + \beta_0$, *where* n *is the length of the input, can be simulated in* INC^k.

Proof (sketch). We consider such a machine $M = (Q, q_0, \delta)$. Take $d = \lceil \log(|Q|) \rceil$. We attribute to each state in Q a word $w \in \mathbb{W}_d$ taking the convention that the initial state q_0 has encoding $0 \cdots 0$. From now on, the distinction between the state and its associated word is omitted.

Let us consider the encoding of two stacks $s_1 = a_1 a_2 \cdots a_i \in \mathbb{W}$ and $s_2 = b_1 b_2 \cdots b_j \in \mathbb{W}$ of length less or equal than $\alpha_1 \cdot \log(n) + \alpha_0$:

$$\mathsf{P}(s_1, s_2) = \mathsf{I}(a_1)\mathsf{I}(b_1)\mathsf{I}(a_2) \cdots \mathsf{I}(a_i)\mathsf{I}(\#)\mathsf{I}(b_2) \cdots \mathsf{I}(b_j)\mathsf{I}(\#)\mathsf{I}(\#) \cdots \mathsf{I}(\#)$$

where $\mathsf{I}(\mathtt{0}) = 10, \mathsf{I}(\mathtt{1}) = 11$ and $\mathsf{I}(\#) = 00$, in such a way that this word has length exactly $2(\alpha_1 \cdot \log(n) + \alpha_0 + 1)$. The "+1" origins from the extra character # which separates the two (tails of the) stacks. For convenience we use a typewriter font for the encoding I. Then, the encoding of stacks above is written

$$\mathsf{P}(s_1, s_2) = \mathtt{a_1}\,\mathtt{b_1}\,\mathtt{a_2}\,\mathtt{a_3} \cdots \mathtt{a_i}\,\#\,\mathtt{b_2}\,\mathtt{b_3} \cdots \mathtt{b_j}\,\#\,\# \cdots \#.$$

To perform the computations for some input of size n, we use a *configuration tree* which is a perfectly balanced tree of height $d + 2(\alpha_1 \cdot \log(n) + \alpha_0 + 1)$. It is used as a map from all[1] configurations to (some currently computed) values $\{\mathbf{0}, \mathbf{1}, \bot\}$. Given a configuration $K = (q, w_1, w_2)$, the leaf obtained following the path $q\mathsf{P}(w_1, w_2)$ from the root of the configuration tree is the stored value for that configuration. In other words, given a configuration tree t, the value corresponding to the configuration (q, s_1, s_2) is $\mathsf{d}_{q\mathsf{P}(s_1,s_2)}(t)$.

We describe now the process of the computation. The initial valued configuration tree has all leaves labeled by $\bot$ (this tree can be defined by explicit structural recursion, cf. Lemma 9). The strategy will be to update the leaves of the initial valued configuration tree, as many times as the running time of the machine. We will show that updates can be performed by a MIP-function. Then, we use Lemma 11 to iterate this update function. After this process, the output can be read on the left-most branch of the configuration tree, that is the path of the initial configuration $(q_0, \epsilon, \epsilon)$. So, to finish the proof, we have to show that such an update can be done by MIP-recursion.

Lemma 14. *There exists a* MIP*-definable function* $\mathsf{next}(x, y)$ *which takes as input the currently computed valued configuration tree and the input tree, and which returns the configuration tree updated according to the explanations above.*

[1] Actually, all configurations with stacks smaller than $O(\log(n))$.

$\mathsf{next}(x, y)$ works by finite case distinction just calling auxiliary functions. By Lemma 6 it is shown MIP-definable[2]:

$$\mathsf{next}(\mathsf{t}_{d+4}, y) = \mathsf{t}_{d+4}[(x_{q\mathtt{ab}} \leftarrow \mathsf{next}_{q,a,b}(x_{q\mathtt{ab}}, \mathsf{t}_{d+4}, y))_{q\in\mathbb{W}_d, \mathtt{a}\in\mathbb{W}_2, \mathtt{b}\in\mathbb{W}_2}]$$

where $\mathsf{next}_{q,a,b}$ are the auxiliary functions. The role of these functions is to update the part of the configuration tree they correspond to. More precisely, each path corresponding to a state q and bits a, b identifies a subtree containing all configurations with state q and top bits a, b. $\mathsf{next}_{q,a,b}$ updates this subtree using MIP recursion.

The definition of these auxiliary functions depends on the kind of states (accepting, rejecting, etc) and, for action states, on the top bits of the stacks.

- *Accepting and rejecting states.* We define

$$\mathsf{next}_{q,a,b}(x, t, y) = \mathsf{const}_{\mathbf{1}}(x) \text{ if } q \text{ is accepting}$$
$$\mathsf{next}_{q,a,b}(x, t, y) = \mathsf{const}_{\mathbf{0}}(x) \text{ if } q \text{ is rejecting}$$

and use Lemma 7 to get MIP-definability.

- *Reading states.* We only provide the definition corresponding to a $1, 1$-reading state. Other cases are similar, $\mathsf{next}_{q,a,b}(x, t, y) = \mathsf{read}(x, \mathsf{d}_a(y))$ with:

$$\mathsf{read}(\mathsf{t}_2, y) = (\mathsf{read}'(x_{00}, y) \star \mathsf{read}(x_{01}, y)) \star (\mathsf{read}(x_{10}, \mathsf{d}_{10}(y)) * \mathsf{read}(x_{11}, \mathsf{d}_{11}(y)))$$
$$\mathsf{read}'(x_0 \star x_1, y) = \mathsf{read}'(x_0, y) \star \mathsf{read}(x_1, y)$$
$$\mathsf{read}(c, y) = \bot$$
$$\mathsf{read}'(c, y) = \mathsf{cond}(y, \mathbf{0}, \mathbf{1}, \bot, \bot)$$

- *Action states.* These are the hard cases. To compute the value of such configurations, we need the value of its two successor configurations. The key point is that the transitions of a configuration $(q, a_1 \cdots a_i, b_1 \cdots b_j)$ to its successors are entirely determined by the state q and the two top bits a_1 and b_1 so that $\mathsf{next}_{q,a_1,b_1}$ "knows" exactly which transition it must implement. We have to distinguish the four cases where we push or pop an element on one of the two stacks: 1. $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{push}_1(\mathrm{a}_0))$; 2. $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{pop}_1)$; 3. $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{push}_2(\mathrm{b}_0))$; 4. $\delta(q, \mathrm{a}_1, \mathrm{b}_1) = (q', q'', \mathsf{pop}_2)$.

Let us see first how these action modify the encoding of configurations. So, we suppose the current configuration to be $K = (q, a_1 \cdots a_i, b_1 \cdots b_j)$. By assumption, the stacks of q' are the same as for q, so that the encoding of the first successor of K is

$$q'\,\mathtt{a}_1\,\mathtt{b}_1\,\mathtt{a}_2\,\mathtt{a}_3 \cdots \mathtt{a}_\mathtt{i}\,\#\,\mathtt{b}_2\,\mathtt{b}_3 \cdots \mathtt{b}_\mathtt{j}\,\#\,\# \cdots \#$$

For the second successor of K, the encoding depends on the four possible actions:

1. $q''\,\mathtt{a}_0\,\mathtt{b}_1\,\mathtt{a}_1\;\mathtt{a}_2\;\mathtt{a}_3 \cdots \mathtt{a}_\mathtt{i}\;\#\;\mathtt{b}_2\,\mathtt{b}_3 \cdots \mathtt{b}_\mathtt{j}\,\# \cdots \#$
2. $q''\,\mathtt{a}_2\,\mathtt{b}_1\,\mathtt{a}_3 \cdots \mathtt{a}_\mathtt{i}\;\#\;\mathtt{b}_2\,\mathtt{b}_3 \cdots \mathtt{b}_\mathtt{j}\;\#\;\#\,\# \cdots \#$
3. $q''\,\mathtt{a}_1\,\mathtt{b}_0\,\mathtt{a}_2\;\mathtt{a}_3 \cdots \mathtt{a}_\mathtt{i}\;\#\;\mathtt{b}_1\;\mathtt{b}_2\;\mathtt{b}_3 \cdots \mathtt{b}_\mathtt{j}\,\# \cdots \#$
4. $q''\,\mathtt{a}_1\,\mathtt{b}_2\,\mathtt{a}_2\;\mathtt{a}_3 \cdots \mathtt{a}_\mathtt{i}\;\#\;\mathtt{b}_3 \cdots \mathtt{b}_\mathtt{j}\;\#\;\#\,\# \cdots \#$

[2] Since, wrt the simulation, Equations for $m < d + 4$ play no role, we do not write them explicitly.

As for accepting and rejecting states, we will use auxiliary functions $\mathsf{next}_{\circ,1}$, $\mathsf{next}_{\circ,2,b_1}$, $\mathsf{next}_{\circ,3,b_1}$, and $\mathsf{next}_{\circ,4}$, which correspond to the four cases mentioned above (and where $\circ$ is $\wedge$ or $\vee$ according to the state q). Then we use Lemma 7 to show the functions $\mathsf{next}_{q,a_1,b_1}$ defined by MIP-recursion.

We come back now to the definition of the four auxiliary functions $\mathsf{next}_{\circ,1}$, $\mathsf{next}_{\circ,2,b_1}$, $\mathsf{next}_{\circ,3,b_1}$, and $\mathsf{next}_{\circ,4}$. The principle of their definition is to follow in parallel the paths of the two successor configurations. To do that, we essentially use substitution of parameters, in the mutual in place recursion scheme.

1. For the case $\delta(q,\mathrm{a}_1,\mathrm{b}_1)=(q',q'',\mathsf{push}_1(\mathrm{a}_0))$, we define $\mathsf{next}_{q,a_1,b_1}(x,t,y)=\mathsf{next}_{\circ,1}(x,\mathsf{d}_{q'a_1b_1}(t),\mathsf{d}_{q''a_0b_1a_1}(t))$. With respect to the configuration tree encoding and to the definition of next, observe that $\mathsf{next}_{\circ,1}(x,u,v)$ is fed with the arguments $(\mathsf{d}_{qa_1b_1}(t),\mathsf{d}_{q'a_1b_1}(t),\mathsf{d}_{qa_0b_1a_1}(t))$ where t is the configuration tree to be updated. So that the height of the last argument is two less than the others (one bit of the stack is encoded as two bits in the configuration tree). In this case, we can go in parallel, with the only *previso* that the second stack is shorter. Equations below cope with that technical point. Formally we define $\mathsf{next}_{\circ,1}$ as:

$$\begin{aligned}
\mathsf{next}_{\circ,1}(\mathrm{t}_2,u,v)&=\mathrm{t}_2[(x_w\leftarrow\mathsf{next}_{\circ,1'}(x_w,\mathsf{d}_w(u),\mathsf{d}_w(v)))_{w\in\mathbb{W}_2}]\\
\mathsf{next}_{\circ,1'}(\mathrm{t}_4,u,v)&=\mathrm{t}_4[(x_w\leftarrow\mathsf{next}_{\circ,1'}(x_w,\mathsf{d}_w(u),\mathsf{d}_w(v)))_{w\in\mathbb{W}_4}]\\
\mathsf{next}_{\circ,1'}(\mathrm{t}_2[x_w\leftarrow c_w],u,v)&=\mathrm{t}_2[(x_w\leftarrow\mathsf{d}_w(u)\circ v)_{w\in\mathbb{W}_2}]
\end{aligned}$$

where the c_w are to be taken in $\{\mathbf{0},\mathbf{1},\bot\}$ and $\circ$ is the conditional corresponding to the state.

2. If $\delta(q,\mathrm{a}_1,\mathrm{b}_1)=(q',q'',\mathsf{pop}_1)$, we define $\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x,t,y)=\mathsf{next}_{\circ,2,\mathrm{b}_1}(x,\mathsf{d}_{q'a_1b_1}(t),\mathsf{d}_{q''(t)})$. In that case, it is the last argument which is the bigger one.

$$\begin{aligned}
\mathsf{next}_{\circ,2,b_1}(\mathrm{t}_2,u,v)&=\mathrm{t}_2[(x_w\leftarrow\mathsf{next}'_{\circ,2}(x_w,\mathsf{d}_w(u),\mathsf{d}_{wb_1}(v)))_{w\in\mathbb{W}_2}]\\
\mathsf{next}'_{\circ,2}(\mathrm{t}_2,u,v)&=\mathrm{t}_2[(x_w\leftarrow\mathsf{next}'_{\circ,2}(x_w,\mathsf{d}_w(u),\mathsf{d}_w(v)))_{w\in\mathbb{W}_2}]\\
\mathsf{next}'_{\circ,2}(c,u,v)&=u\circ\mathsf{d}_{00}(v)
\end{aligned}$$

3. If $\delta(q,\mathrm{a}_1,\mathrm{b}_1)=(q',q'',\mathsf{push}_2(\mathrm{b}_0))$, we define $\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}$ by the equation:

$$\begin{aligned}
\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x,t,y)&=\mathsf{next}_{\circ,3,\mathrm{b}_1}(x,\mathsf{d}_{q'a_1b_1}(t),\mathsf{d}_{q''a_1b_0(t)})\\
\mathsf{next}_{\circ,3,b_1}(\mathrm{t}_2,u,v)&=(\mathsf{next}_{\circ,1}(x_{00},\mathsf{d}_{00}(u),\mathsf{d}_{00b_1}(v))\star\mathsf{next}_{\circ,1}(x_{01},\mathsf{d}_{01}(u),\mathsf{d}_{01b_1}(v)))\star\\
&\quad(\mathsf{next}_{\circ,3,b_1}(x_{10},\mathsf{d}_{10}(u),\mathsf{d}_{10}(v))\star\mathsf{next}_{\circ,3,b_1}(x_{11},\mathsf{d}_{11}(u),\mathsf{d}_{11}(v)))\\
\mathsf{next}_{\circ,3,b_1}(c,u,v)&=\bot
\end{aligned}$$

4. For the last case, that is $\delta(q,\mathrm{a}_1,\mathrm{b}_1)=(q',q'',\mathsf{pop}_2)$, we use four auxiliary arguments to remind the first letter read on the stack of the second successor.

$$\begin{aligned}
\mathsf{next}_{q,\mathrm{a}_1,\mathrm{b}_1}(x,t,y)&=\mathsf{next}_{\circ,4,\epsilon}(x,\mathsf{d}_{q'a_1b_1}(t),\mathsf{d}_{q''00}(t),\mathsf{d}_{q''01}(t),\\
&\qquad\qquad\mathsf{d}_{q''10}(t),\mathsf{d}_{q''11}(t))\\
\mathsf{next}_{\circ,4,00}(\mathrm{t}_2,u,v_{00},v_{01},v_{10},v_{11})&=\mathrm{t}_2[(x_w\leftarrow\mathsf{next}'_{\circ,2}(x_w,\mathsf{d}_w(u),v_w))_{w\in\mathbb{W}_2}]\\
\mathsf{next}_{\circ,4,v}(\mathrm{t}_2,u,v_{00},v_{01},v_{10},v_{11})&=\mathrm{t}_2[(x_w\leftarrow\mathsf{next}_{\circ,4,w}(x_w,\mathsf{d}_w(u),\mathsf{d}_w(v_{00}),\\
&\qquad\qquad\mathsf{d}_w(v_{01}),\mathsf{d}_w(v_{10}),\mathsf{d}_w(v_{11}))_{w\in\mathbb{W}_2}]\\
\mathsf{next}_{\circ,4,v'}(c,u,v_{00},v_{01},v_{10},v_{11})&=\bot
\end{aligned}$$

with $v\in\{\epsilon,01,10,11\}$ and $v'\in\mathbb{W}_0\cup\mathbb{W}_2$.

5 Compilation of Recursive Definitions to Circuit

This section is devoted to the proof of the Proposition:

Proposition 15. *For $k \geq 1$, any function in INC^k is computable in NC^k.*

We begin with some observations. All along, n denotes the size of the input. First, to simulate theoretic functions in INC^k, we will forget the tree structure and make the computations on the words made by the leaves. Actually, since the trees are always full balanced binary trees, we could restrict our attention to input of size 2^k for some k.

Second, functions defined by explicit structural recursion can be computed by NC^1 circuits. This is a direct consequence of the fact that explicit structural recursion is a particular case of LRRS-recursion as defined in Leivant and Marion [14].

Third, by induction on the definition of functions, one proves the key Lemma:

Lemma 16. *Given a function $f \in \mathsf{INC}^k$, there are (finitely many)* MIP*-functions $h_1, \ldots, h_m$ and polynomials $P_1, \ldots, P_m$ of degree smaller than k such that $f(\bar{t}, \bar{u}) = h_1^{P_1(\log(n))}(\cdots h_m^{P_m(\log(n))}(g(\bar{u}))\ldots)$ where g is defined by structural recursion.*

Now, the compilation of functions to circuits relies on three main ingredients. First point, we show that each function h_i as above can be computed by a circuit:

1. of fixed height with respect to the input (the height depends only on the definition of the functions),
2. with a linear number of gates with respect to the size of the first input of the circuit (corresponding to the recurrence argument),
3. with the number of output bits equal to the number of input bits of its first argument.

According to 1), we note H the maximal height of the circuits corresponding to the h_i's.

Second point, since there are $\sum_{i=1..m} P_i(\log(n))$ applications of such h_i, we get a circuit of height bounded by $H \times \sum_{i=1..m} P_i(\log(n)) = O(\log^k(n))$. That is a circuit of height compatible with NC^k. Observe that we have to add as a first layer a circuit that computes g. According to our second remark, this circuit has a height bounded by $O(log(n))$, so that the height of the whole circuit is of the order $O(\log^k(n))$.

Third point, the circuits corresponding to g, being in NC^1, have a polynomial number of gates with respect to n and a polynomial number of output bits with respect to n. Observe that the output of g is exactly the recurrence argument of some h_i whose output is itself the first argument of the next h_i, and so on. So that according to item 3) of the first point, the size of the input argument of each of the h_i is exactly the size of the output of g. Consequently, according to item 2) above, the number of circuit gates is polynomial.

Since all constructions are uniform, we get the expected result.

5.1 NC^0 Circuits for Mutual Recursion

In this section, we prove that functions defined by mutual in place recursion can be computed by NC^0 circuits with a linear number of gates wrt the size of the first argument. Since MIP-functions keep the shape of their first argument, we essentially have to build a circuit for each bit of this argument.

Lemma 17. *Explicitely defined boolean functions can be defined without use of* $\star$.

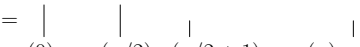
Lemma 18. *Explicitly defined boolean functions are in* NC^0.

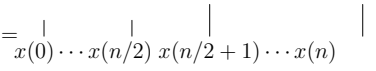
Proof. Consider the following circuits. To stress the fact that circuits are uniform, we put the size of the arguments into the brackets. n corresponds to the size of x, n_0 to the size of x_0 and so on. $x(k)$ for $k \in \mathbb{N}$ corresponds to the k-th bit of the input x. The "long" wires correspond to the outputs. Shorter ones are simply forgotten.

$C_0[n]:$ 

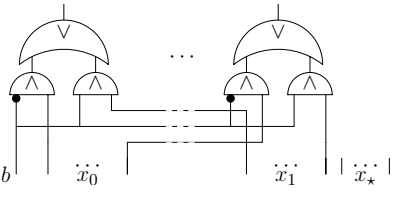
$C_1[n]:$

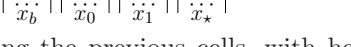
$C_{\mathtt{d}_0}[1] = C_{\mathtt{d}_1}[1] =$

$C_{\mathtt{d}_0}[2+n] =$ 

$C_{\mathtt{d}_1}[2+n] =$ 

$C_{\pi_i^j}[n_1, \cdots, n_j]:$

$C_{\mathtt{cond}}[1, n_0, n_0, n_\star] =$ 

$C_{\mathrm{cond}}[2+n_b, n_0, n_1, n_\star] =$ 

We see that composing the previous cells, with help of Lemma 17, we can build a circuit of fixed height (wrt to the size of input) for any explicitly defined boolean function. Observe that the constructions are clearly uniform.

5.2 Simulation of Time Recursion

Lemma 19. *Any* MIP*-function can be computed by a circuit of fixed height wrt the size of the input.*

Proof. Let us consider a set $(f_i)_{i\in I}$ of MIP-functions. Write their equations as follows:

$$f_i(t_0 \star t_1, \bar{u}) = f_{p(i,0)}(t_0, \bar{\sigma}_{i,0}(\bar{u})) \star f_{p(i,1)}(t_1, \bar{\sigma}_{i,1}(\bar{u}))$$
$$f_i(c, \bar{u}) = g_i(c, \bar{u})$$

where $p(i,b) \in I$ is an explicit (finite) mapping of the indices, $\bar{\sigma}_{i,0}$ and $\bar{\sigma}_{i,1}$ are vectors of $\star$-free explicitely defined functions and the functions $g_{i,c}$ (and consequently the g_i) are explicitly defined boolean functions.

First, observe that any of these explicitly defined functions g_i can be computed by some circuit B_i of fixed height as seen in Lemma 18. Since I is finite, we call M the maximal height of these circuits $(B_i)_{i\in I}$.

Suppose we want to compute $f_i(t, \bar{x})$ for some t and $\bar{x}$ which have both size smaller than n. Remember that the shape of the output is exactly the shape of the recurrence argument t. So, to any k-th bit of the recurrence argument t, we will associate a circuit computing the corresponding output bit, call this circuit C_k. Actually, we will take for each k, $C_k \in \{B_i : i \in I\}$. Putting all the circuits $(C_k)_k$ in parallel, we get a circuit that computes all the output bits of f_i, and moreover, this circuit has a height bounded by M. So, the last point is to show that for each k, we may compute uniformly the index i of the circuit B_i corresponding to C_k and the inputs of the circuit C_k.

To denote the k-th bit of the input, consider its binary encoding where we take the path in the full binary tree t ending at this k-th bit. Call this path w. Notice first that w itself has logarithmic size wrt n, the size of t. Next, observe that any sub-tree of the inputs can be represented in logarithmic size by means of its path. Since all along the computations, the arguments $\bar{u}$ are sub-trees of the input, we can accordingly represent them within the space bound.

To represent the value of a subterm of some input, we use the following data structure. Consider the record type $\mathtt{st} = \{\mathtt{r}; \mathtt{w}; \mathtt{h}\}$. The field `r` says to which input the value corresponds to. `r = 0` corresponds to t, `r = 1` correspond to x_1 and so on. `w` gives the path to the value (in that input). For convenience, we keep its height `h`. In summary {`r=i;w=w';h=m`} corresponds to the subtree $\mathtt{d}_{w'}(u_i)$ (where we take the convention that $t = u_0$). We use the '.' notation to refer to a field of a record. We consider then the data structure $\mathtt{val} = \mathtt{st} + \{0, 1\}$. Variables `u, v` coming next will be of that "type".

To compute the function $(\sigma_{i,b})_{i\in I, b\in\{0,1\}}$ appearing in the definition of the $(f_i)_{i\in I}$, we compose the programs:

```
zero(u){
  return 0;
}
```

```
one(u){
  return 1;
}
```

```
pi_i_j(u_1,...u_j){
  return u_i;
 }
```

```
d0(u){ if(u == 0 || u == 1 || u.h = 0)  return u;
         else  return [r=u.r;w=u.w 0;h= u.h-1];   }

d1(u){ if(u == 0 || u == 1 || u.h == 0) return u;
         else return [r=u.r;w=u.w 1;h= u.h-1];    }

cond(u_b,u_0,u_1,u_s){
  if (u_b == 0 ||
     (u_b.h == 0 && last-bit(u_b.w) == 0))
     return u_0;
  elseif (u_b == 1||
         (u_b.h == 0 && last-bit(u_b.w) == 1))
             return u_1;
  else  return u_s;
}
```

Then we compute the values of i and the $\bar{u}$ in $g_i(c, \bar{u})$ corresponding to the computation of the k-th bits of the output. Take $d + 1$ the maximal arity of functions in $(f_i)_{i \in I}$. To simplify the writing, we take it (wlog) as a common arity for all functions.

```
G(i,w,u_0,...,u_d){
 //u_0 corresponds to t,
   if(w == epsilon) {
   return(i,u_0,...,u_d);
   }
   else{
   a := pop(w); //get the first letter of w
   w := tail(w); //remove the first letter to w
   switch(i,a){//i in I, a in {0,1}
         case (i1,0):
         v_0 = d_0(u_0);
         foreach 1 <= k <= d:
               v_k = sigma_i1_0_k(u_0,...,u_d);
               //use the sigma defined above
               next_i = p_i1_0;
               //the map p is hard-encoded
         break;
         ...
         case (im,1):
         v_0 = d_1(u_0);
         foreach 1 <= k <= d:
               v_k = sigma_im_1_k(u_0,...,u_d);
               next_i = p_im_1;
         break;
        }
        return G(next_i,w,d_a(u_0),v_1,...,v_d);
   }
}
```

Observe that this program is a tail recursive program. As a consequence, to compute it, one needs only to store the recurrence arguments, that is a finite

number of variables. Since the value of these latter variables can be stored in logarithmic space, the computation itself can be performed within the bound. Finally, the program returns the name `i` of the circuit that must be build, a pointer on each of the inputs of the circuit with their size. It is then routine to build the corresponding circuit at the corresponding position `w`.

References

1. Aspinall, D., Beringer, L., Hofmann, M., Loidl, H.-W., Momigliano, A.: A program logic for resources. Theor. Comput. Sci. 389(3), 411–445 (2007)
2. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural complexity II. EATCS Monographs of Theoretical Computer Science, vol. 22. Springer, Heidelberg (1990)
3. Bellantoni, S., Cook, S.: A new recursion-theoretic characterization of the poly-time functions. Computational Complexity 2, 97–110 (1992)
4. Bellantoni, S., Oitavem, I.: Separating NC along the δ axis. Theoretical Computer Science 318, 57–78 (2004)
5. Bonfante, G., Kahle, R., Marion, J.-Y., Oitavem, I.: Towards an implicit characterization of NC^k. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 212–224. Springer, Heidelberg (2006)
6. Bonfante, G., Marion, J.-Y., Péchoux, R.: A characterization of alternating log time by first order functional programs. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 90–104. Springer, Heidelberg (2006)
7. Chandra, A.K., Kožen, D.J., Stockmeyer, L.J.: Alternation. Journal ACM 28, 114–133 (1981)
8. Girard, J.-Y.: Light linear logic. Information and Computation 143(2), 175–204 (1998)
9. Immerman, N.: Descriptive Complexity. Springer, Heidelberg (1998)
10. Kristiansen, L., Jones, N.D.: The flow of data and the complexity of algorithms. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 263–274. Springer, Heidelberg (2005)
11. Leivant, D.: A foundational delineation of computational feasiblity. In: Proceedings of the Sixth IEEE Symposium on Logic in Computer Science (LICS 1991) (1991)
12. Leivant, D.: Predicative recurrence and computational complexity I: Word recurrence and poly-time. In: Clote, P., Remmel, J. (eds.) Feasible Mathematics II, pp. 320–343. Birkhäuser, Basel (1994)
13. Leivant, D.: A characterization of NC by tree recurrence. In: Foundations of Computer Science 1998, pp. 716–724. IEEE Computer Society, Los Alamitos (1998)
14. Leivant, D., Marion, J.-Y.: A characterization of alternating log time by ramified recurrence. Theoretical Computer Science 236(1–2), 192–208 (2000)
15. Marion, J.-Y.: Predicative analysis of feasibility and diagonalization. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 290–304. Springer, Heidelberg (2007)
16. Niggl, K.-H., Wunderlich, H.: Certifying polynomial time and linear/polynomial space for imperative programs. SIAM J. Comput. 35(5), 1122–1147 (2006)
17. Oitavem, I.: Characterizing NC with tier 0 pointers. Mathematical Logic Quarterly 50, 9–17 (2004)
18. Ruzzo, W.L.: On uniform circuit complexity. Journal of Computer and System Sciences 22, 365–383 (1981)
19. Simmons, H.: The realm of primitive recursion. Archive for Mathematical Logic 27, 177–188 (1988)

Extensional Uniformity for Boolean Circuits*

Pierre McKenzie[1], Michael Thomas[2], and Heribert Vollmer[2]

[1] Dép. d'informatique et de recherche opérationnelle, Université de Montréal, C.P. 6128, succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
mckenzie@iro.umontreal.ca
[2] Institut für Theoretische Informatik, Leibniz Universität Hannover, Appelstr. 4, 30167 Hannover, Germany
{thomas, vollmer}@thi.uni-hannover.de

Abstract. Imposing an extensional uniformity condition on a non-uniform circuit complexity class $\mathcal{C}$ means simply intersecting $\mathcal{C}$ with a uniform class $\mathcal{L}$. By contrast, the usual intensional uniformity conditions require that a resource-bounded machine be able to exhibit the circuits in the circuit family defining $\mathcal{C}$. We say that $(\mathcal{C}, \mathcal{L})$ has the *Uniformity Duality Property* if the extensionally uniform class $\mathcal{C} \cap \mathcal{L}$ can be captured intensionally by means of adding so-called $\mathcal{L}$*-numerical predicates* to the first-order descriptive complexity apparatus describing the connection language of the circuit family defining $\mathcal{C}$.

This paper exhibits positive instances and negative instances of the Uniformity Duality Property.

Keywords: Boolean circuits, uniformity, descriptive complexity.

1 Introduction

A family $\{C_n\}_{n\geq 1}$ of Boolean circuits is *uniform* if the way in which C_{n+1} can differ from C_n is restricted. Generally, uniformity is imposed by requiring that some form of a resource-bounded constructor on input n be able to fully or partially describe C_n (see [1, 5, 8, 14, 19] or refer to [22] for an overview). Circuit-based language classes can then be compared with classes that are based on a finite computing mechanism such as a Turing machine.

Recall the gist of descriptive complexity. Consider the set of words $w \in \{a, b\}^\star$ having no b at an even position. This language is described by the FO[$<$, EVEN] formula $\neg\exists i\big(\text{EVEN}(i) \wedge P_b(i)\big)$. In such a first-order formula, the variables range over positions in w, a predicate P_σ for $\sigma \in \{a, b\}$ holds at i iff $w_i = \sigma$, and a *numerical* predicate, such as the obvious 1-ary EVEN predicate here, holds at its arguments iff these arguments fulfill the specific relation.

The following viewpoint has emerged [3, 5, 6] over two decades: *when a circuit-based language class is characterized using first-order descriptive complexity, the circuit uniformity conditions spring up in the logic in the form of restrictions on the set of numerical predicates allowed.*

* Supported in part by DFG VO 630/6-1, by the NSERC of Canada and by the (Québec) FQRNT.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 64–78, 2008.
© Springer-Verlag Berlin Heidelberg 2008

As a well studied example [5, 12], $\mathrm{FO}[<,+,\times] = \text{DLOGTIME-uniform } \mathrm{AC}^0 \subsetneq$ non-uniform $\mathrm{AC}^0 = \mathrm{FO}[\mathbf{arb}]$, where the latter class is the class of languages definable by first-order formulae entitled to *arb*itrary numerical predicates (we use a logic and the set of languages it captures interchangeably when this brings no confusion).

In a related vein but with a different emphasis, Straubing [21] presents a beautiful account of the relationship between automata theory, formal logic and (non-uniform) circuit complexity. Straubing concludes by expressing the proven fact that $\mathrm{AC}^0 \subsetneq \mathrm{ACC}^0$ and the celebrated conjectures that $\mathrm{AC}^0[q] \subsetneq \mathrm{ACC}^0$ and that $\mathrm{ACC}^0 \subsetneq \mathrm{NC}^1$ as instances of the following conjecture concerning the class REG of regular languages:

$$\mathcal{Q}[\mathbf{arb}] \cap \mathrm{REG} = \mathcal{Q}[\mathbf{reg}]. \tag{1}$$

In Straubing's instances, $\mathcal{Q}$ is an appropriate set of quantifiers chosen from $\{\exists\} \cup \{\exists^{(q,r)} : 0 \leq r < q\}$ and **reg** is the set of *reg*ular numerical predicates, that is, the set of those numerical predicates of arbitrary arity definable in a formal sense by finite automata. We stress the point of view that intersecting $\{\exists\}[\mathbf{arb}] = \mathrm{FO}[\mathbf{arb}]$ with REG to form $\mathrm{FO}[\mathbf{arb}] \cap \mathrm{REG}$ in conjecture (1) amounts to imposing uniformity on the non-uniform class $\mathrm{FO}[\mathbf{arb}]$. And once again, imposing uniformity has the effect of restricting the numerical predicates: it is a proven fact that $\mathrm{FO}[\mathbf{arb}] \cap \mathrm{REG} = \mathrm{FO}[\mathbf{reg}]$, and conjecture (1) expresses the hope that this phenomenon extends from $\{\exists\}$ to other $\mathcal{Q}$, which would determine much of the internal structure of NC^1. We ask:

1. Does the duality between uniformity in a circuit-based class and numerical predicates in its logical characterization extend beyond NC^1?
2. What would play the role of the regular numerical predicates in such a duality?
3. Could such a duality help understanding classes such as the context-free languages in AC^0?

To tackle the first question, we note that intersecting with REG is just one out of many possible ways in which one can "impose uniformity". Indeed, if $\mathcal{L}$ is any uniform language class, one can replace $\mathcal{Q}[\mathbf{arb}] \cap \mathrm{REG}$ by $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ to get another uniform subclass of $\mathcal{Q}[\mathbf{arb}]$. For example, consider any "formal language class" (in the loose terminology used by Lange when discussing language theory versus complexity theory [14]), such as the class CFL of context-free languages. Undoubtedly, CFL is a uniform class of languages. Therefore, the class $\mathcal{Q}[\mathbf{arb}] \cap \mathrm{CFL}$ is another uniform class well worth comparing with $\mathcal{Q}[<,+]$ or $\mathcal{Q}[<,+,\times]$. Of course, $\mathrm{FO}[\mathbf{arb}] \cap \mathrm{CFL}$ is none other than the poorly understood class $\mathrm{AC}^0 \cap \mathrm{CFL}$, and when Q is a quantifier given by some word problem of a nonsolvable group, $(\mathrm{FO}+\{Q\})[\mathbf{arb}] \cap \mathrm{CFL}$ is the poorly understood class $\mathrm{NC}^1 \cap \mathrm{CFL}$ alluded to 20 years ago [11].

The present paper thus considers classes $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ for various $\mathcal{Q}$ and $\mathcal{L}$. To explain its title, we note that the constructor-based approach defines uniform classes by specifying their properties: such definitions are *intensional* definitions.

By contrast, viewing $\mathcal{Q}[\mathbf{arb}] \cap \mathrm{REG}$ as a uniform class amounts to an *extensional* definition, namely one that selects the members of $\mathcal{Q}[\mathbf{arb}]$ that will collectively form the uniform class. In this paper we set up the extensional uniformity framework and we study classes $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ for $\mathcal{Q} \supseteq \{\exists\}$.

Certainly, the uniform class $\mathcal{L}$ will determine the class of numerical predicates we have to use when trying to capture $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$, as Straubing does for $\mathcal{L} = \mathrm{REG}$, as an intensionally uniform class. A contribution of this paper is to provide a meaningful definition for the set $\mathcal{L}^{\mathbb{N}}$ of $\mathcal{L}$*-numerical predicates*. Informally, $\mathcal{L}^{\mathbb{N}}$ is the set of relations over the natural numbers that are definable in the sense of Straubing [21, Section III.2] by a language over a singleton alphabet drawn from $\mathcal{L}$. When $\mathcal{L}$ is REG, the $\mathcal{L}$-numerical predicates are precisely Straubing's regular numerical predicates.

Fix a set $\mathcal{Q}$ of monoidal or groupoidal quantifiers in the sense of [5, 16, 22]. (As prototypical examples, the reader unfamiliar with such quantifiers may think of the usual existential and universal quantifiers, of Straubing's "there exist r modulo q" quantifiers, or of threshold quantifiers such as "there exist a majority" or "there exist at least t"). We propose the *Uniformity Duality Property for* $(\mathcal{Q}, \mathcal{L})$ as a natural generalization of conjecture (1):

Uniformity Duality Property for $(\mathcal{Q}, \mathcal{L})$

$$\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L} = \mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}.$$

Barrington, Immerman and Straubing [5] have shown that $\mathcal{Q}[\mathbf{arb}]$ equals $\mathrm{AC}^0[\mathcal{Q}]$, that is, non-uniform AC^0 with $\mathcal{Q}$ gates. Behle and Lange [6] have shown that $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}]$ equals $\mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]$-uniform $\mathrm{AC}^0[\mathcal{Q}]$, that is, uniform $\mathrm{AC}^0[\mathcal{Q}]$ where the direct connection language of the circuit families can be described by means of the logic $\mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]$. Hence the Uniformity Duality Property can be restated in circuit complexity-theoretic terms as follows:

Uniformity Duality Property for $(\mathcal{Q}, \mathcal{L})$, 2nd form

$$\mathrm{AC}^0[\mathcal{Q}] \cap \mathcal{L} \;=\; \mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]\text{-uniform } \mathrm{AC}^0[\mathcal{Q}] \;\cap\; \mathcal{L}.$$

By definition, $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L} \supseteq \mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}$. The critical question is whether the reverse inclusion holds. Intuitively, the Uniformity Duality Property states that the "extensional uniformity induced by intersecting $\mathcal{Q}[\mathbf{arb}]$ with $\mathcal{L}$" is a strong enough restriction imposed on $\mathcal{Q}[\mathbf{arb}]$ to permit expressing the uniform class using the $\mathcal{L}$-numerical predicates, or in other words: the extensional uniformity given by intersecting the non-uniform class with $\mathcal{L}$ coincides with the intensional uniformity condition given by first-order logic with $\mathcal{L}$-numerical predicates. Further motivation for this definition of $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}$ is as follows:

- when constructors serve to define uniform classes, they have access to input lengths but not to the inputs themselves; a convenient logical analog to this is to use the unary alphabet languages from $\mathcal{L}$ as a basis for defining the extra numerical predicates

– if the closure properties of $\mathcal{L}$ differ from the closure properties of $\mathcal{Q}[\mathbf{arb}]$, then $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L} = \mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}]$ may fail trivially (this occurs for example when $\mathcal{L} =$ CFL and $\mathcal{Q} = \{\exists\}$ since the non-context-free language $\{a^n b^n c^n : n \geq 0\}$ is easily seen to belong to $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}]$ by closure under intersection of the latter); hence intersecting $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}]$ with $\mathcal{L}$ before comparing it with $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ is necessary to obtain a reasonable generalization of Straubing's conjecture for classes $\mathcal{L}$ that are not Boolean-closed.

We now state our results, classified, loosely, as foundational observations (F) or technical statements (T). We let $\mathcal{L}$ be *any* class of languages.

(F) By design, the Uniformity Duality Property for $(\mathcal{Q}, \mathrm{REG})$ is precisely Straubing's conjecture (1), hence its conjectured validity holds the key to the internal structure of NC^1.

(F) The Uniformity Duality Property for $(\{\exists\}, \mathrm{NEUTRAL})$ is precisely the Crane Beach Conjecture [4]; here, NEUTRAL is the class of languages L that have a neutral letter, i.e., a letter e that may be arbitrarily inserted into or deleted from words without changing membership in L. The Crane Beach conjecture, stating that any neutral letter language in $\mathrm{AC}^0 = \mathrm{FO}[\mathbf{arb}]$ can be expressed in $\mathrm{FO}[<]$, was motivated by attempts to develop a purely automata-theoretic proof that *Parity*, a neutral letter language, is not in AC^0. The Crane Beach Conjecture was ultimately refuted [4], but several of its variants have been studied. Thus [4]:
  – the Uniformity Duality Property for $(\{\exists\}, \mathrm{NEUTRAL})$ fails
  – the Uniformity Duality Property for $(\{\exists\}, \mathrm{NEUTRAL} \cap \mathrm{REG})$ holds
  – the Uniformity Duality Property for $(\{\exists\}, \mathrm{NEUTRAL} \cap \{\text{two-letter languages}\})$ holds.

(T) Our definition for the set $\mathcal{L}^{\mathbb{N}}$ of $\mathcal{L}$-numerical predicates parallels Straubing's definition of regular numerical predicates. For kernel-closed language classes $\mathcal{L}$ that are closed under homomorphisms, inverse homomorphisms and intersection with a regular language, we furthermore characterize $\mathcal{L}^{\mathbb{N}}$ as the set of predicates expressible as one generalized unary $\mathcal{L}$-quantifier applied to an $\mathrm{FO}[<]$-formula. (Intuitively, $\mathcal{L}$-numerical predicates are those predicates definable in first-order logic with one "oracle call" to a language from $\mathcal{L}$.)

(T) We characterize the numerical predicates that surround the context-free languages: first-order combinations of $\mathrm{CFL}^{\mathbb{N}}$ suffice to capture all semilinear predicates over $\mathbb{N}$; in particular, $\mathrm{FO}[<, +] = \mathrm{FO}[\mathrm{DCFL}^{\mathbb{N}}] = \mathrm{FO}[\mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}]$, where DCFL denotes the deterministic context-free languages and BC(CFL) is the Boolean closure of CFL.

(T) We deduce that, despite the fact that $\mathrm{FO}[\mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}]$ contains all the semilinear relations, the Uniformity Duality Property fails for $(\{\exists\}, \mathcal{L})$ in each of the following cases:
  – $\mathcal{L} = \mathrm{CFL}$
  – $\mathcal{L} = \mathrm{VPL}$, the "visibly pushdown languages" recently introduced by [2]
  – $\mathcal{L} =$ Boolean closure of the deterministic context-free languages

- $-\mathcal{L}$ = Boolean closure of the linear context-free languages
- $-\mathcal{L}$ = Boolean closure of the context-free languages.

The crux of the justifications of these negative results is a proof that the complement of the "Immerman language", used in disproving the Crane Beach Conjecture, is context-free.

(T) At the opposite end of the spectrum, while it is clear that the Uniformity Duality Property holds for the set of all languages and any $\mathcal{Q}$, we show that the Uniformity Duality Property already holds for $(\mathcal{Q}, \mathcal{L})$ whenever $\mathcal{Q}$ is a set of groupoidal quantifiers and $\mathcal{L} = \text{NTIME}(n)^{\mathcal{L}}$; thus it holds for, e. g., the rudimentary languages, DSPACE(n), CSL and PSPACE.

The rest of this paper is organized as follows. Section 2 contains preliminaries. Section 3 defines the $\mathcal{L}$-numerical predicates and introduces the Uniformity Duality Property formally. The context-free numerical predicates are investigated in Section 4, and the duality property for classes of context-free languages is considered in Section 5. Section 6 shows that the duality property holds when $\mathcal{L}$ is "large enough". Section 7 concludes with a summary and a discussion. For the sake of brevity, proofs are omitted and will be included in the full verion.

2 Preliminaries

2.1 Complexity Theory

We assume familiarity with standard notions in formal languages, automata and complexity theory.

When dealing with circuit complexity classes, all references will be made to the non-uniform versions unless otherwise stated. Thus AC^0 refers of the Boolean functions computed by constant-depth polynomial-size unbounded-fan-in $\{\vee, \wedge, \neg\}$-circuits. And DLOGTIME-uniform AC^0 refers to the set of those functions in AC^0 computable by a circuit family having a direct connection language decidable in time $O(\log n)$ on a deterministic Turing machine (cf. [5, 22]).

2.2 First-Order Logic

Let $\mathbb{N}$ be the natural numbers $\{1, 2, 3, \ldots\}$ and let $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A *signature* σ is a finite set of relation symbols with fixed arity and constant symbols. A σ-structure $\mathfrak{A} = \langle \mathcal{U}^{\mathfrak{A}}, \sigma^{\mathfrak{A}} \rangle$ consists of a set $\mathcal{U}^{\mathfrak{A}}$, called *universe*, and a set $\sigma^{\mathfrak{A}}$ that contains an *interpretation* $R^{\mathfrak{A}} \subseteq (\mathcal{U}^{\mathfrak{A}})^k$ for each k-ary relation symbol $R \in \sigma$. We fix the interpretations of the "standard" numerical predicates $<$, $+$, $\times$, etc. to their natural interpretations. By BIT we will denote the binary relation $\{(x, i) \in \mathbb{N}^2 : \text{bit } i \text{ in the binary representation of } x \text{ is } 1\}$. For logics over strings with alphabet Σ, we will use signatures extending $\sigma_\Sigma = \{P_a : a \in \Sigma\}$ and identify $w = w_1 \cdots w_n \in \Sigma^\star$ with $\mathfrak{A}_w = \langle \{1, \ldots, n\}, \sigma^{\mathfrak{A}_w} \rangle$, where $P_a^{\mathfrak{A}_w} = \{i \in \mathbb{N} : w_i = a\}$ for all $a \in \Sigma$. We will not distinguish between a relation symbol and its interpretation, when the meaning is clear from the context.

Let $\mathcal{Q}$ be a set of (first-order) quantifiers. We denote by $\mathcal{Q}[\sigma]$ the set of first-order formulae over σ using quantifiers from $\mathcal{Q}$ only. The set of all $\mathcal{Q}[\sigma]$-formulae will be referred to as the *logic* $\mathcal{Q}[\sigma]$. In case $\mathcal{Q} = \{\exists\}$ ($\mathcal{Q} = \{\exists\} \cup \mathcal{Q}'\}$), we will also write FO$[\sigma]$ (FO+$\mathcal{Q}'[\sigma]$, respectively). When discussing logics over strings, we will omit the relation symbols from σ_Σ.

Say that a language $L \subseteq \Sigma^\star$ is *definable* in a logic $\mathcal{Q}[\sigma]$ if there exists a $\mathcal{Q}[\sigma]$-formula φ such that $\mathfrak{A}_w \models \varphi \iff w \in L$ for all $w \in \Sigma^\star$, and say that a relation $R \subseteq \mathbb{N}^n$ is definable by a $\mathcal{Q}[\sigma]$-formula if there exists a formula φ with free variables $x_1, \ldots, x_n$ that defines R for all sufficiently large initial segment of $\mathbb{N}$, i. e., if $\langle \{1, \ldots, m\}, \sigma \rangle \models \varphi(c_1, \ldots, c_n) \iff (c_1, \ldots, c_n) \in R$ for all $m \geq c_{\max}$, where $c_{\max} = \max\{c_1, \ldots, c_n\}$ [20, Section 3.1]. By abuse of notation, we will write $L \in \mathcal{Q}[\sigma]$ (or $R \in \mathcal{Q}[\sigma]$) to express that a language L (a relation R, resp.) is definable by a $\mathcal{Q}[\sigma]$-formula and use a logic and the set of languages and relations it defines interchangeably.

3 The Uniformity Duality Property

In order to generalize conjecture (1), we propose Definition 3.2 as a simple generalization of the regular numerical predicates defined using $\mathcal{V}$-structures by Straubing [21, Section III.2].

Definition 3.1. *Let $\mathcal{V}_n = \{x_1, \ldots, x_n\}$ be a nonempty set of variables and let Σ be a finite alphabet. A* $\mathcal{V}_n$-structure *is a sequence*

$$\underline{w} = (a_1, V_1) \cdots (a_m, V_m) \in (\Sigma \times \mathfrak{P}(\mathcal{V}_n))^\star$$

such that $a_1, \ldots, a_m \in \Sigma$ and the nonempty sets among $V_1, \ldots, V_m$ form a partition of $\mathcal{V}_n$ (the underscore distinguishes $\mathcal{V}_n$-structures from ordinary strings). Define $\Gamma_n = \{0\} \times \mathfrak{P}(\mathcal{V}_n)$. We say that a $\mathcal{V}_n$-structure $\underline{w}$ is unary *if $\underline{w} \in \Gamma_n^\star$, i. e., if $a_1 \cdots a_n$ is defined over the singleton alphabet $\{0\}$; in that case, we define the* kernel *of $\underline{w}$,* kern$(\underline{w})$*, as the maximal prefix of $\underline{w}$ that does not end with $(0, \emptyset)$; to signify that $x_i \in V_{c_i}$ for all $1 \leq i \leq n$, we also write* kern$(\underline{w})$ *as $[x_1 = c_1, \ldots, x_n = c_n]$ and we let $\underline{w}^{\mathbb{N}}$ stand for $(c_1, \ldots, c_n)$.*

We define Struc$_n$ *as the language of all such words in $\Gamma_n^\star$ that are unary $\mathcal{V}_n$-structures and let* Struc $= \bigcup_{n>0}$ Struc$_n$.

Any set L of unary $\mathcal{V}_n$-structures naturally prescribes a relation over the natural numbers. Hence, a set of such L prescribes a set of relations, or numerical predicates, over $\mathbb{N}$.

Definition 3.2. *Let $L \subseteq \Gamma_n^\star$ be a* unary $\mathcal{V}_n$-language*, that is, a set of unary $\mathcal{V}_n$-structures. Let $L^{\mathbb{N}} = \{\underline{w}^{\mathbb{N}} : \underline{w} \in L\}$ denote the relation over $\mathbb{N}^n$ defined by L. Then the $\mathcal{L}$-*numerical predicates *are defined as*

$$\mathcal{L}^{\mathbb{N}} = \{L^{\mathbb{N}} : L \in \mathcal{L} \text{ and } L \subseteq \text{Struc}\}.$$

We say that a language L is kernel-closed *if, for every $\underline{w} \in L$,* kern$(\underline{w}) \in L$*. We further say that a language class $\mathcal{L}$ is* kernel-closed *if, for every $L \in \mathcal{L}$ there exists an $L' \in \mathcal{L}$ such that $L^{\mathbb{N}} = L'^{\mathbb{N}}$ and L' is kernel-closed.*

We point out the following facts, where we write $\equiv_q r$ for the unary predicate $\{x : x \equiv r \mod q\}$.

Proposition 3.3. *Let* APER *and* NEUTRAL *denote the set of aperiodic languages and the set of languages having a neutral letter respectively. Then*

1. $\mathrm{APER}^{\mathbb{N}} = \mathrm{FO}[<]$,
2. $\mathrm{REG}^{\mathbb{N}} = (\mathrm{AC}^0 \cap \mathrm{REG})^{\mathbb{N}} = \mathrm{FO}[<, \{\equiv_q r : 0 \leq r < q\}] = \mathbf{reg}$, *and*
3. $\mathrm{NEUTRAL}^{\mathbb{N}} \subseteq \mathrm{FO}[<]$.

Having discussed the $\mathcal{L}$-numerical predicates, we can state the property expressing the dual facets of uniformity, namely, intersecting with an a priori uniform class on the one hand, and adding the corresponding numerical predicates to first-order logics on the other.

Property 3.4 (Uniformity Duality for ($\mathcal{Q}$,$\mathcal{L}$)). *Let $\mathcal{Q}$ be a set of quantifiers and let $\mathcal{L}$ be a language class, then*

$$\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L} = \mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}.$$

As $\mathcal{Q}[\mathbf{arb}] = \mathrm{AC}^0[\mathcal{Q}]$ [5] and $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] = \mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]$-uniform $\mathrm{AC}^0[\mathcal{Q}]$ [6], the above property equivalently states that

$$\mathrm{AC}^0[\mathcal{Q}] \cap \mathcal{L} = \mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]\text{-uniform } \mathrm{AC}^0[\mathcal{Q}] \cap \mathcal{L}.$$

As a consequence of Proposition 3.3 (1–2), the Uniformity Duality Property is equivalent to the instances of the Straubing conjectures obtained by setting $\mathcal{Q}$ and $\mathcal{L}$ as we expect, for example $\mathcal{Q} \subseteq \{\exists\} \cup \{\exists^{(q,r)} : 0 \leq r < q\}$ and $\mathcal{L} = \mathrm{REG}$ yield exactly (1). Similarly, as a consequence of Proposition 3.3 (3), the Uniformity Duality Property is equivalent to the Crane Beach Conjecture if $\mathrm{FO}[<] \subseteq \mathcal{L}$. Property 3.4 is thus false when $\mathcal{Q} = \{\exists\}$ and $\mathcal{L}$ is the set NEUTRAL of all neutral letter languages. For some other classes, the Crane Beach Conjecture and thus Property 3.4 hold: consider for example the case $\mathcal{L} = \mathrm{REG} \cap \mathrm{NEUTRAL}$ [4], or the case $\mathcal{Q} = \{\exists\}$ and $\mathcal{L} \subseteq \mathrm{NEUTRAL} \cap \mathrm{FO}[+]$. Accordingly the Uniformity Duality Property both generalizes the conjectures of Straubing et al. and captures the intuition underlying the Crane Beach Conjecture. Encouraged by this unification, we will take a closer look at the Uniformity Duality in the case of first-order logic and context-free languages in the next section.

In the rest of this section, we present an alternative characterization of $\mathcal{L}^{\mathbb{N}}$ using FO[<]-transformations and unary Lindström quantifiers. This is further justification for our definition of $\mathcal{L}$-numerical predicates. The reader unfamiliar with this topic may skip to the end of Section 3.

Digression: Numerical Predicates and Generalized Quantifiers

Generalized or Lindström quantifiers provide a very general yet coherent approach to extending the descriptive complexity of first-order logics [17]. Since we only deal with unary Lindström quantifiers over strings, we will restrict our definition to this case.

Definition 3.5. *Let $\Delta = \{\mathrm{a}_1, \ldots, \mathrm{a}_t\}$ be an alphabet, $\varphi_1, \ldots, \varphi_{t-1}$ be* FO[<]*-formulae, each with $k+1$ free variables $x_1, x_2, \ldots, x_k, y$, and let $\vec{x}$ abbreviate $x_1, x_2, \ldots, x_k$. Further, let* $\textsc{Struct}(\sigma)$ *denote the set of finite structures $\mathfrak{A} = \langle \mathcal{U}^{\mathfrak{A}}, \sigma^{\mathfrak{A}} \rangle$ over σ. Then $\varphi_1, \ldots, \varphi_{t-1}$ define an* FO[<]*-transformation*

$$[\varphi_1(\vec{x}), \ldots, \varphi_{t-1}(\vec{x})] \colon \textsc{struct}(\{<, x_1, \ldots, x_k\}) \to \Delta^\star$$

as follows: Let $\mathfrak{A} \in \textsc{struct}(\{<, x_1, \ldots, x_k\})$, $x_i^{\mathfrak{A}} = c_i \in \mathcal{U}^{\mathfrak{A}}$, $1 \leq i \leq k$, and $s = |\mathcal{U}^{\mathfrak{A}}|$, then $[\varphi_1(\vec{x}), \ldots, \varphi_{t-1}(\vec{x})](\mathfrak{A}) = v_1 \cdots v_s \in \Delta^\star$, where

$$v_i = \begin{cases} \mathrm{a}_1, & \text{if } \mathfrak{A} \models \varphi_1(c_1, \ldots, c_k, i), \\ \mathrm{a}_j, & \text{if } \mathfrak{A} \models \varphi_j(c_1, \ldots, c_k, i) \wedge \bigwedge_{l=1}^{j-1} \neg\varphi_l(c_1, \ldots, c_k, i), 1 < j < t, \\ \mathrm{a}_t, & \text{if } \mathfrak{A} \models \bigwedge_{l=1}^{t-1} \neg\varphi_l(c_1, \ldots, c_k, i). \end{cases}$$

A language $L \subseteq \Delta^\star$ and an FO[<]*-transformation $[\varphi_1(\vec{x}), \ldots, \varphi_{t-1}(\vec{x})]$ now naturally define a* (unary) Lindström quantifier $\mathcal{Q}_L^{\mathrm{un}}$ *via*

$$\mathfrak{A} \models \mathcal{Q}_L^{\mathrm{un}} y[\varphi_1(\vec{x}, y), \ldots, \varphi_{t-1}(\vec{x}, y)] \iff [\varphi_1(\vec{x}), \ldots, \varphi_{t-1}(\vec{x})](\mathfrak{A}) \in L.$$

Finally, the set of relations definable by formulae $\mathcal{Q}_L^{\mathrm{un}} y[\varphi_1(\vec{x}, y), \ldots, \varphi_{t-1}(\vec{x}, y)]$, where $L \in \mathcal{L}$ and $\varphi_1, \ldots, \varphi_{t-1} \in$ FO[<], will be denoted by $\mathcal{Q}_{\mathcal{L}}^{\mathrm{un}}$FO[<].

The notation $[\varphi_1(\vec{x}), \ldots, \varphi_{t-1}(\vec{x})]$ is chosen to distinguish the variables in $\vec{x}$ from y; the variables in $\vec{x}$ are interpreted by $\mathfrak{A}$ whereas y is utilized in the transformation.

Theorem 3.6. *Let $\mathcal{L}$ be a kernel-closed language class which is closed under homomorphisms, inverse homomorphisms and intersection with regular languages, then $\mathcal{L}^{\mathbb{N}} = \mathcal{Q}_{\mathcal{L}}^{\mathrm{un}}$FO[<]; that is, the $\mathcal{L}$-numerical predicates correspond to the predicates definable using a unary Lindström quantifier over $\mathcal{L}$ and an* FO[<]*-transformation.*

We stress that the above result provides a logical characterization of the $\mathcal{L}$-numerical predicates for all kernel-closed classes $\mathcal{L}$ forming a cone, viz. a class of languages $\mathcal{L}$ closed under homomorphisms, inverse homomorphisms and intersection with regular languages [10]. As the closure under these operations is equivalent to the closure under rational transductions (i. e., transductions performed by finite automata [7]), we obtain:

Corollary 3.7. *Let $\mathcal{L}$ be kernel-closed and closed under rational transductions, then $\mathcal{L}^{\mathbb{N}} = \mathcal{Q}_{\mathcal{L}}^{\mathrm{un}}$FO[<].*

4 Characterizing the Context-Free Numerical Predicates

In order to examine whether the Uniformity Duality Property for first-order logics holds in the case of context-free languages, we first need to consider the counterpart of the regular numerical predicates, that is, $\mathrm{CFL}^{\mathbb{N}}$. Our results in this section

will relate $\mathrm{CFL}^{\mathbb{N}}$ to addition w. r. t. to first-order combinations, and are based upon a result by Ginsburg [9]. Ginsburg showed that the number of repetitions per fragment in bounded context-free languages corresponds to a subset of the semilinear sets. For a start, note that addition is definable in $\mathrm{DCFL}^{\mathbb{N}}$.

Lemma 4.1. *Addition is definable in* $\mathrm{DCFL}^{\mathbb{N}}$.

Next, we restate the result of Ginsburg in order to prepare ground for the examination of the context-free numerical predicates. In the following, let $w^\star$ abbreviate $\{w\}^\star$ and say that a language $L \subseteq \Sigma^\star$ is *bounded* if there exists an $n \in \mathbb{N}$ and $w_1, \ldots, w_n \in \Sigma^+$ such that $L \subseteq w_1^\star \cdots w_n^\star$.

Definition 4.2. *A set* $R \subseteq \mathbb{N}_0^n$ *is* stratified *if*

1. *each element in* R *has at most two non-zero coordinates,*
2. *there are no integers* i, j, k, l *and* $x = (x_1, \ldots, x_n), x' = (x'_1, \ldots, x'_n)$ *in* R *such that* $1 \leq i < j < k < l \leq n$ *and* $x_i x'_j x_k x'_l \neq 0$.

Moreover, a set $S \subseteq \mathbb{N}^n$ *is said to be* stratified semilinear *if it is expressible as a finite union of linear sets, each with a stratified set of periods; that is,* $S = \bigcup_{i=1}^{m} \{\vec{\alpha}_{i0} + \sum_{j=1}^{n_i} k \cdot \vec{\alpha}_{ij} : k \in \mathbb{N}_0\}$, *where* $\vec{\alpha}_{i0} \in \mathbb{N}^n$, $\vec{\alpha}_{ij} \in \mathbb{N}_0^n$, $1 \leq j \leq n_i$, $1 \leq i \leq m$, *and each* $P_i = \{\vec{\alpha}_{ij} : 1 \leq j \leq n_i\}$ *is stratified.*

Theorem 4.3 ([9, Theorem 5.4.2]). *Let* Σ *be an alphabet and* $L \subseteq w_1^\star \cdots w_n^\star$ *be bounded by* $w_1, \ldots, w_n \in \Sigma^+$. *Then* L *is context-free if and only if the set*

$$E(L) = \left\{(e_1, \ldots, e_n) \in \mathbb{N}_0^n : w_1^{e_1} \ldots w_n^{e_n} \in L\right\}$$

is a stratified semilinear set.

Theorem 4.3 relates the bounded context-free languages to a strict subset of the semilinear sets. The semilinear sets are exactly those sets definable by FO[+]-formulae. There are however sets in FO[+] that are undefinable in $\mathrm{CFL}^{\mathbb{N}}$: e. g., if $R = \{(x, 2x, 3x) : x \in \mathbb{N}\}$ was definable in $\mathrm{CFL}^{\mathbb{N}}$ then $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n : n \in \mathbb{N}\} \in \mathrm{CFL}$. Hence, FO[+] can not be captured by $\mathrm{CFL}^{\mathbb{N}}$ alone. Yet, addition is definable in $\mathrm{CFL}^{\mathbb{N}}$, therefore we will in the following investigate the relationship between first-order logic with addition, FO[+], and the Boolean closure of CFL, BC(CFL).

Theorem 4.4. $\mathrm{BC}(\mathrm{CFL}^{\mathbb{N}}) \subseteq \mathrm{BC}(\mathrm{CFL})^{\mathbb{N}} \subseteq \mathrm{FO}[+]$.

That is, the relations definable in the Boolean closure of the context-free unary V_n-languages are captured by FO[+]. Hence, $\mathrm{FO}[\mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}] \subseteq \mathrm{FO}[+]$. Now Lemma 4.1 yields the following corollary.

Corollary 4.5. $\mathrm{FO}[\mathrm{DCFL}^{\mathbb{N}}] = \mathrm{FO}[\mathrm{CFL}^{\mathbb{N}}] = \mathrm{FO}[\mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}] = \mathrm{FO}[+]$.

We note that in particular, for any $k \in \mathbb{N}$, the inclusion $(\bigcap_k \mathrm{CFL})^{\mathbb{N}} \subsetneq \mathrm{FO}[+]$ holds, where $\bigcap_k \mathrm{CFL}$ denotes the languages definable as the intersection of $\leq k$ context-free languages: this is deduced from embedding numerical predicates

derived from the infinite hierarchy of context-free languages by Liu and Weiner into $\mathrm{CFL}^{\mathbb{N}}$ [18]. Hence,

$$\mathrm{CFL}^{\mathbb{N}} \subsetneq \cdots \subsetneq (\bigcap_k \mathrm{CFL})^{\mathbb{N}} \subsetneq (\bigcap_{k+1} \mathrm{CFL})^{\mathbb{N}} \subsetneq \cdots \subsetneq (\bigcap \mathrm{CFL})^{\mathbb{N}} \subseteq \mathrm{FO}[+].$$

Unfortunately, we could neither prove nor refute $\mathrm{FO}[+] \subseteq \mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}$. The difficulty in comparing $\mathrm{FO}[+]$ and $\mathrm{BC}(\mathrm{CFL})^{\mathbb{N}}$ comes to some extent from the restriction on the syntactic representation of tuples in CFL; viz., context-free languages may only compare distances between variables, whereas the tuples defined by unary $\mathcal{V}_n$-languages count positions from the beginning of a word. This difference matters only for language classes that are subject to similar restrictions as the context-free languages (e. g., the regular languages are not capable of counting, the context-sensitive languages have the ability to convert between these two representations). To account for this special behavior, we will render precisely $\mathrm{CFL}^{\mathbb{N}}$ in Theorem 4.6.

But there is more to be taken into account. Consider, e. g., the relation $R = \{(x, x, x) : x \in \mathbb{N}\}$. R is clearly definable in $\mathrm{CFL}^{\mathbb{N}}$, yet the set $E(L)$ of the defining language L, $L^{\mathbb{N}} = R$, is not stratified semilinear. Specifically, duplicate variables and permutations of the variables do not increase the complexity of a unary $\mathcal{V}_n$-language L but affect $L^{\mathbb{N}}$.

Let t be an order type of $\vec{x} = (x_1, \ldots, x_n)$ and say that a relation $R \subseteq \mathbb{N}^n$ has order type t if, for all $\vec{x} \in R$, $\vec{x}$ has order type t. For $\vec{x}$ of order type t, let $\vec{x}' = (x'_1, \ldots, x'_m)$, $m \leq n$, denote the variables in $\vec{x}$ with mutually distinct values and let π_t denote a permutation such that $x'_{\pi_t(i)} < x'_{\pi_t(i+1)}$, $1 \leq i < m$. We define functions $\mathit{sort} \colon \mathfrak{P}(\mathbb{N}^n) \to \mathfrak{P}(\mathbb{N}^m)$ and $\mathit{diff} \colon \mathfrak{P}(\mathbb{N}^n) \to \mathfrak{P}(\mathbb{N}_0^n)$ as

$$\mathit{sort}(R) = \big\{\pi_t(\vec{x}') : \vec{x} \in R \text{ has order type } t\big\},$$

$$\mathit{diff}(R) = \Big\{(x_i)_{1 \leq i \leq n} : \Big(\sum_{j=1}^{i} x_j\Big)_{1 \leq i \leq n} \in R\Big\}.$$

The function *sort* rearranges the components of R in an ascending order and eliminates duplicates, whereas *diff* transforms a tuple $(x_1, \ldots, x_n)$ with $x_1 < x_2 < \cdots < x_n$ into $(x_1, x_2 - x_1, x_3 - x_2 - x_1, \ldots, x_n - \sum_{i=1}^{n-1} x_i)$, a representation more "suitable" to CFL (cf. $E(L)$ in Theorem 4.3).

Theorem 4.6. *Let $R \subseteq \mathbb{N}^n$. $R \in \mathrm{CFL}^{\mathbb{N}}$ if and only if there exists a partition $R = R_1 \cup \cdots \cup R_k$ such that each $\mathit{diff}\big(\mathit{sort}(R_i)\big)$, $1 \leq i \leq k$, is a stratified semilinear set.*

5 The Uniformity Duality and Context-Free Languages

Due to the previous section, we may express the Uniformity Duality Property for context-free languages using Corollary 4.5 in the following more intuitive way: let $\mathcal{Q} = \{\exists\}$ and $\mathcal{L}$ be such that $\mathrm{FO}[\mathcal{L}^{\mathbb{N}}] = \mathrm{FO}[<, +]$ (e. g., $\mathrm{DCFL} \subseteq \mathcal{L} \subseteq \mathrm{BC}(\mathrm{CFL})$), then the Uniformity Duality Property for $(\{\exists\}, \mathcal{L})$ is equivalent to

$$\mathrm{FO}[\mathbf{arb}] \cap \mathcal{L} = \mathrm{FO}[<, +] \cap \mathcal{L}. \tag{2}$$

We will hence examine whether (2) holds, and see that this is not the case.

For a binary word $u = u_{n-1}u_{n-2}\cdots u_0 \in \{0,1\}^\star$, we write $\widehat{u}$ for the integer $u_{n-1}2^{n-1} + \cdots + 2u_1 + u_0$. Recall the Immerman language $L_I \subseteq \{0,1,\mathrm{a}\}^\star$, that is, the language consisting of all words of the form $x_1\mathrm{a}x_2\mathrm{a}\cdots\mathrm{a}x_{2^n}$, where $x_i \in \{0,1\}^n$, $\widehat{x}_i + 1 = \widehat{x}_{i+1}$, $1 \le i < 2^n$, and $x_1 = 0^n$, $x_{2^n} = 1^n$. For example, $00\mathrm{a}01\mathrm{a}10\mathrm{a}11 \in L_I$ and $000\mathrm{a}001\mathrm{a}010\mathrm{a}011\mathrm{a}100\mathrm{a}101\mathrm{a}110\mathrm{a}111 \in L_I$. We prove that despite its definition involving arithmetic, L_I is simply the complement of a context-free language.

Lemma 5.1. *The complement $\overline{L_I}$ of the Immerman language is context-free.*

For a language $L \subseteq \Sigma^\star$, let Neutral(L) denote L supplemented with a neutral letter $\mathrm{e} \notin \Sigma$, i.e., Neutral(L) consists of all words in L with possibly arbitrary repeated insertions of the neutral letter. The above Lemma implies that Neutral(L_I) $\in$ BC(CFL). From [4] we know that Neutral(L_I) $\in$ FO[**arb**] $\setminus$ FO[$<,+$]. This finally leads to the following:

Theorem 5.2. FO[**arb**] $\cap$ BC(CFL) $\supsetneq$ FO[$<,+$] $\cap$ BC(CFL).

Theorem 5.2 implies that the Uniformity Duality Property fails for $\mathcal{Q} = \{\exists\}$ and $\mathcal{L} = \mathrm{BC(CFL)}$, since $\mathrm{FO}[<, \mathrm{BC(CFL)}^{\mathbb{N}}] = \mathrm{FO}[<,+]$. Yet, it even provides a witness for the failure of the duality property in the case of $\mathcal{L} = \mathrm{CFL}$, as the context-free language Neutral($\overline{L_I}$) lies in FO[**arb**] $\setminus$ FO[$<,+$]. We will state this result as a corollary further below. For now, consider the modified Immerman language R_I defined as L_I except that the successive binary words are reversed in alternance, i. e.,

$$R_I = \{\ldots, 000\mathrm{a}(001)^R\mathrm{a}010\mathrm{a}(011)^R\mathrm{a}100\mathrm{a}(101)^R\mathrm{a}110\mathrm{a}(111)^R, \ldots\}.$$

R_I is the intersection of two deterministic context-free languages. Even more, the argument in Lemma 5.1 can actually be extended to prove that the complement of R_I is a linear CFL. Hence,

Theorem 5.3. *1.* FO[**arb**] $\cap$ BC(DCFL) $\supsetneq$ FO[$<,+$] $\cap$ BC(DCFL).
2. FO[**arb**] $\cap$ BC(LinCFL) $\supsetneq$ FO[$<,+$] $\cap$ BC(LinCFL).

The role of neutral letters in the above theorems suggests taking a closer look at Neutral(CFL). As the Uniformity Duality Property for $(\{\exists\}, \mathrm{Neutral(CFL)})$ would have it, all neutral-letter context-free languages in AC^0 would be regular and aperiodic. This is, however, not the case as witnessed by Neutral($\overline{L_I}$). Hence,

Corollary 5.4. *In the case of $\mathcal{Q} = \{\exists\}$, the Uniformity Duality Property fails in all of the following cases.*

1. $\mathcal{L} = \mathrm{CFL}$,
2. $\mathcal{L} = \mathrm{BC(CFL)}$,
3. $\mathcal{L} = \mathrm{BC(DCFL)}$,
4. $\mathcal{L} = \mathrm{BC(LinCFL)}$,
5. $\mathcal{L} = \mathrm{Neutral(CFL)}$.

Remark 5.5. The class VPL of visibly pushdown languages [2] has gained prominence recently because it shares with REG many useful properties. But despite having access to a stack, the VPL-numerical predicates coincide with $\mathrm{REG}^{\mathbb{N}}$, for each word may only contain constantly many characters different from $(0, \emptyset)$. It follows that the Uniformity Duality Property fails for VPL and first-order quantifiers: consider, e. g., $L = \{\mathrm{a}^n\mathrm{b}^n : n > 0\} \in \mathrm{FO}[\mathbf{arb}] \cap (\mathrm{VPL} \setminus \mathrm{REG})$ then $L \in \mathrm{FO}[\mathbf{arb}] \cap \mathrm{VPL}$ but $L \notin \mathrm{FO}[<, \mathrm{VPL}^{\mathbb{N}}] \cap \mathrm{VPL}$.

6 The Duality in Higher Classes

We have seen that the context-free languages do not exhibit our conjectured Uniformity Duality. In this section we will show that the Uniformity Duality Property holds if the extensional uniformity condition imposed by intersecting with $\mathcal{L}$ is quite loose, in other words, if the language class $\mathcal{L}$ is powerful.

Recall the notion of non-uniformity introduced by Karp and Lipton [13].

Definition 6.1. *For a complexity class $\mathcal{L}$, denote by $\mathcal{L}/\mathrm{poly}$ the class $\mathcal{L}$ with polynomial advice. That is, $\mathcal{L}/\mathrm{poly}$ is the class of all languages L such that, for each L, there is a function $f\colon \mathbb{N} \to \{0,1\}^\star$ with*

1. *$|f(x)| \leq p(|x|)$, for all x, and*
2. *$L^f = \{\langle x, f(|x|)\rangle : x \in L\} \in \mathcal{L}$,*

where p is a polynomial depending on $\mathcal{L}$. Without loss of generality, we will assume $|f(x)| = |x|^k$ for some $k \in \mathbb{N}$.

Note that, using the above notation, DLOGTIME-uniform $\mathrm{AC}^0/\mathrm{poly} = \mathrm{AC}^0$. As we further need to make the advice strings accessible in a logic, we define the following predicates.

Following [5], we say that a Lindström quantifier Q_L is *groupoidal*, if $L \in \mathrm{CFL}$.

Definition 6.2. *Let $\mathcal{Q}$ be any set of groupoidal quantifiers. Further, let $L \in$ DLOGTIME-uniform $\mathrm{AC}^0[\mathcal{Q}]/\mathrm{poly}$ and let f be the function for which $L^f \in$ DLOGTIME-uniform $\mathrm{AC}^0[\mathcal{Q}]$. Let $r = 2kl + 1$, where k and l are chosen such that the circuit family recognizing L in DLOGTIME-uniform AC^0 has size n^l and $|f(x)| = |x|^k$. We define $\textsc{Advice}^f_{L,\mathcal{Q}} \in \mathrm{FO}{+}\mathcal{Q}[\mathbf{arb}]$ to be the ternary relation*

$$\textsc{Advice}^f_{L,\mathcal{Q}} = \{(i, n, n^r) : \textit{bit } i \textit{ of } f(n) \textit{ equals } 1\},$$

and denote the set of all relations $\textsc{Advice}^f_{L,\mathcal{Q}}$, for $L \in \mathcal{L}$, by $\textsc{Advice}_{\mathcal{L},\mathcal{Q}}$.

The intention of $\textsc{Advice}^f_{L,\mathcal{Q}}$ is to encode the advice string as a numerical relation. A point in this definition that will become clear later is the third argument of the $\textsc{Advice}^f_{L,\mathcal{Q}}$-predicate; it will pad words in the corresponding unary $\mathcal{V}_n$-language to the length of the advice string. This padding will be required for Theorem 6.4.

Theorem 6.3. *Let $\mathcal{L}$ be a language class and $\mathcal{Q}$ be a set of groupoidal quantifiers. Then the Uniformity Duality Property for $(\{\exists\} \cup \mathcal{Q}, \mathcal{L})$ holds if $\textsc{Bit} \in \mathcal{L}^{\mathbb{N}}$ and $\textsc{Advice}_{\mathcal{L},\mathcal{Q}} \in \mathcal{L}^{\mathbb{N}}$.*

We can now give a lower bound beyond which the Uniformity Duality Property holds. Let $\mathrm{NTIME}(n)^{\mathcal{L}}$ denote the class of languages decidable in linear time by nondeterministic Turing machines with oracles from $\mathcal{L}$.

Theorem 6.4. *Let $\mathcal{Q}$ be any set of groupoidal quantifiers and suppose $\mathcal{L} = \mathrm{NTIME}(n)^{\mathcal{L}}$. Then the Uniformity Duality Property for $(\{\exists\} \cup \mathcal{Q}, \mathcal{L})$ holds.*

Corollary 6.5. *Let $\mathcal{Q}$ be any set of groupoidal quantifiers. The Uniformity Duality Property holds for $(\{\exists\} \cup \mathcal{Q}, \mathcal{L})$ if $\mathcal{L}$ equals the deterministic context-sensitive languages* $\mathrm{DSPACE}(n)$*, the context-sensitive languages* CSL*, the rudimentary languages (i. e., the linear time hierarchy [24]),* PH*,* PSPACE*, or the recursively enumerable languages.*

7 Conclusion

For a set $\mathcal{Q}$ of quantifiers and a class $\mathcal{L}$ of languages, we have suggested that $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ defines an (extensionally) uniform complexity class. After defining the notion of $\mathcal{L}$-numerical predicates, we have proposed comparing $\mathcal{Q}[\mathbf{arb}] \cap \mathcal{L}$ with its subclass $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}$, a class equivalently defined as the (intensionally) uniform circuit class $\mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}]$-uniform $\mathrm{AC}^0[\mathcal{Q}] \cap \mathcal{L}$.

We have noted that the duality property, defined to hold when both classes above are equal, encompasses Straubing's conjecture (1) as well as some positive and some negative instances of the Crane Beach Conjecture.

We have then investigated the duality property in specific cases with $\mathcal{Q} = \{\exists\}$. We have seen that the property fails for several classes $\mathcal{L}$ involving the context-free languages. Exhibiting these failures has required new insights, such as characterizations of the context-free numerical predicates and a proof that the complement of the Immerman language is context-free, but these failures have prevented successfully tackling complexity classes such as $\mathrm{AC}^0 \cap \mathrm{CFL}$. Restricting the class of allowed relations on the left hand side of the uniformity duality property from **arb** to a subclass might lead to further insight and provide positive examples of this modified duality property (and address, e.g., the class of context-free languages in different uniform versions of AC^0). Methods from embedded finite model theory should find applications here.

More generally, the duality property widens our perspective on the relationship between uniform circuits and descriptive complexity beyond the level of NC^1. We have noted for example that the property holds for any set of groupoidal quantifiers $\mathcal{Q} \supseteq \{\exists\}$ and complexity classes $\mathcal{L}$ that are closed under nondeterministic linear-time Turing reductions.

A point often made is that a satisfactory uniformity definition should apply comparable resource bounds to a circuit family and to its constructor. For instance, although P-uniform NC^1 has merit [1], the classes AC^0-uniform NC^1 and NC^1-uniform NC^1 [5] seem more fundamental, provided that one can make sense of the apparent circularity. As a by-product of our work, we might suggest $\mathrm{FO}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}$ as the minimal "uniform subclass of $\mathcal{L}$" and thus as a meaningful (albeit restrictive) definition of $\mathcal{L}$*-uniform* $\mathcal{L}$. Our choice of $\mathrm{FO}[<]$ as the

"bottom class of interest" is implicit in this definition and results in the containment of $\mathcal{L}$-*uniform* $\mathcal{L}$ in (non-uniform) AC^0 for any $\mathcal{L}$. Progressively less uniform subclasses of $\mathcal{L}$ would be the classes $\mathcal{Q}[<, \mathcal{L}^{\mathbb{N}}] \cap \mathcal{L}$ for $\mathcal{Q} \supseteq \{\exists\}$.

Restating hard questions such as conjecture (1) in terms of a unifying property does not make these questions go away. But the duality property raises further questions. As an example, can the duality property for various $(\mathcal{Q}, \mathcal{L})$ be shown to hold or to fail when $\mathcal{Q}$ includes the majority quantifier? This could help develop incisive results concerning the class TC^0. To be more precise, let us consider $\mathcal{Q} = \{\exists, \mathrm{MAJ}\}$. The majority quantifier is a particular groupoidal (or, *context-free*) quantifier [16], hence it seems natural to consider the Uniformity Duality Property for $(\{\exists, \mathrm{MAJ}\}, \mathrm{CFL})$:

$$\mathrm{FO{+}MAJ}[\mathbf{arb}] \cap \mathrm{CFL} = \mathrm{FO{+}MAJ}[<, +] \cap \mathrm{CFL}. \tag{3}$$

It is not hard to see that the Immerman language in fact is in $\mathrm{FO{+}MAJ}[<, +]$, hence our Theorem 5.2 that refutes (2), the Uniformity Duality Property for $(\mathrm{FO}, \mathrm{BC(CFL)})$, does not speak to whether (3) holds. (Another prominent example that refutes (2) is the "Wotschke language" $\mathrm{W} = \{(\mathrm{a}^n\mathrm{b})^n : n \geq 0\}$, again a co-context-free language [23]. Similar to the case of the Immerman language we observe that $\mathrm{W} \in \mathrm{FO{+}MAJ}[<, +]$, hence W does not refute (3) either.)

Observe that $\mathrm{FO{+}MAJ}[\mathbf{arb}] = \mathrm{TC}^0$ [5] and that, on the other hand, $\mathrm{FO{+}MAJ}[<, +] = \mathrm{MAJ}[<] = \mathrm{FO}[+]$-uniform linear fan-in TC^0 [6, 15]. Let us call this latter class sTC^0 (for *small* TC^0 or *strict* TC^0). It is known that $\mathrm{sTC}^0 \subsetneq \mathrm{TC}^0$ [16]. Hence we conclude that if (3) holds, then in fact $\mathrm{TC}^0 \cap \mathrm{CFL} = \mathrm{sTC}^0 \cap \mathrm{CFL}$. Thus, if we can show that some language in the Boolean closure of the context-free languages is not in sTC^0, we have a new TC^0 lower bound. Thus, to separate TC^0 from a superclass it suffices to separate sTC^0 from a superclass, a possibly less demanding goal. This may be another reason to look for appropriate uniform classes $\mathcal{L}$ such that

$$\mathrm{FO{+}MAJ}[\mathbf{arb}] \cap \mathcal{L} = \mathrm{FO{+}MAJ}[<, +] \cap \mathcal{L}.$$

Acknowledgements

We would like to thank Klaus-Jörn Lange (personal communication) for suggesting Lemma 5.1. We also acknowledge helpful discussions on various topics of this paper with Christoph Behle, Andreas Krebs, Klaus-Jörn Lange and Thomas Schwentick. We also acknowledge helpful comments from the anonymous referees.

References

1. Allender, E.: P-uniform circuit complexity. Journal of the Association for Computing Machinery 36, 912–928 (1989)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: Proc. of the 16th Annual ACM Symposium on Theory of Computing, pp. 202–211 (2004)

3. Mix Barrington, D.A., Immerman, N.: Time, hardware, and uniformity. In: Proceedings 9th Structure in Complexity Theory, pp. 176–185. IEEE Computer Society Press, Los Alamitos (1994)
4. Mix Barrington, D.A., Immerman, N., Lautemann, C., Schweikardt, N., Thérien, D.: First-order expressibility of languages with neutral letters or: The Crane Beach Conjecture. Journal of Computer and System Sciences 70, 101–127 (2005)
5. Mix Barrington, D.A., Immerman, N., Straubing, H.: On uniformity within NC^1. Journal of Computer and System Sciences 41(3), 274–306 (1990)
6. Behle, C., Lange, K.-J.: FO[<]-Uniformity. In: Proceedings of the 21st Annual IEEE Conference on Computational Complexity (CCC 2006), pp. 183–189 (2006)
7. Berstel, J.: Transductions and Context-Free Languages. Leitfäden der angewandten Mathematik und Mechanik LAMM, vol. 38. Teubner (1979)
8. Borodin, A.: On relating time and space to size and depth. SIAM Journal on Computing 6, 733–744 (1977)
9. Ginsburg, S.: The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York (1966)
10. Ginsburg, S., Greibach, S., Hopcroft, J.: Abstract families of languages. Memoirs of the Amer. Math. Soc. 87 (1969)
11. Ibarra, O., Jiang, T., Ravikumar, B.: Some subclasses of context-free languages in NC^1. Information Processing Letters 29, 111–117 (1988)
12. Immerman, N.: Expressibility and parallel complexity. SIAM Journal on Computing 18, 625–638 (1989)
13. Karp, R., Lipton, R.: Turing machines that take advice. L'enseignement mathématique 28, 191–209 (1982)
14. Lange, K.-J.: Complexity theory and formal languages. In: Dassow, J., Kelemen, J. (eds.) IMYCS 1988. LNCS, vol. 381, pp. 19–36. Springer, Heidelberg (1989)
15. Lange, K.-J.: Some results on majority quantifiers over words. In: 19th IEEE Conference on Computational Complexity, pp. 123–129 (2004)
16. Lautemann, C., McKenzie, P., Schwentick, T., Vollmer, H.: The descriptive complexity approach to LOGCFL. Journal of Computer and Systems Sciences 62(4), 629–652 (2001)
17. Lindström, P.: First order predicate logic with generalized quantifiers. Theoria 32, 186–195 (1966)
18. Liu, L., Weiner, P.: An infinite hierarchy of intersections of context-free languages. Mathematical Systems Theory 7, 185–192 (1973)
19. Ruzzo, W.L.: On uniform circuit complexity. Journal of Computer and Systems Sciences 21, 365–383 (1981)
20. Schweikardt, N.: Arithmetic, first-order logic, and counting quantifiers. ACM Transactions on Computational Logic 6(3), 634–671 (2005)
21. Straubing, H.: Finite Automata, Formal Logic, and Circuit Complexity. Birkhäuser, Boston (1994)
22. Vollmer, H.: Introduction to Circuit Complexity – A Uniform Approach. Texts in Theoretical Computer Science. Springer, Heidelberg (1999)
23. Wotschke, D.: The Boolean closure of the deterministic and nondeterministic context-free languages. In: GI 1973. LNCS, vol. 1, pp. 113–121. Springer, Heidelberg (1973)
24. Wrathall, C.: Rudimentary predicates and relative computation. SIAM Journal on Computing 7(2), 194–209 (1978)

Pure Pointer Programs with Iteration

Martin Hofmann and Ulrich Schöpp

Ludwig-Maximilians-Universität München
D-80538 Munich, Germany

Abstract. Many LOGSPACE algorithms are naturally described as programs that operate on a structured input (e.g. a graph), that store in memory only a constant number of pointers (e.g. to graph nodes) and that do not use pointer arithmetic. Such "pure pointer algorithms" thus are a useful abstraction for studying the nature of LOGSPACE-computation.

In this paper we introduce a formal class PURPLE of pure pointer programs and study them on locally ordered graphs. Existing classes of pointer algorithms, such as Jumping Automata on Graphs (JAGs) or Deterministic Transitive Closure (DTC) logic, often exclude simple programs. PURPLE subsumes these classes and allows for a natural representation of many graph algorithms that access the input graph by a constant number of pure pointers. It does so by providing a primitive for iterating an algorithm over all nodes of the input graph in an unspecified order.

Since pointers are given as an abstract data type rather than as binary digits we expect that logarithmic-size worktapes cannot be encoded using pointers as is done, e.g. in totally-ordered DTC logic. We show that this is indeed the case by proving that the property "the number of nodes is a power of two," which is in LOGSPACE, is not representable in PURPLE.

1 Introduction

One of the central open questions in theoretical computer science is whether LOGSPACE equals PTIME and more broadly an estimation of the power of LOGSPACE computation.

While these questions remain as yet inaccessible, one may hope to get some useful insights by studying the expressive power of programming models or logics that are motivated by LOGSPACE but are idealised and thus inherently weaker. Examples of such formalisms that have been proposed in the literature are *Jumping Automata on Graphs* (JAGs) [2] and *Deterministic Transitive Closure (*DTC*) logic* [3]. Both are based on the popular intuition that a LOGSPACE computation on some structure, e.g. a graph, is one that stores only a constant number of graph nodes. Many usual LOGSPACE algorithms obey this intuition and are representable in those formalisms. Interestingly, there are also natural LOGSPACE algorithms that do not fall into this category. Reingold's algorithm for st-connectivity in undirected graphs [13], for example, uses not only a constant number of graph nodes, but also a logarithmic number of boolean variables, which are used to exhaustively search the neighbourhood of nodes up to a logarithmic depth.

Indeed, Cook & Rackoff [2] show that st-connectivity is not computable with JAGs. On the other hand, JAGs cannot compute some other problems either, for which there

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 79–93, 2008.
© Springer-Verlag Berlin Heidelberg 2008

does exist an algorithm obeying the above intuition, such as a test for acyclicity in graphs. Thus, important though this result is, we cannot see it evidence that "constant number of graph nodes" is strictly weaker than LOGSPACE.

Likewise, DTC logic without a total order on the graph nodes is fairly weak and brittle [8,5], being unable to express properties such as that the input graph has an even number of nodes.

By assuming a total order on the input structure these deficiencies are removed, but DTC logic becomes as strong as LOGSPACE, because the total order can be used to simulate logarithmically sized work tapes [3].

We thus find that neither JAGs nor DTC logic adequately formalise the intuitive concept of "using a constant number of graph variables only." In this paper we introduce a formalism that fills this gap. Our main technical result shows that full LOGSPACE does not enter through the backdoor by some encoding, as is the case if one assumes a total order on the input structure.

One reader remarked that it was known that LOGSPACE is more than "constant number of pointers", but up until the present contribution there was no way of even rigorously formulating such a claim because the existing formalisms are either artificially weak or acquire an artificial strength by using the total order in a "cheating" kind of way.

To give some intuition for the formalism introduced in this paper, let us recall that a JAG is a finite automaton accepting a locally ordered graph (the latter means that the edges emanating from any node are uniquely identified by numbers from 1 to the degree of that node). In addition to its finite state a JAG has a finite (and fixed) number of pebbles that may be placed on graph nodes and may be moved along edges according to the state of the automaton. The automaton can check whether two pebbles lie on the same node and obtains its input in this way. Formally, one may say that the input alphabet of a JAG is the set of equivalence relations on the set of pebbles.

JAGs cannot visit all nodes of the input graphs and therefore are incapable of evaluating DTC formulas with quantifiers. To give a concrete example, the property whether a graph contains a node with a self-loop is not computable with JAGs.

Rather than as an automaton, we may understand a JAG as a `while`-program whose variables are partitioned into two types: boolean variables and graph variables. For boolean variables the usual operations are available, whereas for graph variables one only has equality test and, for each i, a successor operation to move a graph variable along the i-th edge.

Our proposal, which we call PURPLE for "PURe Pointer LanguagE", consists of adding to this programming language theoretic version of JAGs a `forall`-loop construct ($\texttt{forall}\ x\ \texttt{do}\ P$) whose meaning is to set the graph variable x successively to all graph nodes in some arbitrary order and to evaluate the loop body P after each such setting. The important point is that the order is arbitrary and will in general be different each time a `forall`-loop is evaluated. A program computes a function or predicate only if it gives the same (and correct) result for all such orderings.

The `forall`-loop in PURPLE can be used to evaluate first-order quantifiers and thus to encode DTC logic on locally ordered graphs. Moreover, PURPLE is strictly more expressive than that logic. The following PURPLE-program checks whether the input

graph has an even number of nodes: $(b := \mathrm{true};\ \texttt{forall}\ x\ \texttt{do}\ b := \neg b)$. It is known that this property is not expressible in locally-ordered DTC logic, which establishes strictness of the inclusion.

Beside the introduction of PURPLE, the main technical contribution of this paper is a proof that PURPLE is not as powerful as all of logarithmic space and that thus in particular one cannot use the `forall`-loop to somehow simulate counting, as one can in totally ordered DTC logic [3]. We do this by showing that the property "the number of nodes is a power of two" is not computable in PURPLE. We believe that st-connectivity in undirected graphs is not computable in PURPLE either.

In order to justify the naturality of PURPLE we can invoke, besides the fact that formulas of locally-ordered DTC logic may be easily evaluated, the fact that iterations over elements of a data structure in an unspecified order are a common programming pattern, being made available e.g. in the Java library for the representation of sets as trees or hash maps. The Java API for the `iterator` method in the interface `Set` or its implementation `HashSet` says that "the elements are returned in no particular order".

Efficient implementations of such data structures, e.g. as splay trees, will use a different order of iteration even if the contents of the data structure are the same. Thus, a client program should not depend upon the order of iteration. A spin-off of PURPLE is a rigorous formalisation of this independence.

This research was supported by the DFG project *programming language aspects of sublinear space complexity* (ProPlatz).

2 Pointer Structures

We define the class of structures that serve as input to pure pointer programs.

Definition 1. *Let $L = \{l_1, \ldots, l_n\}$ be a finite set of operation labels. A* pointer structure *on L, an L-model for short, is a set U with n unary functions $l_1, \ldots, l_n \colon U \to U$.*

An L-model can be viewed as the current state of a program with pointers pointing uniformly to records whose fields are labelled $\{l_1, \ldots, l_n\}$. For example, if L is $\{car, cdr\}$ then an L-model is a heap layout of a LISP-machine. We show in the next section how various kinds of graphs can be represented as L-models.

A *homomorphism* $\sigma \colon U \to U'$ between L-models U and U' is a function $\sigma \colon U \to U'$ such that $l(\sigma(x)) = \sigma(l(x))$ holds for all $l \in L$ and $x \in U$. A bijective homomorphism is called an *isomorphism*.

3 Pure Pointer Programs

Pure pointer programs take L-models as input. Unlike general programs with pointers they are not permitted to modify the input, but only to inspect it using a constant number of variables holding elements of U. In addition, pure pointer programs have a finite local state represented by a constant number of boolean variables. The pointer language PURPLE is parameterised by a finite set of *operation labels* $L = \{l_1, \ldots, l_n\}$.

Terms. There are two types of terms, one for boolean values and one for pointers into the universe U. Fix countably infinite sets Vars and Vars_B of pointer variables and

boolean variables. We make the convention that x, y denote pointer variables and b, c denote boolean variables. The terms are then generated by the grammars below.

$$t^B ::= \text{true} \mid \text{false} \mid b \mid \neg t^B \mid t^B \wedge t^B \mid t^B \vee t^B \mid t^U = t^U$$
$$t^U ::= x \mid t^U.l_1 \mid \cdots \mid t^U.l_n$$

The intention is that pointer terms denote elements of the universe U of an L-model and that the term $x.l$ denotes the result of applying the unary operation l in this model to x. The only direct observation about pointers is the equality test $t = t'$.

Programs. The set of PURPLE programs is defined by the following grammar.

$$P ::= \texttt{skip} \mid P; P \mid x := t^U \mid b := t^B \mid \texttt{if}\ t^B\ \texttt{then}\ P\ \texttt{else}\ P \\ \mid \texttt{while}\ t^B\ \texttt{do}\ P \mid \texttt{forall}\ x\ \texttt{do}\ P$$

We abbreviate $(\texttt{if}\ b\ \texttt{then}\ P\ \texttt{else}\ \texttt{skip})$ by $(\texttt{if}\ b\ \texttt{then}\ P)$. The intended behaviour of $(\texttt{forall}\ x\ \texttt{do}\ P)$ is that the pointer variable x iterates over U in some unspecified order, visiting each element exactly once, and P is executed after each setting of x.

On certain classes of L-models the power of PURPLE coincides with LOGSPACE. This is in particular the case if one of the functions l_i is the successor function induced by a total ordering on U. In general, however, PURPLE fails to capture all of LOGSPACE, as we show in Sect. 5. Since we are interested mainly in pointer programs on locally ordered graphs, let us now discuss different possible choices of operation labels for working with locally ordered graphs.

Graphs of constant degree. Pointer algorithms on locally ordered graphs of some fixed out-degree d are most easily represented as an L-model with L being $\{succ_1, \ldots, succ_d\}$ and U being the set of nodes in G. We write $\mathcal{A}_d(G)$ for this L-model.

Graphs of unbounded degree. For graphs of unbounded degree, it is more suitable to use pointer programs with three labels $succ$, $next$ and $prec$. Each locally ordered graph G determines a model $\mathcal{A}(G)$ of these labels. The universe of $\mathcal{A}(G)$ is the set $U = \{\langle v, i\rangle \mid v \in V,\ 0 \le i \le deg(v)\}$, where V denotes the node set of G. A pair $\langle v, i\rangle \in U$ with $i > 0$ represents an outgoing edge from v. Such pairs are often called *darts*, especially in the case of undirected graphs, where each undirected edge is represented by two darts, one for each direction in which the edge can be traversed. We include objects of the form $\langle v, 0\rangle$ to model the nodes themselves; thus the universe consists of the disjoint union of the nodes and the darts. The operation labels are interpreted by

$$succ(v, i) = \begin{cases} \langle succ_i(v), 0\rangle & \text{if } i \ge 1, \\ \langle v, 0\rangle & \text{if } i = 0, \end{cases}$$
$$next(v, i) = \langle v, \min(i + 1, deg(v))\rangle,$$
$$prec(v, i) = \langle v, \max(i - 1, 0)\rangle.$$

Using $next$ and $prec$ one can iterate over the darts on a node and using $succ$ one can follow the edge identified by a dart.

The presentation of graphs by darts and operations on them is commonly used in the description of LOGSPACE-algorithms [1,13,10], but it is also prevalent in other contexts, see [7] and the discussion there.

We note that with the dart representation, the `forall`-loop iterates over all darts and not the nodes of the graph. Iteration over all nodes can nevertheless be implemented easily. Since a program can recognise if x is a dart of the form $\langle v, 0\rangle$ by testing whether $x = x.prec$ is true, one can implement a `forall`-loop that visits only darts of the form $\langle v, 0\rangle$ and this amounts to iteration over all nodes.

While we have now introduced two representations for graphs of bounded degree, it is not hard to see that it does not matter which representation we use, as each program with labels $succ_1, \ldots, succ_d$ can be translated into a program with labels $succ, prec, next$ that recognises the same graphs, and vice versa.

3.1 Examples

We give two examples to illustrate the use of PURPLE. The first simple example program decides the property that all nodes have even in-degree. First we define a program $E(x, y, b)$ with variables x, y and b, such that after evaluation of $E(x, y, b)$ the boolean variable b is true if and only if there is an edge from the node given by x to that given by y. Such a program may be defined as:

$$
\begin{array}{l}
b := \mathrm{false};\ x' := x \\
\texttt{while}\ \neg(x' = x'.next)\ \texttt{do}\ x' := x'.next; \\
\texttt{while}\ \neg(x' = x'.prec)\ \texttt{do}\ (b := b \vee (y = x'.succ);\ x' := x'.prec)
\end{array}
$$

Herein, x' should be chosen afresh for each occurrence of $E(x, y, b)$ within a larger program. The following program then computes if the in-degree of all nodes is even.

$$
\begin{array}{l}
even := \mathrm{true}; \\
\texttt{forall}\ x\ \texttt{do} \\
\quad c := \mathrm{true}; \\
\quad \texttt{forall}\ y\ \texttt{do}\ (\texttt{if}\ y = y.prec\ \texttt{then}\ (E(y, x, b);\ \texttt{if}\ b\ \texttt{then}\ c := \neg c)); \\
\quad even := even \wedge c
\end{array}
$$

In the inner `forall`-loop we have a test for $(y = y.prec)$, so that the body of this loop is executed once for each graph node, rather than for each dart. In this way, the inner `forall`-loop is used to iterate over all nodes that have an edge to x. In the body we then make the assignment $c := \neg c$ for all nodes y that have an edge to x.

For a second, more substantial, example we show that acyclicity of undirected graphs can be decided in PURPLE, which is a well-known LOGSPACE-complete problem [1]. We next describe the LOGSPACE-algorithm of Cook & McKenzie [1] and then show that it can be written directly in PURPLE. The fact that PURPLE can express a LOGSPACE-complete problem does not conflict with PURPLE being strictly weaker than LOGSPACE, since not all reductions can be expressed in PURPLE.

Let G be an undirected locally ordered graph with node set V and let U be the universe of $\mathcal{A}(G)$. Let σ be the permutation on U such that $\sigma(v, 0) = \langle v, 0\rangle$ holds and such that $\sigma(v, i) = \langle w, j\rangle$ implies both $succ(v, i) = w$ and $succ(w, j) = v$. Note that in an undirected graph each edge between v and w is given by two half-edges, one from v to w and one from w to v. Thus, if the dart $\langle v, i\rangle$ represents one half of

an edge in G, then $\sigma(v,i)$ represents the other half of the same edge. Let π be the permutation on U satisfying $\pi(v,0) = \langle v,0\rangle$, $\pi(v,i) = \langle v, i+1\rangle$ for $1 \leq i < deg(v)$ and $\pi(v, deg(v)) = \langle v, \min(1, deg(v)\rangle$.

Consider now the composite permutation $\pi \circ \sigma$ on U. It implements a way of exploring the graph G in depth-first-search order. That is, the walks in depth-first-search order are the obtained as the sequences of darts $x_0, x_1, \ldots$ defined by $x_{i+1} = (\pi \circ \sigma)(x_i)$. Since these sequences are generated by the composite permutation $\pi \circ \sigma$, it is easy to see that they can be generated in logarithmic space.

Being able to construct walks in depth-first-search order, one can use the following characterisation of acyclicity of undirected graphs to decide this property in LOGSPACE. An undirected graph is acyclic, if it does not have self-loops and if, for any node v and any integer i, the walk that starts by taking the i-th edge from v and proceeds in depth-first-search order does not visit v again until it traverses the i-th edge from v in the opposite direction. This is formulated precisely in the following lemma, a proof of which can be found in [1].

Lemma 2. *The undirected graph G is acyclic if and only if the following property holds for all $x_0 \in \{\langle v,i\rangle \mid v \in V,\ 1 \leq i \leq deg(v)\}$: If the walk $x_0, x_1, \ldots$ is defined by $x_{i+1} = (\pi \circ \sigma)(x_i)$ and $k > 0$ is the least number such that x_0 and x_k are darts on the same node, then both $k > 1$ and $\sigma(x_{k-1}) = x_0$ hold.*

It now only remains to show that the property in this lemma can be decided in PURPLE. A program $P_{\pi\circ\sigma}(x)$ implementing the permutation $\pi \circ \sigma$ can be written easily, since the `forall`-loop allows one to iterate over all the neighbours of any given node. Moreover, it is easy to write a program $P_{=}(x, y, b)$ that sets the boolean variable b to true if x and y are darts on the same node and to false otherwise. With these programs, the property of the above lemma can be decided in PURPLE as follows:

$acyclic := \text{true}$
`forall` x `do`
  `if` $\neg(x = x.prec)$ `then`
    $\mathit{keq0} := \text{true};\ \mathit{kleq1} := \text{true};\ x_0 := x;\ \mathit{returned} := \text{false};$
    `while` $\neg \mathit{returned}$ `do`
      $(\mathit{kleq1} := \mathit{keq0};\ \mathit{keq0} := \text{false};\ x' := x;\ P_{\pi\circ\sigma}(x);\ P_{=}(x_0, x, \mathit{returned}));$
    $P_\sigma(x');\ \mathit{acyclic} := \mathit{acyclic} \wedge \neg \mathit{kleq1} \wedge (x' = x)$

3.2 Operational Semantics

PURPLE is defined with the intention that an input must be accepted or rejected regardless of the order in which the `forall`-loops are run through. In this section we give the operational semantics of PURPLE, thus making this intention precise.

The operational semantics of PURPLE with operation labels in L is parameterised by an L-model $\mathcal{A} = (U, \boldsymbol{l})$ and is formulated in terms of a small-step transition relation $\longrightarrow_{\mathcal{A}}$. To define this transition relation, we define a set of *extended programs* that have annotations for keeping track of which variables have already been visited in the

Assignment

$$\langle x := t^U, q, \rho\rangle \longrightarrow_{\mathcal{A}} \langle \texttt{skip}, q, \rho[x \mapsto [\![t^U]\!]_{q,\rho}]\rangle$$
$$\langle b := t^B, q, \rho\rangle \longrightarrow_{\mathcal{A}} \langle \texttt{skip}, q[b \mapsto [\![t^B]\!]_{q,\rho}], \rho\rangle$$

Composition

$$\langle \texttt{skip}; P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P,\ q,\ \rho\rangle \qquad \frac{\langle P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P',\ q',\ \rho'\rangle}{\langle P; Q,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P'; Q,\ q',\ \rho'\rangle}$$

Conditional

$$\langle \texttt{if}\ t\ \texttt{then}\ P\ \texttt{else}\ Q,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P,\ q,\ \rho\rangle \text{ if } [\![t]\!]_{e,\rho} = \text{true}$$
$$\langle \texttt{if}\ t\ \texttt{then}\ P\ \texttt{else}\ Q,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle Q,\ q,\ \rho\rangle \text{ if } [\![t]\!]_{e,\rho} = \text{false}$$

`while`-loop

$$\langle \texttt{while}\ t\ \texttt{do}\ P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle \texttt{skip},\ q,\ \rho\rangle \text{ if } [\![t]\!]_{q,\rho} = \text{false}$$
$$\langle \texttt{while}\ t\ \texttt{do}\ P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P;\ \texttt{while}\ t\ \texttt{do}\ P,\ q,\ \rho\rangle \text{ if } [\![t]\!]_{q,\rho} = \text{true}$$

`for`-loop

$$\langle \texttt{for}\ x \in \emptyset\ \texttt{do}\ P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle \texttt{skip},\ q,\ \rho\rangle$$
$$\langle \texttt{for}\ x \in W\ \texttt{do}\ P,\ q,\ \rho\rangle \longrightarrow_{\mathcal{A}} \langle P;\ \texttt{for}\ x \in W \setminus \{v\}\ \texttt{do}\ P,\ q,\ \rho[x \mapsto v]\rangle \text{ for any } v \in W$$

Fig. 1. Operational Semantics

computation of the `forall`-loops. The set of extended programs consists of PURPLE-programs in which the `forall`-loops are not restricted to an iteration over the whole universe U, but where ($\texttt{for}\ x \in W\ \texttt{do}\ P$) is allowed for any subset W of U. We identify ($\texttt{forall}\ x\ \texttt{do}\ P$) with ($\texttt{for}\ x \in U\ \texttt{do}\ P$).

The transition relation $\longrightarrow_{\mathcal{A}}$ is a binary relation on configurations. A *configuration* is a triple $\langle P, q, \rho\rangle$, where P is an extended program, $q\colon Vars_B \to 2$ is an assignment of boolean variables and $\rho\colon Vars \to U$ is an assignment of pointer variables. The inference rules defining $\longrightarrow_{\mathcal{A}}$ appear in Fig. 1. In this figure, we denote by $[\![t]\!]_{q,\rho}$ the evident interpretations of terms with respect to the variable assignments q and ρ. The operational semantics is standard for all but the `for`-loop. We note, in particular, that the rules for the `for`-loop make the transition system non-deterministic.

We say that a program P is *strongly terminating* if for all $\mathcal{A}$ the computation of P on $\mathcal{A}$ always terminates, i.e. for all q and ρ there is no infinite reduction sequence of $\longrightarrow_{\mathcal{A}}$ starting from $\langle P,\ q,\ \rho\rangle$ and in particular there are ρ' and q' such that $\langle P,\ q,\ \rho\rangle \longrightarrow^*_{\mathcal{A}} \langle \texttt{skip}, q',\ \rho'\rangle$ holds.

To define what it means for an L-model to be recognised by a program, we choose a distinguished boolean variable *result* that indicates the outcome of a computation.

Definition 3 (Recognition). *A set X of L-models is* recognised *by a program P, if P is strongly terminating and, for all L-models $\mathcal{A}$ and all ρ, ρ', q and q' satisfying $\langle P,\ q,\ \rho\rangle \longrightarrow^*_{\mathcal{A}} \langle \texttt{skip}, q',\ \rho'\rangle$, one has $q'(result) = \text{true}$ if and only if $\mathcal{A} \in X$.*

Our notion of recognition should not be confused with the usual definition of acceptance for existentially (nondeterministic) or universally branching Turing machines; in contrast to those concepts it is completely symmetrical in X vs. $\overline{X}$. If the input is in X then all runs must accept; if the input is not in X then all runs must reject. In particular, not even for strongly terminating P can we *in general* define "the language of P". A program whose result depends on the traversal order does not recognise any set at all.

3.3 Basic Properties

In contrast to formalisms that depend on a global ordering, PURPLE is closed under isomorphism. This is formulated by the following lemma, in which we write σP for the program obtained from P by replacing each occurrence of $(\texttt{for}\ x \in W\ \texttt{do}\ P)$ by $(\texttt{for}\ x \in \sigma W\ \texttt{do}\ P)$. Note that if P is a PURPLE-program proper, i.e., not an extended one, then $\sigma P = P$ holds. Its proof is a straightforward induction.

Lemma 4. *Let $\sigma\colon U \to V$ be an isomorphism of L-models. Then $\langle P, q, \rho\rangle \longrightarrow_U \langle P', q', \rho'\rangle$ implies $\langle \sigma P, q, \sigma \circ \rho\rangle \longrightarrow_V \langle \sigma P', q', \sigma \circ \rho'\rangle$.*

The straightforward proof of the following lemma is based on the fact that the number of global configurations is polynomial in the input size.

Lemma 5. *For any program P with labels in L, there exists a* `while`*-free program P' that recognises the same sets of finite L-models.*

4 Related Models of Computation

Based on the intuition that computation with logarithmic space amounts to computation with a constant number of pointers, a number of formalisms of pure pointer algorithms have been proposed as approximations of LOGSPACE.

4.1 Jumping Automata on Graphs

Cook & Rackoff [2] introduce *Jumping Automata on Graphs* (JAGs) in order to study space lower bounds for reachability problems on directed and undirected graphs. Jumping automata on graphs are a model of pure pointer algorithms on locally ordered graphs. Each JAG may be described as a `forall`-free PURPLE-program over the operation labels $succ_1$, $succ_2$, ... and vice versa. Therefore, a JAG may move on the graph only by traversing edges and by jumping one graph variable to the position of another variable. As a result, JAGs can only compute local properties of the input graph. If, for instance, all the graph variables are in some connected component of the input graph then they will remain in it throughout the whole computation.

Cook & Rackoff show that it is possible to prove upper bounds on the expressivity of JAGs [2]. They show that both on directed and on undirected graphs reachability cannot be solved by them. Together with the local character of JAG computations, this can be used to show that many natural LOGSPACE-properties of graphs cannot be computed by JAGs. For instance, JAGs cannot compute the parity function and they cannot decide whether or not the input graph is acyclic.

While PURPLE is more expressive than JAGs, we hope that nevertheless separation results along the lines of the existing ones for JAGs, e.g. [2,4], could be achievable for PURPLE by further elaborating the pumping techniques used to establish those.

One criticism of Jumping Automata on Graphs as a computation model is that JAGs are artificially weak on directed graphs. Since, with the operations $succ_1$, $succ_2$, ..., edges can only be traversed in the forward direction, there is no way for a JAG to reach a node that has only outgoing edges, for example. One solution to this problem is to work with graphs having a local ordering both on the outgoing and on the incoming edges of each node, so that edges can be traversed in both directions [5]. The `forall`-loop of PURPLE represents another possible solution, since we can use it to iterate over all the nodes that have an edge to a given node, as we have shown in Sect. 3 above.

4.2 Deterministic Transitive Closure Logic

In the context of descriptive complexity theory DTC-logic was introduced as a logical characterisation of LOGSPACE on ordered structures [9]. The formulae of this logic are built from the connectives of first-order logic and a connective DTC for deterministic transitive closure. The formula $\mathrm{DTC}[\varphi(\boldsymbol{x},\boldsymbol{y})](\boldsymbol{s},\boldsymbol{t})$ expresses that, for all variable assignments ν, the pair $([\![\boldsymbol{s}]\!]_\nu, [\![\boldsymbol{t}]\!]_\nu)$ is in the transitive closure of the relation $\{(\boldsymbol{u},\boldsymbol{v}) \mid \mathcal{A} \models_\nu \varphi[\boldsymbol{u},\boldsymbol{v}] \wedge \forall \boldsymbol{z}.\, \varphi[\boldsymbol{u},\boldsymbol{z}] \Rightarrow \boldsymbol{z} = \boldsymbol{v}\}$, see e.g. [9].

While on structures with a totally ordered universe DTC-logic captures all of LOGSPACE, it is strictly weaker on unordered structures. A typical example of a property that cannot be expressed without an ordering is whether or not the universe has an even number of elements. If graphs are represented without any ordering by an edge relation $E(-,-)$, then DTC logic on graphs is very weak indeed. Grädel & McColm [8] have shown that there exist families of graphs on which DTC without any ordering is no more expressive than first-order logic.

Unordered DTC logic is nevertheless interesting, since on locally ordered graphs it captures an interesting class of pure pointer algorithms. Locally ordered graphs may be used in the logic by allowing, in addition to the binary edge relation $E(-,-)$, a ternary relation $F(-,-,-)$, such that $F(v,-,-)$ is a total ordering on $\{w \mid E(v,w)\}$ [5], for any v. With such a graph representation, DTC can encode JAGs and it is strictly more expressive, since it allows first-order quantification [5]. With suitable restrictions on the formulae, it is furthermore possible to characterise smaller classes of pointer algorithms on locally ordered graphs, such as the class given by Tree Walking Automata [11].

We next observe that PURPLE subsumes DTC logic on locally ordered graphs.

Proposition 6. *For each closed* DTC *formula φ for locally ordered graphs there exists a program P_φ such that, for any finite locally ordered graph G, $G \models \varphi$ holds if and only if P_φ recognises $\mathcal{A}(G)$.*

First-order quantifiers can be evaluated directly using the `forall`-loop. To see that $\mathrm{DTC}[\varphi(\boldsymbol{x},\boldsymbol{y})](\boldsymbol{a},\boldsymbol{b})$ can be evaluated, note that using the `forall`-loop we can iterate over all tuples $\boldsymbol{y}$, so that we can compute the unique $\boldsymbol{y}$ such that $\varphi(\boldsymbol{x},\boldsymbol{y})$ holds, if such a unique $\boldsymbol{y}$ exists, and we can recognise when such a unique $\boldsymbol{y}$ does not exist.

The converse of this proposition is not true, of course, since there is no DTC-formula that expresses that the input graph has an even number of nodes [5].

Although DTC-logic on locally ordered graphs is interesting, there are still many open questions regarding its expressive power. As far as we know, it is not known if DTC-logic on locally ordered graphs can express directed or undirected reachability. The best result we know is that of Etessami & Immerman [5], who show that undirected reachability cannot be expressed by a formula of the form $\mathrm{DTC}[\varphi(\boldsymbol{x},\boldsymbol{y})](\boldsymbol{s},\boldsymbol{t})$, where φ is a first-order formula (without a total ordering not every formula can be expressed in this way).

One reason for the lack of results on the expressive power of DTC on locally ordered graphs may be that at present there are no simple Ehrenfeucht-Fraïssé games for it; and such games are the main tool for proving inexpressivity results in finite model theory. Most of the existing results have been proved either directly or by reduction to a proof that uses automata-theoretic techniques. Etessami & Immerman, for example, obtain their inexpressivity result for undirected reachability by reduction to the corresponding result of Cook & Rackoff for JAGs. The relative success of automata-theoretic methods is part of the motivation for studying the programming language PURPLE.

Furthermore, when viewed as a model of pointer algorithms, DTC logic is somewhat unnaturally restricted. To implement universal quantification, say on a LOGSPACE Turing Machine, one needs to have a form of iteration over all possible pointers. If it is possible to iterate over all pointers, then it should also be possible to write a program for the parity function, even without any knowledge about a total ordering of the pointers. But this cannot be done in DTC. If we view the universal quantifier as a form of iteration that works without a total ordering, then it is more restricted than it needs to be.

The problem that a logic cannot express counting properties such as parity is often addressed in the literature by extending the logic with a totally ordered universe of numbers (of the same size as the first universe) and perhaps also counting quantifiers, see e.g. [5,9]. Such an addition appears to be quite a jump in expressivity. For instance, in view of Reingold's algorithm for undirected reachability, it is likely that undirected reachability becomes expressible in such a logic [Ganzow & Grädel, personal communication]. However, we believe that this problem is not expressible in PURPLE and in view of the Prop. 6 also not in DTC.

Another option of increasing the expressive power of DTC to include functions such as parity is to consider order-independent queries [9]. An order-independent query is a DTC formula that has access to a total ordering on the universe, but whose value does not depend on which particular ordering is chosen. Superficially, there appears to be a similarity to the `forall`-loop in PURPLE. However, order-independent queries are strictly stronger than PURPLE. They correspond to the version of PURPLE, in which each program is guaranteed that all `forall`-loops iterate over the universe in the same order, even though this order may be different from run to run. Of course, in either system (PURPLE with fixed traversal order and order-independent queries) one can use the order to capture all of LOGSPACE.

5 Counting

Our goal in this section is to show that the behaviour of an arbitrary program on the discrete graph with n nodes can be described abstractly and independently of n. From

this it will follow that PURPLE-programs are unable to detect whether n has certain arithmetic properties such as being a power of two.

Write G_n for the discrete graph with n nodes and write $V_n = \{1, \ldots, n\}$ for its set of nodes. Since this graph has constant degree 0, the L-model with universe V_n induced by it does not have any operations.

Fix a finite set M of graph variables. We show that no program with graph variables in M can recognise the set of all graphs G_n where n is a power of two. Since M is arbitrary, this will be enough to show the result for all PURPLE-programs.

The proof idea is to show that whether or not a (while-free) program P accepts G_n for sufficiently large n depends only on the initial value of boolean variables, the initial incidence relation of the pointer variables and the remainder modulo l of the graph size n for some l. In order to prove this by induction on programs we associate to each program P an abstraction $[\![P]\!]$, which given the initial valuation of the boolean variables q_0, the initial incidence relation E_0 and the size n modulo l yields a triple $(q, E, f) = [\![P]\!](q_0, E_0, n \bmod l)$ that characterises the final configuration of a computation as follows: q is the final valuation of the boolean variables, E is the final incidence relation of pointer variables, and $f\colon M \to M + \{\mathit{fresh}\}$ is a function that tells for each variable x whether it moves to a "fresh" node, i.e. one that was not occupied at the start of the computation, or assumes the position that some other variable $f(x) \in M$ had in the initial configuration. The exact position of the "fresh" variables will of course depend on the order in which `forall`-loops are being worked off. In fact, we will show that with an appropriate choice of ordering any position of the "fresh" variables can be realised, so long as it respects E.

For example, the abstraction of the program $(z := x;\ \texttt{forall}\ x\ \texttt{do}\ y := x)$ would map (q_0, E_0, l) to (q_0, E, f), where E is the equivalence relation generated by (x, y), and f is given by $f(x) = f(y) = \mathit{fresh}$ and $f(z) = x$. This means that for any n large enough, the program has a run on G_n that ends in a state where z assumes the position of x in the start configuration and where x and y lie on a node not occupied in the start configuration. Moreover, E specifies that x and y must lie on the same node.

Notice that the abstraction characterises the result of *some* run of the program. In the example, there also exists a run in which the last node offered by the `forall`-loop happens to be the (old) value of x, in which case x, y, z are all equal. The purpose of the abstraction is to show that certain sets cannot be recognised. Since for a set to be recognised the result must be the same (and correct) for all runs it suffices to exhibit (using the abstraction) a single run that yields a wrong result. This existential nature of the abstraction is made more precise in Def. 10 and Lemma 11.

Definition 7. *Let Σ denote the set of equivalence relations on M. For each environment ρ we write $[\rho] \in \Sigma$ for the equivalence relation given by $x[\rho]y \iff \rho(x) = \rho(y)$. Since the meaning of a boolean term t depends only on the induced equivalence relation, we define $[\![t^B]\!]_{q,E}$ as $[\![t^B]\!]_{q,\rho}$ for any ρ with $[\rho] = E \in \Sigma$.*

Definition 8. *The set F of* moves *is given by $F := M \to M + \{\mathit{fresh}\}$.*

The intention of a *move* $f \in F$ is that if $f(x) = y \neq \mathit{fresh}$ holds then variable x is set to the (old value of) variable y and if $f(x) = \mathit{fresh}$ holds then x is moved to a fresh location. This is formalised by the next definition.

Definition 9. *Let $\rho, \rho' \colon Vars \to V_n$ for some n and let $E' \in \Sigma$ and $f \in F$. We say that ρ' is* compatible with (E', ρ, f) *if $[\rho'] = E'$ holds and for all $x \in M$ we have*

- $f(x) = y \neq \mathit{fresh}$ *implies* $\rho'(x) = \rho(y)$*; and*
- $f(x) = \mathit{fresh}$ *implies* $\rho'(x) \notin \mathrm{im}(\rho)$.

In the rest of this section, we write Q for the set $Vars_B \to 2$ of boolean states.

Definition 10. *Let P be a program, $k, l > 0$ and*

$$B : Q \times \Sigma \times \mathbb{Z}/l\mathbb{Z} \longrightarrow Q \times \Sigma \times F$$

be a function. Say that (B, k, l) represents the behaviour of P on discrete graphs *if for all $n > k$, $q \in Q$ and $\rho \colon Vars \to V_n$ there exists $\rho' \colon Vars \to V_n$ with*

$$\langle P, q, \rho\rangle \longrightarrow^*_{G_n} \langle \mathtt{skip}, q', \rho'\rangle,$$

such that ρ' is compatible with (E', ρ, f) whenever $B(q, [\rho], n \bmod l) = (q', E', f)$.

Notice that in contrast to the definition of recognition we only require that *for some* evaluation of $\langle P, q, \rho\rangle$ the predicted behaviour is matched. This is appropriate because the intended use of this concept is negative: in order to show that no program can recognise a certain class of discrete graphs we should exhibit for each program a run that defies the purported behaviour. Of course, this also helps in the subsequent proofs since it is then us who can control the order of iteration through `forall`-loops.

Lemma 11. *Suppose (B, k, l) represents the behaviour of P on discrete graphs. Then whenever $n \geq k$ and $q \in Q$ and $\rho \colon Vars \to V_n$ and $B(q, [\rho], n \bmod l) = (q', E', f)$ then $\langle P, q, \rho\rangle \longrightarrow^*_{G_n} \langle \mathtt{skip}, q', \rho_1\rangle$ holds for all ρ_1 compatible with (E', ρ, f).*

Proof. Choose ρ' compatible with (E', ρ, f) that satisfies $\langle P, q, \rho\rangle \longrightarrow^*_{G_n} \langle \mathtt{skip}, q', \rho'\rangle$. Such ρ' must exist by the definition of "represents." We have $\rho'(x) = \rho(f(x)) = \rho_1(x)$ whenever $f(x) = y \neq \mathit{fresh}$ holds and $\rho'(x), \rho_1(x) \notin \mathrm{im}(\rho)$ whenever $f(x) = \mathit{fresh}$ holds. Hence we can find an automorphism $\sigma : G_n \to G_n$ satisfying $\sigma \circ \rho = \rho$ and $\sigma \circ \rho' = \rho_1$. The claim then follows from Lemma 4. □

Theorem 12. *There exist numbers k, l (depending on the number of variables in M) such that each* `while`*-free program P with graph variables in M is represented by $(\llbracket P\rrbracket, k, l)$ for some function $\llbracket P\rrbracket$.*

Proof. Put $N = |Q| \cdot |\Sigma| \cdot 2^{|M|\cdot|M|}$ and $k = 3|M| + N$ and $l = N!$.

We prove the claim by induction on P. For basic programs the statement is obvious.

Case $P = P_1; P_2$**.** Suppose we are given (q, E, t) where $t \in \mathbb{Z}/l\mathbb{Z}$. Write $\llbracket P_1\rrbracket(q, E, t) = (q_1, E_1, f_1)$ as well as $\llbracket P_2\rrbracket(q_1, E_1, t) = (q_2, E_2, f_2)$. Define $f \in F$ by

$$f(x) = f_1(f_2(x)), \text{ if } f_2(x) \in M;$$
$$f(x) = \mathit{fresh}, \text{ if } f_2(x) = \mathit{fresh}$$

Put $\llbracket P\rrbracket(q, E, t) = (q_2, E_2, f)$. Fix some n with $n \bmod l = t$ and $\rho \colon Vars \to V_n$ with $[\rho] = E$ and, using the induction hypothesis, choose ρ_1 compatible with (E_1, ρ, f_1)

and ρ_2 compatible with (E_2, ρ_1, f_2). Invoking Lemma 11 we may assume without loss of generality that $f_2(x) = \mathit{fresh}$ implies $\rho_2(x) \notin \mathrm{im}(\rho)$. We now claim that ρ_2 is compatible with (E_2, ρ, f), which establishes the current case. To see this claim pick $x \in M$ and suppose that $f_2(x) = y$ and $f_1(y) = z \in M$. Then $f(x) = z$ and $\rho_2(x) = \rho_1(y) = \rho(z)$ as required. If $f_2(x) = \mathit{fresh}$ then $f(x) = \mathit{fresh}$ and $\rho_2(x) \notin \mathrm{im}(\rho)$ by assumption on ρ_2. If, finally, $f_2(x) = y \in M$ and $f_1(y) = \mathit{fresh}$ then $\rho_2(x) = \rho_1(y) \notin \mathrm{im}(\rho)$ by compatibility of ρ_1.

Case `if` s^B `then` P_1 `else` P_2**.** Define $[\![P]\!](q, E, t) = [\![P_1]\!](q, E, t)$ when $[\![s]\!]_{q,E} = \mathrm{true}$ and $[\![P]\!](q, E, t) = [\![P_2]\!](q, E, t)$ when $[\![s]\!]_{q,E} = \mathrm{false}$.

Case `forall` x `do` P_1**.** We note that $k \geq 3|M|$ holds. Now, given (q, E, t) choose n minimal with $n \geq k$ and $n \bmod l = t$ and some $\rho\colon \mathit{Vars} \to V_n$ with $[\rho] = E$ and assume w.l.o.g. that $\mathrm{im}(\rho) \subseteq \{1, \ldots, |M|\}$. We then iterate through the graph nodes $\{1, \ldots, n\}$ in ascending order. Fresh nodes are chosen from $\{|M|+1, \ldots, 3|M|\}$. Since there are only $|M|$ variables we have enough space in this interval as to satisfy any request for fresh nodes possibly arising during the evaluation of P_1. Formally, we choose sequences ρ_i and q_i in such a way that

1. $\rho_0 = \rho$, $q_0 = q$;
2. $\langle P_1, q_i, \rho_i[x \mapsto i{+}1]\rangle \longrightarrow^*_{G_n} \langle \mathtt{skip}, q_{i+1}, \rho_{i+1}\rangle$;
3. for all $y \in M$, $\rho_{i+1}(y) \notin \mathrm{im}(\rho_i[x \mapsto i{+}1])$ implies $\rho_{i+1} \in \{|M|+1, \ldots, 3|M|\}$.

That such sequences exist follows from the induction hypothesis and Lemma 11.

For $I \in \Sigma$ define $I^+ = I \setminus x \cup \{(x, x)\}$, where $I \setminus x$ is I with all pairs involving x removed.

Putting $E_i = [\rho_i]$ we then get $[\rho_i[x \mapsto i{+}1]] = E_i^+$ for all $i > 3|M|$ and thus, again for $i > 3|M|$:

$$(q_{i+1}, E_{i+1}, f_{i+1}) = [\![P_1]\!](q_i, E_i^+, t)$$

for some sequence f_i.

Thus, for $i > 3|M|$ the incidence relations E_{i+1} no longer depend on ρ_i itself but only on the previous incidence relation E_i (and the valuation of the boolean variables).

Choose now f such that ρ_n is compatible with (E_n, ρ, f) and define $[\![P]\!](q, E, t) = (q_n, E_n, f)$.

Now we have to show that indeed $([\![P]\!], k, l)$ represents the behaviour of P on discrete graphs. To this end fix $m > n \geq k$ with $m \bmod l = t = n \bmod l$ and $\chi \in \mathit{Vars} \to V_m$ with $[\chi] = E$.

In view of Lemma 4 we may assume $\chi(x) = \rho(x)$ for all $x \in M$ so that in particular $\mathrm{im}(\chi) \subseteq \{1, \ldots, |M|\}$. We can now iterate through G_m in ascending order in exactly the same fashion yielding sequences I_i, χ_i, r_i such that

1. $\chi_0 = \chi$, $r_0 = q$;
2. $\langle P_1, r_i, \chi_i[x \mapsto i{+}1]\rangle \longrightarrow^*_{G_m} \langle \mathtt{skip}, r_{i+1}, \chi_{i+1}\rangle$;
3. $\chi_i(y) = \rho_i(y)$ and $q_i = r_i$ for all $y \in M$ and $i \leq n$.
4. for all $y \in M$, $\chi_{i+1}(y) \notin \mathrm{im}(\chi_i[x \mapsto i{+}1])$ implies $\chi_{i+1} \in \{|M|+1, \ldots, 3|M|\}$.

We put $I_i = [\chi_i]$ and find $(r_{i+1}, I_{i+1}, g_{i+1}) = [\![P_1]\!](r_i, I_i^+, t)$ for some function sequence g_i. Now consider the restriction of χ_i to $\{1, \ldots, |M|\}$, i.e., formally define

$\xi_i = \{(x, \chi_i(x)) \mid x \in M,\ \chi_i(x) \in \{1, \ldots, |M|\}\}$. We note that ξ_{i+1} does not depend upon χ_i but only on the incidence relation I_i (and q_i and t, of course). Indeed,

$$\xi_{i+1} = \{(y, v) \mid g_{i+1}(y) = z, (z, v) \in \xi_i\}.$$

In view of the choice of N there must exist indices $3|M| < t < t' \leq 3|M| + N = k$ such that $r_t = r_{t'}$ and $I_t = I_{t'}$ and $\xi_t = \xi_{t'}$. But then we also have $r_{t+d} = r_{t'+d}$ and $I_{t+d} = I_{t'+d}$ and $\xi_{t+d} = \xi_{t'+d}$ for all $d \geq 0$ with $t' + d \leq m$. Now, since $t' - t$ divides l we find that $r_i = r_{i'}$ and $I_i = I_{i'}$ and $\xi_i = \xi_{i'}$ as soon as $t' \leq k \leq n \leq i < i' \leq m$ and $i \cong i'$ modulo l. Hence in particular, $r_m = r_n = q_n$ and $I_m = I_n = E_n$ and $\xi_m = \xi_n$. Thus,

$$\langle P, \chi, q\rangle \longrightarrow^*_{G_m} \langle \mathtt{skip}, q_n, \chi_m\rangle$$

with χ_m compatible with (E_n, ρ, f_n) as required. □

Corollary 13. *Checking whether the input is a discrete graph with n nodes with n a power of two is possible in deterministic logarithmic space but not in* PURPLE.

Proof. To program this in LOGSPACE count the number of nodes on a work tape in binary and see whether its final content has the form 10^*. Suppose there was a PURPLE-program P recognising this class of graphs in the sense of Def. 3. By Lemma 5 we can assume that P is `while`-free. Then Theorem 12 furnishes $(\llbracket P\rrbracket, k, l)$ representing P on discrete graphs. Let *result* be the boolean variable in P containing the return value. Let n be a power of two such that $n > k$ holds and $n + l$ is not a power of two. Let ρ, q be arbitrary initial values. Now, since P purportedly recognises G_n we must have $\llbracket P\rrbracket(q, [\rho], n \bmod l) = (q', \_, \_)$ with $q'(\mathit{result}) = \mathrm{true}$. Now let χ be a valuation of the variables in G_{n+l} satisfying $[\rho] = [\chi]$. We then get $\langle P, \chi, q\rangle \longrightarrow^*_{G_{n+l}} \langle \mathtt{skip}, q', \_\rangle$, which contradicts the assumption on P, since on all runs of P on G_{n+l} the value of the boolean variable *result* would have to be false. Recall the explanation after Def. 3. □

A reader of an earlier version of this paper suggested an alternative, purportedly simpler route to this result. From Theorem 12 one can conclude that if X is a property of discrete graphs then the set $\{a^n \mid G_n \in X\}$ is a regular set over the unary alphabet $\{a\}$; in fact, every regular set over this alphabet arises in this way. Given that all iterations through n indistinguishable discrete nodes look essentially the same and that the internal control of a PURPLE-program is a finite state machine it should not be able to do anything more than a DFA when run on a unary word.

We cannot see, how to turn this admittedly convincing intuition into a rigorous proof and would like to point out that a PURPLE-program can test for equality of nodes, thus it can store certain nodes from an earlier iteration and then find out when they appear in a subsequent one. Indeed, if the order of traversal were always the same then we could use this feature to define a total ordering on the nodes and thus program all of LOGSPACE including the question whether the cardinality of the universe is a power of two. This would be true even if the traversal order was not fixed a priori but the same for all iterations in a given run of a program. Thus, any proof of Theorem 12 must exploit the fact that an input is recognised or rejected only if this is the case for all possible traversal sequences. Doing so rigorously is what takes up most of the work in our proof.

6 Conclusion

We believe that PURPLE captures a natural class of pure pointer programs within LOGSPACE. By showing that PURPLE is unable to express arithmetic properties, we have demonstrated that it is not merely a reformulation of LOGSPACE but defines a standalone class whose properties are worth of study in view of its motivation from practical programming with pointers.

On the one hand, PURPLE strictly subsumes JAGs and DTC logic and can therefore express many pure pointer algorithms in LOGSPACE. On the other hand, Reingold's algorithm for st-reachability in undirected graphs uses counting registers of logarithmic size. We believe that it is not possible to solve undirected reachability in PURPLE and therefore not in DTC-logic. The details will appear elsewhere.

We also consider it an important contribution of our work to have formalised the notion that the order of iteration through a data structure may not be relied upon. Such provisos often appear in the documentation of library functions like Java's iterators. Our notion of recognition in Def. 3 captures this and, as argued at the end of Sec. 5, it measurably affects the computing strength (otherwise all of LOGSPACE could be computed).

The appearance of "freshness" and the accompanying $\forall\exists$-coincidence expressed in Lemma 11 suggest some rather unexpected connection to the semantic study of name generation and α-conversion [6,12]. It remains to be seen whether this is merely coincidence or whether techniques and results can be fruitfully transferred in either direction.

References

1. Cook, S.A., McKenzie, P.: Problems complete for deterministic logarithmic space. Journal of Algorithms 8(3), 385–394 (1987)
2. Cook, S.A., Rackoff, C.: Space lower bounds for maze threadability on restricted machines. SIAM Journal of Computing 9(3), 636–652 (1980)
3. Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Springer, Heidelberg (1995)
4. Edmonds, J., Poon, C.K., Achlioptas, D.: Tight lower bounds for st-connectivity on the NNJAG model. SIAM Journal of Computing 28(6), 2257–2284 (1999)
5. Etessami, K., Immerman, N.: Reachability and the power of local ordering. Theoretical Computer Science 148(2), 261–279 (1995)
6. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363 (2002)
7. Gonthier, G.: A computer-checked proof of the four-colour theorem, `http://research.microsoft.com/~gonthier`
8. Grädel, E., McColm, G.L.: On the power of deterministic transitive closures. Information and Computation 119(1), 129–135 (1995)
9. Immerman, N.: Descriptive Complexity. Springer, Heidelberg (1999)
10. Johannsen, J.: Satisfiability problems complete for deterministic logarithmic space. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 317–325. Springer, Heidelberg (2004)
11. Neven, F., Schwentick, T.: On the power of tree-walking automata. In: Welzl, E., Montanari, U., Rolim, J. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 547–560. Springer, Heidelberg (2000)
12. Pitts, A.M., Stark, I.D.B.: Observable properties of higher order functions that dynamically create local names, or: What's new? In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 122–141. Springer, Heidelberg (1993)
13. Reingold, O.: Undirected st-connectivity in log-space. In: STOC, pp. 376–385 (2005)

Quantified Positive Temporal Constraints*

Witold Charatonik and Michał Wrona

Institute of Computer Science
University of Wrocław

Abstract. A positive temporal template (or a positive temporal constraint language) is a relational structure whose relations can be defined over countable dense linear order without endpoints using a relational symbol $\leq$, logical conjunction and disjunction. This paper gives a complete complexity characterization for quantified constraint satisfaction problems (QCSP) over positive temporal languages. Although the constraint satisfaction problem (CSP) for an arbitrary positive temporal language is trivial (all these templates are closed under constant functions), the corresponding QCSP problems are decidable in LOGSPACE or complete for one of the following classes: NLOGSPACE, P, NP or PSPACE.

1 Introduction

Constraint Satisfaction Problems provide a uniform approach to research on a wide variety of combinatorial problems. Undisputedly, the most interesting problem in this area is to verify Dichotomy Conjecture posed by Feder and Vardi [1]. It says that every constraint satisfaction problem on a finite domain is either tractable or NP-complete and was inspired by Schaefer's Dichotomy Theorem for CSP on a two element set [2]. When algebraic approach came on the scene the works on dichotomy conjecture were sped up [3]. Although the main goal has not yet been attained, many interesting results were published and many interesting techniques were developed [4,5]. Besides earlier results on constraints over infinite domains [6,7], a new approach was quite recently proposed and developed by Manuel Bodirsky [8] and co-authors. This framework concentrates on relational structures that are ω-categorical. Many results concerning both CSP and QCSP [9] over finite domains were generalized to infinite ones. Moreover, new results were established. Among them there are full characterizations of complexity for both CSP and QCSP of equality constraint languages [10,11].

Our paper is the next step in this research area. In general, we consider quantified constraint satisfaction problems for sets of relations definable over $\langle \mathbb{Q}, < \rangle$. In particular, we restrict ourselves to templates definable with $\wedge, \vee$ and $\leq$, i.e., we do not consider negation. We name such relations and languages positive temporal. As in [12], we refer to an arbitrary relation defined over $\langle \mathbb{Q}, < \rangle$ as a temporal relation.

Our main contribution is a complexity characterization of QCSP problems over positive temporal languages summarized in Theorem 1 below. We follow the algebraic approach to constraint satisfaction problems: we first classify positive temporal languages depending on their surjective polymorphisms and then give the complexity of QCSP for each obtained class.

* Work partially supported by Polish Ministry of Science and Education grant 3 T11C 042 30.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 94–108, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Theorem 1 (The Main Theorem). *Let Γ be a language of positive temporal relations, then one of the following holds.*

1. *Each relation in Γ is definable by a conjunction of equations ($x_1 = x_2$) and then QCSP(Γ) is decidable in LOGSPACE.*
2. *Each relation in Γ is definable by a conjunction of weak inequalities ($x_1 \leq x_2$). If there exists a relation in Γ that is not definable as a conjunction of equalities, then QCSP(Γ) is NLOGSPACE-complete.*
3. *Each relation in Γ is definable by a formula of the form $\bigwedge_{i=1}^{n}(x_{i_1} \leq x_{i_2} \vee \ldots \vee x_{i_1} \leq x_{i_k})$ and then, provided Γ satisfies neither condition 1 nor 2, the set QCSP(Γ) is P-complete.*
4. *Each relation in Γ is definable by a formula of the form $\bigwedge_{i=1}^{n}(x_{i_2} \leq x_{i_1} \vee \ldots \vee x_{i_k} \leq x_{i_1})$ and then provided Γ satisfies neither condition 1 nor 2, the set QCSP(Γ) is P-complete.*
5. *Each relation in Γ is definable by a formula of the form $\bigwedge_{i=1}^{n}(x_{i_1} = y_{i_1} \vee \ldots \vee x_{i_k} = y_{i_k})$ and then provided Γ does not belong to any of the classes 1–4, the set QCSP(Γ) is NP-complete.*
6. *The problem QCSP(Γ) is PSPACE-complete.*

Related work. The complete characterization is quite complex and does not fit into one paper. Therefore some parts of Theorem 1 are proved in a companion paper [13]. In particular, we prove there that each QCSP problem over positive temporal relations is either in P or is NP-hard. Here we provide the complete characterization of the NP-hard case, distinguishing between NP-complete and PSPACE-complete cases (items 5 and 6 of Theorem 1). We also give complexity proofs for items 3 and 4 of Theorem 1, leaving the algebraic characterization in [13].

Quantified constraint satisfaction problems over temporal relations were investigated in [11,14]. In particular, it is shown there that quantified problems from item 1 and 2 of Theorem 1 belong to LOGSPACE and NLOGSPACE, respectively. Our result substantially improves these results in the sense that we consider a strictly more expressive class of constraint languages. As in [11] we use the surjective preservation theorem.

The area of *CSP* may be often seen as a good framework for many problems in AI. In context of our characterization the well-motivated AND/OR precedence constraints [15] should be noted. They are closely related to languages from items 3 and 4. It might be said that we consider quantified positive variations of AND/OR precedence constraints.

In a very recent paper [12] the authors give a classification of *CSP* over temporal languages depending on their polymorphisms. Although it sounds similar, it is different from our classification. We deal with positive temporal languages and surjective polymorphisms, which are used to classify *QCSP* problems (as opposed to *CSP* problems considered in [12]). In the case of positive temporal languages, the classification based on polymorphisms is trivial: all these languages fall into the same class because they are all closed under constant functions — as a consequence all *CSP* problems for positive temporal languages are trivial. To obtain our classification we use methods different from those used in [12].

Outline of the paper. In Section 2, we give some preliminaries. Among others, we recall a definition of a surjective polymorphism and surjective preservation theorem, which is the most important tool in algebraic approach to *QCSP*. In [13] it is shown that the problem *QCSP*(Γ) is NP-hard if and only if Γ has essentially unary surjective polymorphisms only. Sections 3 and 4 are devoted to classify positive temporal languages preserved by essentially unary surjections. In Section 3 we show that there are only five different classes of positive temporal relations with different surjective unary polymorphisms. If a positive temporal language Γ is closed under all unary surjective polymorphisms, then, as it was shown in [11], each relation of Γ may be defined as in item 5 of Theorem 1 and *QCSP*(Γ) is NP-complete. If a positive temporal language is preserved by some non-trivial subset of unary surjections, then *QCSP*(Γ) is PSPACE-complete. We prove it in Section 4. The last section contains a complexity proof for cases 3 and 4.

2 Preliminaries

Relational structures. In most cases we follow the notation from [8,11]. We consider only relations defined over countable domains and hence whenever we write a domain or D we mean a countable set. Let τ be some relational (in this paper always finite) signature i.e., a set of relational symbols with assigned arity. Then Γ is a τ-structure over domain D if for each relational symbol R_i from τ, it contains a relation of according arity defined on D. In the rest of the paper we usually say relational language (or pattern) instead of relational structure. Moreover, we use the same notation for relational symbols and relations.

Automorphisms of Γ constitute a group with respect to composition. An orbit of a k-tuple t in Γ is the set of all tuples of the form $\langle \Pi(t_1), \ldots, \Pi(t_k)\rangle$ for all automorphisms Π. We say that a group of automorphisms of Γ is oligomorphic if for each k it has a finite number of orbits of k-tuples. Although there are many different ways of introducing a concept of ω-categorical structures we do it by the following theorem [16].

Theorem 2 (Engeler, Ryll-Nardzewski, Svenonius). *Let Γ be a relational structure. Then Γ is ω-categorical if and only if the automorphism group of Γ is oligomorphic.*

Polymorphisms. Let R be a relation of arity n defined over D. We say that a function $f : D^m \to D$ is a polymorphism of R if for all $a^1, \ldots, a^m \in R$ (where a^i, for $1 \leq i \leq m$, is a tuple $\langle a^i_1, \ldots, a^i_n\rangle$), we have $\langle f(a^1_1, \ldots, a^m_1), \ldots, f(a^1_n, \ldots, a^m_n)\rangle \in R$. Then we say that f preserves R or that R is closed under f. A polymorphism of Γ is a function that preserves all relations of Γ. By *Pol*(Γ) we denote the set of polymorphisms of Γ, and by *sPol*(Γ) — the set of surjective polymorphisms.

An operation f of arity n is *essentially unary* if there exists a unary operation f_0 such that $f(x_1, \ldots, x_n) = f_0(x_i)$ for some fixed $i \in \{1, \ldots, n\}$. An operation that is not essentially unary is called *essential*.

Quantified constraint satisfaction problems. Let Γ contain $R_1, \ldots, R_k$. Then a conjunctive positive formula (*cp-formula*) over Γ is a formula of the following form:

$$Q_1 x_1 \ldots Q_n x_n (R_1(\boldsymbol{v_1}) \wedge \ldots \wedge R_k(\boldsymbol{v_k})), \tag{1}$$

where $Q_i \in \{\forall, \exists\}$ and $\boldsymbol{v_j}$ are vectors of variables $x_1, \ldots, x_n$.

A *QCSP*(Γ) is a problem to decide whether a given cp-formula without free variables over the structure Γ is true or not. Note that by downward Löwenheim-Skolem Theorem we can focus on countable domains only.

If all quantifiers in (1) are existential then such a cp-formula is called positive primitive (*pp-formula*). A problem to decide whether a given pp-formula over Γ is satisfiable is well-known as a *constraint satisfaction problem*.

A relation R has a *cp-definition* in Γ if there exists a cp-formula $\phi(x_1, \ldots, x_n)$ over Γ such that for all $a_1, \ldots, a_n$ we have $R(a_1, \ldots, a_n)$ iff $\phi(a_1, \ldots, a_n)$ is true. The set of all relations cp-definable in Γ is denoted by $[\Gamma]$.

Lemma 1 ([11]). *Let Γ_1, Γ_2 be relational languages. If every relation in Γ_1 has a cp-definition in Γ_2, then QCSP(Γ_1) is log-space reducible to QCSP(Γ_2).*

The following results link $[\Gamma]$ with *sPol*(Γ). The idea behind Theorem 3 is that the more Γ can express, in the sense of cp-definability, the less polymorphisms are contained in *sPol*(Γ). Moreover, the converse is also true. This theorem is called *surjective preservation theorem*.

Theorem 3 ([11]). *Let Γ be an ω-categorical structure. Then a relation R has a cp-definition in Γ if and only if R is preserved by all surjective polymorphisms of Γ.*

As a direct consequence of Lemma 1 and Theorem 3 we obtain the following.

Corollary 1 ([11]). *Let Γ_1, Γ_2 be ω-categorical structures. If sPol(Γ_2) $\subseteq$ sPol(Γ_1), then QCSP(Γ_1) is log-space reducible to QCSP(Γ_2).*

Games and cp-definitions. Sometimes it is useful to see a cp-formula ψ without free variables as a two-player game. The game consists of alternating moves of existential and universal player. All variables are evaluated in the order they occur in the quantifier prefix, the existential player evaluates existentially quantified variables and the universal player evaluates universally quantified variables. At the end of the game, the players establish a valuation q from the variables of ψ into the set of rational numbers. We say that one variable is earlier (later) than the another one if it occurs earlier (later) in the quantifier prefix. If at the end of the game, the valuation q satisfies ψ, then the existential player wins; otherwise, the universal player is the winner. If the existential player has a winning strategy, then ψ is true; otherwise, if there exists a winning strategy for the universal player, then ψ is false.

Quantified Equality Constraints. Concerning patterns that allow equations and all logical connectives the following classification is known [11].

1. **Negative languages.** Relations of such a language are definable as CNF-formulas whose clauses are either equalities ($x = y$) or disjunctions of disequalities ($x_1 \neq y_1 \vee \ldots \vee x_k \neq y_k$). For each negative Γ the problem *QCSP*(Γ) is contained in LOGSPACE.
2. **Positive languages.** Relations may be defined as a conjunction of disjunctions of equalities ($x_1 = y_1 \vee \ldots \vee x_k = y_k$). For each positive Γ not being negative the problem *QCSP*(Γ) is NP-complete.
3. In any other case the problem *QCSP*(Γ) is PSPACE-complete.

Note that the class 1 from Theorem 1 is a subset of Negative languages and the class 5 is just the class of Positive languages.

To give our characterization we need the following result. It may be inferred from lemmas given in Section 7 in [11].

Lemma 2. *Let Γ be an equational positive constraint language that is preserved by an essential operation on D with infinite image. Then Γ is preserved by all operations, and Γ is negative.*

Corollary 2. *If an equational positive constraint language Γ is positive, but not negative, then sPol(Γ) contains only essentially unary polymorphisms.*

Quantified Positive Temporal Constraints. Now, we focus on positive temporal relations announced in the introduction. All of them are defined over the set of rational numbers using a relational symbol $\leq$ and connectives $\wedge, \vee$. Therefore our results concerning positive temporal relations generalize those for positive equality languages. Since the only relational symbol we use is interpreted as a weak linear order over rational numbers, for each positive temporal structure Γ the set *sPol*(Γ) contains all automorphisms that preserve order, i.e., all increasing unary surjections $f : \mathbb{Q} \to \mathbb{Q}$. Thus, using Theorem 2, it is not hard to see that all positive temporal languages are ω-categorical.

In [13] we show that each temporal relation is closed not only under all increasing functions but also under all weakly increasing surjections.

3 Surjective Unary Polymorphisms of Temporal Relations

This section examines positive temporal relations that are closed only under surjective unary polymorphisms. We want to divide this subset of positive temporal languages into classes each of which contains Γ_1 and Γ_2 if and only if *sPol*$(\Gamma_1) =$ *sPol*(Γ_2) (or equivalently $[\Gamma_1] = [\Gamma_2]$). Such a classification facilitates providing complexity results — see Theorem 3.

First we give some preliminary definitions. A permutation of a finite set is a bijection from this set to itself. Let $A = \{a_1, \ldots, a_n\}$ be a finite ordered set such that $a_1 < \ldots < a_n$. We say that a permutation π of A is a cycle of A if there exists $i \leq n$ such that $\pi(a_i) < \pi(a_{i+1}) < \ldots < \pi(a_n) < \pi(a_1) < \ldots < \pi(a_{i-1})$. Similarly, π is a reversed cycle if there exists $i \leq n$ such that $\pi(a_i) > \pi(a_{i+1}) > \ldots > \pi(a_n) > \pi(a_1) > \ldots > \pi(a_{i-1})$.

Definition 1. *We say that a relation R is closed under all permutations (respectively, under all cycles or reversed cycles) if for every tuple $\langle q_1, \ldots, q_n\rangle \in R$ and every permutation (respectively, every cycle or reversed cycle) π of the set $\{q_1, \ldots, q_n\}$ we have $\langle \pi(q_1), \ldots, \pi(q_n)\rangle \in R$.*

Note that in the definition above we permute the set $\{q_1, \ldots, q_n\}$ (and not the set of indices $\{1, \ldots, n\}$), which may have less then n elements if $q_1, \ldots, q_n$ are not pairwise distinct.

The preceding definitions concern closure under various kinds of permutations. Although they may look quite similar to closure under polymorphisms, they are different.

Below we give some, important for us, examples of (unary) surjective polymorphisms of positive temporal relations. They are all of the type: $\mathbb{Q} \to \mathbb{Q}$.

Definition 2. *We say that a surjection $f : \mathbb{Q} \to \mathbb{Q}$ is weakly half-increasing (respectively weakly half-decreasing) if there exist two irrational real numbers x and y such that*

- *f restricted to the set $\{q \in \mathbb{Q} \mid q < x\}$ as well as f restricted to the set $\{q \in \mathbb{Q} \mid q > x\}$ is weakly increasing (respectively, weakly decreasing), and*
- *for all $q < x$ we have $f(q) > y$ (respectively $f(q) < y$) and for all $q > x$ we have $f(q) < y$ (respectively $f(q) > y$).*

A weakly half-increasing or weakly half-decreasing function is called weakly half-monotone.

Example 1. Recall that all countable dense linear orders without endpoints are isomorphic. In particular, $\mathbb{Q}$ and $\mathbb{Q}\setminus\{0\}$ are isomorphic, so we may identify $\mathbb{Q}$ with $\mathbb{Q}\setminus\{0\}$ and think of 0 as an irrational number in $\mathbb{Q} \setminus \{0\}$. Then the function $f : \mathbb{Q} \setminus \{0\} \to \mathbb{Q} \setminus \{0\}$ defined by $f(q) = \frac{1}{q}$ is weakly half-decreasing and the function defined by $f(q) = \frac{-1}{q}$ is weakly half-increasing.

The unary operation $- : \mathbb{Q} \to \mathbb{Q}$ is defined as $-(x) = -x$ in usual sense.
The rest of this section is devoted to prove the following result.

Theorem 4. *Let Γ be a set of positive temporal relations such that sPol(Γ) contains only essentially unary functions. Then exactly one of the following cases holds.*

1. *sPol(Γ) is the set of all unary surjections of $\mathbb{Q}$.*
2. *sPol(Γ) is the set of all weakly increasing, weakly decreasing or weakly half-monotone surjections of $\mathbb{Q}$.*
3. *sPol(Γ) is the set of all weakly increasing or weakly decreasing surjections of $\mathbb{Q}$.*
4. *sPol(Γ) is the set of all weakly increasing or weakly half-increasing surjections of $\mathbb{Q}$.*
5. *sPol(Γ) is the set of all weakly increasing surjections of $\mathbb{Q}$.*

A similar classification considering (not necessarily surjective) unary polymorphisms was obtained in [12]. Weakly half-increasing polymorphisms correspond in some way to the function cyc from that paper. In turn, positive temporal relations preserved by weakly half-decreasing functions correspond to temporal relations closed under $-$ and cyc.

As indicated in Theorem 4, there are only four interesting classes of unary polymorphisms of positive temporal relations: weakly increasing, weakly decreasing, weakly half-increasing, and weakly half-decreasing. The following lemmas say that if some positive temporal relation is closed under one polymorphism of a given class, then it is closed under all polymorphisms of this class.

Lemma 3. *If sPol(R) contains a weakly decreasing unary surjection f, then it contains all weakly decreasing unary surjections.*

Lemma 4. *If sPol(R) contains a weakly half-increasing unary surjection f, then it contains all weakly half-increasing unary surjections. If sPol(R) contains a weakly half-decreasing unary surjection f, then it contains all weakly decreasing, all weakly half-increasing and all weakly half-decreasing unary surjections.*

Now, we relate various surjective polymorphisms to closures under various kinds of permutations (see for example Definition 1). In particular, Lemma 5 below implies that the set of positive temporal relations closed under all permutations equals to the set of positive languages from [11].

Lemma 5. *A positive temporal relation R is closed under all permutations iff sPol(R) contains all unary surjections of $\mathbb{Q}$.*

Lemma 6. *A positive temporal relation R is closed under all cycles iff sPol(R) contains all weakly half-increasing surjections of $\mathbb{Q}$.*

Lemma 7. *A positive temporal relation R is closed under all reversed cycles iff sPol(R) contains all weakly half-decreasing surjections of $\mathbb{Q}$.*

Since we are interested in surjective functions, we can claim the following.

Lemma 8. *Let f be a unary, surjective operation on Q, then there exist:*

- *an infinite, strictly monotone sequence $(a_n)_{n\in\mathbb{N}}$ of rational numbers such that $\lim_{n\to\infty} f(a_n) = +\infty$*
- *an infinite, strictly monotone sequence $(b_n)_{n\in\mathbb{N}}$ of rational numbers such that $\lim_{n\to\infty} f(b_n) = -\infty$*

To prove Theorem 4, we show that if *sPol*(Γ) contains any function that is neither weakly monotone nor weakly half-monotone, then it contains all unary rational functions or equivalently, by Lemma 5, is closed under all permutations.

Lemma 9. *Let R be a positive temporal relation such that sPol(R) contains a function f that is neither weakly increasing nor weakly decreasing nor weakly half-monotone. Let $(c_n)_{n\in\mathbb{N}}$ and $(d_n)_{n\in\mathbb{N}}$ be two strictly monotone sequences satisfying the following: $\lim_{n\to\infty} f(c_n) = +\infty$ and $\lim_{n\to\infty} f(d_n) = -\infty$. Then R is closed under all permutations.*

Proof. (of Theorem 4) Suppose that *sPol*(Γ) contains only essentially unary functions. If *sPol*(Γ) contains a function f that is neither weakly monotone nor weakly half-monotone, then by Lemma 8 we find two strictly monotone sequences $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ such that $\lim_{n\to\infty} f(a_n) = +\infty$ and $\lim_{n\to\infty} f(b_n) = -\infty$. Then by Lemma 9 every relation in Γ is closed under all permutations, so by Lemma 5 *sPol*(R) contains all essentially unary surjections. Hence *sPol*(Γ) is the set of all essentially unary surjections of $\mathbb{Q}$ and we are in case 1.

Now assume that *sPol*(Γ) contains only weakly monotone or weakly half-monotone surjections. There are four cases, depending on whether *sPol*(Γ) contains a weakly decreasing surjection or a weakly half-increasing surjection.

If *sPol*(Γ) contains a weakly decreasing surjection and a weakly half-increasing surjection, then it contains a weakly half-decreasing surjection and by Lemmas 3 and 4, it contains all weakly decreasing and all weakly half-monotone surjections of $\mathbb{Q}$, so we are in case 2.

If *sPol*(Γ) contains a weakly decreasing surjection and it does not contain any weakly half-increasing surjection, then by lemmas 3 and 4 it contains all weakly decreasing and it does not contain any weakly half-monotone surjections, so we are in case 3.

If *sPol*(Γ) does not contain any weakly decreasing surjection and it contains a weakly half-increasing surjection, then by Lemma 4 it contains all weakly half-increasing and it does not contain weakly decreasing surjections, so we are in case 4.

Finally, if *sPol*(Γ) does not contain any weakly decreasing surjection and it does not contain any weakly half-increasing surjection, then by Lemma 4 it does not contain any weakly decreasing nor weakly half-monotone surjection, so we are in case 5. □

Example 2. Recall from [11] that $(x_1 = x_2 \vee x_1 = x_3)$ is closed under all essentially unary surjections of $\mathbb{Q}$ — see also Section 2. Now, for each of the classes 2–5 of Theorem 4 we give representatives, that is, relations $R_{(5)}$–$R_{(2)}$ each of which belongs to exactly one of these classes.

The relation $R_{(5)}$ defined by $R_{(5)}(x_1, x_2, x_3) := (x_1 \leq x_2 \vee x_2 \leq x_3)$, as all positive temporal relations, is closed under all weakly increasing functions. Observe that $\langle 1, 2, 3\rangle \in R_{(5)}$, but $\langle -1, -2, -3\rangle \notin R_{(5)}$, so $R_{(5)}$ is not closed under weakly decreasing functions (and by Lemma 4 it is not closed under half-decreasing functions). Similarly, $\langle 1, 3, 2\rangle \in R_{(5)}$, but $\langle 3, 2, 1\rangle \notin R_{(5)}$, so $R_{(5)}$ is not closed under cycles (and thus it is not closed under weakly half-increasing functions). The relation $R_{(4)}$ defined by $R_{(4)}(x_1, x_2, x_3) := (x_1 \leq x_2 \vee x_2 \leq x_3) \wedge (x_2 \leq x_3 \vee x_3 \leq x_1) \wedge (x_3 \leq x_1 \vee x_1 \leq x_2)$ is a conjunction of the relations $(x_{\Pi(1)} \leq x_{\Pi(2)} \vee x_{\Pi(2)} \leq x_{\Pi(3)})$ where Π ranges over all cycles of the set $\{1, 2, 3\}$, so it is closed under all cycles. Since $\langle 1, 2, 3\rangle \in R_{(4)}$ and $\langle 3, 2, 1\rangle \notin R_{(4)}$, it is not closed under weakly decreasing or weakly half-decreasing functions. It is easy to observe that the relation $R_{(3)}$ defined by $R_{(3)}(x_1, x_2, x_3) := (x_1 \leq x_2 \vee x_2 \leq x_3) \wedge (x_3 \leq x_2 \vee x_2 \leq x_1)$ is closed under weakly decreasing functions. Since $\langle 2, 1, 3\rangle \in R_{(3)}$ and $\langle 3, 2, 1\rangle \notin R_{(3)}$, this relation is not closed under cycles and by Lemmas 6 and 4 it is not closed under any weakly half-monotone surjection. Let a relation $R_{(2)}$ be defined as a conjunction of the clauses $(x_{\Pi(1)} \leq x_{\Pi(2)} \vee x_{\Pi(2)} \leq x_{\Pi(3)} \vee x_{\Pi(3)} \leq x_{\Pi(4)})$ where Π ranges over all cycles and reversed cycles of the set $\{1, 2, 3, 4\}$, so it obviously must be closed under all cycles and reversed cycles. Note that cycles and reversed cycles are 8 out of total 24 permutations of the set $\{1, 2, 3, 4\}$ (this explains why we could not use a ternary relation as an example here — all permutations of the set $\{1, 2, 3\}$ are either cycles or reversed cycles). To see that $R_{(2)}$ is not closed under all permutations observe that $\langle 4, 3, 2, 1\rangle \notin R_{(2)}$, but $\langle 2, 1, 3, 4\rangle \in R_{(2)}$.

Finally, we show that all these relations ($R_{(2)}$–$R_{(5)}$) are closed under essentially unary surjections only. Let $R(x_1, \ldots, x_k)$ where $k = 3, 4$ be one of these relations. Then a relation $\bigwedge_{\Pi \in S_k} R(x_{\Pi(1)}, \ldots, x_{\Pi(k)})$ where S_k is a set of all permutations on k elements is equivalent to a relation R' defined by $\bigvee_{i \neq j} x_i = x_j$. Because R' is positive and non-negative, by Corollary 2 and Theorem 3, we have that R is closed under unary surjections only.

4 PSPACE-Complete Positive Temporal Languages

Recall from Section 2 the complexity characterization of equational languages. By corollaries 2 and 1, the problem $QCSP(\Gamma)$ where Γ is closed under essentially unary functions only is NP-hard. Likewise we know that *QCSP* for languages from item 1 of Theorem 4 is NP-complete. This section is devoted to show PSPACE-completeness for *QCSP* of languages with surjective polymorphisms from items 2–5 of Theorem 4.

Membership in PSPACE is the simpler part of the proof and is common for all, not only positive, temporal relations.

Proposition 1. *For every temporal language Γ, the problem $QCSP(\Gamma)$ is decidable in PSPACE.*

In the rest of the section we prove hardness. Note that the set of surjective polymorphisms from item 2 contains sets of surjective polymorphisms from each of items 2–5. Therefore, by Theorem 3 and Corollary 1, it is enough to show PSPACE-hardness of *QCSP* for positive temporal languages closed under all weakly monotone and all weakly half-monotone surjections only.

Theorem 5. *Let Γ be a set of positive languages closed only under essentially unary functions. If $sPol(\Gamma)$ is the set of all weakly increasing, weakly decreasing and weakly half-monotone surjections of $\mathbb{Q}$, then $QCSP(\Gamma)$ is PSPACE-hard.*

Because of Theorem 3 and Corollary 1, it is enough to choose just one language with appropriate set of surjective polymorphisms and show PSPACE hardness for this language. Our choice is the language Γ_{Circle} defined below. We show that it is closed only under all weakly increasing, weakly decreasing and weakly half-monotone surjections of $\mathbb{Q}$. In fact, it is enough to show that Γ_{circle} is closed only under unary surjections of $\mathbb{Q}$ and that is closed under all reversed cycles – see lemmas 4 and 7. Finally, we show that $QCSP(\Gamma_{Circle})$ is PSPACE-hard and in consequence we prove Theorem 5.

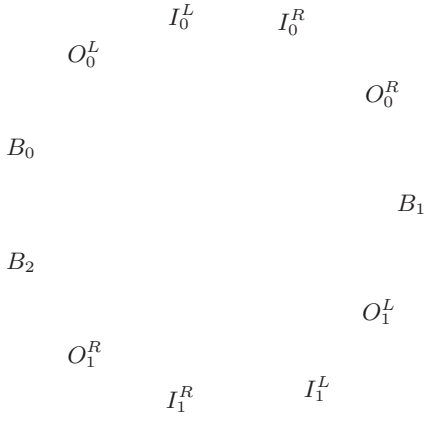


Fig. 1. The representation of the set $Arenas$

Definition of Γ_{Circle}. First we present some auxiliary relations that shorten the definition. Let $\boldsymbol{v}_A$ be a vector $\langle B_0, O_0^L, I_0^L, I_0^R, O_0^R, B_1, O_1^L, I_1^L, I_1^R, O_1^R, B_2\rangle$ of variables ranging over $\mathbb{Q}$. The corresponding set of variables is denoted by Var_A. In the following we call Var_A the set of arena variables. We sometimes see a vector $\boldsymbol{v_A}$ as a function from $\{0,\dots,10\}$ to Var_A.

Let $Arenas$ be a set of vectors $\Pi(\boldsymbol{v_A})$ for all cycles and reversed-cycles Π of the set $\{0,\dots,10\}$. Note that the set $Arenas$ may be represented in some way using Fig. 1. To obtain one of linear orders that is represented by this circle, we have just to tear it apart and orientate. If we orientate it clockwise, then we represent some $\Pi_C(\boldsymbol{v_A})$ where Π_C is a cycle. Otherwise, if we orientate it anticlockwise, then we represent some $\Pi_{RC}(\boldsymbol{v_A})$ where Π_{RC} is a reversed cycle.

Now, for each $\boldsymbol{v} \in Arenas$ we define a relation $Prefix_{\boldsymbol{v}} := \neg(y_0 < \dots < y_{10})$ where $\boldsymbol{v} = \langle y_0,\dots,y_{10}\rangle$. At this point we probably owe the reader one more explanation. Sometimes, when we think it is intuitive, we use negation as well as implication in the definition of relations. Nevertheless, they should be treated just as notational shortcuts and all relations we claim to be positive temporal are indeed definable by conjunction, disjunction, and $\le$. In particular, $Prefix_{\boldsymbol{v}}$ may be defined as $\bigvee_{i=0}^{9} y_i \ge y_{i+1}$. Nevertheless, the situation where $Prefix_{\boldsymbol{v}}$ is falsified is more important for us. Then the arena variables are arranged in some linear order represented by a circle in Fig. 1.

In general, our intention is to model (see Definition 3) a boolean relation. Arena variables set in some order presented in Fig. 1 constitute some kind of arena. When some other variable is set strictly between O_0^L and O_0^R then we see its value as a boolean zero, and if some variable is set strictly between O_1^L and O_1^R then we see its value as a boolean one. We need also I_0^L, I_0^R, and I_1^L, I_1^R. Sometimes we want to say: 'If a variable x is equal to zero, then a variable y is also equal to zero'. Unfortunately, concerning positive temporal relations we are unable to write something like $(O_0^L < x < O_0^R) \to (O_0^L < y < O_0^R)$. Instead we write $(O_0^L < x < O_0^R) \to (I_0^L \le y \le I_0^R)$ and assure that I_0^L, I_0^R are always strictly between O_0^L, O_0^R. Similarly, we assure that I_1^L, I_1^R are always strictly between O_1^L, O_1^R. This is the general idea, but sometimes because of technical reasons we also use O_0^{L1}, O_0^{L2} etc.

Concerning positive temporal relations closed under reversed cycles it is hard to say that some variable must be set on the left (or on the right) of the another variable. Far more natural is to say that some variable is inside the interval set by values of other variables or outside such an interval. We define $In_{\boldsymbol{v}}(x, y_1, y_2)$ equal to $((y_1 < y_2) \to (y_1 \le x \le y_2)) \wedge ((y_2 < y_1) \to (x \le y_2 \vee x \ge y_1))$ if $\boldsymbol{v} = \Pi_C(\boldsymbol{v_A})$ for some cycle Π_C; and equal to $((y_1 < y_2) \to (x \le y_1 \vee x \ge y_2)) \wedge ((y_2 < y_1) \to (y_2 \le x \le y_1))$ if $\boldsymbol{v} = \Pi_{RC}(\boldsymbol{v_A})$ for some reversed-cycle Π_{RC}. Similarly, we define $Out_{\boldsymbol{v}}(x, y_1, y_2)$ equal to $((y_1 < y_2) \to (x \le y_1 \vee x \ge y_2)) \wedge ((y_2 < y_1) \to (y_2 \le x \le y_1))$ if $\boldsymbol{v} = \Pi_C(\boldsymbol{v_A})$ for some cycle Π_C; and equal to $((y_1 < y_2) \to (y_1 \le x \le y_2)) \wedge ((y_2 < y_1) \to (x \le y_2 \vee x \ge y_1))$ if $\boldsymbol{v} = \Pi_{RC}(\boldsymbol{v_A})$ for some reversed-cycle Π_{RC}.

Example 3. For every $\boldsymbol{v} \in Arenas$ the following formulas are always true:

1. $(\neg Prefix_{\boldsymbol{v}}) \to In_{\boldsymbol{v}}(I_0^R, O_0^R, O_0^L)$
2. $(\neg Prefix_{\boldsymbol{v}}) \to Out_{\boldsymbol{v}}(O_0^R, I_0^R, I_0^L)$

The positive temporal language Γ_{Circle} consists of three relations: *UImp*, *BImp*, and *Final*. Each relation $R \in \Gamma_{Circle}$ is of the form $\bigwedge_{\boldsymbol{v} \in Arenas} \phi^R_{\boldsymbol{v}}$. By using this conjunction we assure that R is closed under all cycles and reversed cycles.

1. First of our relations is $\mathit{UImp}(\boldsymbol{v_A}, p, O^L, O^R, f, I^L, I^R)$ with

$$\phi^{UImp}_{\boldsymbol{v}} := \mathit{Prefix}_{\boldsymbol{v}} \vee \mathit{Out}_{\boldsymbol{v}}(p, O^L, O^R) \vee \mathit{In}_{\boldsymbol{v}}(f, I^L, I^R). \tag{2}$$

   The name *UImp* stands for unary implication. It is justified by the context in which we use it. If both $\mathit{Prefix}_{\boldsymbol{v}}$ and $\mathit{Out}_{\boldsymbol{v}}(p, O^L, O^R)$ are falsified, then $\mathit{In}_{\boldsymbol{v}}(f, I^L, I^R)$ must be satisfied. We use this relation to express the implication: 'if v represents different values in an appropriate order and p is a value in the interval from I^L to I^R, then f also is a value in this interval'.
2. We have also binary implication $\mathit{BImp}(\boldsymbol{v_A}, p_1, p_2, O^L, O^R, f, I^L, I^R)$ with

$$\phi^{BImp}_{\boldsymbol{v}} := \mathit{Prefix}_{\boldsymbol{v}} \vee \mathit{Out}_{\boldsymbol{v}}(p_1, O^L, O^R) \vee \mathit{Out}_{\boldsymbol{v}}(p_2, O^L, O^R) \vee \mathit{In}_{\boldsymbol{v}}(f, I^L, I^R). \tag{3}$$

   If $\mathit{Prefix}_{\boldsymbol{v}}$ as well as $\mathit{Out}_{\boldsymbol{v}}(p_1, O^L, O^R)$ and $\mathit{Out}_{\boldsymbol{v}}(p_2, O^L, O^R)$ are falsified, then $\mathit{In}_{\boldsymbol{v}}(f, I^L, I^R)$ must be satisfied.
3. Finally there is $\mathit{Final}(\boldsymbol{v_A}, f_0, f_1)$ with

$$\phi^{Final}_{\boldsymbol{v}} := \mathit{Prefix}_{\boldsymbol{v}} \vee \mathit{Out}_{\boldsymbol{v}}(f_0, B_0, B_2) \vee \mathit{Out}_{\boldsymbol{v}}(f_1, B_0, B_2). \tag{4}$$

   We want to see it in the following way. If $\mathit{Prefix}_{\boldsymbol{v}}$ is falsified, then $\mathit{Out}_{\boldsymbol{v}}(f_0, B_0, B_2)$ or $\mathit{Out}_{\boldsymbol{v}}(f_1, B_0, B_2)$ must be satisfied.

Lemma 10. *The positive temporal language Γ_{Circle} is closed under weakly increasing, weakly decreasing, and weakly half monotone surjections only.*

PSPACE-hardness of QCSP(Γ_{Circle}). The hardness proof for *QCSP*(Γ_{Circle}) is based on the proof of PSPACE-hardness of *QCSP*($x_1 \neq x_2 \vee x_1 = x_3$) from [11]. We define analogous notions and follow analogous reasoning.

Definition 3. *A relation $R \subseteq \{0,1\}^n$ is force definable if there exists a prenex formula*

$$\Phi_{R,f_0,f_1}(\boldsymbol{v_A}, O^{L1}_0, O^{R1}_0, O^{L1}_1, O^{R1}_1, x_1, \ldots, x_n) = \mathcal{Q}\phi$$

over Γ_{Circle} that satisfies all of the following.

1. *$\mathcal{Q}$ is a quantifier prefix and ϕ is a quantifier-free part.*
2. *The quantifier prefix $\mathcal{Q}$ contains f_0 and f_1 as its two last variables, and they are both existentially quantified.*
3. *The set of free variables is equal to $\{\boldsymbol{v_A}, O^{L1}_0, O^{R1}_0, O^{L1}_1, O^{R1}_1, x_1, \ldots, x_n\}$.*
4. *Let $\boldsymbol{t} \in \{0,1\}^n$ and let $\boldsymbol{v} \in Arenas$. Let variables from Var_A be set to satisfy $\neg(\mathit{Prefix}_{\boldsymbol{v}})$ and let variables $O^{L1}_0, O^{R1}_0, O^{L1}_1, O^{R1}_1$ be set to satisfy*
   - *$\mathit{In}_{\boldsymbol{v}}(O^{L1}_0, B_0, O^L_0)$,*
   - *$\mathit{In}_{\boldsymbol{v}}(O^{R1}_0, O^R_0, B_1)$,*
   - *$\mathit{In}_{\boldsymbol{v}}(O^{L1}_1, B_1, O^L_1)$, and*
   - *$\mathit{In}_{\boldsymbol{v}}(O^{R1}_1, O^R_1, B_2)$.*

Further, let x_k for $k \in \{1, \ldots, n\}$ are set to satisfy $In_{\boldsymbol{v}}(x_k, I_i^L, I_i^R)$ iff $\boldsymbol{t}_k = i$ for $i = 0, 1$. Then the sentence $\Phi' := \mathcal{Q}(\phi \wedge \neg(In_{\boldsymbol{v}}(f_0, I_0^L, I_0^R) \wedge In_{\boldsymbol{v}}(f_1, I_1^L, I_1^R)))$ is false iff $\boldsymbol{t} \in R$.

5. *If values of arena variables satisfy $Prefix_{\boldsymbol{v}}$ for all $\boldsymbol{v} \in Arenas$; or, in case $Prefix_{\boldsymbol{v}}$ is falsified for some $\boldsymbol{v} \in Arenas$, free variables $O_0^{L1}, O_0^{R1}, O_1^{L1}, O_1^{R1}$ are set to satisfy*
   - *$In_{\boldsymbol{v}}(O_0^{L1}, I_0^L, I_0^R) \wedge In_{\boldsymbol{v}}(O_0^{R1}, I_0^L, I_0^R) \vee$*
   - *$In_{\boldsymbol{v}}(O_1^{L1}, I_1^L, I_1^R) \wedge In_{\boldsymbol{v}}(O_1^{R1}, I_1^L, I_1^R)$;*

   then Φ' is always true.
6. *(monotonicity) For any setting to the free variables of Φ_{R,f_0,f_1}, if the formula Φ' is true, then changing the value of any variable x_i to satisfy $(Out_{\boldsymbol{v}}(x_i, O_0^{L1}, O_0^{R1}) \vee Out_{\boldsymbol{v}}(x_i, O_1^{L1}, O_1^{R1}))$ preserves the truth of Φ'.*

As it was described in Section 2 we can see a sentence as a two-player game. The intuition behind Definition 3 is as follows. If free variables of Φ_{R,f_0,f_1} are set according to conditions from item 4 and $\boldsymbol{t} \in R$, then the universal player has a strategy to force the existential player to satisfy $In_{\boldsymbol{v}}(f_0, I_0^L, I_0^R)$ and $In_{\boldsymbol{v}}(f_1, I_1^L, I_1^R)$ where $\boldsymbol{v} \in Arenas$ and $Prefix_{\boldsymbol{v}}$ is falsified. But if $Prefix_{\boldsymbol{v}}$ is falsified for some $\boldsymbol{v} \in Arenas$ and the condition from item 5 is fulfilled, then the existential player is able to falsify $In_{\boldsymbol{v}}(f_0, I_0^L, I_0^R)$ or $In_{\boldsymbol{v}}(f_1, I_1^L, I_1^R)$.

Note that variables $O_0^{L1}, O_0^{R1}, O_1^{L1}, O_1^{R1}$ are different from $O_0^L, O_0^R, O_1^L, O_1^R$.

Lemma 11. *There exists a polynomial-time algorithm that, given a boolean circuit C as input, produces a force definition of the relation R_C containing, as tuples, exactly the satisfying assignments of the circuit.*

Similarly as in [11], we reduce from succinct graph unreachability. In this problem, the input is a boolean circuit with $2c$ inputs that represent a graph G whose vertices are the tuples in $\{0,1\}^c$. There is a directed edge $(\boldsymbol{X}, \boldsymbol{Y})$ in the graph iff the circuit returns true given the input $(\boldsymbol{X}, \boldsymbol{Y})$. The question is to decide whether or not there is a directed path from $\boldsymbol{S}$ to $\boldsymbol{T}$. This problem is known to be *PSPACE*-complete.

Define $R_i \subseteq \{0,1\}^{2c}$ to be the relation containing exactly the tuples $(\boldsymbol{X}, \boldsymbol{Y})$ such that there exists a directed path in G from $\boldsymbol{X}$ to $\boldsymbol{Y}$ of length less than or equal to 2^i. Then there is a path in G from $\boldsymbol{S}$ to $\boldsymbol{T}$ iff $(\boldsymbol{S}, \boldsymbol{T}) \in R_c$.

From Lemma 11 it is not hard to infer that R_0 is computable in polynomial time. Now, by induction we show that R_c is also computable by a polynomial algorithm.

Lemma 12. *The force definition of the relation R_c is computable in polynomial time.*

Proof. (of Theorem 5) Let $\Phi_{R_c,g_0,g_1}(\boldsymbol{v_A}, O_0^L, O_0^R, O_1^L, O_1^R, \boldsymbol{x}, \boldsymbol{y}) = Q_c\phi_c$ be a force definition of R_c. We use it now to give an instance of $\boldsymbol{QCSP}(\Gamma_{Circle})$ that is true if and only if there is no path from $\boldsymbol{S}$ to $\boldsymbol{T}$ in the succinctly represented graph.

The instance created is

$$\forall \boldsymbol{v_A} Q_c\phi_c \wedge \boldsymbol{x} = \boldsymbol{s} \wedge \boldsymbol{y} = \boldsymbol{t} \wedge \mathit{Final}(\boldsymbol{v_A}, g_0, g_1)$$

where $\boldsymbol{s}_i = \boldsymbol{t}_j = I_k^L$ if $\boldsymbol{S}_i = \boldsymbol{T}_j = k$ for all $1 \leq i, j \leq c$ and $k = 0, 1$.

The universal player starts the game. To have a chance to win (to falsify) the sentence he must set arena variables to falsify $Prefix_{\boldsymbol{v}}$ for some $\boldsymbol{v} \in Arenas$. Otherwise each

clause from ϕ_c is satisfied already at the beginning. If there is a path from $\boldsymbol{s}$ to $\boldsymbol{t}$, then the universal player can enforce the existential player to satisfy $In_{\boldsymbol{v}}(g_0, I_0^L, I_0^R)$ and $In_{\boldsymbol{v}}(g_1, I_1^L, I_1^R)$. It is not hard to verify that it contradicts $\mathit{Final}(\boldsymbol{v_A}, g_0, g_1)$ — see (4). If $(\boldsymbol{s}, \boldsymbol{t}) \notin R_c$, then the existential player can satisfy $Out_{\boldsymbol{v}}(g_0, B_0, B_2)$ or $Out_{\boldsymbol{v}}(g_1, B_0, B_2)$; and in consequence satisfy $\mathit{Final}(\boldsymbol{v_A}, g_0, g_1)$. □

5 PTIME-Complete Positive Temporal Languages

This section is devoted to give the complexity proof for classes from items 3 and 4 of Theorem 1. We focus here of the former case, the latter one is dual and hence the whole reasoning is similar in both cases.

Let R^k_{Left} be equal to $(x_1 \leq x_2 \vee \ldots \vee x_1 \leq x_k)$, and Γ_{Left} be the set of relations R^k_{Left} for each natural number $k \geq 2$. In [13] it is shown that $\mathit{QCSP}(\Gamma_{Left})$ is log-space equivalent to $\mathit{QCSP}(\Gamma)$ where Γ is the language from case 3 of Theorem 1. Furthermore, we have that $\mathit{QCSP}(x_1 \leq x_2 \vee x_1 \leq x_2)$ is log-space equivalent to $\mathit{QCSP}(\Gamma_{Left})$. Observe that the definition of each R^k_{Left} have a simple tree-like structure where x_1 is a root and $x_2, \ldots, x_k$ are sons of x_1. Moreover, to denote a root of a clause C we write $root(C)$ and to denote a set of sons – $sons(C)$.

In this section whenever we write: a formula, we think of a cp-formula over Γ_{Left}.

Lemma 13. *Let ϕ be a formula defining a positive temporal relation. Assume that ϕ contains clauses $C_1 := (y \leq x_1 \vee \ldots \vee y \leq x_k)$ and $C_2 := (x_1 \leq z_1 \vee \ldots \vee x_1 \leq z_l)$. Then ϕ and ϕ' given by $\phi \wedge C_3$, where $C_3 := (y \leq z_1 \vee \ldots \vee y \leq z_l \vee y \leq x_2 \vee \ldots \vee y \leq x_k)$, are equivalent, that is, they define the same relation.*

In the following, we sometimes refer to a quantifier-free formula ϕ as to a set of clauses. We say that a set of clauses ϕ is $\mathit{TClosed}$ if for all pairs of clauses of the form $(y \leq x_1 \vee \ldots \vee y \leq x_k)$ and $(x_1 \leq z_1 \vee \ldots \vee x_1 \leq z_l)$, the clause $(y \leq z_1 \vee \ldots \vee y \leq z_l \vee y \leq x_2 \vee \ldots y \vee \leq x_k)$ also belongs to ϕ. By $\mathit{TClosure}(\phi)$ we denote the least $\mathit{TClosed}$ superset of ϕ.

By a simple induction, from Lemma 13 we can obtain the following.

Corollary 3. *Formulas ϕ and $\mathit{TClosure}(\phi)$ are equivalent.*

We show that the universal player has a winning strategy if and only if $\mathit{TClosure}(\phi)$ contains a clause of the form $(y \leq x_1 \vee \ldots \vee y \leq x_k)$ such that for each disjunct $y \leq x_i$ where $1 \leq i \leq k$ we have that either y or x_i is later and universal. We call such a clause *ultimate*.

Lemma 14. *Let ψ be a sentence and let Q be its quantifier prefix and ϕ its quantifier-free part. Then ψ is false if and only if $\mathit{TClosure}(\phi)$ contains an ultimate clause.*

To show the exact complexity of case 3 of Theorem 1 we use the emptiness problem for context-free grammars. It is well known that this problem is P-complete. We assume that the reader is familiar with the notion of the context-free grammar. By $\mathcal{L}(G)$ we denote a language generated by a context free-grammar $G = \langle N, \Sigma, R, S \rangle$.

Theorem 6. *Let Γ be a positive temporal language such that each of its relation is definable by a formula of the form $\bigwedge_{i=1}^{n}(x_{i_1} \leq x_{i_2} \vee \ldots \vee x_{i_1} \leq x_{i_k})$ and it is neither definable as a conjunction of equalities nor as a conjunction of inequalities. Then the problem QCSP(Γ) is P-complete*

Proof. (About Membership) To obtain the result we give a logspace reduction from the problem *QCSP*$(x_1 \leq x_2 \vee x_1 \leq x_3)$ to the emptiness problem for context-free grammars. Let ψ be an instance of *QCSP*$(x_1 \leq x_2 \vee x_1 \leq x_3)$ with a quantifier prefix Q and a quantifier free-part ϕ. We construct a context-free grammar G_ψ such that $\mathcal{L}(G_\psi) \neq \emptyset$ if and only if ψ is false. By Lemma 14, it is enough to show G_ψ that generates a non-empty language if and only if $TClosure(\phi)$ contains an ultimate clause.

The reduction runs as follows. For each variable x of ψ, we show a grammar G_ψ^x that generates an empty language if and only if there is no ultimate clause C in $TClosure(\phi)$ with x being a root of C. Further, by G_ψ we take a grammar such that $\mathcal{L}(G_\psi) = \bigcup_{x \in Var(\psi)} \mathcal{L}(G_\psi^x)$ where $Var(\psi)$ is a set of all variables of ψ. Recall that the set of context-free grammars is closed under union and note that $\|G_\psi\| \leq c{*}\Sigma_{x \in Var(\psi)} \left\|G_\psi^x\right\|$ for some constant c.

We now turn to the definition of $G_x = \langle N_x, \Sigma_x, R_x, A_x \rangle$. For each variable y of ψ we introduce a nonterminal A_y. The set Σ_x contains a terminal symbol a_y for each variable y that is universal and later than x. If x is universal, then there is also a terminal a_y for each variable y that is earlier than x. Further, for each clause of the form $(x_1 \leq x_2 \vee x_1 \leq x_3)$ we have a rule $A_{x_1} \to A_{x_2}A_{x_3}$. For each terminal symbol a_y in Σ_x there is also a rule $A_y \to a_y$. It is clear, that such a reduction may be provided using logarithmic space.

Now, if $\mathcal{L}(G_\psi^x)$ contains a word $a_{x_1} \ldots a_{x_k}$, then, by a simple induction, we can show that a clause $(x \leq x_1 \vee \ldots \vee x \leq x_k)$ belongs to $TClosure(\phi)$. Since each x_i for $1 \leq i \leq k$ is universal and later than x or provided x is universal, earlier than x; this clause is ultimate. Similarly, if any ultimate clause $(x \leq x_1 \vee \ldots \vee x \leq x_k)$ belongs to $TClosure(\phi)$, then we can construct a parse tree that witnesses $a_{x_1} \ldots a_{x_k} \in \mathcal{L}(G_\psi^x)$

(About Hardness) The hardness proof is quite similar. This time we give a logspace reduction from the emptiness problem to the problem *QCSP*$(x_1 \leq x_2 \vee x_1 \leq x_3)$. □

Acknowledgements. We thank Jerzy Marcinkowski for turning our attention to [11].

References

1. Feder, T., Vardi, M.: Monotone monadic SNP and constraint satisfaction. In: Proceedings of 25th ACM Symposium on the Theory of Computing (STOC), pp. 612–622 (1993)
2. Schaefer, T.: The complexity of satisfiability problems. In: Proceedings 10th ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226 (1978)
3. Jeavons, P., Cohen, D., Gyssens, M.: Closure properties of constraints. Journal of the ACM 44, 527–548 (1997)
4. Bulatov, A.: A dichotomy theorem for constraints on a three-element set. In: Proceedings 43rd IEEE Symposium on Foundations of Computer Science (FOCS 2002), Vancouver, Canada, pp. 649–658 (2002)

5. Cohen, D., Jeavons, P.: The complexity of constraints languages. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming. Elsevier, Amsterdam (2006)
6. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM 26(11), 832–843 (1983)
7. Krokhin, A., Jeavons, P., Jonsson, P.: A complete classification of complexity in Allens algebra in the presence of a non-trivial basic relation. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 83–88. Morgan Kaufmann, San Francisco (2001)
8. Bodirsky, M.: Constraint Satisfaction Problems with Infinite Domains. PhD thesis, Humboldt-Universität zu Berlin (2004), http://www2.informatik.hu-berlin.de/~bodirsky/publications/diss.html
9. Boerner, F., Bulatov, A., Jeavons, P., Krokhin, A.: Quantified constraints: Algorithms and complexity. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 58–70. Springer, Heidelberg (2003)
10. Bodirsky, M., Kára, J.: The complexity of equality constraint languages. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 114–126. Springer, Heidelberg (2006)
11. Bodirsky, M., Chen, H.: Quantified equality constraints. In: 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), Proceedings, Wroclaw, Poland, July 10-12, pp. 203–212. IEEE Computer Society, Los Alamitos (2007)
12. Bodirsky, M., Kára, J.: The complexity of temporal constraint satisfaction problems. In: Ladner, R.E., Dwork, C. (eds.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, pp. 29–38. ACM, New York (2008)
13. Charatonik, W., Wrona, M.: Tractable positive quantified constraint satisfaction problems (submitted, 2008), http://www.ii.uni.wroc.pl/~mwrona/publications/TQP.pdf
14. Bodirsky, M., Chen, H.: Qualitative temporal and spatial reasoning revisited. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 194–207. Springer, Heidelberg (2007)
15. Möhring, R.H., Skutella, M., Stork, F.: Scheduling with and/or precedence constraints. SIAM J. Comput. 33(2), 393–415 (2004)
16. Hodges, W.: A Shorter Model Theory. Cambridge University Press, Cambridge (1997)

Non-uniform Boolean Constraint Satisfaction Problems with Cardinality Constraint*

Nadia Creignou[1], Henning Schnoor[2], and Ilka Schnoor[3]

[1] LIF (CNRS UMR 6166), Université d'Aix-Marseille, 163 avenue de Luminy, F-13288 Marseille, France
creignou@lif.univ-mrs.fr
[2] Institut für Informatik, Christian-Albrechts-Universität Kiel, Christian-Albrechts-Platz 4, D-24098 Kiel, Germany
schnoor@ti.informatik.uni-kiel.de
[3] Institut für Theoretische Informatik, Universität Lübeck, Ratzeburger Allee 160, D-23538 Lübeck, Germany
schnoor@tcs.uni-luebeck.de

Abstract. We study the computational complexity of Boolean constraint satisfaction problems with cardinality constraint. A Galois connection between clones and co-clones has received a lot of attention in the context of complexity considerations for constraint satisfaction problems. This connection fails when considering constraint satisfaction problems that support in addition a cardinality constraint. We prove that a similar Galois connection, involving a weaker closure operator and partial polymorphisms, can be applied to such problems. Thus, we establish dichotomies for the decision as well as for the counting problems in Schaefer's framework.

1 Introduction

The success of Boolean constraint satisfaction problems (CSPs) is due to two features: they provide a framework in which various combinatorial problems (including NP-complete ones) can be adequately expressed, and which is practically efficient since highly optimized solvers are available. Therefore, Boolean constraint satisfaction problems are an important test-bed for questions about computational complexity and algorithms. In particular the non-uniform version, $\textsc{Csp}(\Gamma)$ has been extensively studied from the computational complexity point of view. In this context a finite set of Boolean relations Γ, called a constraint language, is fixed. An input of such a problem is a Γ-formula. Such a formula is a conjunction of "clauses", each of which consisting in an application of some relation from Γ to variables. This framework captures many well-known combinatorial problems, as for instance the famous NP-complete problem 3Sat. The complexity study of these problems started in 1978 with the seminal paper of

* Supported by the DAAD postdoc program. Work done in part while the second and third authors worked at the University of Hannover.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 109–123, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Schaefer [Sch78]. He proved a remarkable dichotomy theorem: $\textsc{Csp}(\Gamma)$ is either in P or NP-complete. Since then many other algorithmic problems related to Γ-formulas have been investigated: including counting [CH96], non-monotonic reasoning [KK03, CZ06], equivalence and isomorphism [BHRV02, BHRV04, BH05], optimization [Cre95, KSTW01, RV03], parameterized complexity [Mar05], and many others (see [CV08] for a complete survey).

All the constraints that appear in non-uniform CSPs are local ones: each applies to a fixed number of variables. However in practice one is often faced with constraints of global nature, which involve all the variables occurring in the input. Due to the wide embrace of global constraints in the constraint programming community, we believe that the computational complexity of non-uniform CSPs supporting additional global constraints is worth being investigated. To this aim, we focus here on Boolean constraint satisfaction problems that support in addition a constraint of global nature, namely a cardinality constraint. Given a Γ-formula, a feasible solution is a satisfying assignment that fulfills in addition some cardinality constraint on the number of variables set to 1. More precisely we are interested in two problems $\textsc{Bal-Csp}(\Gamma)$ and $\textsc{k-Ones}(\Gamma)$. For the balanced constraint satisfaction problem $\textsc{Bal-Csp}(\Gamma)$, the global constraint is that the assignment is balanced, i.e., it sets the same number of variables to 0 and 1. For $\textsc{k-Ones}(\Gamma)$, the requirement is that exactly k variables (where k is given in the input) are set to 1. These two global constraints are well-known in constraint programming and appear in the Global constraint catalog (see http://www.emn.fr/x-info/sdemasse/gccat/index.html). The balanced constraint also arises naturally in some optimization problems. For example Min-Bisection can be seen as Min-Cut with the restriction that the two sets of vertices have the same cardinality. Other optimization problems can be expressed as a Boolean constraint satisfaction problem where a feasible solution is a balanced assignment. Recently, there was an increased interest in optimization problems supporting an additional cardinality constraint, see e.g. [Svi01, BK05, BHM08].

Satisfiability problems with an additional cardinality constraint first appeared in [KSTW01]. The authors studied the problem $\textsc{Max-Ones}(\Gamma)$, in which a solution is a satisfying assignment that sets at least k variables to 1[1]. The problem $\textsc{k-Ones}(\Gamma)$ was already studied from the point of view of parameterized complexity in [Mar05] (our results differ from his, since in our problems k is part of the input instance). The study of the complexity of $\textsc{Bal-Csp}(\Gamma)$ and $\textsc{k-Ones}(\Gamma)$ was initiated in [BK05]. The authors identified a polynomial time case, obtained individual hardness results for specific constraint languages and conjectured a dichotomy classification. In this paper we prove that the conjecture holds. We prove a full complexity classification for the two problems $\textsc{Bal-Csp}(\Gamma)$ and $\textsc{k-Ones}(\Gamma)$. Moreover, we also tackle the corresponding counting problems and prove a dichotomy classification FP/#P-complete.

For this we use new algebraic tools. In order to obtain a complexity classification for constraint satisfaction problems, the main idea is to compare the

[1] They studied this problem as an optimization problem, and were interested in approximability properties.

so-called *expressive power* of constraint languages. Roughly speaking, given two constraint languages Γ_1 and Γ_2, if Γ_2 is more expressive than Γ_1, then any Γ_1-formula (i.e., a conjunction of Γ_1-clauses, each of which being an application of some relation from Γ_1 to variables) can be transformed into a Γ_2-formula. In the last decade, a Galois correspondence between the lattice of Boolean relations and the lattice of Boolean functions, together with Post's lattice has turned out to be one of the most successful tools to derive complexity results for Boolean constraint satisfaction problems. Indeed, this Galois correspondence relates the expressive power of a constraint language to its set of *polymorphisms*, i.e., algebraic closure properties.The structure of the polymorphism sets, so called clones, is well-known and is described by Post's lattice [Pos41]. This Galois connection gives a procedure transforming Γ_1-formulas into equivalent Γ_2-formulas. However, the newly constructed Γ_2-formulas contain additional existentially quantified variables and equality clauses can occur. Due to the additional global constraint, these features make this Galois connection unhelpful to transfer directly results from Post's classes to complexity when there is an additional cardinality constraint.

We prove that we can use a restricted closure, based on *partial polymorphisms* and studied in [Rom81]. These partial polymorphisms form a structure which is a refinement of the clone structure exhibited by Post. However, surprisingly, the complexity classification, when achieved, obeys the borders of Post's lattice.

In Section 2 we introduce the main concepts precisely, and state our results. In Section 3 we present the algebraic method that will be used to obtain the complexity classification. Due to space restrictions, this section focuses on the results needed for this paper and does not give any examples for the involved constructions. See [SS08] for details on this technique. Section 4 is then dedicated to the hardness proofs.

2 Main Result

A *logical relation* of arity k is a relation $R \subseteq \{0,1\}^k$. A *constraint* (or *constraint application*) is a formula $R(x_1, \ldots, x_k)$, where R is a logical relation of arity k and $x_1, \ldots, x_k$ are (not necessarily distinct) variables. An assignment I of truth values to the variables *satisfies* the constraint if $\big(I(x_1), \ldots, I(x_k)\big) \in R$. A *constraint language* Γ is a finite set of logical relations. A *Γ-formula* is a conjunction of constraint applications using only logical relations from Γ. With $\mathrm{Var}(\varphi)$ we denote the set of variables appearing in φ. A formula φ is satisfied by an assignment I if I satisfies all constraints in φ. The satisfiability problem for Γ-formulas is denoted by $\textsc{Csp}(\Gamma)$. Assuming a canonical order on the variables, we can regard assignments as tuples in the obvious way, and say that a formula *defines* or *expresses* the relation of its solutions.

A *balanced assignment* for φ is a truth assignment I that assigns 0 to the same number of variables as 1, that means it fulfills $|\{x \in \mathrm{Var}(\varphi) \mid I(x) = 0\} = |\{x \in \mathrm{Var}(\varphi) \mid I(x) = 1\}|$.

We study here the two following problems.

Problem:	BAL-CSP(Γ)
Input:	A Γ-formula φ
Question:	Is there a balanced assignment that satisfies φ ?
Problem:	K-ONES(Γ)
Input:	A Γ-formula φ and a number $k \in \mathbb{N}$
Question:	Is there a truth assignment setting exactly k variables to true that satisfies φ?

Additionally we look at the counting version associated with each of these problems, i.e., the question of how many "acceptable" (balanced/ with k ones) satisfying truth assignments a given Γ-formula has. These counting problems are denoted by #BAL-CSP(Γ) and #K-ONES(Γ).

Definition 2.1. *A logical relation R is* affine with width 2 *if it is definable by a conjunction of equations, each of which being either a unary clause or a 2XOR-clause, that is of the form $l_1 \oplus l_2$, where l_1, l_2 are literals and $\oplus$ is the exclusive-or operator. A constraint language Γ is affine with width 2 if every relation in Γ is affine with width 2.*

The following is our main result:

Theorem 2.2. *Let Γ be a constraint language.*

- *If Γ is affine with width 2, then* BAL-CSP(Γ) *(respectively,* K-ONES(Γ)*) is decidable in polynomial time. Otherwise it is* NP*-complete.*
- *If Γ is affine with width 2, then* #BAL-CSP(Γ) *(respectively,* #K-ONES(Γ)*) is computable in polynomial time. Otherwise it is* #P*-complete under counting reductions.*

Observe that there is an immediate parsimonious reduction from #BAL-CSP(Γ) to #K-ONES(Γ). It therefore suffices to prove polynomial-time results only for the problems K-ONES and #K-ONES, and hardness results for the problems BAL-CSP and #BAL-CSP. The polynomial side of this theorem is rather easy to prove. As suggested in [BK05] deciding the existence of a satisfying assignment of an affine with width 2 formula that sets exactly k variables to 1 can be reduced to solving an instance of the UNARY-SUBSET-SUM problem. The input of this last problem consists in a set $A = \{a_1, \ldots, a_n\}$ of positive integers and an integer B; the question is whether there exists a subset $A' \subseteq A$ such that the sum of the elements in A' is exactly B. This problem can be solved by examining $w(i, L)$, the number of subsets of $\{a_1, \ldots, a_i\}$ whose sum of elements is exactly L, for $i = 1, \ldots, n$ and $L = 1, \ldots B$. Since $w(i+1, L) = w(i, L) + w(i, L - a_{i+1})$, the quantity we are interested in, $w(n, B)$, can be computed dynamically, in polynomial time when all the integers are encoded in unary. Therefore, if Γ is affine with width 2, then #K-ONES(Γ) (and *a fortiori* #BAL-CSP(Γ)) is computable in polynomial time (details are left out due to space restrictions). In the following, in order to finish the proof of the theorem we focus on hardness results for the problem BAL-CSP(Γ) (resp. #BAL-CSP(Γ)).

3 The Weak Base Method

We now introduce the algebraic tools that our proof relies on. For more background on these notions, see [SS08].

Definition 3.1. *Let Γ be a set of logical relations.*

- *$\langle\Gamma\rangle$ is the set of relations which can be expressed as a formula of the form $\exists x_1 \dots \exists x_k \varphi$, where φ is a $(\Gamma \cup \{=\})$-formula as defined above in which (among others) the variables $x_1, \dots, x_k$ appear.*
- *$\langle\Gamma\rangle_{\nexists}$ is the set of relations which can be expressed as a $\Gamma \cup \{=\}$-formula.*
- *$\langle\Gamma\rangle_{\nexists,\neq}$ is the set of relations which can be expressed as a Γ-formula.*

Let $\Gamma_1 \subseteq \langle\Gamma_2\rangle$. Then a Γ_1-formula can be transformed into a satisfiability-equivalent Γ_2-formula. Thus, it has been proved that $\textsc{Csp}(\Gamma_1)$ can be reduced in logarithmic space to $\textsc{Csp}(\Gamma_2)$ (see [Jea98, ABI+05]). Hence the complexity of $\textsc{Csp}(\Gamma)$ depends only on $\langle\Gamma\rangle$. The set $\langle\Gamma\rangle$ is a relational clone (or a *co-clone*) Accordingly, in order to obtain a full complexity classification for the satisfiability problem we only have to study the co-clones. Interestingly, there exists a Galois correspondence between the lattice of Boolean relations (co-clones) and the lattice of boolean functions (clones) (see [Gei68, BKKR69]). This one-to-one correspondence is established through the operators Pol and Inv defined below.

Definition 3.2. Let $f\colon \{0,1\}^m \to \{0,1\}$ and $R \subseteq \{0,1\}^n$. We say that f is a *polymorphism* of R, if for all $x_1, \dots, x_m \in R$, where $x_i = (x_i[1], x_i[2], \dots, x_i[n])$, we have $\big(f\big(x_1[1], \cdots, x_m[1]\big), f\big(x_1[2], \cdots, x_m[2]\big), \dots, f\big(x_1[n], \cdots, x_m[n]\big)\big) \in R$.

If $f \in \mathrm{Pol}(R)$, we also say that R is *closed under* f, or f *preserves* R. For a set of relations Γ we write $\mathrm{Pol}(\Gamma)$ to denote the set of all polymorphisms of Γ, i.e., the set of all Boolean functions that preserve every relation in Γ. For every Γ, $\mathrm{Pol}(\Gamma)$ is a *clone*, i.e., a set of Boolean functions that contains all projections and is closed under composition. The smallest clone containing a set B of Boolean functions will be denoted by $[B]$ in the sequel (B is also called a *basis* for $[B]$). For a set B of Boolean functions, let $\mathrm{Inv}(B)$ denote the set of all *invariants* of B, i.e., the set of all Boolean relations that are preserved by every function in B. It can be observed that each $\mathrm{Inv}(B)$ is a relational clone. Thus, we may compile a full list of co-clones from the list of clones obtained by Emil Post in [Pos41]. The list of all Boolean clones with finite bases can be found e.g. in [BCRV03]. A compilation of all co-clones with simple bases is given in [BRSV05]. In the following, when discussing about bases for clones or co-clones we implicitly refer to these two lists.

Unfortunately, this Galois connection cannot help *a priori* for the study of CSPs with cardinality constraint. Indeed, existential variables and equality constraints that may occur when transforming a Γ_1-formula into a satisfiability-equivalent Γ_2-formula are problematic, as they can change the set of solutions. Therefore for these problems we have to consider the restricted closure $\langle . \rangle_{\nexists,\neq}$, which allows to translate formulas into equivalent ones.

Proposition 3.3. *Let Γ_1 and Γ_2 be constraint languages with $\Gamma_1 \subseteq \langle \Gamma_2 \rangle_{\not\equiv,\neq}$. Then*

- $\text{Bal-Csp}(\Gamma_1) \leq_m^{\log} \text{Bal-Csp}(\Gamma_2)$ *and* $\text{k-Ones}(\Gamma_1) \leq_m^{\log} \text{k-Ones}(\Gamma_2)$,
- $\#\text{Bal-Csp}(\Gamma_1) \leq_!^{\log} \#\text{Bal-Csp}(\Gamma_2)$ *and* $\#\text{k-Ones}(\Gamma_1) \leq_!^{\log} \#\text{k-Ones}(\Gamma_2)$,

where $\leq_m^{\log}$ denotes a logspace many-one reduction and $\leq_!^{\log}$ a parsimonious (i.e., preserving the number of witnesses) logspace many-one reduction.

The main strategy to prove that for some class of constraint languages, the problems we consider are NP- or #P-hard is to prove that for every language Γ in this class, $\langle \Gamma \rangle_{\not\equiv,\neq}$ contains a language Γ' for which the problem is hard. Proposition 3.3 then implies the result for every language in the class. In [SS08], Schnoor and Schnoor established techniques that allow to prove results in this direction. We briefly explain the main definitions and results.

The main tool is the notion of a *weak base*. Note that in that paper, a different (but proven to be equivalent) definition was given.

Definition 3.4 ([SS08]). *Let $\mathcal{C}$ be a clone. A* weak base *for* $\text{Inv}(\mathcal{C})$ *is a constraint language Γ such that: (i)* $\text{Pol}(\Gamma) = \mathcal{C}$*, (ii) for any constraint language Γ' with* $\text{Pol}(\Gamma') = \mathcal{C}$*, it follows that $\Gamma \subseteq \langle \Gamma' \rangle_{\not\equiv}$.*

Since for the balanced satisfiability problem, we need to consider the stricter closure operator $\langle . \rangle_{\not\equiv,\neq}$, we need an additional technical notion. In the following, we consider relations as matrices, where the rows of the matrix correspond to the tuples of the relation (technically, for uniqueness, we need to fix an order on the rows, for example lexicographical ordering). An n-ary relation R is *irredundant* if R, considered as a matrix, does not contain two identical columns, and if there is no i, $1 \leq i \leq n$, such that the value of the i^{th} variable is unconstrained. A set of relations Γ is irredundant if every relation in Γ is irredundant. For representing irredundant relations, equality clauses are not needed. As a corollary the following proposition holds:

Proposition 3.5 ([SS08]). *Let Γ be an irredundant weak base for a co-clone* $\text{Inv}(\mathcal{C})$*. If Γ' is a constraint language with* $\text{Pol}(\Gamma') = \mathcal{C}$*, then $\Gamma \subseteq \langle \Gamma' \rangle_{\not\equiv,\neq}$.*

We now explain how to construct weak bases. For a set of Boolean functions $\mathcal{F}$, the *$\mathcal{F}$-closure* of a relation R, denoted by $\mathcal{F}(R)$, is the minimal superset of R that is closed under every function from $\mathcal{F}$. This relation can be obtained from R by repeatedly applying all functions from $\mathcal{F}$, and adding the result to R. We say R is an *$\mathcal{F}$-core* of $\mathcal{F}(R)$.

Definition 3.6 ([SS08]). *Let $\mathcal{C}$ be a clone. Then* $\text{Inv}(\mathcal{C})$ *has* core-size s *if there is a relation R such that $\langle R \rangle = \text{Inv}(\mathcal{C})$ and R has a $\mathcal{C}$-core with cardinality s.*

The relation COLS_s is defined to be the 2^s-ary relation of cardinality s such that the columns of COLS_s contain every possible s-ary binary vector (the order is irrelevant, we fix an arbitrary one in order for the notion to be well-defined). In the following by $\text{COLS}_s(l, -)$ we denote the l-th row vector of COLS_s, and by $\text{COLS}_s(-, k)$ its k-th column vector.

Theorem 3.7 ([SS08]). *Let $\mathcal{C}$ be a clone. Suppose that* $\mathrm{Inv}(\mathcal{C})$ *has core-size s. Then the relation* $\mathcal{C}\,(\mathrm{COLS}_s)$ *is a weak base of* $\mathrm{Inv}(\mathcal{C})$.

With this theorem one can construct weak bases for all Boolean co-clones for which we know finite bases (since finite bases give us core-sizes). This fits our purpose. Indeed, there are only 8 clones that have no finite basis, namely S_0, S_{01}, S_{02}, S_{00} and S_1, S_{11}, S_{12}, S_{10}. These clones $\mathcal{C}$ are exactly the ones for which there exists no finite constraint language Γ such that $\langle\Gamma\rangle = \mathrm{Inv}(\mathcal{C})$ (see [BRSV05]), and therefore will not be involved in our study. An explicit example for the construction of Theorem 3.7 is given in the proof of Theorem 4.8.

4 Proofs of Hardness Results

4.1 Another Statement of the Main Result

As discussed above the Galois correspondence between relational co-clones and Post's classes cannot help *a priori* for our problems. However, it turns out that the complexity classification, when achieved, obeys the border among co-clones (see Figure 1), and so the Galois connection holds *a posteriori*. The clones corresponding to our polynomial-time cases are highlighted in Figure 1.

Corollary 4.1. *Let Γ be a constraint language.*

- *If* $\Gamma \subseteq \mathrm{Inv}(\mathrm{D}_1)$, *then* Bal-Csp($\Gamma$) *(respectively,* k-Ones(Γ)*) is decidable in polynomial time. Otherwise it is* NP*-complete.*
- *If* $\Gamma \subseteq \mathrm{Inv}(\mathrm{D}_1)$, *then* #Bal-Csp($\Gamma$) *(respectively,* #k-Ones(Γ)*) is computable in polynomial time. Otherwise it is* #P*-complete under counting reductions.*

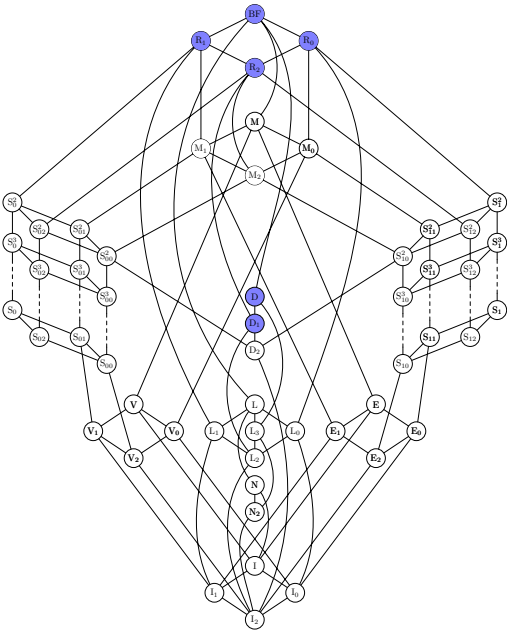


Fig. 1. Post's lattice

Our main theorem can be reformulated as above since it is well known that $\mathrm{Inv}(\mathrm{D}_1)$ is the set of all affine with width two relations (see e.g., [CKZ08]). Thus, Theorem 2.2 will be proved by an exhaustive examination of the clones in Post's lattice. As mentioned before, it suffices to prove hardness results for the balanced versions of our problems. These results are organized as follows: In Section 4.2, we prove hardness for a set of individual

relations. Section 4.3 uses these results and our algebraic techniques to extend these results to entire co-clones, without the need to construct a concrete weak base for these. Section 4.4 then contains the results for the remaining co-clones, where the proof requires to study individual weak bases.

4.2 Some Basic Hardness Results

First we will take advantage of the symmetry of Post's lattice. The dual relation of a relation R, is given by $\text{dual}(R) = \{(1-a_1, \ldots, 1-a_n) \ : \ (a_1, \ldots, a_n) \in R\}$. If Γ is a set of relations, then $\text{dual}(\Gamma) = \{\text{dual}(R) \ : \ R \in \Gamma\}$.

Proposition 4.2. *For any constraint language Γ,* $\#\text{Bal-Csp}(\text{dual}(\Gamma)) \leq_!^{\log} \#\text{Bal-Csp}(\Gamma)$.

In the following we will use specific relations: $C_0 = \{0\}$, $C_1 = \{1\}$, 1-in-3 $= \{001, 010, 100\}$, Imp $= \{00, 01, 11\}$, $\text{Or}^2 = \{01, 10, 11\}$, $\text{Odd}^2 = \{01, 10\}$ and $\text{Odd}^3 = \{001, 010, 100, 111\}$. Now, we establish hardness results which will serve as base problems for the following hardness proofs.

Lemma 4.3. $\text{Bal-Csp}(\text{Imp})$, $\text{Bal-Csp}(\text{Or}^2)$, *and* $\text{Bal-Csp}(\text{Odd}^3)$ *are* NP*-hard. Their corresponding counting problems are* #P*-hard under counting reductions.*

Proof. Hardness of $\text{Bal-Csp}(\text{Or}^2)$ and $\text{Bal-Csp}(\text{Odd}^3)$ was shown in [BK05]. In order to prove NP-hardness of $\text{Bal-Csp}(\text{Imp})$, we consider the following NP-complete problem (see [GJ79]).

Problem:	k-Closure
Input:	a directed graph $G = (V, E)$ and $k \in \mathbb{N}$
Question:	Is there a $V' \subseteq V$ such that $\lvert V' \rvert = k$ and for all $(u, v) \in E$ it holds $u \in V'$ or $v \notin V'$?

We show k-Closure $\leq_m^{\log}$ $\text{Bal-Csp}(\text{Imp})$. Let $G = (V, E)$ be a directed graph and $k \in \mathbb{N}$. Let $n = |V|$. We construct an {Imp}-formula with variables $X = V \cup \{t_1, \ldots, t_k, f_1, \ldots, f_{n-k}\}$, where $t_1, \ldots, t_k, f_1, \ldots, f_{n-k}$ are all distinct variables and not from V. We set

$$\varphi = \bigwedge_{(u,v)\in E} \text{Imp}(u, v) \wedge \bigwedge_{i=1}^{k} \bigwedge_{x\in X} \text{Imp}(x, t_i) \wedge \bigwedge_{i=1}^{n-k} \bigwedge_{x\in X} \text{Imp}(f_i, x).$$

Observe that a balanced solution for φ sets all t_i's to 1 and all f_i's to 0. Now it is easy to check that φ has a balanced solution if and only if G has a k-closure (the balanced assignment is obtained by assigning 0 to each $v \in V'$).

We now study the counting problems. For #P-hardness of $\#\text{Bal-Csp}(\text{Odd}^3)$ it suffices to show $\#\text{Csp}(\text{1-in-3}) \leq_!^{\log} \#\text{Bal-Csp}(\text{Odd}^3)$, because $\#\text{Csp}(\text{1-in-3})$ is hard for #P [CH96]. Note that, since Csp(1-in-3) is an NP-complete problem [Sch78], the following reduction is also an alternative proof for the NP-hardness

of BAL-CSP(Odd^3). Let $\varphi = \bigwedge_{i=1}^{n} 1\text{-in-}3(x_i, y_i, z_i)$. We construct an Odd^3-formula using additionally to the variables appearing in φ the following new and distinct variables: a_i, b_i, c_i, d_i for every $1 \leq i \leq n$; t^i, f^i for every $1 \leq i \leq k$ where $k = 2\,|\mathrm{Var}(\varphi)| + 4n$; and v' for every $v \in \varphi$. We set:

$$\begin{aligned}\varphi' = \bigwedge_{i=1}^{n} \big\{ & \mathrm{Odd}^3(x_i, y_i, z_i) \wedge \mathrm{Odd}^3(d_i, d_i, d_i) \wedge \\ & \mathrm{Odd}^3(d_i, x_i, a_i) \wedge \mathrm{Odd}^3(d_i, y_i, b_i) \wedge \mathrm{Odd}^3(d_i, z_i, c_i) \,\big\} \\ \wedge \bigwedge_{i=1}^{k} & \mathrm{Odd}^3(t^i, t^i, t^i) \wedge \mathrm{Odd}^3(t^i, f^i, f^1) \wedge \bigwedge_{v \in \mathrm{Var}(\varphi)} \mathrm{Odd}^3(f^1, v, v').\end{aligned}$$

Observe that $|\mathrm{Var}(\varphi')| = 3k$. If I is a satisfying assignment for φ', then necessarily $I(d_i) = I(t^i) = 1$ and $I(a_i) = I(x_i)$, $I(b_i) = I(y_i)$, $I(c_i) = I(z_i)$ and $I(f^i) = (f^1)$. If in addition I is balanced, then $I(f^i) = 0$, $I(v') = 1 - I(v)$ for all variable v in $\mathrm{Var}(\varphi)$, and for no clause $\mathrm{Odd}^3(x_i, y_i, z_i)$ one can have $I(x_i) = I(y_i) = I(z_i) = 1$ (otherwise I will set more than $2n$ variables to 1 among the a_i, b_i, c_i, d_i). As a consequence I satisfies the constraint $1\text{-in-}3(x_i, y_i, z_i)$. From these observations, one can check that there is a one-to-one correspondence between solutions of φ and balanced solutions of φ'. So we showed $\#\mathrm{CSP}(1\text{-in-}3) \leq_!^{\log} \#\mathrm{BAL\text{-}CSP}(\mathrm{Odd}^3)$, which completes the proof.

One can show hardness of $\#\mathrm{BAL\text{-}CSP}(\mathrm{Imp})$ and $\#\mathrm{BAL\text{-}CSP}(\mathrm{Or}^2)$ by proving $\#\mathrm{CSP}(\mathrm{Imp}) \leq_!^{\log} \#\mathrm{BAL\text{-}CSP}(\mathrm{Imp})$ and $\#\mathrm{CSP}(\mathrm{Or}^2) \leq_!^{\log} \#\mathrm{BAL\text{-}CSP}(\mathrm{Or}^2)$. The results then follow since $\#\mathrm{CSP}(\mathrm{Imp})$ and $\#\mathrm{CSP}(\mathrm{Or}^2)$ were shown to be #P-complete in [CH96]. □

4.3 Hardness Results with Unified Proofs

Now we start to look at constraint languages. The first result covers all constraint languages that generate Inv(M), Inv(V), Inv(E), or Inv(I). One can show that for these Γ, $\mathrm{Imp} \in \langle\Gamma\rangle_{\nexists,\neq}$. Hardness for decision and counting now follows from Lemma 4.3 and Proposition 3.3.

Proposition 4.4. *Let Γ be a constraint language such that $\langle\Gamma\rangle \subseteq \mathrm{Inv(I)}$ and $\langle\Gamma\rangle \nsubseteq \mathrm{Inv}(\mathrm{N}_2)$. Then* BAL-CSP($\Gamma$) *is* NP*-hard and* $\#$BAL-CSP(Γ) *is* #P*-hard under counting reductions.*

In the rest of this section we work with weak bases, however we do not need to compute any concrete weak base and we see that weak bases for the above co-clones share some properties.

The next theorem deals with constraint languages that generate one of the following co-clones: $\mathrm{Inv}(\mathrm{M}_1)$, $\mathrm{Inv}(\mathrm{V}_1)$, $\mathrm{Inv}(\mathrm{E}_1)$, $\mathrm{Inv}(\mathrm{S}_{01}^m)$.

Theorem 4.5. *Let Γ be a constraint language such that* $\mathrm{Inv}(\mathrm{M}_1) \subseteq \langle\Gamma\rangle \subsetneq \mathrm{Inv}(\mathrm{I}_1)$. *Then* Bal-Csp($\Gamma$) *is* NP*-hard and* #Bal-Csp(Γ) *is* #P*-hard under counting reductions.*

Proof. Let T-Imp be the relation $C_1 \times \mathrm{Imp}$. First, we prove Bal-Csp(Imp) $\leq^{\log}_{!}$ Bal-Csp(T-Imp). Let $\varphi = \bigwedge_{i=1}^{n} \mathrm{Imp}(x_i, y_i)$. Let t and f be new and distinct variables. We set $\varphi' = \bigwedge_{i=1}^{n} \text{T-Imp}(t, x_i, y_i) \wedge \text{T-Imp}(t,t,t) \wedge \bigwedge_{x \in \mathrm{Var}(\varphi)} \text{T-Imp}(t, f, x)$. If I is a satisfying assignment of φ', then $I(t) = 1$. If in addition I is balanced, then $I(f) = 0$. Hence, there is a one-to-one correspondence between balanced solutions of φ and balanced solutions of φ'. Second, we will show that T-Imp $\in \langle\Gamma\rangle_{\not\equiv,\neq}$. The proposition then follows from Lemma 4.3 and Proposition 3.3.

Let s be a core-size of $\langle\Gamma\rangle$ and let $R = \mathrm{Pol}(\Gamma)(\mathrm{COLS}_s)$. According to Theorem 3.7 it holds that $\{R\}$ is a weak base of $\langle\Gamma\rangle$ which implies $\langle R\rangle_{\not\equiv} \subseteq \langle\Gamma\rangle_{\not\equiv}$. It is enough to show that T-Imp $\in \langle R\rangle_{\not\equiv}$, then, since T-Imp is irredundant, it follows T-Imp $\in \langle\Gamma\rangle_{\not\equiv,\neq}$. We distinguish two cases: $\langle\Gamma\rangle \subseteq \mathrm{Inv}(\mathrm{V}_1)$ and $\langle\Gamma\rangle \not\subseteq \mathrm{Inv}(\mathrm{V}_1)$. Let us first suppose that $\langle\Gamma\rangle \subseteq \mathrm{Inv}(\mathrm{V}_1)$. Let S be the Boolean relation defined by

$$S(t,x,y) \equiv R(x, \underbrace{y,\ldots,y}_{2^{s-1}-1}, \underbrace{t,\ldots,t}_{2^{s-1}}).$$

We show $S =$ T-Imp. Since $\langle\Gamma\rangle \subseteq \mathrm{Inv}(\mathrm{I}_1)$, it holds that $c_1 \in \mathrm{Pol}(\Gamma)$ and therefore $(1,\ldots,1) \in R$ and $(1,1,1) \in S$. Because $\vee \in \mathrm{V}_1 \subseteq \mathrm{Pol}(\Gamma)$ it follows that the nested application of $\vee$ to all tuples of COLS_s is in R, i.e., $(0,1,\ldots,1) = \mathrm{COLS}_s(1,-) \vee \cdots \vee \mathrm{COLS}_s(s,-) \in R$. This means $(1,0,1) \in S$. Since $(\underbrace{0,\ldots,0}_{2^{s-1}}, \underbrace{1,\ldots,1}_{2^{s-1}}) = \mathrm{COLS}_s(1,-) \in R$, it holds that $(1,0,0) \in S$, hence T-Imp $\subseteq S$.

Note that $\mathrm{Pol}(R)$ contains only functions which are both monotone[2] and 1-reproducing[3] because $\mathrm{Pol}(R) \subseteq \mathrm{M}_1$, and M_1 contains exactly the functions with these two properties. Since $\mathrm{COLS}_s(-, 2^s) = (1,\ldots,1)$ and since all polymorphisms of Γ are 1-reproducing, it follows $R(-,2^s) = (1,\ldots,1)$ and therefore it holds for all $a,b \in \{0,1\}$ that $(0,a,b) \notin S$.

Finally assume $(1,1,0) \in S$. Then $u = (1, \underbrace{0,\ldots,0}_{2^{s-1}-1}, \underbrace{1,\ldots,1}_{2^{s-1}}) \in R$. By construction of R it follows that there is an s-ary Boolean function $g \in \mathrm{Pol}(\Gamma)$ such that $g(\mathrm{COLS}_s(1,-),\ldots,\mathrm{COLS}_s(s,-)) = u$. It holds that g is not monotone because $g(0,\ldots,0) = u[1] = 1$ and $g(0,\ldots,0,1) = u[2] = 0$. Since every function from $\mathrm{Pol}(\Gamma)$ is monotone, this is a contradiction. Hence T-Imp $= S$ and therefore T-Imp $\in \langle R\rangle_{\not\equiv} \subseteq \langle\Gamma\rangle_{\not\equiv}$.

[2] An n-ary Boolean function f is called *monotone* if for all $\alpha, \beta \in \{0,1\}^n$ holds: If $\alpha \leq \beta$ then $f(\alpha) \leq f(\beta)$.

[3] f is called 1-*reproducing* if $f(1,\ldots,1) = 1$.

The case where $\langle \Gamma \rangle \not\subseteq \mathrm{Inv}(\mathrm{V}_1)$ can be handled in a similar way in considering the Boolean relation defined by $S(t,x,y) \equiv R(\underbrace{x,\ldots,x}_{2^{s-1}},\underbrace{y,\ldots,y}_{2^{s-1}-1},t)$. □

We look at the co-clones $\mathrm{Inv}(\mathrm{M}_2)$, $\mathrm{Inv}(\mathrm{V}_2)$, $\mathrm{Inv}(\mathrm{E}_2)$, and $\mathrm{Inv}(\mathrm{S}_{00}^m)$ for $m \geq 2$ next.

Theorem 4.6. *Let Γ be a constraint language such that $\mathrm{Inv}(\mathrm{M}_2) \subseteq \langle \Gamma \rangle \subseteq \mathrm{Inv}(\mathrm{V}_2)$. Then* Bal-Csp($\Gamma$) *is* NP*-hard and* #Bal-Csp(Γ) *is* #P*-hard under counting reductions.*

Proof. The proof is similar to the one above. Consider the relation TF-Imp $= C_1 \times C_0 \times \mathrm{Imp}$. Similarly as in Theorem 4.5 we prove that #Bal-Csp(Imp) $\leq_!^{\log}$ #Bal-Csp(TF-Imp) and show that TF-Imp $\in \langle \Gamma \rangle_{\not\exists,\neq}$. For this latter part, suppose that $s \geq 2$ is a core-size of $\langle \Gamma \rangle$. Let $R = \mathrm{Pol}(\Gamma)(\mathrm{COLS}_s)$, the arity of R is $n = 2^s$. One can check that the Boolean relation defined by $S(t,f,x,y) = R(f,\underbrace{x,\ldots,x}_{\frac{n}{4}-1},\underbrace{y,\ldots,y}_{\frac{n}{4}},\underbrace{t,\ldots,t}_{\frac{n}{2}})$ verifies $S =$ TF-Imp. □

The following theorem covers the cases $\mathrm{Inv}(\mathrm{S}_0^m)$ and $\mathrm{Inv}(\mathrm{S}_{02}^m)$ for all $m \geq 2$. The proof follows the same lines as the proofs for Theorems 4.5 and 4.6.

Theorem 4.7. *Let Γ be a constraint language such that $\langle \Gamma \rangle = \mathrm{Inv}(\mathrm{S}_{02}^m)$ or $\langle \Gamma \rangle = \mathrm{Inv}(\mathrm{S}_0^m)$ for some natural number $m \geq 2$. Then* Bal-Csp(Γ) *is* NP*-hard and* #Bal-Csp(Γ) *is* #P*-hard under counting reductions.*

4.4 Hardness Results with Non-unified Proofs

In this section we work with concrete irredundant weak bases in all proofs.

Theorem 4.8. *Let Γ be a constraint language such that $\langle \Gamma \rangle = \mathrm{Inv}(\mathrm{D}_2)$. Then* Bal-Csp($\Gamma$) *is* NP*-hard and* #Bal-Csp(Γ) *is* #P*-hard under counting reductions.*

Proof. Let *maj* be the ternary majority function defined by $maj(a,b,c) = 1$ if and only if $a+b+c \geq 2$. It holds that $[\{maj\}] = \mathrm{D}_2$ and that 3 is a core-size of $\mathrm{Inv}(\mathrm{D}_2)$. It follows from Theorem 3.7 that $R = maj(\mathrm{COLS}_3)$ is a weak base of $\mathrm{Inv}(\mathrm{D}_2)$. It can be verified that

$$R = \begin{pmatrix} 0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,1\,0\,1\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,1\,0\,1\,0\,1\,0\,1 \end{pmatrix}$$

Note that the second row is generated by the coordinatewise application of *maj* to the other three rows, which form COLS_3. Clearly, R is irredundant. According to Proposition 3.5 it holds $\langle R \rangle_{\not\exists,\neq} \subseteq \langle \Gamma \rangle_{\not\exists,\neq}$.

We define Boolean relations S and T in the following way:

$$S(t,f,x,y) = R(f,f,x,x,y,y,t,t),\ T(t,f,v,w,x,y) = R(f,f,v,w,y,x,t,t).$$

It follows $\{S,T\} \subseteq \langle R\rangle_{\not\equiv,\neq} \subseteq \langle \Gamma\rangle_{\not\equiv,\neq}$. Therefore, according to Proposition 3.3, it holds $\#\text{Bal-Csp}(\{S,T\}) \leq^{\log}_{!} \#\text{Bal-Csp}(\Gamma)$.

The following equivalences can be verified:

$$S(t,f,x,y) \equiv \mathrm{C}_1(t) \wedge \mathrm{C}_0(f) \wedge \mathrm{Odd}^2(x,y)$$

$$T(t,f,v,w,x,y) \equiv \mathrm{C}_1(t) \wedge \mathrm{C}_0(f) \wedge \mathrm{Imp}(v,w) \wedge \mathrm{Odd}^2(v,x) \wedge \mathrm{Odd}^2(w,y)$$

We show that $\#\text{Bal-Csp}(\mathrm{Imp}) \leq^{\log}_{!} \#\text{Bal-Csp}(\{S,T\})$. Let $\varphi = \bigwedge_{i=1}^{n} \mathrm{Imp}(x_i,y_i)$ be an Imp-formula. We construct an $\{S,T\}$-formula: let t, f and z', z'' for every $z \in \mathrm{Var}(\varphi)$ be new and distinct variables. We set $\varphi' = \bigwedge_{i=1}^{n} T(t,f,x_i,y_i,x_i',y_i') \wedge S(t,f,x_i',x_i'') \wedge S(t,f,y_i',y_i'')$, then

$$\varphi' \equiv \varphi \wedge \bigwedge_{i=1}^{n} \mathrm{Imp}(x_i,y_i) \wedge \bigwedge_{z\in\varphi} \mathrm{Odd}^2(z,z') \wedge \mathrm{Odd}^2(z',z'') \wedge \mathrm{C}_1(t) \wedge \mathrm{C}_0(f)$$

One can check that there is a one-to-one correspondence between balanced solutions of φ and the balanced solutions of φ'. Hence,

$$\#\text{Bal-Csp}(\mathrm{Imp}) \leq^{\log}_{!} \#\text{Bal-Csp}(\{S,T\}) \leq^{\log}_{!} \#\text{Bal-Csp}(\Gamma).$$

Due to Lemma 4.3 this completes the proof. □

We now cover the cases $\mathrm{Inv(L)}$, $\mathrm{Inv(L_1)}$, $\mathrm{Inv(L_2)}$ and $\mathrm{Inv(L_3)}$.

Theorem 4.9. *Let Γ be a constraint language such that $\langle\Gamma\rangle \in \{\mathrm{Inv(L)}, \mathrm{Inv(L_1)}, \mathrm{Inv(L_2)}, \mathrm{Inv(L_3)}\}$. Then* Bal-Csp($\Gamma$) *is* NP*-hard and* $\#$Bal-Csp(Γ) *is* $\#$P*-hard under counting reductions.*

Proof. We make a case distinction. The proofs in the different cases are very similar. We give here as an example the proof in the case $\langle\Gamma\rangle = \mathrm{Inv(L)}$. The co-clone $\mathrm{Inv(L)}$ has 2 as a core-size, therefore $R = \mathrm{L(COLS_2)}$ is weak base of $\mathrm{Inv(L)}$. It can be verified that $R = \mathrm{Even}^4 = \{(a_1,a_2,a_3,a_4) \ : \ \sum_{i=1}^{4} a_i \equiv 0(2)\}$.

Since Even^4 is obviously irredundant it follows from Proposition 3.5 that $\mathrm{Even}^4 \in \langle\Gamma\rangle_{\not\equiv,\neq}$. Hence $\#\text{Bal-Csp}(\mathrm{Even}^4) \leq^{\log}_{!} \#\text{Bal-Csp}(\Gamma)$ due to Proposition 3.3. Now we show there is a counting reduction from $\#\text{Bal-Csp}(\mathrm{Odd}^3)$ to $\#\text{Bal-Csp}(\mathrm{Even}^4)$, thus completing the proof. Let $\varphi = \bigwedge_{i=1}^{n} \mathrm{Odd}^3(x_i,y_i,z_i)$ be an Odd^3-formula. Let $k = |\mathrm{Var}(\varphi)|$ and let $t, t_1, \ldots, t_k, f, f_1, \ldots, f_k$ be new and distinct variables. We set:

$$\varphi' = \bigwedge_{i=1}^{n} \mathrm{Even}^4(t,x_i,y_i,z_i) \wedge \bigwedge_{i_1}^{k} \mathrm{Even}^4(t,t,t,t_i) \wedge \mathrm{Even}^4(f,f,f,f_i).$$

It can be verified that φ' has exactly twice as many balanced solutions as φ. □

Finally, only four co-clones remain to be examined.

Theorem 4.10. *Let Γ be a constraint language with $\langle\Gamma\rangle \in \{\mathrm{Inv}(\mathrm{I}_2), \mathrm{Inv}(\mathrm{I}_0), \mathrm{Inv}(\mathrm{N}), \mathrm{Inv}(\mathrm{N}_2)\}$. Then* Bal-Csp($\Gamma$) *is* NP*-complete and* #Bal-Csp(Γ) *is complete for* #P *under counting reductions.*

Proof. We make a case distinction. The proofs again are very similar, but not obviously unifiable. As an example, let us deal with the case $\langle\Gamma\rangle = \mathrm{Inv}(\mathrm{I}_2)$.

The co-clone $\mathrm{Inv}(\mathrm{I}_2)$ has 3 as a core-size, therefore $R = \mathrm{I}_2(\mathrm{COLS}_3)$ is a weak base for $\mathrm{Inv}(\mathrm{I}_2)$ according to Theorem 3.7. Because $[\{id\}] = \mathrm{I}_2$, it holds that $R = \mathrm{COLS}_3$. That means

$$R = \begin{pmatrix} 0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,1\,0\,1\,0\,1\,0\,1 \end{pmatrix}.$$

It can be verified that the following equivalence is true:

$$R(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8) \equiv \text{1-in-3}(x_2,x_3,x_5) \wedge \mathrm{C}_0(x_1) \wedge \mathrm{C}_1(x_8) \\ \wedge \mathrm{Odd}^2(x_2,x_7) \wedge \mathrm{Odd}^2(x_3,x_6) \wedge \mathrm{Odd}^2(x_4,x_5)$$

Since R is irredundant, it follows $R \in \langle\Gamma\rangle_{\not\equiv,\neq}$ from Proposition 3.5. Therefore we have $\#\text{Bal-Csp}(R) \leq_!^{\log} \#\text{Bal-Csp}(\Gamma)$ due to Proposition 3.3.

It is known that Csp(1-in-3) is NP-hard [Sch78] and #Csp(1-in-3) is hard for #P under parsimonious reductions [CH96]. Hence, showing $\#\text{Csp(1-in-3)} \leq_!^{\log} \#\text{Bal-Csp}(R)$ completes the proof. Let $\varphi = \bigwedge_{i=1}^{n} \text{1-in-3}(x_i,y_i,z_i)$, and let f, t and v' for every $v \in \mathrm{Var}(\varphi)$ be new and distinct variables. We define the R-formula $\varphi' = \bigwedge_{i=1}^{n} R(f,x_i,y_i,z_i',z_i,y_i',x_i',t)$. According to the above it holds $\varphi' \equiv \varphi \wedge \bigwedge_{v\in\mathrm{Var}(\varphi)} \mathrm{Odd}^2(v,v') \wedge \mathrm{C}_0(f) \wedge \mathrm{C}_1(t)$. Obviously every balanced solution of φ' satisfies φ as well and every solution of φ can be extended uniquely to a balanced solution for φ', thus completing the proof. □

5 Conclusion

We have obtained complete complexity classifications for constraint satisfaction problems that mix local constraints with a global one. We have demonstrated that the weak base method is indeed a useful tool in order to get complexity results for these hybrid CSPs. Our contribution is twofold. On the one hand, our results represent a first encouraging step in the study of global constraints in the framework of non-uniform CSPs. A systematic treatment of global constraints will require an appropriate framework and to develop adequate algebraic tools.

It is somewhat surprising that for the two global constraints considered in this paper, namely balanced solutions and solutions with a variable number of 1s, we achieve the same complexity classification. This is unexpected, since being able to specify the number of ones required in the solution as part of the input seems to be a much stronger requirement than only to demand that the solutions are balanced. The complexity remains the same even if we consider

the counting versions of these problems. This suggests that a comparison of the expressive power and related complexity of different global constraints might be very interesting.

On the other hand, our work shows an application of a new Galois connection for studying the complexity of constraint satisfaction problems. This is interesting on its own. This illuminates the potential of this new Galois connection and hopefully will popularize it.

Finally, as we said in the introduction, balanced assignments arise naturally in many optimization problems. For this reason, as discussed in [BK05], it is natural to investigate the approximability of the balanced optimization problem, Bal-Max-Csp(Γ). The classification of the approximability of this problem is still an open question. We believe that if such a non-trivial complete classification can be achieved, then it will not follow Post's lattice (as it is already proved for the Max-Csp(Γ) problem with no balance requirement, see [Cre95, CV08]).

Acknowledgment

We thank the anonymous referees for helpful comments and corrections.

References

[ABI+05] Allender, E., Bauland, M., Immerman, N., Schnoor, H., Vollmer, H.: The complexity of satisfiability problems: Refining Schaefer's theorem. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 71–82. Springer, Heidelberg (2005)

[BCRV03] Böhler, E., Creignou, N., Reith, S., Vollmer, H.: Playing with Boolean blocks, part I: Post's lattice with applications to complexity theory. ACM-SIGACT Newsletter 34(4), 38–52 (2003)

[BH05] Bauland, M., Hemaspaandra, E.: Isomorphic implication. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 119–130. Springer, Heidelberg (2005); Theory of Computing Systems (to appear)

[BHM08] Bläser, M., Heynen, T., Manthey, B.: Adding cardinality constraints to integer programs with applications to maximum satisfiability. Information Processing Letters 105, 194–198 (2008)

[BHRV02] Böhler, E., Hemaspaandra, E., Reith, S., Vollmer, H.: Equivalence and isomorphism for Boolean constraint satisfaction. In: Bradfield, J.C. (ed.) CSL 2002 and EACSL 2002. LNCS, vol. 2471, pp. 412–426. Springer, Heidelberg (2002)

[BHRV04] Böhler, E., Hemaspaandra, E., Reith, S., Vollmer, H.: The complexity of Boolean constraint isomorphism. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 164–175. Springer, Heidelberg (2004)

[BK05] Bazgan, C., Karpinski, M.: On the complexity of global constraint satisfaction. In: Deng, X., Du, D. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 624–633. Springer, Heidelberg (2005)

[BKKR69] Bodnarchuk, V.G., Kalužnin, L.A., Kotov, V.N., Romov, B.A.: Galois theory for Post algebras I, II. Cybernetics 5, 243–252, 531–539 (1969)

[BRSV05] Böhler, E., Reith, S., Schnoor, H., Vollmer, H.: Bases for Boolean co-clones. Information Processing Letters 96, 59–66 (2005)

[CH96] Creignou, N., Hermann, M.: Complexity of generalized satisfiability counting problems. Information and Computation 125, 1–12 (1996)

[CKZ08] Creignou, N., Kolaitis, P., Zanuttini, B.: Structure identification for Boolean relations and plain bases for co-clones. Journal of Computer and System Sciences (in press, 2008)

[Cre95] Creignou, N.: A dichotomy theorem for maximum generalized satisfiability problems. Journal of Computer and System Sciences 51, 511–522 (1995)

[CV08] Creignou, N., Vollmer, H.: Boolean constraint satisfaction problems: when does Post's lattice help? In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. Springer, Heidelberg (to appear, 2008)

[CZ06] Creignou, N., Zanuttini, B.: A complete classification of the complexity of propositional abduction. SIAM Journal on Computing 36(1), 207–229 (2006)

[Gei68] Geiger, D.: Closed systems of functions and predicates. Pac. J. Math. 27(2), 228–250 (1968)

[GJ79] Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. Freeman, New York (1979)

[Jea98] Jeavons, P.G.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200, 185–204 (1998)

[KK03] Kirousis, L.M., Kolaitis, P.G.: The complexity of minimal satisfiability problems. Information and Computation 187(1), 20–39 (2003)

[KSTW01] Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM Journal on Computing 30, 1863–1920 (2001)

[Mar05] Marx, D.: Parameterized complexity of constraint satisfaction problems. Computational Complexity 14(2), 153–183 (2005)

[Pos41] Post, E.L.: The two-valued iterative systems of mathematical logic. Annals of Mathematical Studies 5, 1–122 (1941)

[Rom81] Romov, B.A.: The algebras of partial functions and their invariants. Cybernetics and Systems Analysis 17(2), 157–167 (1981)

[RV03] Reith, S., Vollmer, H.: Optimal satisfiability for propositional calculi and constraint satisfaction problems. Information and Computation 186(1), 1–19 (2003)

[Sch78] Schaefer, T.J.: The complexity of satisfiability problems. In: Proccedings 10th Symposium on Theory of Computing, pp. 216–226. ACM Press, New York (1978)

[SS08] Schnoor, H., Schnoor, I.: Partial polymorphisms and constraint satisfaction problems. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. Springer, Heidelberg (to appear, 2008)

[Svi01] Sviridenko, M.I.: Best possible approximation algorithm for MAX-SAT with cardinality constraint. Algorithmica 30(3), 398–405 (2001)

Fractional Collections with Cardinality Bounds, and Mixed Linear Arithmetic with Stars

Ruzica Piskac and Viktor Kuncak

LARA - I&C - EPFL
emails: firstname.lastname@epfl.ch
INR 318, Station 15, CH-1015 Lausanne, Switzerland

Abstract. We present decision procedures for logical constraints involving collections such as sets, multisets, and fuzzy sets. Element membership in our collections is given by characteristic functions from a finite universe (of unknown size) to a user-defined subset of rational numbers. Our logic supports standard operators such as union, intersection, difference, or any operation defined pointwise using mixed linear integer-rational arithmetic. Moreover, it supports the notion of cardinality of the collection, defined as the sum of occurrences of all elements. Deciding formulas in such logic has applications in software verification.

Our decision procedure reduces satisfiability of formulas with collections to satisfiability of formulas in an extension of mixed linear integer-rational arithmetic with a "star" operator. The star operator computes the integer cone (closure under vector addition) of the solution set of a given formula. We give an algorithm for eliminating the star operator, which reduces the problem to mixed linear integer-rational arithmetic. Star elimination combines naturally with quantifier elimination for mixed integer-rational arithmetic. Our decidability result subsumes previous special cases for sets and multisets. The extension with star is interesting in its own right because it can encode reachability problems for a simple class of transition systems.

Keywords: verification and program analysis, sets, multisets, fuzzy sets, cardinality operator, mixed linear integer-rational arithmetic

1 Introduction

In this paper we show decidability of a logic for reasoning about collections of elements such as sets, multisets (bags), and fuzzy sets. We present a unified logic that can express all these kinds of collections and supports the cardinality operator on collections.

Our approach represents a collection of elements using its characteristic function $f : E \to R$. Inspired by applications in software verification [9], we assume that the domain E is a finite but of unknown size. The range R depends on the kind of the collection: for sets, $R = \{0, 1\}$; for multisets, $R = \{0, 1, 2, \ldots, \}$; for fuzzy sets, R is the interval $[0, 1]$ of rational numbers, denoted $\mathbb{Q}_{[0,1]}$. With this representation, operations and relations on collections such as union, difference,

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 124–138, 2008.
© Springer-Verlag Berlin Heidelberg 2008

and subset are all expressed using operations of linear arithmetic. For example, the condition $A \cup B = C$ becomes $\forall e{\in}E.\ \max(A(e), B(e)) = C(e)$, a definition that applies whether A, B are sets, multisets, or fuzzy sets. A distinguishing feature of our constraints, compared to many other approaches for reasoning about functions $E \to R$, e.g. [2, Chapter 11], is the presence of the cardinality operator, defined by $|A| = \sum_{e \in E} A(e)$. The resulting language freely combines the uses linear arithmetic at two levels: the level of individual elements, as in the subformula $\max(A(e), B(e)) = C(e)$, and the level of sizes of collections, as in the formula $|A \cup B| + |A \cap B| = |A| + |B|$. The language subsumes constraints such as quantifier-free Boolean Algebra with Presburger Arithmetic [9] and therefore contains both set algebra and integer linear arithmetic. It also subsumes decidable constraints on multisets with cardinality bounds [12, 13].

The contribution of this paper is the decidability of constraints on collections where the range R is the set $\mathbb{Q}$ of all rational numbers. Our constraints can express the condition $(\forall e.int(A(e)) \wedge A(e) \geq 0)$ that the number of occurrences $A(e)$ for each element e is a non-negative integer number, so they subsume the case $R = \{0, 1, 2, \dots\}$ solved in [14, 13], which, in turn, subsumes the case of sets [9]. Moreover, our constraints can express the condition $\forall e.(0 \leq A(e) \leq 1)$, which makes them appropriate for modelling fuzzy sets.

Analogously to [12], our decision procedure is based on a translation of a formula with collections and cardinality constraints into a conjunction of a mixed linear integer-rational arithmetic (MLIRA) formula and a new form of condition, denoted $\boldsymbol{u} \in \{\boldsymbol{v} \mid F(\boldsymbol{v})\}^*$. Here the star operator denotes the integer conic hull of a set of rational vectors [5]. Therefore, $\{\boldsymbol{v} \mid F(\boldsymbol{v})\}^*$ denotes the closure under vector addition of the set of solution vectors $\boldsymbol{v}$ of the MLIRA formula F. Formally,

$$\boldsymbol{u} \in \{\boldsymbol{v} \mid F(\boldsymbol{v})\}^* \ \leftrightarrow\ \exists K \in \{0, 1, 2, \dots\}.\ \exists \boldsymbol{v}_1, \dots, \boldsymbol{v}_K.\ \boldsymbol{u} = \sum_{i=1}^{K} \boldsymbol{v}_i \wedge \bigwedge_{i=1}^{K} F(\boldsymbol{v}_i)$$

The star operator is interesting beyond its use in decidability of constraints on collections. For example, it can express the reachability condition for a transition system whose state is an integer or rational vector and whose transitions increment the vector by a solution of a given formula [13].

In contrast to the previous work [12, 13], the formula F in this paper is not restricted to integers, but can be arbitrary MLIRA formula. Consequently, we are faced with the problem of solving an extension of satisfiability of MLIRA formulas with the conditions $\boldsymbol{u} \in \{\boldsymbol{v} \mid F(\boldsymbol{v})\}^*$ where F is an arbitrary MLIRA formula. To solve this problem, we describe a finite and effectively computable representation of the solution set $S = \{\boldsymbol{v} \mid F(\boldsymbol{v})\}$. We use this representation to express the condition $\boldsymbol{u} \in S^*$ as a new MLIRA formula. This gives a "star elimination" algorithm. As one consequence, we obtain a unified decision procedure for sets, multisets, and fuzzy sets in the presence of the cardinality operator. As another consequence, we obtain the decidability of the extension of quantified mixed linear constraints [18] with stars.

Examples of constraints on sets. For each set variable s we assume the constraint $\forall e.(s(e) = 0 \vee s(e) = 1)$.

formula	informal description
$x \notin \texttt{content} \wedge \texttt{size} = \texttt{card content} \longrightarrow$ $(\texttt{size} = 0 \leftrightarrow \texttt{content} = \emptyset)$	using invariant on size to prove correctness of an efficient emptiness check
$x \notin \texttt{content} \wedge \texttt{size} = \texttt{card content} \longrightarrow$ $\texttt{size} + 1 = \texttt{card}(\{x\} \cup \texttt{content})$	maintaining correct size when inserting fresh element into set
$\texttt{size} = \texttt{card content} \wedge$ $\texttt{size1} = \texttt{card}(\{x\} \cup \texttt{content}) \longrightarrow$ $\texttt{size1} \leq \texttt{size} + 1$	maintaining size after inserting an element into set
$\texttt{content} \subseteq \texttt{alloc} \wedge$ $x_1 \notin \texttt{alloc} \wedge$ $x_2 \notin \texttt{alloc} \cup \{x_1\} \wedge$ $x_3 \notin \texttt{alloc} \cup \{x_1\} \cup \{x_2\} \longrightarrow$ $\texttt{card}\,(\texttt{content} \cup \{x_1\} \cup \{x_2\} \cup \{x_3\}) =$ $\texttt{card content} + 3$	allocating and inserting three objects into a container data structure
$\texttt{content} \subseteq \texttt{alloc0} \wedge x_1 \notin \texttt{alloc0} \wedge$ $\texttt{alloc0} \cup \{x_1\} \subseteq \texttt{alloc1} \wedge x_2 \notin \texttt{alloc1} \wedge$ $\texttt{alloc1} \cup \{x_2\} \subseteq \texttt{alloc2} \wedge x_3 \notin \texttt{alloc2} \longrightarrow$ $\texttt{card}\,(\texttt{content} \cup \{x_1\} \cup \{x_2\} \cup \{x_3\}) =$ $\texttt{card content} + 3$	allocating and inserting at least three objects into a container data structure
$x \in C \wedge C_1 = (C \setminus \{x\}) \wedge$ $\texttt{card}(\texttt{alloc1} \setminus \texttt{alloc0}) \leq 1 \wedge$ $\texttt{card}(\texttt{alloc2} \setminus \texttt{alloc1}) \leq \texttt{card}\, C_1 \longrightarrow$ $\texttt{card}\,(\texttt{alloc2} \setminus \texttt{alloc0}) \leq \texttt{card}\, C$	bound on the number of allocated objects in a recursive function that incorporates container C into another container

Examples of constraints on multisets. For each multiset variable m we assume the constraint $\forall e.int(m(e)) \wedge A(e) \geq 0$.

$\texttt{size} = \texttt{card content} \wedge$ $\texttt{size1} = \texttt{card}(\{x\} \uplus \texttt{content}) \longrightarrow$ $\texttt{size1} = \texttt{size} + 1$	maintaining size after inserting an element into multiset

Examples of constraints on fuzzy sets. For each fuzzy set variable f we assume the constraint $\forall e.0 \leq f(e) \leq 1$.

$2\|A\| \neq 2\|B\| + 1$	example formula valid over multisets but invalid over fuzzy sets
$(\forall e.U(e) = 1) \rightarrow \|A \cap B\| + \|A \cup B\| \leq \|A\| + \|U\|$	example formula valid over fuzzy sets but invalid over multisets
$(\forall e.C(e) = \lambda A(e) + (1 - \lambda)B(e)) \rightarrow$ $A \cap B \subseteq C \subseteq A \cup B$	basic property of convex combination of fuzzy sets [19], for any fixed constant $\lambda \in [0, 1]$

Fig. 1. Example constraints in our class

2 Examples

Figure 1 shows small example formulas over sets, multisets, and fuzzy sets that are expressible in our logic. The examples for sets and multisets are based on verification conditions from software verification [9]. The remaining examples illustrate basic differences in valid formulas over multisets and fuzzy sets.

We illustrate our technique on one of the examples shown in Figure 1: we show that formula $\forall e.U(e) = 1 \rightarrow |A \cap B| + |A \cup B| \leq |A| + |U|$ is valid where U, A, and B are fuzzy sets. To prove formula validity, we prove unsatisfiability of its negation, conjoined with the constraints ensuring that the collections are fuzzy sets:

$$\begin{array}{l} \forall e.U(e) = 1 \ \wedge \ |A| + |U| < |A \cap B| + |A \cup B| \ \wedge \\ \forall e.0 \leq A(e) \leq 1 \ \wedge \ \forall e.0 \leq B(e) \leq 1 \ \wedge \ \forall e.0 \leq U(e) \leq 1 \end{array}$$

We first reduce the formula to the normal form, as follows. We flatten the formula by introducing fresh variables n_i for each cardinality operator. The formula reduces to:

$$\begin{array}{l} n_1 + n_2 < n_3 + n_4 \ \wedge \ n_1 = |A| \ \wedge \ n_2 = |U| \ \wedge \ n_3 = |A \cap B| \ \wedge \ n_4 = |A \cup B| \ \wedge \\ \forall e.U(e) = 1 \ \wedge \ \forall e.0 \leq A(e) \leq 1 \ \wedge \ \forall e.0 \leq B(e) \leq 1 \ \wedge \ \forall e.0 \leq U(e) \leq 1 \end{array}$$

We next apply the definition of the cardinality operator, $|C| = \sum_{e \in E} C(e)$:

$$\begin{array}{l} n_1 + n_2 < n_3 + n_4 \ \wedge \ n_1 = \sum_{e \in E} A(e) \ \wedge \ n_2 = \sum_{e \in E} U(e) \ \wedge \\ n_3 = \sum_{e \in E} (A \cap B)(e) \ \wedge \ n_4 = \sum_{e \in E} (A \cup B)(e) \ \wedge \\ \forall e.U(e) = 1 \ \wedge \ \forall e.0 \leq A(e) \leq 1 \ \wedge \ \forall e.0 \leq B(e) \leq 1 \ \wedge \ \forall e.0 \leq U(e) \leq 1 \end{array}$$

Operators $\cup$ and $\cap$ are defined pointwise using ite operator:

$$\begin{array}{l} (C_1 \cup C_2)(e) = \max\{C_1(e), C_2(e)\} = \mathsf{ite}(C_1(e) \leq C_2(e), C_2(e), C_1(e)) \\ (C_1 \cap C_2)(e) = \min\{C_1(e), C_2(e)\} = \mathsf{ite}(C_1(e) \leq C_2(e), C_1(e), C_2(e)), \end{array}$$

where $\mathsf{ite}(A, B, C)$ is the standard *if-then-else* operator, denoting B when A is true and C otherwise. Using these definitions, the example formula becomes:

$$\begin{array}{l} n_1 + n_2 < n_3 + n_4 \ \wedge \ n_1 = \displaystyle\sum_{e \in E} A(e) \ \wedge \ n_2 = \sum_{e \in E} U(e) \ \wedge \\ n_3 = \displaystyle\sum_{e \in E} \mathsf{ite}(A(e) \leq B(e), A(e), B(e)) \ \wedge \ n_4 = \sum_{e \in E} \mathsf{ite}(A(e) \leq B(e), B(e), A(e)) \ \wedge \\ \forall e.U(e) = 1 \ \wedge \ \forall e.0 \leq A(e) \leq 1 \ \wedge \ \forall e.0 \leq B(e) \leq 1 \ \wedge \ \forall e.0 \leq U(e) \leq 1 \end{array}$$

Using vectors of integers, we then group all the sums into one, and also group all universally quantified constraints:

$$\begin{array}{l} n_1 + n_2 < n_3 + n_4 \wedge \big(n_1, n_2, n_3, n_4\big) = \\ \displaystyle\sum_{e \in E} \big(A(e), \ U(e), \ \mathsf{ite}(A(e) \leq B(e), A(e), B(e)), \ \mathsf{ite}(A(e) \leq B(e), B(e), A(e))\big) \\ \qquad \wedge \ \forall e. \ \big(U(e) = 1 \ \wedge \ 0 \leq A(e) \leq 1 \ \wedge \ 0 \leq B(e) \leq 1 \ \wedge \ 0 \leq U(e) \leq 1\big) \end{array}$$

As we prove in Theorem 1 below, the last formula is equisatisfiable with

$$n_1 + n_2 < n_3 + n_4 \wedge (n_1, n_2, n_3, n_4) \in \\ \{\big(a, u, \mathsf{ite}(a \leq b, a, b), \mathsf{ite}(a \leq b, b, a)\big) \mid u = 1 \ \wedge \ 0 \leq a \leq 1 \ \wedge \ 0 \leq b \leq 1\}^*$$

The subject of this paper are general techniques for solving such satisfiability problems that contain a MLIRA formula and a star operator applied to another MLIRA formula. We next illustrate some of the ideas of the general technique, taking several shortcuts to keep the exposition brief.

Because the value of the variable u is determined ($u = 1$), we can simplify the last formula to:

$$n_1 + n_2 < n_3 + n_4 \ \wedge \ (n_1, n_2, n_3, n_4) \in S^*$$

where $S = \{\big(a, 1, \mathsf{ite}(a \leq b, a, b), \mathsf{ite}(a \leq b, b, a)\big) \mid 0 \leq a \leq 1 \wedge 0 \leq b \leq 1\}$. By case analysis on $a \leq b$, we conclude $S = S_1 \cup S_2$ for

$$S_1 = \{(a, 1, a, b) \mid 0 \leq a \leq 1 \wedge 0 \leq b \leq 1 \wedge a \leq b\} \\ S_2 = \{(a, 1, b, a) \mid 0 \leq a \leq 1 \wedge 0 \leq b \leq 1 \wedge b < a\}$$

This eliminates the ite expressions and we have:

$$n_1 + n_2 < n_3 + n_4 \wedge (n_1, n_2, n_3, n_4) \in (S_1 \cup S_2)^*$$

By definition of star operator, the last condition is equivalent to

$$n_1 + n_2 < n_3 + n_4 \ \wedge \ (n_1, n_2, n_3, n_4) = (n_1^1, n_2^1, n_3^1, n_4^1) + (n_1^2, n_2^2, n_3^2, n_4^2) \wedge \\ (n_1^1, n_2^1, n_3^1, n_4^1) \in S_1^* \ \wedge \ (n_1^2, n_2^2, n_3^2, n_4^2) \in S_2^*$$

Let us characterize the condition $(n_1^1, n_2^1, n_3^1, n_4^1) \in S_1^*$. Let K_1 denote the number of vectors in S_1 whose sum is $(n_1^1, n_2^1, n_3^1, n_4^1)$. By definition of the star operator, there are $a_1^1, \ldots, a_{K_1}^1$ and $b_1^1, \ldots, b_{K_1}^1$ such that $0 \leq a_i^1 \leq b_i^1 \leq 1$ and

$$(n_1^1, n_2^1, n_3^1, n_4^1) = \sum_{i=1}^{K_1} (a_i^1, 1, a_i^1, b_i^1)$$

We obtain that $n_1^1 = n_3^1 = \sum_{i=1}^{K_1} a_i^1 = A_1$, $n_2 = K_1$, $n_4 = \sum_{i=1}^{K_1} b_i^1 = B_1$. The other case for S_2 is analogous and we derive $(n_1^2, n_2^2, n_3^2, n_4^2) = (A_2, K_2, B_2, A_2)$.

This way we eliminate the star operator and the example formula becomes:

$$n_1 + n_2 < n_3 + n_4 \wedge (n_1, n_2, n_3, n_4) = (A_1, K_1, A_1, B_1) + (A_2, K_2, B_2, A_2)$$

This formula further reduces to $K_1 + K_2 < B_1 + B_2$. If we apply the definitions of B_i and properties of b_i^j we obtain the following formula:

$$K_1 + K_2 < \sum_{i=1}^{K_1} b_i^1 + \sum_{i=1}^{K_2} b_i^2 \ \wedge \ \bigwedge_{i=1}^{K_1} b_i^1 \leq 1 \wedge \bigwedge_{i=1}^{K_2} b_i^2 \leq 1$$

In this case, it is easy to see that the resulting formula is contradictory. This shows that the initial formula is valid over fuzzy sets. Our paper shows that, in

general, such formulas are equivalent to existentially quantified MLIRA formulas, despite the fact that their initial formulation involves sums with parameters such as K_1 and K_2. This is possible thanks to the special structure of the sets of solutions of MLIRA formulas, which we describe building on results such as [7] and the theory of linear programming.

Having seen the use of our method to prove formula validity, we illustrate its use in producing counterexamples by showing that the original formula is invalid over multisets. Restricting the range of each collection to integers and using the same reduction, we derive formula

$$n_1 + n_2 < n_3 + n_4 \wedge$$
$$(n_1, n_2, n_3, n_4) \in \{\big(a, 1, \mathsf{ite}(a \leq b, a, b), \mathsf{ite}(a \leq b, b, a)\big) \mid a, b \in \mathbb{N}\}^*$$

Applying again a similar case analysis, we deduce $K_1 + K_2 < \sum_{i=1}^{K_1} b_i^1 + \sum_{i=1}^{K_2} b_i^2$ where all b_i^j's are non-negative integers. This formula is satisfiable, for example, with a satisfying variable assignment $K_1 = 1, b_1^1 = 2$ and $K_2 = 0$. The correspondence of Theorem 1 then allows us to construct a multiset counterexample. Because $K_2 = 0$, no vector from S_2 contributes to sum and we consider only S_1. Variable K_1 denotes the number of elements of a domain set E, so we consider the domain set $E = \{e_1\}$. Multisets A, B and U are defined by $A(e_1) = 1$, $B(e_1) = 2$, and $U(e_1) = 1$. It can easily be verified that this is a counterexample for validity of the formula over multisets.

3 From Collections to Stars

This section describes the translation from constraints on collections to constraints that use star operator. We first present the syntax of our constraints and clarify the semantics of selected constructs (the semantics of the remaining constructs can be derived from their translation into simpler ones).

We model each collection f as a function whose domain is a finite set E of unknown size and whose range is the set of rational numbers. When the constraints imply that the range of f is $\{0, 1\}$, then f models sets, when the range of f are non-negative integers, then f denotes standard multisets (bags), in which an element can occur multiple times. We call the number of occurrences of an element e, denoted $f(e)$, the *multiplicity* of an element. When the range of f is restricted to be in interval $[0, 1]$, then f describes a fuzzy set [19].

In addition to standard operations on collections (such as plus, union, intersection, difference) we also allow the cardinality operator, defined as $|f| = \sum_{e \in E} f(e)$. This is the desired definition for sets and multisets and we believe it is a natural notion for fuzzy sets over a finite universe E. Figure 2 shows a context-free grammar of our formulas involving collections.

Semantics of some less commonly known operators is defined as follows: $\mathsf{ite}(A, B, C)$ denotes the if-then-else expression, which evaluates to B when A is true and evaluates to C when A is false. The $\mathsf{setof}(C)$ operator takes as an argument collection C and returns the set of all elements for which $C(e)$ is positive. To constrain a variable s to denote a set, use formula $\forall e.s(e) = 0 \vee s(e) = 1$. To

top-level formulas:
$F ::= A \mid F \wedge F \mid \neg F$
$A ::= C{=}C \mid C \subseteq C \mid \forall e.\mathsf{F}^{\mathsf{in}} \mid \mathsf{A}^{\mathsf{out}}$
outer linear arithmetic formulas:
$\mathsf{F}^{\mathsf{out}} ::= \mathsf{A}^{\mathsf{out}} \mid \mathsf{F}^{\mathsf{out}} \wedge \mathsf{F}^{\mathsf{out}} \mid \neg \mathsf{F}^{\mathsf{out}}$
$\mathsf{A}^{\mathsf{out}} ::= \mathsf{t}^{\mathsf{out}} \leq \mathsf{t}^{\mathsf{out}} \mid \mathsf{t}^{\mathsf{out}}{=}\mathsf{t}^{\mathsf{out}} \mid (\mathsf{t}^{\mathsf{out}}, \ldots, \mathsf{t}^{\mathsf{out}}){=}\sum_{\mathsf{F}^{\mathsf{in}}}(\mathsf{t}^{\mathsf{in}}, \ldots, \mathsf{t}^{\mathsf{in}})$
$\mathsf{t}^{\mathsf{out}} ::= k \mid \ |C| \ \mid K \mid \mathsf{t}^{\mathsf{out}} + \mathsf{t}^{\mathsf{out}} \mid K \cdot \mathsf{t}^{\mathsf{out}} \mid \lfloor \mathsf{t}^{\mathsf{out}} \rfloor \mid \mathsf{ite}(\mathsf{F}^{\mathsf{out}}, \mathsf{t}^{\mathsf{out}}, \mathsf{t}^{\mathsf{out}})$
inner linear arithmetic formulas:
$\mathsf{F}^{\mathsf{in}} ::= \mathsf{A}^{\mathsf{in}} \mid \mathsf{F}^{\mathsf{in}} \wedge \mathsf{F}^{\mathsf{in}} \mid \neg \mathsf{F}^{\mathsf{in}}$
$\mathsf{A}^{\mathsf{in}} ::= \mathsf{t}^{\mathsf{in}} \leq \mathsf{t}^{\mathsf{in}} \mid \mathsf{t}^{\mathsf{in}}{=}\mathsf{t}^{\mathsf{in}}$
$\mathsf{t}^{\mathsf{in}} ::= f(e) \mid K \mid \mathsf{t}^{\mathsf{in}} + \mathsf{t}^{\mathsf{in}} \mid K \cdot \mathsf{t}^{\mathsf{in}} \mid \lfloor \mathsf{t}^{\mathsf{in}} \rfloor \mid \mathsf{ite}(\mathsf{F}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}})$
expressions about collections:
$C ::= c \mid \emptyset \mid C \cap C \mid C \cup C \mid C \uplus C \mid C \setminus C \mid C \setminus\setminus C \mid \mathsf{setof}(C)$
terminals:
c - collection variable; e - index variable (fixed)
k - rational variable; K - rational constant

Fig. 2. Quantifier-Free Formulas about Collection with Cardinality Operator

constraint a variable m to denote a multiset, use formula $(\forall e.int(m(e)) \wedge m(e) \geq 0)$. Here $int(x)$ is a shorthand for $\lfloor x \rfloor = x$ where $\lfloor x \rfloor$ is the largest integer smaller than or equal to x.

A decision procedure for checking satisfiability of the subclass of integer formulas was described in [12]. The novelty of constraints in Figure 2 compared to the language in [12] is the presence of the floor operator $\lfloor x \rfloor$ and not only integer but also rational constants. All variables in our current language are interpreted over rationals, but any of them can be restricted to be integer using the constraint $int(x)$.

To reduce reasoning about collections to reasoning in linear arithmetic with stars, we follow the idea from [12] and convert a formula to the *sum normal form*.

Definition 1. *A formula is in sum normal form iff it is of the form*

$$P \ \wedge \ (u_1, \ldots, u_n) = \sum_{e \in E} (t_1, \ldots, t_n) \ \wedge \ \forall e.F$$

where P is a quantifier-free linear arithmetic formula with no collection variables, and where variables in $t_1, \ldots, t_n$ and F occur only as expressions of the form $c(e)$ for a collection variable c and e the fixed index variable.

Figure 3 summarizes the process of transforming formula into sum normal form.[1] The previous example section illustrated this idea. As another example, consider a negation of a formula that verifies the change in the size of a list after insertion of an element: $|x| = 1 \wedge |L \uplus x| \neq |L| + 1$. The sum normal form of this formula is: $k_1 \neq k_2 + 1 \ \wedge \ (1, k_1, k_2) = \sum_{e \in E}(x(e), y(e), L(e)) \ \wedge \ \forall e.y(e) = L(e) + x(e)$.

[1] Note that the part $\forall e.F$ could be omitted from normal form definition and expressed as an additional component of the sum. However, its use leads to somewhat simpler constraints.

INPUT: formula in the syntax of Figure 2
OUTPUT: formula in sum normal form (Definition 1)

1. Flatten expressions that we wish to eliminate:
   $C[exp] \rightsquigarrow (x = exp \wedge C[x])$
   where exp is one of the expressions $\emptyset$, $c_1 \cup c_2$, $c_1 \cap c_2$, $c_1 \uplus c_2$, $c_1 \setminus c_2$, $\mathsf{setof}(c_1)$, $|c_1|$, and where the occurrence of exp is not already in a top-level conjunct of the form $x = exp$ or $exp = x$ for some variable x.
2. Reduce colection relations to pointwise linear arithmetic conditions:
   $C[c_0 = \emptyset] \rightsquigarrow C[\forall e.\, c_0(e) = 0]$
   $C[c_0 = c_1 \cap c_2] \rightsquigarrow C[\forall e.\, c_0(e) = \mathsf{ite}(c_1(e) \leq c_2(e), c_1(e), c_2(e))]$
   $C[c_0 = c_1 \cup c_2] \rightsquigarrow C[\forall e.\, c_0(e) = \mathsf{ite}(c_1(e) \leq c_2(e), c_2(e), c_1(e))]$
   $C[c_0 = c_1 \uplus c_2] \rightsquigarrow C[\forall e.\, c_0(e) = c_1(e) + c_2(e)]$
   $C[c_0 = c_1 \setminus c_2] \rightsquigarrow C[\forall e.\, c_0(e) = \mathsf{ite}(c_1(e) \leq c_2(e), 0, c_1(e) - c_2(e))]$
   $C[c_0 = c_1 \setminus\!\setminus c_2] \rightsquigarrow C[\forall e.\, c_0(e) = \mathsf{ite}(c_2(e) = 0, c_1(e), 0)]$
   $C[c_0 = \mathsf{setof}(c_1)] \rightsquigarrow C[\forall e.\, c_0(e) = \mathsf{ite}(0 < c_1(e), 1, 0)]$
   $C[c_1 \subseteq c_2] \rightsquigarrow C[\forall e.\, (c_1(e) \leq c_2(e))]$
   $C[c_1 = c_2] \rightsquigarrow C[\forall e.\, (c_1(e) = c_2(e))]$
3. Express each cardinality operator using a sum:
   $C[\,|c|\,] \rightsquigarrow C[\sum_{e \in E} c(e)]$
4. Express negatively occurring pointwise definitions using the sum:
   $C[\forall e.F] \rightsquigarrow C[0 = \sum_{e \in E} \mathsf{ite}(F(e), 0, 1)]$
5. Flatten any sums that are not already top-level conjuncts:
   $C[\,(u_1, \ldots, u_n) = \sum_F (t_1, \ldots, t_n)\,] \rightsquigarrow (w_1, \ldots, w_n) = \sum_F (t_1, \ldots, t_n) \wedge C[\bigwedge_{i=1}^{n} u_i = w_i\,]$
6. Eliminate conditions from sums:
   $C[\,\sum_F (t_1, \ldots, t_n)\,] \rightsquigarrow C[\sum_{e \in E} (\mathsf{ite}(F, t_1, 0), \ldots, \mathsf{ite}(F, t_n, 0))\,]$
7. Group all sums into one:
   $P \wedge \bigwedge_{i=1}^{q} (u_1^i, \ldots, u_{n_i}^i) = \sum_{e \in E} (t_1^i, \ldots, t_{n_i}^i) \rightsquigarrow$
   $P \wedge (u_1^1, \ldots, u_{n_1}^1, \ldots, u_1^q, \ldots, u_{n_q}^q) = \sum_{e \in E} (t_1^1, \ldots, t_{n_1}^1, \ldots, t_1^q, \ldots, t_{n_q}^q)$
8. Group all pointwise defined operations into one:
   $P \wedge \bigwedge_{i=1}^{q} (\forall e.F_i) \rightsquigarrow P \wedge \forall e. \bigwedge_{i=1}^{q} F_i$

Fig. 3. Algorithm for reducing collections formulas to sum normal form

Formulas in sum normal form contain only one top-level sum which ranges over elements of an existentially quantified set E. To study such constraints we introduce the star operator.

Definition 2 (Star operator, integer conic hull [5]). *Let C be a set of rational vectors. Define $C^* = \{\boldsymbol{v}_1 + \ldots + \boldsymbol{v}_K \mid K \in \{0, 1, 2, \ldots\}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_K \in C\}$.*

The fact that the bound variable K in Definition 2 ranges over non-negative integers as opposed to rational or real numbers differentiates the integer conic hull (star) from the notion of conic hull in linear programming [17].

Theorem 1. *A formula* $(u_1, \ldots, u_n) = \sum_{e \in E}(t_1, \ldots, t_n) \wedge \forall e.F$ *is equisatisfiable with the formula* $(u_1, \ldots, u_n) \in \{(t'_1, \ldots, t'_n) \mid x_i \in \mathbb{Q} \wedge F'\}^*$ *where* t'_j *and* F' *are* t_j *and* F *respectively in which each* $c_i(e)$ *is replaced by a fresh variable* x_i*.*

Proof. $\Leftarrow$): Assume $(u_1, \ldots, u_n) \in \{(t'_1, \ldots, t'_n) \mid x_i \in \mathbb{Q} \wedge F'\}^*$ is satisfiable. Then there exists an integer $k \geq 0$ such that $(u_1, \ldots, u_n) = \sum_{j=1}^{k}(t_1^j, \ldots, t_n^j)$. We define set E to consist of k distinct elements, $E = \{e_1, \ldots, e_k\}$. Every variable x_i occurring in $t'_1, \ldots, t'_n$ and F' corresponds to the collection c_i. Let x_i^j denote the value of x_i in jth summand $(t_1^j, \ldots, t_n^j)$. Define each collection c_i by $c_i(e_j) = x_i^j$. The finite set E and collections c_i defined as above make formula $(u_1, \ldots, u_n) = \sum_{e \in E}(t_1, \ldots, t_n) \ \wedge \ \forall e.F$ satisfiable.

$\Rightarrow$): The other direction is analogous. Given E, for each $e_j \in E$ we obtain a set of values $c_i(e_j)$ that give the values for x_i in jth summand. ∎

Applying Theorem 1 to our example of insertion into a list, we obtain that

$$k_1 \neq k_2 + 1 \ \wedge \ (1, k_1, k_2) = \sum_{e \in E}(x(e), y(e), L(e)) \ \wedge \ \forall e.y(e) = L(e) + x(e)$$

is equisatisfiable with

$$k_1 \neq k_2 + 1 \ \wedge \ (1, k_1, k_2) \in \{(x, y, L) \mid y = L + x\}^*$$

Thanks to Theorem 1, in the rest of the paper we investigate the satisfiability problem for such formulas, whose syntax is given in Figure 4. These formulas are sufficient to check satisfiability for formulas in Figure 2. In Section 6 we present a more general decidable language that allows nesting of terms, logical operations, quantifiers, and stars.

top-level, outer linear arithmetic formulas:
$\mathsf{F}^{\mathsf{out}} ::= \mathsf{A}^{\mathsf{out}} \mid \mathsf{F}^{\mathsf{out}} \wedge \mathsf{F}^{\mathsf{out}} \mid \neg\mathsf{F}^{\mathsf{out}}$
$\mathsf{A}^{\mathsf{out}} ::= \mathsf{t}^{\mathsf{out}} \leq \mathsf{t}^{\mathsf{out}} \mid \mathsf{t}^{\mathsf{out}} = \mathsf{t}^{\mathsf{out}} \mid (\mathsf{t}^{\mathsf{out}}, \ldots, \mathsf{t}^{\mathsf{out}}) \in \{(\mathsf{t}^{\mathsf{in}}, \ldots, \mathsf{t}^{\mathsf{in}}) \mid \mathsf{F}^{\mathsf{in}}\}^*$
$\mathsf{t}^{\mathsf{out}} ::= \mathsf{k}^{\mathsf{out}} \mid K \mid \mathsf{t}^{\mathsf{out}} + \mathsf{t}^{\mathsf{out}} \mid K \cdot \mathsf{t}^{\mathsf{out}} \mid \lfloor \mathsf{t}^{\mathsf{out}} \rfloor \mid \mathsf{ite}(\mathsf{F}^{\mathsf{out}}, \mathsf{t}^{\mathsf{out}}, \mathsf{t}^{\mathsf{out}})$
inner linear arithmetic formulas:
$\mathsf{F}^{\mathsf{in}} ::= \mathsf{A}^{\mathsf{in}} \mid \mathsf{F}^{\mathsf{in}} \wedge \mathsf{F}^{\mathsf{in}} \mid \neg\mathsf{F}^{\mathsf{in}}$
$\mathsf{A}^{\mathsf{in}} ::= \mathsf{t}^{\mathsf{in}} \leq \mathsf{t}^{\mathsf{in}} \mid \mathsf{t}^{\mathsf{in}} = \mathsf{t}^{\mathsf{in}}$
$\mathsf{t}^{\mathsf{in}} ::= \mathsf{k}^{\mathsf{in}} \mid K \mid \mathsf{t}^{\mathsf{in}} + \mathsf{t}^{\mathsf{in}} \mid K \cdot \mathsf{t}^{\mathsf{in}} \mid \lfloor \mathsf{t}^{\mathsf{in}} \rfloor \mid \mathsf{ite}(\mathsf{F}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}}, \mathsf{t}^{\mathsf{in}})$
terminals:
$\mathsf{k}^{\mathsf{in}}, \mathsf{k}^{\mathsf{out}}$ - rational variable (two disjoint sets); K - rational constants

Fig. 4. Syntax of Mixed Integer-Rational Linear Arithmetic with Star

4 Separating Mixed Constraints

As justified in previous sections, we consider the satisfiability problem for $G(\boldsymbol{r}, \boldsymbol{w}) \wedge \boldsymbol{w} \in \{\boldsymbol{x} \mid F(\boldsymbol{x})\}^*$ where F and G are quantifier-free, mixed linear integer-rational arithmetic (MLIRA) formulas.

Our goal is to give an algorithm for constructing another MLIRA formula F' such that $\boldsymbol{w} \in \{\boldsymbol{x} \mid F(\boldsymbol{x})\}^*$ is equivalent to $\exists \boldsymbol{w}'.F'(\boldsymbol{w}', \boldsymbol{w})$. This will reduce the satisfiability problem to the satisfiability of $G(\boldsymbol{r}, \boldsymbol{w}) \wedge F'(\boldsymbol{w}', \boldsymbol{w})$.

As a first stage towards this goal, this section shows how to represent the set $\{\boldsymbol{x} \mid F(\boldsymbol{x})\}$ using solutions of pure integer constraints and solutions of pure rational constraints. We proceed in several steps.

Step 1. Eliminate the floor functions from F using integer and real variables, applying from left to right the equivalence

$$C(\lfloor t \rfloor) \leftrightarrow \exists y_Q \in \mathbb{Q}.\exists y_Z \in \mathbb{Z}.\ t = y_Q \wedge y_Z \leq y_Q < y_Z + 1 \wedge C(y_Z)$$

The result is an equivalent formula without the floor operators, where some of the variables are restricted to be integer.

Step 2. Transform F into linear programming problems, as follows. First, eliminate if-then-else expressions by introducing fresh variables and using disjunction (see e.g. [12]). Then transform formula to negation normal form. Eliminate $t_1 = t_2$ by transforming it into $t_1 \leq t_2 \wedge t_2 \leq t_1$. Eliminate $t_1 \neq t_2$ by transforming it into $t_1 < t_2 \vee t_2 < t_1$. Following [4, Section 3.3], replace each $t_1 < t_2$ with $t_1 + \delta \leq t_2$ where δ is a special variable (the same for all strict inequalities), for which we require $0 < \delta \leq 1$. We obtain for some d matrices A_i for $1 \leq i \leq d$ such that

$$F(\boldsymbol{x}) \ \leftrightarrow\ \exists \boldsymbol{y}^Z \in \mathbb{Z}^{d_Z}.\exists \boldsymbol{y}^Q \in \mathbb{Q}^{d_Q}.\exists \delta \in \mathbb{Q}_{(0,1]}.\ \bigvee_{i=1}^{d} A_i \cdot (\boldsymbol{x}, \boldsymbol{y}^Z, \boldsymbol{y}^Q) \leq \boldsymbol{b}$$

where $A_i \cdot (\boldsymbol{x}, \boldsymbol{y}^Z, \boldsymbol{y}^Q)$ denotes multiplication of matrix A_i by the vector $(\boldsymbol{x}, \boldsymbol{y}^Z, \boldsymbol{y}^Q)$ obtained by stacking vectors $\boldsymbol{x}$, $\boldsymbol{y}^Z$, and $\boldsymbol{y}^Q$.

Step 3. Represent the rational variables $\boldsymbol{x}$, $\boldsymbol{y}^Q$ as a sum of its integer part and its fractional part from $\mathbb{Q}_{[0,1]}$, obtaining

$$\begin{aligned} F(\boldsymbol{x}) \ \leftrightarrow\ \Big(&\exists (\boldsymbol{x}^Z, \boldsymbol{y}^Z) \in \mathbb{Z}^{d'_Z}.\exists (\boldsymbol{x}^R, \boldsymbol{y}^R) \in \mathbb{Q}_{[0,1]}^{d'_Q}.\exists \delta \in \mathbb{Q}_{(0,1]}. \\ &\boldsymbol{x} = \boldsymbol{x}^Z + \boldsymbol{x}^R \ \wedge\ \bigvee_{i=1}^{d} A'_i \cdot (\boldsymbol{x}^Z, \boldsymbol{y}^Z, \boldsymbol{x}^R, \boldsymbol{y}^R) \leq \boldsymbol{b}'\Big) \end{aligned}$$

Note that $\boldsymbol{w} \in \{\boldsymbol{x} \mid \exists \boldsymbol{y}.H(\boldsymbol{x}, \boldsymbol{y})\}^*$ is equivalent to

$$\exists \boldsymbol{w}'.(\boldsymbol{w}, \boldsymbol{w}') \in \{(\boldsymbol{x}, \boldsymbol{y}) \mid H(\boldsymbol{x}, \boldsymbol{y})\}^*$$

In other words, we can push existential quantifiers to the top-level of the formula. Therefore, the original problem (after renaming) becomes

$$\begin{aligned} &G(\boldsymbol{r}, \boldsymbol{w}) \ \wedge\ \exists \boldsymbol{z}.\ (\boldsymbol{u}^Z, \boldsymbol{u}^Q, \Delta) \ \in \\ &\{(\boldsymbol{x}^Z, \boldsymbol{x}^R, \delta) \mid \bigvee_{i=1}^{d} A_i \cdot (\boldsymbol{x}^Z, \boldsymbol{x}^R, \delta) \leq \boldsymbol{b}_i,\ \boldsymbol{x}^Z \in \mathbb{Z}^{d_Z}, \boldsymbol{x}^R \in \mathbb{Q}_{[0,1]}^{d_R}, \delta \in \mathbb{Q}_{(0,1]}\}^* \end{aligned}$$

where the vector $\boldsymbol{z}$ contains a subset of variables $\boldsymbol{u}^Z, \boldsymbol{u}^Q, \Delta$.

Step 4. Separate integer and rational parts, as follows. Consider one of the disjuncts $A \cdot (\boldsymbol{x}^Z, \boldsymbol{y}^R, \delta) \leq \boldsymbol{b}$. For $A = [A_Z\ A_R\ c]$ this linear condition can be written as $A_Z \boldsymbol{x}^Z + A_R \boldsymbol{x}^R + \boldsymbol{c}\delta \leq \boldsymbol{b}$, that is

$$A_R \boldsymbol{x}^R + \boldsymbol{c}\delta \leq \boldsymbol{b} - A_Z \boldsymbol{x}^Z \tag{1}$$

Because the right-hand side is integer, for $\boldsymbol{a}$ denoting $\lceil A_R \boldsymbol{x}^R + \boldsymbol{c}\delta \rceil$ (left-hand side rounded up), the equation becomes $A_R \boldsymbol{x}^R + \boldsymbol{c}\delta \leq \boldsymbol{a} \leq \boldsymbol{b} - A_Z \boldsymbol{x}^Z$. Because $\boldsymbol{x}^R \in \mathbb{Q}^{d_Q}_{[0,1]}, \delta \in \mathbb{Q}_{(0,1]}$, vector $\boldsymbol{a}$ is bounded by the norm M_1 of the matrix $[A_R\ \boldsymbol{c}]$. Formula (1) is therefore equivalent to the finite disjunction

$$\bigvee_{\boldsymbol{a} \in \mathbb{Z}^d, ||\boldsymbol{a}|| \leq M_1} A_Z \boldsymbol{x}^Z \leq \boldsymbol{b} - \boldsymbol{a} \wedge A_R \boldsymbol{x}^R + \boldsymbol{c}\delta \leq \boldsymbol{a} \tag{2}$$

Note that each disjunct is a conjunction of a purely integer constraint and a purely rational constraint.

Step 5. Propagate star through disjunction, using the property

$$\boldsymbol{w} \in \{\boldsymbol{x} \mid \bigvee_{i=1}^{n} H_i(\boldsymbol{x})\}^* \ \leftrightarrow\ \exists \boldsymbol{w}_1, \ldots, \boldsymbol{w}_n.\ \boldsymbol{w} = \sum_{i=1}^{n} \boldsymbol{w}_i \ \wedge\ \bigwedge_{i=1}^{n} \boldsymbol{w}_i \in \{\boldsymbol{x} \mid H_i(\boldsymbol{x})\}^*$$

The final result is an equivalent conjunction of a MLIRA formula and an existentially quantified conjunction of formulas of the form

$$(\boldsymbol{u}^Z, \boldsymbol{u}^Q, \Delta) \in \{(\boldsymbol{x}^Z, \boldsymbol{x}^R, \delta) \mid \begin{array}{l} A_Z \boldsymbol{x}^Z \leq \boldsymbol{b}_Z, A_R \cdot (\boldsymbol{x}^R, \delta) \leq \boldsymbol{b}_R, \\ \boldsymbol{x}^Z \in \mathbb{Z}^{d_Z}, \boldsymbol{x}^R \in \mathbb{Q}^{d_R}_{[0,1]}, \delta \in \mathbb{Q}_{(0,1]}\}^* \end{array} \tag{3}$$

5 Eliminating Star Operator from Formulas

The previous section sets the stage for the following star-elimination theorem, which is the core result of this paper.

Theorem 2. *Let F be a quantifier-free MLIRA formula. Then there exist effectively computable integer vectors $\boldsymbol{a}_i$ and $\boldsymbol{b}_{ij}$ and effectively computable rational vectors $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n$ with coordinates in $\mathbb{Q}_{[0,1]}$ such that formula* (3) *is equivalent to a formula of the form*

$$\begin{array}{l} \exists K \in \mathbb{N}.\ \exists \mu_1, \ldots, \mu_q, \nu_{11}, \ldots, \nu_{qq_q} \in \mathbb{N}.\ \exists \beta_1, \ldots, \beta_n \in \mathbb{Q}. \\ \big(\boldsymbol{u}^Z = \sum_{i=1}^{q} (\mu_i \boldsymbol{a}_i + \sum_{j=1}^{q_i} \nu_{ij} \boldsymbol{b}_{ij})\ \wedge \bigwedge_{i=1}^{q} (\mu_i = 0 \rightarrow \sum_{j=1}^{q_i} \nu_{ij} = 0)\ \wedge\ (\sum_{i=1}^{q} \mu_i = K)\big) \\ \qquad \wedge\ \ \big((K = 0 \wedge \Delta = 0 \wedge \boldsymbol{u}^Q = \boldsymbol{0})\ \vee \\ \qquad (K \geq 1 \wedge \Delta > 0 \wedge (\boldsymbol{u}^Q, \Delta) = \sum_{i=1}^{n} \beta_i \boldsymbol{c}_i\ \wedge\ \bigwedge_{i=1}^{n} \beta_i \geq 0\ \wedge\ \sum_{i=1}^{n} \beta_i = K)\big) \end{array} \tag{4}$$

Proof. For a set of vectors S and an integer variable K, we define $KS = \{v_1 + \ldots + v_K \mid v_1, \ldots, v_K \in S\}$. Formula (3) is satisfiable iff there exists non-negative integer $K \in \mathbb{N}$ such that both

$$\boldsymbol{u}^Z \in K\{\boldsymbol{x}^Z \mid A_Z \boldsymbol{x}^Z \leq \boldsymbol{b}^Z\} \tag{5}$$

and

$$(\boldsymbol{u}^Q, \Delta) \in K\{(\boldsymbol{x}^R, \delta) \mid A_R \cdot (\boldsymbol{x}^R, \delta) \leq \boldsymbol{b}^R, \boldsymbol{x}^R \in \mathbb{Q}_{[0,1]}^{d_R}, \delta \in \mathbb{Q}_{(0,1]}\} \tag{6}$$

hold. We show how to describe (5) and (6) as existentially quantified MLIRA formulas that share the variable K.

To express formula (5) as a MLIRA formula, we use the fact that solutions of integer linear arithmetic formulas are semilinear sets (see [7], [11, Proposition 2]). Semilinear sets are finite unions of sets of a form $\{\boldsymbol{a}\} + \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n\}^*$. A sum of two sets is the Minkowski sum: $A + B = \{\boldsymbol{a} + \boldsymbol{b} \mid \boldsymbol{a} \in A,\ \boldsymbol{b} \in B\}$. It was shown in [14, 12] that if S is a semilinear set then $\boldsymbol{u} \in S^*$ can be expressed as Presburger arithmetic formula. In particular, formula (5) is equivalent to

$$\begin{aligned}\exists \mu_1, \ldots, \mu_q, \nu_{11}, \ldots, \nu_{qq_q} \in \mathbb{N}.\ \boldsymbol{u}^Z = \textstyle\sum_{i=1}^{q} (\mu_i \boldsymbol{a}_i + \sum_{j=1}^{q_i} \nu_{ij} \boldsymbol{b}_{ij}) \wedge \\ \bigwedge_{i=1}^{q} (\mu_i = 0 \rightarrow \textstyle\sum_{j=1}^{q_i} \nu_{ij} = 0)\ \wedge\ (\sum_{i=1}^{q} \mu_i = K)\end{aligned} \tag{7}$$

where vectors $\boldsymbol{a}_i$'s and $\boldsymbol{b}_{ij}$ can be computed effectively from A_Z and b^Z.

We next characterize condition (6). Renaming variables and incorporating the boundedness of $\boldsymbol{x}, \delta$ into the linear inequations, we can write such condition in the form

$$(\boldsymbol{u}^Q, \Delta) \in K\{(\boldsymbol{x}, \delta) \mid A \cdot (\boldsymbol{x}, \delta) \leq \boldsymbol{b}, \delta > 0\} \tag{8}$$

Here $A \cdot (\boldsymbol{x}, \delta) \leq b$ subsumes the conditions $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1}$, $0 \leq \delta \leq 1$. From the theory of linear programming [17] it follows that the set $\{(\boldsymbol{x}, \delta) \mid A \cdot (\boldsymbol{x}, \delta) \leq \boldsymbol{b}\}$ is a polyhedron, and because the solution set is bounded, it is in fact a polytope. Therefore, there exist finitely many vertices $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n \in \mathbb{Q}_{[0,1]}^d$ for some d such that $A \cdot (\boldsymbol{x}, \delta) \leq \boldsymbol{b}$ is equivalent to

$$\exists \lambda_1, \ldots, \lambda_n \in \mathbb{Q}_{[0,1]}.\ \sum_{i=1}^{n} \lambda_i = 1\ \wedge\ (\boldsymbol{x}, \delta) = \sum_{i=1}^{n} \lambda_i \boldsymbol{c}_i$$

Consequently, (8) is equivalent to

$$\begin{aligned}\exists \boldsymbol{u}_1, \ldots, \boldsymbol{u}_K.\ (\boldsymbol{u}^Q, \Delta) = \sum_{j=1}^{K} \boldsymbol{u}_j\ \wedge\ \exists \lambda_{11}, \ldots, \lambda_{Kn}. \\ \bigwedge_{j=1}^{K} \Big(\bigwedge_{i=1}^{n} \lambda_{ij} \geq 0\ \wedge\ \sum_{i=1}^{n} \lambda_{ij} = 1\ \wedge\ (\boldsymbol{u}_j, \delta_j) = \sum_{i=1}^{n} \lambda_{ij} \boldsymbol{c}_i \wedge \delta_j > 0 \Big)\end{aligned} \tag{9}$$

It remains to show that the above condition is equivalent to

$$\begin{aligned}\exists \beta_1, \ldots, \beta_n.\ \big((K = 0 \wedge \Delta = 0 \wedge \boldsymbol{u} = \boldsymbol{0}) \vee \\ (K \geq 1 \wedge \Delta > 0 \wedge (\boldsymbol{u}^Q, \Delta) = \sum_{i=1}^{n} \beta_i \boldsymbol{c}_i\ \wedge\ \bigwedge_{i=1}^{n} \beta_i \geq 0\ \wedge\ \sum_{i=1}^{n} \beta_i = K) \big)\end{aligned} \tag{10}$$

Case $K = 0$ is trivial, so assume $K \neq 0$. Consider a solution of (9). Letting $\beta_i = \sum_{j=1}^{K} \lambda_{ij}$ we obtain a solution of (10). Conversely, consider a solution of (10). Letting $\alpha_{ij} = \beta_i/K$, $\boldsymbol{u}_j = \boldsymbol{u}/K$, $\delta_j = \Delta/n$ we obtain a solution of (9). This shows the equivalence of (9) and (10).

Conjoining formulas (10) and (7) we complete the proof of Theorem 2. ∎

Satisfiability checking for collection formulas. Because star elimination (as well as the preparatory steps in Section 4) introduce only existential quantifiers, and the satisfiability of MLIRA formulas is decidable (see e.g. [4, 3]), we obtain the decidability of the initial formula $G(\boldsymbol{r}, \boldsymbol{w}) \wedge \boldsymbol{w} \in \{\boldsymbol{x} \mid F(\boldsymbol{x})\}^*$. Thanks to transformation to sum normal form and Theorem 1, we obtain the decidability of formulas involving sets, multisets and fuzzy sets.

6 Language with Nested Star Operators and Quantifiers

Theorem 2 can be combined with quantifier elimination for MLIRA formulas [18] to decide a language that permits nested uses of quantifiers and stars. Figure 5 summarizes the syntax of one such language. Note that the expression $(r_1, \ldots, r_n) \in \{(t_1, \ldots, t_n) | F\}^*$ has the same meaning as before and its only free variables are in $r_1, \ldots, r_n$ (the variables in $\{(t_1, \ldots, t_n) | F\}$ are all bound). To decide constraints in this language, we eliminate stars and quantifiers starting from the innermost ones. If the innermost operator is a quantifier, we eliminate it as in [18]. If the innermost operator is a star, we use results of Section 4 and Theorem 2 while keeping all existential quantifiers explicitly to preserve equivalence of the subformula. We obtain an existentially quantifier subformula without stars. We eliminate the generated existential quantifiers by again applying quantifier elimination [18]. Repeating this method we obtain a quantifier-free formula without stars, whose satisfiability can be checked [4, 3].

$$
\begin{array}{rl}
F & ::= A \mid F \wedge F \mid \neg F \mid \exists x.F \\
A & ::= t \leq t \mid t{=}t \mid (t, \ldots, t) {\in} \{(t, \ldots, t) | F\}^* \\
t & ::= k \mid K \mid t + t \mid K \cdot t \mid \lfloor t \rfloor \mid \mathsf{ite}(F, t, t)
\end{array}
$$

Fig. 5. Syntax of Constraints with Nested Stars and Quantifiers

The language of Figure 5 can be further generalized to allow atomic formulas of the form $(t, \ldots, t) \in S$ where the syntax of S is given by

$$S ::= \{(t, \ldots, t) | F\} \mid S \cup S \mid S \setminus S \mid S + S \mid t \cdot S \mid S^*$$

The basic idea is to flatten such set expressions, eliminate operators $\cup$, $\setminus$, $+$ using their definition, and eliminate S^* using the algorithms we just described. The case of $t \cdot S$ is similar to S^* but the value K from Theorem 2 is fixed and given by term t, as opposed to being existentially quantified.

7 Related Work

Logical constraints on collections that do not support cardinality bounds have been studied in the past. Zarba [20] considered decision procedures for quantifier-free multisets but without the cardinality operator, showing that it reduces to quantifier-free pointwise reasoning. The cardinality operator makes that reduction impossible. Notions of the cardinality operator naturally arising from the Feferman-Vaught theorem [6] can express only a finite amount of information for each element $e \in E$, so they are appropriate only for cardinality sets or for the cardinality of the support of the multisets or a fuzzy set. Recently, Lugiez [10] shows the decidability of constraints with a weaker form of such a limited cardinality operator that counts only distinct elements in a multiset, and shows decidability of certain quantifier-free expressible constraints with cardinality operator.

Note that, because our Theorem 1 is only equisatisfiability and not equivalence, we do not obtain decidability of constraints with quantified collections. In fact, although quantified sets with cardinality bounds are decidable [6, 8], quantified multisets with cardinality bounds are undecidable [12, Section 6].

The work in this paper is based on previous results for the special cases of sets [9] and multisets [14, 13, 12]. We rely on the fact that solutions of formulas of Presburger arithmetic are semilinear sets [7]. Bounds on generators of such sets are presented in [15].

Techniques for deciding formulas of MLIRA formulas are part of implementations of modern satisfiability modulo theory theorem provers [4,3,1] and typically use SAT solving techniques along with techniques from mixed integer-linear programming, or the Omega test [16].

8 Conclusions

We have shown decidability of a rich logic for reasoning about collections. The logic is expressive enough for reasoning about sets, multisets, and fuzzy sets as well as their cardinality bounds. Our results also show that star, much like quantifiers, is a natural operator of MLIRA formulas and can also be eliminated. A direct application of our star elimination technique creates an exponentially larger MLIRA formula. We leave for future work the question whether it is possible to generate polynomially large equisatisfiable formulas as for multisets [13].

Acknowledgements. We thank Nikolaj Bjørner for useful discussions.

References

1. Berezin, S., Ganesh, V., Dill, D.L.: An online proof-producing decision procedure for mixed-integer linear arithmetic. In: TACAS (2003)
2. Bradley, A.R., Manna, Z.: The Calculus of Computation. Springer, Heidelberg (2007)

3. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer, Heidelberg (2006)
4. Dutertre, B., de Moura, L.: Integrating Simplex with DPLL(T). Technical Report SRI-CSL-06-01, SRI International (2006)
5. Eisenbrand, F., Shmonin, G.: Carathéodory bounds for integer cones. Operations Research Letters 34(5), 564–568 (2006), http://dx.doi.org/10.1016/j.orl.2005.09.008
6. Feferman, S., Vaught, R.L.: The first order properties of products of algebraic systems. Fundamenta Mathematicae 47, 57–103 (1959)
7. Ginsburg, S., Spanier, E.: Semigroups, Pressburger formulas and languages. Pacific Journal of Mathematics 16(2), 285–296 (1966)
8. Kuncak, V., Nguyen, H.H., Rinard, M.: Deciding Boolean Algebra with Presburger Arithmetic. J. of Automated Reasoning (2006), http://dx.doi.org/10.1007/s10817-006-9042-1
9. Kuncak, V., Rinard, M.: Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In: Pfenning, F. (ed.) CADE 2007. LNCS (LNAI), vol. 4603, pp. 215–230. Springer, Heidelberg (2007)
10. Lugiez, D.: Multitree automata that count. Theor. Comput. Sci. 333(1-2), 225–263 (2005)
11. Lugiez, D., Zilio, S.D.: Multitrees Automata, Presburger's Constraints and Tree Logics. Research report 08-2002, LIF, Marseille, France (June 2002), http://www.lif-sud.univ-mrs.fr/Rapports/08-2002.html
12. Piskac, R., Kuncak, V.: Decision procedures for multisets with cardinality constraints. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 218–232. Springer, Heidelberg (2008)
13. Piskac, R., Kuncak V.: Linear arithmetic with stars. In: Gupta A., Malik S. (Eds.): CAV 2008. LNCS, vol. 5123, pp. 268-280. Springer, Heidelberg (2008)
14. Piskac, R., Kuncak, V.: On linear arithmetic with stars. Technical Report LARA-REPORT-2008-005, EPFL (2008)
15. Pottier, L.: Minimal solutions of linear diophantine systems: Bounds and algorithms. In: Book, R.V. (ed.) RTA 1991. LNCS, vol. 488. Springer, Heidelberg (1991)
16. Pugh, W.: The Omega test: a fast and practical integer programming algorithm for dependence analysis. In: ACM/IEEE conf. Supercomputing (1991)
17. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester (1998)
18. Weispfenning, V.: Mixed real-integer linear quantifier elimination. In: ISSAC, pp. 129–136 (1999)
19. Zadeh, L.A.: Fuzzy sets. Information and Control 8, 338–353 (1965)
20. Zarba, C.G.: Combining multisets with integers. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392. Springer, Heidelberg (2002)

Continuous Fragment of the mu-Calculus

Gaëlle Fontaine*

Institute for Logic, Language and Computation,
Plantage Muidergracht 24, 1018 TV Amsterdam, The Netherlands
gfontain@science.uva.nl
http://staff.science.uva.nl/ gfontain

Abstract. In this paper we investigate the Scott continuous fragment of the modal μ-calculus. We discuss its relation with constructivity, where we call a formula constructive if its least fixpoint is always reached in at most ω steps. Our main result is a syntactic characterization of this continuous fragment. We also show that it is decidable whether a formula is continuous.

Keywords: mu-calculus, automata, Scott continuity, constructive fixpoints, preservation results.

1 Introduction

This paper is a study into the fragment of the modal μ-calculus that we call continuous. Roughly, given a proposition letter p, a formula φ is said to be continuous in p if it monotone in p and if in order to establish the truth of φ at a point, we only need finitely many points at which p is true. The continuous fragment of the μ-calculus is defined as the fragment of the μ-calculus in which $\mu x.\varphi$ is allowed only if φ is continuous in x.

We prove the following two results. First, Theorem 2 gives a natural syntactic characterization of the continuous formulas. Informally, continuity corresponds to the formulas built using the operators $\vee$, $\wedge$, $\Diamond$ and μ. Second, we show in Theorem 3 that it is decidable whether a formula is continuous in p.

We believe that this continuous fragment is of interest for a number of reasons. A first motivation concerns the relation between continuity and another property, constructivity. The constructive formulas are the formulas whose fixpoint is reached in at most ω steps. Locally, this means that a state satisfies a least fixpoint formula if it satisfies one of its *finite* approximations. It is folklore that if a formula is continuous, then it is constructive. The other implication does not strictly hold. However, interesting questions concerning the link between constructivity and continuity remain. In any case, given our Theorem 2, continuity can be considered as the most natural candidate to approximate constructivity syntactically.

* The research of the author has been made possible by VICI grant 639.073.501 of the Netherlands Organization for Scientific Research (NWO).

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 139–153, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Next this fragment can be seen as a natural generalization of PDL in the following way. We define the completely additive formulas as the formulas built using the operators $\vee$, $\Diamond$ and μ. That is, the syntax is the same as for the continuous formulas, except that the conjunction is not allowed. Then it was observed by Yde Venema (personal communication) that PDL coincides with the fragment of the μ-calculus in which $\mu x.\varphi$ is allowed only if φ is completely additive. In this perspective, the continuous fragment appears as a natural extension of PDL.

Another reason for looking at this fragment (which also explains the name) is the link with Scott continuity. A formula is continuous in p iff it is continuous with respect to p in the Scott topology on the powerset algebra (with all other variables fixed). Scott continuity is of key importance in many areas of theoretical computer sciences where ordered structures play a role, such as domain theory (see, e.g., [1]). For many purposes, it is sufficient to check that a construction is Scott continuous in order to show that it is computationally feasible.

Finally our results fit in a model-theoretic tradition of so-called preservation results (see, e.g., [2]). Giovanna D'Agostino and Marco Hollenberg have proved some results of this kind in the case of the μ-calculus (see, e.g., [3] and [4]). Their proofs basically consist in identifying automata corresponding to the desired fragment and in showing that these automata give the announced characterization. The proof of our main result is similar as we also first start by translating our problem in terms of automata. We also mention that a version of our syntactic characterization in the case of first order logic has been obtained by Johan van Benthem in [5].

The paper is organized as follows. First we recall the syntax of the μ-calculus and some basic properties that will be used later on. Next we define the continuous fragment and we show how it is linked to Scott continuity, constructivity and PDL. Finally we prove our main result (Theorem 2) which is a syntactic characterization of the fragment and we show that it is decidable whether a formula is continuous (Theorem 3). We end the paper with questions for further research.

2 Preliminaries

We introduce the language and the Kripke semantic for the μ-calculus.

Definition 1. *Let Prop be a finite set of proposition letters and let Var be a countable set of variables. The* formulas of the μ-calculus *are given by*

$$\varphi \ ::= \ \top \mid p \mid x \mid \varphi \vee \varphi \mid \neg\varphi \mid \Diamond\varphi \mid \mu x.\varphi,$$

where p ranges over the set Prop and x ranges over the set Var. In $\mu x.\varphi$, we require that every occurrence of x is under an even number of negations in φ. The notion of closed μ-formula *or* μ-sentence *is defined in the natural way.*

As usual, we let $\varphi \wedge \psi$, $\Box\varphi$ and $\nu x.\varphi$ be abbreviations for $\neg(\neg\varphi \vee \neg\psi)$, $\neg\Diamond\neg\varphi$ and $\neg\mu x.\neg\varphi[\neg x/x]$. For a set of formulas Φ, we denote by $\bigvee\Phi$ the disjunction of formulas in Φ. Similarly, $\bigwedge\Phi$ denotes the conjunction of formulas in Φ.

Finally, we extend the syntax of the μ-calculus by allowing a new construct of the form $\nabla\Phi$, where Φ is a finite set of formulas. We will consider such a formula to be an abbreviation of $\bigwedge\{\Diamond\varphi : \varphi \in \Phi\} \wedge \Box \bigvee \Phi$. Remark that in [6], David Janin and Igor Walukiewicz use the notation $a \rightarrow \Phi$ for $\nabla_a\Phi$.

For reasons of a smooth presentation, we restrict to the unimodal fragment. All the results can be easily extended to the setting where we have more than one basic modality.

Definition 2. *A* Kripke frame *is a pair (M, R), where M is a set and R a binary relation on M. A* Kripke model *$\mathcal{M}$ is a triple (M, R, V) where (M, R) is a Kripke frame and $V : Prop \rightarrow \mathcal{P}(M)$ a valuation.*

If sRt, we say that t is a successor *of s and we write $R(s)$ to denote the set $\{t \in M : sRt\}$. A* path *is a (finite or infinite) sequence $s_0, s_1, \ldots$ such that $s_i R s_{i+1}$ (for all $i \in \mathbb{N}$).*

Definition 3. *Given a μ-formula φ, a model $\mathcal{M} = (M, R, V)$ and an assignment $\tau : Var \rightarrow \mathcal{P}(M)$, we define a subset $[\![\varphi]\!]_{\mathcal{M},\tau}$ of M that is interpreted as the set of points at which φ is true. This subset is defined by induction in the usual way. We only recall that*

$$[\![\mu x.\varphi]\!]_{\mathcal{M},\tau} = \bigcap\{U \subseteq M : [\![\varphi]\!]_{\mathcal{M},\tau[x:=U]} \subseteq U\},$$

where $\tau[x := U]$ is the assignment τ' such that $\tau'(x) = U$ and $\tau'(y) = \tau(y)$ for all $y \neq x$.

Observe that the set $[\![\mu x.\varphi]\!]_{\mathcal{M},\tau}$ is the least fixpoint of the map $\varphi_x : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$ defined by $\varphi_x(U) := [\![\varphi]\!]_{\mathcal{M},\tau[x:=U]}$, for all $U \subseteq M$. Similarly, for a proposition letter p, we can define the map $\varphi_p : \mathcal{P}(M) \rightarrow \mathcal{P}(M)$ by $\varphi_p(U) := [\![\varphi]\!]_{\mathcal{M}[p:=U],\tau}$, where $\mathcal{M}[p := U]$ is the model (M, R, V') with $V'(p) = U$ and $V'(p') = V(p')$, for all $p' \neq p$.

If $s \in [\![\varphi]\!]_{\mathcal{M},\tau}$, we write $\mathcal{M}, s \Vdash_\tau \varphi$ and we say that φ true *at $s \in \mathcal{M}$ under the assignment τ. If φ is a sentence, we simply write $\mathcal{M}, s \Vdash \varphi$.*

A formula φ is monotone *in a proposition letter p if for all models $\mathcal{M} = (M, R, V)$, all assignments τ and all sets $U, U' \subseteq M$ satisfying $U \subseteq U'$, we have $\varphi_p(U) \subseteq \varphi_p(U')$. The notion of monotonicity in a variable x is defined in an analogous way.*

Finally we use the notation $\varphi \models \psi$ if for all models $\mathcal{M}$ and all points $s \in \mathcal{M}$, we have $\mathcal{M}, s \Vdash \varphi$ implies $\mathcal{M}, s \Vdash \psi$.

When deciding whether a sentence is true at a point s, it only depends on the points accessible (in possibly many steps) from s. These points together with the relation and the valuation inherited from the original model form the submodel generated by s. We will use this notion later on and we briefly recall the definition.

Definition 4. *Let $\mathcal{M} = (M, R, V)$ be a model. A subset N of M is* downward closed *if for all s and t, sRt and $t \in N$ imply that $s \in N$. N is* upward closed *if for all s and t, sRt and $s \in N$ imply that $t \in N$.*

A model $\mathcal{N} = (N, S, U)$ *is a* generated submodel *of* $\mathcal{M}$ *if* $N \subseteq M$, N *is upward closed,* $S = R \cap (N \times N)$ *and* $U(p) = V(p) \cap N$, *for all* $p \in Prop$. *If* N' *is a subset of* M, *we say that* $\mathcal{N} = (N, S, U)$ *is the* submodel generated *by* N' *if* $\mathcal{N}$ *is a generated submodel and if* N *is the smallest upward closed set containing* N'.

In our proof, it will be often more convenient to work with a certain kind of Kripke models. That is, we will suppose that the models we are dealing with are trees such that each point (except the root) has infinitely many bisimilar siblings. We make this definition precise and we give the results needed to justify this assumption.

Definition 5. *A point* s *is a* root *of a model* $\mathcal{M} = (M, R, V)$ *if for every* t *distinct from* s, *there is a path from* s *to* t. $\mathcal{M}$ *is a* tree *if it has a root, every point distinct from the root has a unique predecessor and* R *is acyclic (that is, there is no non-empty path starting at a point* t *and ending in* t*).*

A model $\mathcal{M} = (M, R, V)$ *is* ω-expanded *if it is a tree such that for all* $s \in M$ *and all successors* t *of* s, *there are infinitely many distinct successors of* s *that are bisimilar to* t.

Proposition 1. *Let* $\mathcal{M} = (M, R, V)$ *be a model and let* $s \in M$. *There exists a tree* $\mathcal{M}' = (M', R', V')$ *that is* ω*-expanded such that* s *and the root* s' *of* $\mathcal{M}'$ *are bisimilar. In particular, for all* μ*-sentences* φ, $\mathcal{M}, s \Vdash \varphi$ *iff* $\mathcal{M}', s' \Vdash \varphi$.

Another way to look at formulas of the μ-calculus is to consider automata. In [6], David Janin and Igor Walukiewicz define a notion of automaton that operates on Kripke models and that corresponds exactly to the μ-calculus.

Definition 6. *A* μ-automaton $\mathbb{A}$ *over a finite alphabet* Σ *is a tuple* (Q, q_0, δ, Ω) *such that* Q *is a finite set of states,* $q_0 \in Q$ *is the initial state,* $\delta : Q \times \Sigma \to \mathcal{P}\mathcal{P}(Q)$ *is the transition map and* $\Omega : Q \to \mathbb{N}$ *is the parity function.*

Given a frame $\mathcal{M} = (M, R, V)$ *with a labeling* $L : M \to \Sigma$ *and a point* $s \in M$, *an* $\mathbb{A}$*-game in* $\mathcal{M}$ *with starting position* (s, q_0) *is played between two players, the Duplicator and the Spoiler. The game is as follows: If we are in position* (t, q) *(where* $t \in M$ *and* $q \in Q$*), the Duplicator has to make a move. The Duplicator chooses a marking* $m : Q \to \mathcal{P}\{u : tRu\}$ *and then a description* D *in* $\delta(q, L(t))$. *If* $u \in m(q)$, *we say that* u *is* marked *with* q.

The marking and the description have to satisfy the two following properties. First, if $q' \in D$, *there exists a successor* u *of* t *that is marked with* q'. *Second, if* u *is a successor of* t, *there exists* $q' \in D$ *such that* u *is marked with* q'. *After the Duplicator has chosen a marking* m, *the Spoiler plays a position* (u, q') *such that* $t \in m(q')$.

Either player wins the game if the other player cannot make a move. An infinite match $(s, q_0), (s_1, q_1), \ldots$ *is won by the Duplicator if the smallest element of* $\{\Omega(q) : q$ *appears infinitely often in* $q_0, q_1, \ldots\}$ *is even.*

We say that $(\mathcal{M}, s)$ *is* accepted *by* $\mathbb{A}$ *if the Duplicator has a winning strategy in the* $\mathbb{A}$*-game in* $\mathcal{M}$ *with starting position* (s, q_0).

Remark that a model (M, R, V) can be seen as a frame (M, R) with a labeling $L : M \to \mathcal{P}(Prop)$ defined by $L(t) = \{p \in Prop : t \in V(p)\}$, for all $t \in M$.

Theorem 1. *[6] For every μ-automaton $\mathbb{A}$ (over the alphabet $\mathcal{P}(Prop)$), there is a sentence φ such that for all models $\mathcal{M}$ and all points $s \in \mathcal{M}$, $\mathbb{A}$ accepts $(\mathcal{M}, s)$ iff $\mathcal{M}, s \Vdash \varphi$. Conversely, for every sentence φ, there is a μ-automaton $\mathbb{A}$ such that for all models $\mathcal{M}$ and all points $s \in \mathcal{M}$, $\mathbb{A}$ accepts $(\mathcal{M}, s)$ iff $\mathcal{M}, s \Vdash \varphi$.*

3 Continuity

We define the notion of continuity for a formula and we show the connection with the Scott continuity. We also mention that these formulas are constructive and that there is a natural connection with PDL.

Definition 7. *Fix a proposition letter p. A sentence φ is* continuous in p *if for all models $\mathcal{M} = (M, R, V)$ and all $s \in M$, we have*

$$\mathcal{M}, s \Vdash \varphi \text{ iff } \exists\ F \subseteq V(p) \text{ s.t. } F \text{ is finite and } \mathcal{M}[p := F], s \Vdash \varphi.$$

The notion of continuity in x *(where x is a variable) is defined similarly.*

That is, a formula φ is continuous in p iff it is monotone in p and whenever φ is true at a point in a model, we only need finitely many points where p is true in order to establish the truth of φ.

Continuity and Scott Continuity

It does not seem very natural that a formula satisfying such a property should be called continuous. In fact, it is equivalent to require that the formula is Scott continuous with respect to p in the powerset algebra (with all other proposition letters fixed). In the next paragraph, we recall the definition of the Scott topology and we briefly show that the notion of Scott continuity and our last definition coincide.

Definition 8. *Let $\mathcal{M} = (M, R, V)$ be a model. A family $\mathcal{F}$ of subsets of M is* directed *if for all $U_1, U_2 \in \mathcal{F}$, there exists $U \in \mathcal{F}$ such that $U \supseteq U_1 \cup U_2$.*

A Scott open set *in the powerset algebra $\mathcal{P}(M)$ is a family $\mathcal{O}$ of subsets of M that is closed under upset (that is, if $U \in \mathcal{O}$ and $U' \supseteq U$, then $U' \in \mathcal{O}$) and such that for all directed family $\mathcal{F}$ satisfying $\bigcup \mathcal{F} \in \mathcal{O}$, the intersection $\mathcal{F} \cap \mathcal{O}$ is non-empty.*

As usual, a map $f : \mathcal{P}(M) \to \mathcal{P}(M)$ is Scott continuous *if for all Scott open sets $\mathcal{O}$, the set $f^{-1}[\mathcal{O}] = \{f^{-1}(U) : U \in \mathcal{O}\}$ is Scott open.*

Fix a proposition letter p. A sentence φ is Scott continuous in p *if for all models $\mathcal{M} = (M, R, V)$, the map $\varphi_p : \mathcal{P}(M) \to \mathcal{P}(M)$ is Scott continuous.*

Remark that the Scott topology can be defined in an arbitrary partial order (see, e.g., [7]). It is a fairly standard result that a map f is Scott continuous iff it preserves directed joins. That is, for all directed family $\mathcal{F}$, we have $f(\bigcup \mathcal{F}) = \bigcup f[\mathcal{F}]$ (where $f[\mathcal{F}] = \{f(U) : U \in \mathcal{F}\}$). Now we check that our notion of continuity defined in a Kripke semantic framework is equivalent to the standard definition of Scott continuity.

Proposition 2. *A sentence is continuous in p iff it is Scott continuous in p.*

Proof. For the direction from left to right, let φ be a continuous sentence in p. Fix a model $\mathcal{M} = (M, R, V)$. We show that the map $\varphi_p : \mathcal{P}(M) \to \mathcal{P}(M)$ preserves directed joins.

Let $\mathcal{F}$ be a directed family. It follows from the monotonicity of φ that the set $\bigcup \varphi_p[\mathcal{F}]$ is a subset of $\varphi_p(\bigcup \mathcal{F})$. Thus, it remains to show that $\varphi_p(\bigcup \mathcal{F}) \subseteq \bigcup \varphi_p[\mathcal{F}]$. Take s in $\varphi_p(\bigcup \mathcal{F})$. That is, the formula φ is true at s in the model $\mathcal{M}[p := \bigcup \mathcal{F}]$. As φ is continuous in p, there is a finite subset F of $\bigcup \mathcal{F}$ such that φ is true at s in $\mathcal{M}[p := F]$. Now, since F is a finite subset of $\bigcup \mathcal{F}$ and since $\mathcal{F}$ is directed, there exists a set U in $\mathcal{F}$ such that F is a subset of U. Moreover, as φ is monotone, $\mathcal{M}[p := F], s \Vdash \varphi$ implies $\mathcal{M}[p := U], s \Vdash \varphi$. Therefore, s belongs to $\varphi_p(U)$ and in particular, s belongs to $\bigcup \varphi_p[\mathcal{F}]$. This finishes to show that $\varphi_p(\bigcup \mathcal{F}) \subseteq \bigcup \varphi_p[\mathcal{F}]$.

For the direction from right to left, let φ be a Scott continuous sentence in p. First we show that φ is monotone in p. Let $\mathcal{M} = (M, R, V)$ be a model. We check that $\varphi_p(U) \subseteq \varphi_p(U')$, in case $U \subseteq U'$. Suppose $U \subseteq U'$ and let $\mathcal{F}$ be the set $\{U, U'\}$. The family $\mathcal{F}$ is clearly directed and satisfies $\bigcup \mathcal{F} = U'$. Using the fact that φ_p preserves directed joins, we get that $\varphi_p(U') = \varphi_p(\bigcup \mathcal{F}) = \bigcup \varphi_p[\mathcal{F}]$. By definition of $\mathcal{F}$, we have $\bigcup \varphi_p[\mathcal{F}] = \varphi_p(U) \cup \varphi_p(U')$. Putting everything together, we obtain that $\varphi_p(U') = \varphi_p(U) \cup \varphi_p(U')$. Thus, $\varphi_p(U) \subseteq \varphi_p(U')$.

To show that φ is continuous in p, it remains to show that if $\mathcal{M}, s \Vdash \varphi$, then there exists a finite subset F of $V(p)$ such that $\mathcal{M}[p := F], s \Vdash \varphi$. Suppose that the formula φ is true at s in $\mathcal{M}$. That is, s belongs to $\varphi_s(V(p))$. Now let $\mathcal{F}$ be the family $\{F \subseteq V(p) : F \text{ finite}\}$. It is not hard to see that $\mathcal{F}$ is a directed family satisfying $\bigcup \mathcal{F} = V(p)$. Since φ_p preserves directed joins, we obtain $\varphi_p(V(p)) = \varphi_p(\bigcup \mathcal{F}) = \bigcup \varphi_p[\mathcal{F}]$. From $s \in \varphi_p(V(p))$, it then follows that $s \in \bigcup \varphi_p[\mathcal{F}]$. Therefore, there exists $F \in \mathcal{F}$ such that $s \in \varphi_p(F)$. That is, F is a finite subset of $V(p)$ such that $\mathcal{M}[p := F], s \Vdash \varphi$.

Continuity and Constructivity

A formula is constructive if its fixpoint is reached in at most ω steps. Formally, we have the following definition.

Definition 9. *Fix a proposition letter p. A sentence φ is* constructive *in p if for all models $\mathcal{M} = (M, R, V)$, the least fixpoint of the map $\varphi_p : \mathcal{P}(M) \to \mathcal{P}(M)$ is equal to $\bigcup\{\varphi_p^i(\emptyset) : i \in \mathbb{N}\}$ (where φ_p^i is defined by induction by $\varphi_p^0 = \varphi_p$ and $\varphi_p^{i+1} = \varphi_p \circ \varphi_p^i$).*

Locally, this means that given a formula φ constructive in p and a point s in a model at which $\mu p.\varphi$ is true, there is some natural number n such that s belongs to the finite approximation $\varphi_p^n(\emptyset)$. We observe that a continuous formula is constructive.

Proposition 3. *A sentence φ continuous in p is constructive in p.*

Proof. Let φ be a sentence continuous in p and let $\mathcal{M} = (M, R, V)$ be a model. We show that the least fixpoint of φ_p is $\bigcup\{\varphi_p^i(\emptyset) : i \in \mathbb{N}\}$.

Let $\mathcal{F}$ be the family $\{\varphi_p^i(\emptyset) : i \in \mathbb{N}\}$. It is enough to check that $\varphi_p(\bigcup \mathcal{F}) = \bigcup \mathcal{F}$. First remark that $\mathcal{F}$ is directed. Therefore, $\varphi_p(\bigcup \mathcal{F}) = \bigcup \varphi_p[\mathcal{F}]$. It is also easy to prove that $\bigcup \varphi_p[\mathcal{F}] = \bigcup \mathcal{F}$. Putting everything together, we obtain that $\varphi_p(\bigcup \mathcal{F}) = \bigcup \mathcal{F}$ and this finishes the proof.

Remark that a constructive sentence might not be continuous.

Example 1. Let φ be the formula $\Box p \wedge \Box\Box\bot$. Basically, φ is true at a point s in a model if the depth of s is less or equal to 2 (that is, there are no t and t' satisfying $sRtRt'$) and all successors of s satisfy p. It is not hard to see that φ is not continuous in p. However, we have that for all models $\mathcal{M} = (M, R, V)$, $\varphi_p^2(\emptyset) = \varphi_p^3(\emptyset)$. In particular, φ is constructive in p.

Example 2. Let ψ be the formula $\nu x.p \wedge \Diamond x$. The formula ψ is true at a point s if there is a infinite path starting from s and at each point of this path, p is true. This sentence is not continuous in p. However, it is constructive, since for all models $\mathcal{M} = (M, R, V)$, we have $\psi_p(\emptyset) = \emptyset$.

Observe that in the previous examples we have $\mu p.\varphi \equiv \mu p.\Box\Box\bot$ and $\mu p.\psi \equiv \mu p.\bot$. Thus, there is a continuous sentence (namely $\Box\Box\bot$) that is equivalent to φ, modulo the least fixpoint operation. Similarly, there is a continuous sentence (the formula $\bot$) that is equivalent to ψ, modulo the least fixpoint operation. This suggests the following question.

Question 1 (Yde Venema). Given a constructive formula φ, can we find a continuous formula ψ satisfying $\mu p.\varphi \equiv \mu p.\psi$?

The answer is still unknown and this could be a first step for further study of the relation between continuity and constructivity.

Decidability of constructivity is also an interesting question. We would like to mention that in [8], Martin Otto proved that it is decidable in EXPTIME whether a basic modal formula $\varphi(p)$ is bounded. We recall that a basic modal formula $\varphi(p)$ is bounded if there is a natural number n such that for all models $\mathcal{M}$, we have $\varphi_p^n(\emptyset) = \varphi_p^{n+1}(\emptyset)$.

Continuity and PDL

We finish this section by few words about the connection between the continuous fragment and PDL. We start by defining the completely additive formulas.

Definition 10. *Let P be a subset of $Prop$ and let X be a subset of Var. The set of* completely additive formulas *with respect to $P \cup X$ is defined by induction in the following way:*

$$\varphi \ ::= \ \top \mid p \mid x \mid \psi \mid \varphi \vee \varphi \mid \Diamond\varphi \mid \mu y.\chi,$$

where p is in P, x is in X, ψ is a formula of the μ-calculus such that the proposition letters of ψ and the variables of ψ do not belong to $P \cup X$ and χ is completely additive with respect to $P \cup X \cup \{y\}$.

We define the *completely additive fragment* as the fragment of the μ-calculus in which $\mu x.\varphi$ is allowed only if φ is completely additive with respect to x. As mentioned in the introduction, it was observed by Yde Venema that this fragment coincides with test-free PDL.

Similarly, we define the *continuous fragment* as the fragment of the μ-calculus in which $\mu x.\varphi$ is allowed only if φ is continuous in x. It is routine to check that any completely additive formula with respect to p is continuous in p (and the proof is similar to the proof of Lemma 1 below). In particular, the completely additive fragment is included in the continuous fragment. That is, PDL is a subset of the continuous fragment. We remark that this inclusion is strict. An example is the formula $\varphi = \mu x.(\Diamond(p \wedge x) \wedge \Diamond(q \wedge x))$. This formula belongs to the continuous fragment but is not equivalent to a formula in PDL. Roughly, the sentence φ is true at a point s if there is a finite binary tree-like submodel starting from s, such that each non-terminal node of the tree has a child at which p is true and a child at which q is true. This example was given by Johan van Benthem in [9].

4 Syntactic Characterization of the Continuous Fragment

In this section, we give a characterization of the continuous fragment of the μ-calculus. The main result states that the sentences which are continuous in p are exactly the sentences such that p and the variables are only in the scope of the operators $\vee$, $\wedge$, $\Diamond$ and μ. These formulas are formally defined as the set $CF(p)$.

Definition 11. *Let P be a subset of $Prop$ and let X be a subset of Var. The set of formulas $CF(P \cup X)$ is defined by induction in the following way:*

$$\varphi ::= \top \mid p \mid x \mid \psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \mu y.\chi,$$

where p is in P, x is in X, ψ is a formula of the μ-calculus such that the proposition letters of ψ and the variables of ψ do not belong to $P \cup X$ and χ belongs to $CF(P \cup X \cup \{y\})$. We abbreviate $CF(\{p\})$ to $CF(p)$.

As a first property, we mention that the formulas in $CF(P \cup X)$ are closed under composition.

Proposition 4. *If φ is in $CF(P \cup X \cup \{p\})$ and ψ is in $CF(P \cup X)$, then $\varphi[\psi/p]$ belongs to $CF(P \cup X)$.*

Proof. By induction on φ.

Next we observe that the sentences in $CF(p)$ are continuous.

Lemma 1. *A sentence φ in $CF(p)$ is continuous in p.*

Proof. We prove by induction on φ that for all sets $P \subseteq Prop$ and $X \subseteq Var$, $\varphi \in CF(P \cup X)$ implies that φ is continuous in p and in x, for all $p \in P$ and all $x \in X$. We focus on the inductive step $\varphi = \mu y.\chi$, where χ is in $CF(P \cup X \cup \{y\})$.

We also restrict ourselves to show that φ is continuous in p, for a proposition letter p in P.

Fix a proposition letter $p \in P$. First we introduce the following notation. For a model $\mathcal{M} = (M, R, V)$, an assignment τ and a subset U of W, we let $\chi_y^U : \mathcal{P}(M) \to \mathcal{P}(M)$ be the map defined by $\chi_y^U(W) = [\![\chi]\!]_{\mathcal{M}[p:=U],\tau[y:=W]}$, for all $W \subseteq M$. We also denote by $f(U)$ the least fixpoint of χ_y^U.

Now we show that φ is monotone in p. That is, for all models $\mathcal{M} = (M, R, V)$, all assignments τ and all subsets U, U' of M such that $U \subseteq U'$, we have $\mathcal{M}[p := U], s \Vdash_\tau \mu y.\chi$ implies $\mathcal{M}[p := U'], s \Vdash_\tau \mu y.\chi$. Fix a model $\mathcal{M} = (M, R, V)$, an assignment τ and sets $U, U' \subseteq M$ satisfying $U \subseteq U'$. Suppose $\mathcal{M}[p := U_0], s \Vdash_\tau \mu y.\chi$. That is, s belongs to the least fixpoint $f(U)$of the map χ_y^U. Since χ is monotone in p, we have that for all $W \subseteq M$, $\chi_y^U(W) \subseteq \chi_y^{U'}(W)$. It follows that the least fixpoint $f(U)$ of the map χ_y^U is a subset of the least fixpoint of the map $\chi_y^{U'}$. Putting this together with $s \in f(U)$, we get that s belongs to the least fixpoint of $\chi_y^{U'}$. That is, $\mathcal{M}[p := U'], s \Vdash_\tau \mu y.\chi$ and this finishes the proof that φ is monotone in p.

Next suppose that $\mathcal{M}, s \Vdash_\tau \mu y.\chi$, for a point s in a model $\mathcal{M} = (M, R, V)$. That is, s belongs to the least fixpoint L of the map χ_y. Now let $\mathcal{F}$ be the set of finite subsets of $V(p)$. We need to find a set $F \in \mathcal{F}$ satisfying $\mathcal{M}[p := F], s \Vdash_\tau \mu y.\chi$. Or equivalently, we have to show that there exists $F \in \mathcal{F}$ such that s belongs to the least fixpoint $f(F)$ of the map χ_y^F.

Let $\mathcal{G}$ be the set $\{f(F) : F \in \mathcal{F}\}$. It is routine to show that $\mathcal{G}$ is a directed family. Since L is the least fixpoint of χ_y, we have that for all $U \subseteq M$, $\chi_y(U) \subseteq U$ implies $L \subseteq U$. So if we can prove that $\chi_y(\bigcup \mathcal{G}) \subseteq \bigcup \mathcal{G}$, we will obtain $L \subseteq \bigcup \mathcal{G}$. Putting this together with $s \in L$, it will follow that $s \in \bigcup\{f(F) : F \in \mathcal{F}\}$. Therefore, in order to show that $s \in f(F)$ for some $F \in \mathcal{F}$, it is sufficient to prove that $\chi_y(\bigcup \mathcal{G}) \subseteq \bigcup \mathcal{G}$.

Assume $t \in \chi_y(\bigcup \mathcal{G})$. Since $\mathcal{G}$ is a directed family and χ is Scott continuous in y, we have $\chi_y(\bigcup \mathcal{G}) = \bigcup \chi_y(\mathcal{G})$. Thus, there exists $F_0 \in \mathcal{F}$ such that $t \in \chi_y(f(F_0))$. Now since χ is continuous in p, there exists a finite set $F_1 \subseteq V(p)$ such that $t \in \chi_y^{F_1}(f(F_0))$. Let F be the set $F_0 \cup F_1$. Since χ is monotone in p, $t \in \chi_y^{F_1}(f(F_0))$ implies $t \in \chi_y^F(f(F_0))$. It also follows from the monotonicity in p that for all $U \subseteq M$, $\chi_y^{F_0}(U) \subseteq \chi_y^F(U)$. Therefore, the least fixpoint $f(F_0)$ of $\chi_y^{F_0}$ is a subset of the least fixpoint $f(F)$ of χ_y^F. Using the fact that χ is monotone in y and the inclusion $f(F_0) \subseteq f(F)$, we obtain $\chi_y^F(f(F_0)) \subseteq \chi_y^F(f(F))$. Putting this together with $t \in \chi_y^F(f(F_0))$, we get $t \in \chi_y^F(f(F))$. Moreover, since $f(F)$ is a fixpoint of χ_y^F, we have $\chi_y^F(f(F)) = f(F)$. Hence, t belongs to $f(F)$. In particular, t belongs to $\bigcup \mathcal{G}$ and this finishes the proof.

We also prove the converse: the sentences in $CF(p)$ are enough to characterize the continuous fragment of the μ-calculus. The proof is inspired by the one given by Marco Hollenberg in [4], where he shows that a sentence is distributive in p over unions iff it is equivalent to $\langle \pi \rangle p$, for some p-free μ-program π.

Theorem 2. *A sentence φ is continuous in p iff it is equivalent to a sentence in $CF(p)$.*

Proof. By Lemma 1, we only need to prove the implication from left to right. Let φ be a sentence continuous in p. We need to find a formula χ in $CF(p)$ that is equivalent to φ.

The proof consists in constructing a finite set $\Pi \subseteq CF(p)$ such that

$$\varphi \equiv \bigvee\{\psi : \psi \in \Pi \text{ and } \psi \models \varphi\}. \tag{1}$$

Indeed, if there is such a set Π, we can define χ as the formula $\bigvee\{\psi : \psi \in \Pi \text{ and } \psi \models \varphi\}$. Clearly, χ belongs to $CF(p)$ and is equivalent to ψ.

We define Π as the set of sentences in $CF(p)$, which correspond to μ-automata with at most k states, where k is a natural number that we will define later and which depends on φ. In order to define k, we introduce the following notation. First, let $\mathbb{A} = (Q, q_0, \delta, \Omega)$ be a μ-automaton corresponding to φ. For $q \in Q$, let φ_q denote the sentence corresponding to the automaton we get from $\mathbb{A}$ by changing the initial state from q_0 to q.

Next we denote by Sort0 be the set of sentences of the form

$$\bigwedge\{p' : p' \in Prop\backslash\{p\}, p' \in \sigma\} \wedge \bigwedge\{\neg p' : p' \in Prop\backslash\{p\}, p' \notin \sigma\},$$

where σ is a subset of $Prop\backslash\{p\}$. For a point s in a model, there is a unique formula in Sort0 true at s. This formula gives us exactly the set of proposition letters in $Prop\backslash\{p\}$ which are true at s. Sort1 is the set of all sentences of the form

$$\bigwedge\{\varphi_q[\bot/p] : q \in S\} \wedge \bigwedge\{\neg\varphi_q[\bot/p] : q \notin S\},$$

where S is a subset of Q. Finally Sort2 contains all sentences of the form $\chi \wedge \nabla\Psi$, where $\chi \in$ Sort0 and Ψ is a subset of Sort1. As for the formulas in Sort0, it is easy to see that given a model $\mathcal{M}$ and a point s in $\mathcal{M}$, there is exactly one sentence in Sort1 and one sentence in Sort2 which are true at s. Remark finally that Sort0, Sort1 and Sort2 are sets of sentences which do not contain p.

Now we can define Π as the set of sentences in $CF(p)$, which correspond to μ-automata with at most $|\text{Sort2}| \cdot 2^{|Q|+1}$ states. Since there are only finitely many such automata modulo equivalence, Π is finite (up to equivalence). It is also immediate that Π is a subset of $CF(p)$. Thus it remains to show that equivalence (1) holds.

From right to left, equivalence (1) is obvious. For the direction from left to right, suppose that $\mathcal{M} = (M, R, V)$ is a model such that $\mathcal{M}, s \Vdash \varphi$, for some point s. We need to find a sentence $\psi \in \Pi$ satisfying $\psi \models \varphi$ and such that $\mathcal{M}, s \Vdash \psi$. Equivalently, we can construct an automaton $\mathbb{A}'$ corresponding to a formula $\psi \in \Pi$ such that $\psi \models \varphi$ and $\mathcal{M}, s \Vdash \psi$. That is, we can construct an automaton $\mathbb{A}'$ with at most $|\text{Sort2}| \cdot 2^{|Q|+1}$ states, corresponding to a sentence in $CF(p)$, such that $\mathbb{A}'$ accepts $(\mathcal{M}, s)$ and satisfying $\mathbb{A}' \Vdash \mathbb{A}$ (that is, for all models $\mathcal{M}'$ and all $s' \in \mathcal{M}'$, if $\mathbb{A}'$ accepts $(\mathcal{M}', s')$, then $\mathbb{A}$ accepts $(\mathcal{M}', s')$).

By Proposition 1, we may assume that $\mathcal{M}$ is a tree with root s and that $\mathcal{M}$ is ω-expanded. Since φ is continuous, there is a finite subset F of $V(p)$ such that $\mathcal{M}[p := F], s \Vdash \varphi$. Let T be the minimal downward closed set that contains F. Using T, we define the automaton $\mathbb{A}'$. Roughly, the idea is to define the set of states of $\mathbb{A}'$ as the set T together with an extra point $a_\top$. However, we need to make sure that the set of states of $\mathbb{A}'$ contains at most $|\mathrm{Sort2}| \cdot 2^{|Q|+1}$ elements. There is of course no guarantee that $T \cup \{a_\top\}$ satisfies this condition. The solution is the following. We define for every point in T its representation, which encodes the information we might need about the point. Then we can identify the points having the same representation in order to "reduce" the cardinality of T.

Before defining the automaton $\mathbb{A}'$, we introduce some notation. Given a point t in $\mathcal{M}[p := F]$, there is a unique sentence in Sort2 that is true at t. We denote it by $s2(t)$. Next if t belongs to F, we define the color $col(t)$ of t as 1 and otherwise, the color of t is 0. We let $Q(t)$ be the set $\{q \in Q : \mathcal{M}[p := F], t \Vdash \varphi_q\}$. Finally, we define the representation map $r : M \to (\mathrm{Sort2} \times Q \times \{0,1\}) \cup \{a_\top\}$ by

$$r(t) = \begin{cases} (s2(t), Q(t), col(t)) & \text{if } t \in T, \\ a_\top & \text{otherwise.} \end{cases}$$

The automaton $\mathbb{A}' = (Q', q'_0, \delta', \Omega')$ is a μ-automaton over the alphabet $\mathrm{Sort2} \times \{0,1\}$. Its set of states Q' is given by

$$Q' = \{r(t) : t \in T\} \cup \{a_\top\},$$

and its initial state q'_0 is $r(s)$. Next for all $(\sigma, i) \in \mathrm{Sort2} \times \{0,1\}$, the set $\delta'(q', (\sigma, i))$ is defined by

$$\delta'(q', (\sigma, i)) = \begin{cases} \big\{r[R(u)] : u \in T \text{ and } r(u) = r(t)\big\} & \\ & \text{if } q' = r_t,\ \sigma = s2(t) \text{ and } i = col(t), \\ \{\{a_\top\}, \emptyset\} & \text{if } q' = a_\top, \\ \emptyset & \text{otherwise.} \end{cases}$$

Intuitively, when the automaton is in the state $q' = r(t)$ and it reads the label $(s2(t), col(t))$, the Duplicator has to pick a successor u of t that is in T and this induces a description in $\delta'(r(t), (s2(t), col(t)))$. As soon as the automaton reaches the state $a_\top$, either the match is finite or the automaton stays in the state $a_\top$. In all other cases, the Duplicator gets stuck.

Finally, the map Ω' is such that $\Omega'(a_\top) = 0$ and $\Omega'(q') = 1$, for all $q' \neq a_\top$. In other words, the only way the Duplicator can win an infinite match is to reach the state $a_\top$ and to stay there.

Remark that a model $\mathcal{M}' = (M', R', V')$ can be seen as a frame (M', R') with a labeling $L' : M' \to \mathrm{Sort2} \times \{0,1\}$ defined by $L'(t') = (s2(t'), 1)$ if p is true at t' and $L'(t') = (s2(t'), 0)$ otherwise. Thus, the automaton $\mathbb{A}'$ can operate on models.

In order to extract the formula ψ from this automaton, it will be convenient to think of the alphabet of $\mathbb{A}'$ not being the set $\mathrm{Sort2} \times \{0,1\}$ but the set

$\mathcal{P}(\text{Sort2} \cup \{p\})$. The idea is to see a pair $(\sigma, i) \in \text{Sort2} \times \{0,1\}$ as the set $\{\sigma\}$ if $i = 0$ or as the set $\{\sigma, p\}$ if $i = 1$. More precisely, if $\rho \subseteq \text{Sort2} \cup \{p\}$, the transition map would associate to the pair (q', ρ) the set $\delta'(q', (\sigma, 0))$ if $\rho = \{\sigma\}$ for some $\sigma \in \text{Sort2}$, the set $\delta'(q', (\sigma, 1))$ if $\rho = \{\sigma, p\}$ for some $\sigma \in \text{Sort2}$ and the empty set otherwise.

Now if we think to the formulas of Sort2 as proposition letters, it follows from Theorem 1 that $\mathbb{A}'$ is equivalent to a sentence ψ whose proposition letters belong to $\text{Sort2} \cup \{p\}$. Such a formula ψ is also a sentence with proposition letters in $Prop$, in an obvious way. To finish the proof, we need to show that ψ is equivalent to a sentence which is in Π, ψ is true at s and $\psi \models \varphi$.

Claim 1. ψ is equivalent to a sentence in Π.

Proof. The intuition is the following. In order to win an $\mathbb{A}'$-match, the Duplicator has to reach the state $a_\top$ and then, the match is basically over. It seems natural that such a property can be expressed using only least fixpoints (and no greatest fixpoint).

Next we also need to make sure that in a formula corresponding to $\mathbb{A}'$, neither p nor any variable is in the scope of the operator $\Box$. This is guaranteed by the presence of the state $a_\top$ in any non-empty description that the Duplicator might pick. Very informally, each description corresponds to a subformula (of the sentence corresponding to the automaton) which starts with the operator ∇. Using the fact that $a_\top$ belongs to any of these descriptions (except the empty one) and corresponds to the sentence $\top$, we can show that the ∇ operator can be replaced by the modal operator $\Diamond$.

Formally the proof is the following. First observe that $\mathbb{A}'$ has at most $|\text{Sort2}| \cdot 2^{|Q|+1}$ states. Thus in order to show that ψ is equivalent to a formula in Π, it is sufficient to show that ψ is equivalent to a sentence in $CF(p)$.

For $q' \in Q'$ and $S' \subseteq Q'$, we define the translation $tr(S', q')$ of q' with respect to S'. The translation $tr(S', q')$ is a formula in the language whose set of proposition letters is $Prop$ and whose set of variables is $Var \cup Q'$. For those q' that are equal to $r(t) = (s2(t), Q(t), col(t))$ and $S' \subseteq Q'$, we have

$$tr(S', q') := s2(t) \wedge col(t).p \wedge \bigvee \left\{ \bigwedge \left\{ \Diamond \mu q''.tr\,(S' \backslash \{q''\}, s') : q'' \in r[R(u)] \text{ and } q'' \in S' \right\} \wedge \bigwedge \left\{ \Diamond q'' : q'' \in r[R(u)] \text{ and } q'' \notin S' \right\} : u \in T \text{ and } r(u) = q' \right\}$$

where $col(t).p$ is p if $col(t) = 1$ and $\top$ if $col(t) = 0$. By convention, $\bigwedge \emptyset = \top$. For all $S' \subseteq Q'$, we define $tr(S', a_\top)$ by $\top$.

It is routine to show that $tr(S', q')$ is a well-defined sentence with proposition letters in $Prop \cup (Q' \backslash S')$ and that it belongs to $CF(\{p\} \cup (Q' \backslash S'))$. The proofs are by induction on the cardinality of S'. In particular, $tr(Q', q'_0)$ belongs to $CF(p)$. Therefore, in order to prove the claim, it is enough to show that ψ is equivalent to $tr(Q', q'_0)$. The proof is in the appendix.

Claim 2. $\mathcal{M}, s \Vdash \psi$.

Proof. The proof is rather straightforward. Details are given in the appendix.

Claim 3. $\psi \models \varphi$.

Proof. Suppose $\mathcal{M}' = (M', R', V')$ is a model such that $\mathcal{M}', s' \Vdash \psi$, for some point s'. That is, the Duplicator has a winning strategy in the $\mathbb{A}'$-game in $\mathcal{M}'$ with starting position (s', q'_0). We have to show that $\mathcal{M}', s' \Vdash \varphi$. That is, we need to find a winning strategy for the Duplicator in the $\mathbb{A}$-game in $\mathcal{M}'$ with starting position (s', q'_0).

We say that a point t' is marked with a state q' if there is an $\mathbb{A}'$-match during which the Duplicator plays according to his winning strategy and the point t' is marked with q'. Let T' be the set of points marked with a state $q' \neq a_\top$. When we define the strategy for the Duplicator in the $\mathbb{A}$-game, the idea is roughly to make sure that if $t' \in T'$ and $q'(t') = r(t)$, then positions of the form (t', q) are played only if $q \in Q(t)$. Details are given in the appendix.

As a corollary of this last proof, we obtain that it is decidable whether a formula is continuous in p.

Theorem 3. *It is decidable whether a formula is continuous in p.*

Proof. Fix a proposition letter p. Let Π be the set of sentences in $CF(p)$ which correspond to μ-automata with at most $|\mathrm{Sort2}| \cdot 2^{|Q|+1}$ states. Now there are only finitely many such automata (modulo equivalence). There is also an effective translation from μ-automata to μ-sentences. Finally it is easy to verify whether a formula is in $CF(p)$. Therefore, we can compute Π.

It follows from the proof of Theorem 2 that a sentence φ is continuous in p iff $\varphi \equiv \{\psi : \psi \in \Pi \text{ and } \psi \models \varphi\}$. That is, φ is continuous in p iff there exists a subset Ψ of Π such that $\varphi \equiv \bigvee \Psi$. Therefore, in order to decide if φ is continuous in p, we can compute all the subsets Ψ of Π and check whether φ is equivalent to a disjunct $\bigvee \Psi$. Since the μ-calculus if finitely axiomatizable and has the finite model property, it is decidable whether φ is equivalent to a disjunct $\bigvee \Psi$ and this completes the proof.

Looking at the decision procedure presented in the proof of Theorem 3, we can see that the complexity is at most 4EXPTIME. That is, it involves four interlocked checking procedures, each of them being of complexity at most EXPTIME. This result is not very satisfying and we are looking for a better algorithm.

Finally, we mention that a similar syntactic characterization can be obtained in the case of basic modal logic. More precisely, a basic modal formula is continuous in p iff it belongs to the modal fragment $CF_m(p)$ of $CF(p)$. We give a formal definition of $CF_m(p)$ and a sketch of the proof in the appendix.

Definition 12. *Let P be a subset of Prop. $CF_m(P)$ is defined by induction in the following way:*

$$\varphi \ ::= \ \top \mid p \mid \psi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi,$$

where p belongs to P and no proposition letters of ψ is in P.

Corollary 1. *A basic modal formula is continuous in p iff it belongs to* $CF_m(p)$.

5 Conclusion and Further Work

We defined the continuous fragment of the μ-calculus and showed how it relates to Scott continuity. We also started to investigate the relation between continuity and constructivity. Finally, we gave a syntactic characterization of the continuous formulas and we proved that it decidable whether a formula is continuous.

This work can be continued in various directions. To start with, it would be interesting to clarify the link between continuity and constructivity. In particular, we could try to answer the following question: given a constructive formula φ, can we find a continuous formula ψ satisfying $\mu p.\varphi \equiv \mu p.\psi$?

Next we observe that in the proof of Theorem 2, the construction of the automaton $\mathbb{A}'$ depends on the model $\mathcal{M}$ and the point s at which φ is true. Is it possible to construct an automaton $\mathbb{A}'$ by directly transforming the automaton $\mathbb{A}$ that is equivalent to φ? Such a construction might help us to find a better lower complexity bound for the decision procedure (for the membership of a formula in the continuous fragment).

We believe that it might be interesting to generalize our approach. As mentioned earlier, similar results to our characterization have been obtained by Giovanna D'Agostino and Marco Hollenberg in [3]. Is there any general pattern that can be found in all these proofs?

We could also extend this syntactic characterization to other settings. For example, we can try to get a similar result if we restrict our attention to the class of finitely branching models.

Finally, we would like to mention that in [10], Daisuke Ikegami and Johan van Benthem proved that the μ-calculus is closed under taking product update. Using their method together with our syntactic characterization, it is possible to show that the set of continuous formulas is closed under taking product update.

Acknowledgments. I am grateful to Johan van Benthem for proposing this work, interesting comments and suggestions for the introduction. Many thanks also to my supervisor Yde Venema for his valuable help and support. Finally special thanks to Balder ten Cate for comments and inspiring emails.

References

1. Abramsky, S., Jung, A.: Domain Theory. In: Abramsky, S., Gabbay Dov, M., Maibaum, T.S.M. (eds.) Handbook for Logic in Computer Science. Clarendon Press, Oxford (1994)
2. Chang, C., Keisler, H.: Model Theory. North Holland, Amsterdam (1973)
3. D'Agostino, G., Hollenberg, M.: Logical Questions Concerning The mu-Calculus: Interpolation, Lyndon and Los-Tarski. Journal of Symbolic Logic, 310–332 (2000)
4. Hollenberg, M.: Logic and Bisimulation. PhD thesis, Utrecht University, Zeno Institute of Philosophy (1998)

5. van Benthem, J.: Exploring Logical Dynamics. CSLI Publications/Cambridge University Press (1996)
6. Janin, D., Walukiewicz, I.: Automata for the modal μ-calculus and related results. In: Hájek, P., Wiedermann, J. (eds.) MFCS 1995. LNCS, vol. 969, pp. 552–562. Springer, Heidelberg (1995)
7. Gierz, G., Hofman, K., Keimel, K., Lawson, J., Mislove, M., Scott, D.: A Compendium of Continuous Lattices. Springer, Heidelberg (1980)
8. Otto, M.: Eliminating Recursion in the mu-Calculus. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563. Springer, Heidelberg (1999)
9. van Benthem, J.: Modal Frame Correspondences and Fixed-Points. Studia Logica, 133–155 (2006)
10. Ikegami, D., van Benthem, J.: Modal Fixed-Point Logic and Changing Models. ILLC Preprint PP-2008-19 (2008)

On the Relations between the Syntactic Theories of $\lambda\mu$-Calculi

Alexis Saurin

INRIA-Futurs & École polytechnique (LIX)
saurin@lix.polytechnique.fr

Abstract. Since Parigot's seminal article on "an algorithmic interpretation of classical natural deduction" [15], $\lambda\mu$-calculus has been extensively studied both as a typed and an untyped language. Among the studies about the call-by-name lambda-mu-calculus authors used different presentations of the calculus that were usually considered as equivalent from the computational point of view. In particular, most of the papers use one of three variants of the calculus initially introduced by Parigot: *(i)* Parigot's syntax, *(ii)* an extended calculus that satisfies Böhm theorem and *(iii)* a second variant by de Groote he considered when designing an abstract machine for $\lambda\mu$-calculus that contains one more reduction rule.

In a previous work [20] we showed that contrarily to Parigot's calculus that does not enjoy separation property as shown by David and Py [3], de Groote's initial calculus, that we refer to as $\Lambda\mu$-calculus, does enjoy the separation property. This evidence the fact that the calculi are really different and suggest that the relationships between the $\lambda\mu$-calculi should be made clear. This is the purpose of the present work.

We first introduce four variants of call-by-name $\lambda\mu$-calculus, establish some results about reductions in $\Lambda\mu$-calculus and then investigate the relationships between the $\lambda\mu$-calculi. We finally introduce a type system for $\Lambda\mu$-calculus and prove subject reduction and strong normalization.

Keywords: Classical λ-calculi, $\lambda\mu$-calculi, Streams, Confluence, Type Systems, Strong Normalization.

1 Introduction

Curry-Howard in Classical Logic. Curry-Howard correspondence was first designed as a correspondence between intuitionistic natural deduction and simply typed λ-calculus. Extending the correspondence to classical logic resulted in strong connections with control operators in functional programming languages as first noticed by Griffin [8]. In particular, $\lambda\mu$-calculus [15] was introduced by Michel Parigot as an extension of λ-calculus isomorphic to an alternative presentation of classical natural deduction [14] in which one can encode usual control operators and in particular the `call/cc` operator.

Variants of $\lambda\mu$-calculi. Several variants of Parigot's $\lambda\mu$-calculus are considered in the literature. Their relationship is usually not made clear: semantics of

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 154–168, 2008.
© Springer-Verlag Berlin Heidelberg 2008

those languages can differ as well as their syntactical properties. For instance, $\Lambda\mu$-calculus satisfies Böhm theorem while $\lambda\mu\eta$ does not. Moreover, it is sometimes unclear what properties are true in which variant of $\lambda\mu$. One of the purposes of this paper is to study and compare different variants of call-by-name $\lambda\mu$-calculus in order to stress their differences as well as their relationships and to understand how it may affect the expressiveness and semantics of those calculi.

$\lambda\mu$-calculus and Separation. $\lambda\mu$-calculus became one of the most standard ways to study classical lambda-calculi. As a result, the calculus has been more and more studied and more fundamental questions arose. Among them, knowing whether separation property (also called Böhm theorem [2,11]) holds for $\lambda\mu$-calculus was one of the important questions in the study of $\lambda\mu$-calculus, since separation is a fundamental property relating syntax and semantics in a very delicate way. In 2001, David & Py proved that separation fails in $\lambda\mu$-calculus (more precisely the calculus we shall call $\lambda\mu\eta$) by exhibiting a counter-example to separation [3]. In a previous work, we introduced an extension to $\lambda\mu$-calculus, $\Lambda\mu$-calculus, for which we proved that separation holds [20]. We will further develop the meta-theory of $\Lambda\mu$-calculus in this paper by proving its confluence and by characterizing more precisely the canonical normal forms of $\Lambda\mu$-calculus which are the "values" that we shall separate. Moreover, we introduce a type system for $\Lambda\mu$-calculus which has the property of allowing more terms to be typed while keeping subject reduction and strong normalization.

We regard the type system introduced in this paper, $\Lambda_{\mathcal{S}}$, as a first step towards the study a typed separation theorem [22,11,6,7] for $\Lambda\mu$-calculus that we leave for future work.

Structure of the Paper. First we present the four variants of CBN $\lambda\mu$-calculus that we shall study in this paper. Secondly we prove confluence of $\Lambda\mu$-calculus in Section 3 and compare the equational theories of the $\lambda\mu$-calculi in Section 4. Finally we study in Section 5 a new type system for $\Lambda\mu$-calculus, $\Lambda_{\mathcal{S}}$.

2 Four $\lambda\mu$-Calculi

In this section, we present the four variants of call-by-name $\lambda\mu$-calculus that we shall study in this paper. We shall use in this paper an alternative notation for $\lambda\mu$-terms that we introduced and justified in a previous work [20], writing $(t)\alpha$ instead of $[\alpha]t$.

2.1 Parigot's Original Calculus: $\lambda\mu$

In 1992, Michel Parigot introduced $\lambda\mu$, an extension of λ-calculus providing "an algorithmic interpretation of classical natural deduction" [15] by allowing for a proof-program correspondence *à la* Curry-Howard [10] between $\lambda\mu$-calculus and classical natural deduction.

$$\frac{}{\Gamma, x:\mathcal{T} \vdash_{\lambda\mu} x \ : \mathcal{T} | \Delta}\ Var_{\mathcal{T}} \qquad \frac{\Gamma \vdash_{\lambda\mu} t \ : B | \Delta, \alpha : A}{\Gamma \vdash_{\lambda\mu} \mu\alpha^{A}.(t)\beta \ : A | (\Delta, \beta : B) \setminus \alpha : A}\ \mu$$

$$\frac{\Gamma, x:\mathcal{T} \vdash_{\lambda\mu} t \ : \mathcal{T}' | \Delta}{\Gamma \vdash_{\lambda\mu} \lambda x^{\mathcal{T}}.t \ : \mathcal{T} \to \mathcal{T}' | \Delta}\ \lambda\text{-Abs} \qquad \frac{\Gamma \vdash_{\lambda\mu} t \ : \mathcal{T} \to \mathcal{T}' | \Delta \qquad \Gamma \vdash_{\lambda\mu} u \ : \mathcal{T} | \Delta}{\Gamma \vdash_{\lambda\mu} (t)u \ : \mathcal{T}' | \Delta}\ \lambda\text{-App}$$

Fig. 1. Type system for $\lambda\mu$-calculus ($\mathcal{T}, A, B$ are the usual simple types with $\to$)

Definition 1 ($\Sigma_{\lambda\mu}$). *The terms of Parigot's $\lambda\mu$-calculus are defined by the following syntax:*

$$t ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.(t)\beta$$

with $x \in \mathcal{V}$ and $\alpha, \beta \in \mathcal{V}_c$, $\mathcal{V}$ and $\mathcal{V}_c$ being two disjoint infinite sets of variables. The set of $\lambda\mu$-terms is noted $\Sigma_{\lambda\mu}$. In $\mu\alpha.(t)\beta$, variable β is in the scope of $\mu\alpha$. Terms of the form $(t)\alpha$ are not elements of $\Sigma_{\lambda\mu}$, but what is usually called named terms and they are generically written n.

Definition 2 ($\lambda\mu$-calculus reduction). *$\lambda\mu$-calculus reduction, written $\longrightarrow_{\lambda\mu}$, is induced by the following four reduction rules:*

$$\begin{array}{lll} (\lambda x.t)u & \longrightarrow_{\beta} & t\,\{u/x\} \\ (\mu\alpha.n)u & \longrightarrow_{\mu} & \mu\alpha.n\,\{(v)u\alpha/(v)\alpha\} \\ (\mu\alpha.n)\beta & \longrightarrow_{\rho} & n\,\{\beta/\alpha\} \\ \mu\alpha.(t)\alpha & \longrightarrow_{\theta} & t \qquad\qquad \text{if } \alpha \notin FV(t) \end{array}$$

$n\,\{(v)u\alpha/(v)\alpha\}$ substitutes without capture the subterms $(v)\alpha$ in n by $(v)u\alpha$.

$\lambda\mu$-calculus satisfies lots of good properties of λ-calculus: confluence of the untyped calculus [15,19,3], subject reduction [15] and strong normalization [17,18] of the typed calculus (see figure 1) being the most central ones.

2.2 $\lambda\mu$-Calculus with Extensionality: Py's $\lambda\mu\eta$

Whereas the critical pair between β and η does not prevent Church-Rosser to hold, the critical pair μ/η does not converge:

$$\mu\alpha.n \longleftarrow_{\eta} \lambda x.(\mu\alpha.n)x \longrightarrow_{\mu} \lambda x.\mu\alpha.n\,\{(v)x\alpha/(v)\alpha\}$$

This is probably why η is not considered in Parigot's original presentation. In his PhD, Walter Py studied confluence properties of $\lambda\mu$-calculus and was interested in Böhm theorem for $\lambda\mu$-calculus. In studying separation, it is needed to have extensionality so that Py added η to $\lambda\mu$-calculus and restored separation with an additional rule, ν, that makes the above diagram to converge.

$\lambda\mu\eta$-calculus is obtained by adding the following rules to $\lambda\mu$-calculus:

$$\begin{array}{lll} \lambda x.(t)x & \longrightarrow_{\eta} & t \qquad\qquad\qquad\qquad \text{if } x \notin FV(t) \\ \mu\alpha.n & \longrightarrow_{\nu} & \lambda x.\mu\alpha.n\,\{(v)x\alpha/(v)\alpha\} \qquad \text{if } x \notin FV(n) \end{array}$$

David & Py proved that separation fails in $\lambda\mu\eta$ [19,3] by exhibiting a counter-example to separation[1]: $w_0, w_1 \in \Sigma_{\lambda\mu}$ that are solvable, not equivalent for the equivalence relation induced by $\lambda\mu\eta$-reduction rules but that no context can distinguish. In a previous work [20], we defined $\Lambda\mu$, an extension to $\lambda\mu\eta$ for which we proved Böhm theorem.

2.3 A $\lambda\mu$-Calculus Satisfying Böhm Theorem: $\Lambda\mu$-Calculus

Definition 3 ($\Sigma_{\Lambda\mu}$). *$\Lambda\mu$-terms are defined by the following syntax:*

$$t ::= \ x \mid \lambda x.t \mid (t)u \mid \mu\alpha.t \mid (t)\alpha$$

where x ranges over a set $\mathcal{V}_t$ of term variables and α ranges over a set $\mathcal{V}_s$ of stream variables. $\mathcal{V}_t$ and $\mathcal{V}_s$ are disjoint.

It is clear, since $\alpha \notin \Sigma_{\Lambda\mu}$ that notation $(t)\alpha$ is not ambiguous with notation $(t)u$. Notice that $\Sigma_{\lambda\mu} \subsetneq \Sigma_{\Lambda\mu}$ and that named terms of definition 1 are elements of $\Sigma_{\Lambda\mu}$. Moreover, terms such as $\mu\alpha.\mu\beta.t$ or $\lambda x.(t)\alpha y$ are in $\Sigma_{\Lambda\mu}$.

Definition 4 ($\Lambda\mu$-calculus reduction). *$\Lambda\mu$-calculus reduction, which is written $\longrightarrow_{\Lambda\mu}$, is induced by the following five reduction rules[2]:*

$(\lambda x.t)u$	$\longrightarrow_{\beta_T}$	$t\{u/x\}$	
$\lambda x.(t)x$	$\longrightarrow_{\eta_T}$	t	*if* $x \notin FV(t)$
$(\mu\alpha.t)\beta$	$\longrightarrow_{\beta_S}$	$t\{\beta/\alpha\}$	
$\mu\alpha.(t)\alpha$	$\longrightarrow_{\eta_S}$	t	*if* $\alpha \notin FV(t)$
$\mu\alpha.t$	$\longrightarrow_{fst}$	$\lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\}$	*if* $x \notin FV(t)$

μ-reduction can be simulated by a fst-reduction followed by a β_T-reduction: $(\mu\alpha.t)u \longrightarrow_{fst} (\lambda x.\mu\alpha.t\{(v)x\alpha/(v)\alpha\})u \longrightarrow_{\beta_T} \mu\alpha.t\{(v)u\alpha/(v)\alpha\}$.

A separation result is stated with respect to a set of observables (the $\beta\eta$-normal forms in λ-calculus). Since *fst* is an expansion rule, there are very few normal forms in $\Lambda\mu$. We thus consider a set of canonical normal forms [3,20]:

Definition 5. *A $\Lambda\mu$-term t is in* **canonical normal form (CNF)** *if it is $\beta_T\eta_T\beta_S\eta_S$-normal and it contains no subterm $(\lambda x.u)\alpha$ nor $(\mu\alpha.u)v$.*

Theorem 1 (Böhm theorem for $\Lambda\mu$-calculus [20]). *If t and t' are two non $\beta_T\eta_T\beta_S\eta_S$fst-equivalent closed canonical normal forms, there exists a context[3] $C[]$ such that $C[t] \longrightarrow^*_{\Lambda\mu} \lambda x.\lambda y.x$ and $C[t'] \longrightarrow^*_{\Lambda\mu} \lambda x.\lambda y.y$.*

Remark 1. In [4], de Groote introduced an extension to $\lambda\mu$-calculus, that we shall refer to as $\lambda\mu_{dG}$ and that is close to $\Lambda\mu$-calculus except it has neither η_T nor *fst*. In [12], Ong considers a calculus built on $\Sigma_{\Lambda\mu}$ which is very close to $\Lambda\mu$ but it is presented as an equational theory. $\lambda\mu_{dG}$ can be considered as a subcalculus $\Lambda\mu$, in the following, we will not consider much this calculus.

[1] w_0, w_1 are obtained by substituting y by $0 = \lambda x, x'.x'$ and $1 = \lambda x, x'.x$ respectively in $w = \lambda x.\mu\alpha.((x)\mu\beta.(x)u_0y\alpha)u_0\alpha$ with $u_0 = \mu\delta.(\lambda z_1, z_2.z_2)\alpha$.

[2] β_T, η_T, β_S, η_S, *fst* correspond respectively to $\beta, \eta, \rho, \theta$ and ν; see [20] for details.

[3] A context that may be "stream applicative": $[]t_1^1 \dots t_{n_1}^1\alpha_1 \dots t_1^k \dots t_{n_k}^k\alpha_k$.

2.4 De Groote's Extended Syntax with ϵ-Reduction: $\lambda\mu\epsilon$

Philippe de Groote introduced an abstract machine for $\lambda\mu$-calculus [5]. In order to do so, he considered another extension to Parigot's $\lambda\mu$, also based on terms of $\Sigma_{\Lambda\mu}$: the $\lambda\mu\epsilon$-calculus. $\lambda\mu\epsilon$ has an additional rule, ϵ-reduction: $\mu\alpha.\mu\beta.t \to_\epsilon \mu\alpha.\,|t|_\beta$ where $|t|_\beta$ is the result of removing all the free occurrences of β in t. A constraint on reduction ρ is added in order not to lose confluence: ρ-reduction is now $\mu\gamma.(\mu\alpha.t)\beta \longrightarrow_\rho \mu\gamma.t\,\{\beta/\alpha\}$. Otherwise there is a critical pair on $(\mu\gamma.\mu\beta.\mu\alpha.t)\delta\zeta$ depending on whether we apply ϵ to $\mu\alpha$ or $\mu\beta$:

$$|t|_\alpha\,\{\delta/\gamma\}\,\{\zeta/\beta\} \leftarrow^2_\rho\leftarrow_\epsilon (\mu\gamma.\mu\beta.\mu\alpha.t)\delta\zeta \to_\epsilon\to^2_\rho |t|_\beta\,\{\delta/\gamma\}\,\{\zeta/\alpha\}$$

Moerover η-reduction cannot be added to $\lambda\mu\epsilon$ or confluence is lost[4]:

$$\mu\alpha.\lambda x.\mu\beta.y \leftarrow_\mu \mu\alpha.\lambda x.(\mu\beta.y)x \to_\eta \mu\alpha.\mu\beta.y \to_\epsilon \mu\alpha.y$$

Definition 6 ($\lambda\mu\epsilon$-calculus reduction). *$\lambda\mu\epsilon$-calculus reduction, which is written $\longrightarrow_{\lambda\mu\epsilon}$, is induced by the following five reduction rules:*

$$\begin{array}{lll}
(\lambda x.t)u & \longrightarrow_\beta & t\,\{u/x\} \\
(\mu\alpha.t)u & \longrightarrow_\mu & \mu\alpha.t\,\{(v)u\alpha/(v)\alpha\} \\
\mu\gamma.(\mu\alpha.t)\beta & \longrightarrow_\rho & \mu\gamma.t\,\{\beta/\alpha\} \\
\mu\alpha.(t)\alpha & \longrightarrow_\theta & t \qquad\qquad \textit{if } \alpha \notin FV(t) \\
\mu\alpha.\mu\beta.t & \longrightarrow_\epsilon & \mu\alpha.\,|t|_\beta
\end{array}$$

In the rest of this paper, we shall prove some new results about $\Lambda\mu$-calculus and prove some properties relating the four calculi introduced in this section. We already state the following:

Lemma 1. *$\lambda\mu$-calculus is stable by $\Lambda\mu$-reductions and $\lambda\mu\epsilon$-reductions:*

- *if $t \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\Lambda\mu} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu\eta} u$;*
- *if $t \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu\epsilon} u$, then $u \in \Sigma_{\lambda\mu}$ and $t \to^\star_{\lambda\mu} u$.*

3 Syntactical Results for Pure $\Lambda\mu$-Calculus

3.1 $\Lambda\mu$-Calculus Reduction System

Definition 7 ($\boldsymbol{\beta}$, $\boldsymbol{\beta}^{var}$, $\boldsymbol{\eta}$, $\boldsymbol{fst}^-$). *We consider the following subsystems of $\Lambda\mu$-reduction or $\Lambda\mu$-equivalence:*

- *$\boldsymbol{\beta}$ is the subsystem made of reductions β_T and β_S;*
- *$\boldsymbol{\eta}$ is the subsystem made of reductions η_T and η_S;*
- *$\boldsymbol{\beta}fst$ is the subsystem $\beta_T\beta_S fst$ and $\boldsymbol{\beta\eta}fst$ for the full $\Lambda\mu$-reduction system;*
- *$\boldsymbol{\beta}^{var}$ for the subsystem of $\boldsymbol{\beta}$ that reduces a $\boldsymbol{\beta}$-redex only when the argument is a variable: $(\lambda x.\mu\alpha.t)y\beta \longrightarrow^\star_{\boldsymbol{\beta}^{var}} t\,\{y/x\}\,\{\beta/\alpha\}$;*

[4] Many details on problems of confluence in $\lambda\mu\epsilon$ with η can be found in Py's thesis [19].

- *fst^- is the restriction of fst to redexes $t = \mu\alpha.t'$ which are applied to a term u or such that t' contains at least one subterm $(\lambda x.u)\alpha$;*
- *$\longrightarrow_{\Lambda\mu^-}$ is $\longrightarrow_{\boldsymbol{\beta\eta}fst^-}$;*
- *$\sim_{\Lambda\mu}$ and $\sim_{\Lambda\mu^-}$ are the equivalences associated with $\longrightarrow_{\Lambda\mu}$ and $\longrightarrow_{\Lambda\mu^-}$.*

Remark 2. $\sim_{\Lambda\mu}=\sim_{\Lambda\mu^-}$ but we do not consider the reduction system $\Lambda\mu^-$ since it is not confluent. Indeed, there are different canonical normal forms that are $\sim_{\Lambda\mu}$-equivalent (and thus $\sim_{\Lambda\mu^-}$-equivalent) but that are normal forms for $\longrightarrow_{\Lambda\mu^-}$ (as ensured by proposition 5, page 160).

3.2 Confluence of $\Lambda\mu$-Calculus

In this section, we prove confluence of $\Lambda\mu$-calculus for μ-closed terms:

Theorem 2. *$\Lambda\mu$-calculus is confluent on μ-closed terms: for any t, t', t'' μ-closed $\Lambda\mu$-terms, there exists $u \in \Sigma_{\Lambda\mu}$ such that if $t \longrightarrow^\star_{\Lambda\mu} t', t''$ then $t', t'' \longrightarrow^\star_{\Lambda\mu} u$.*

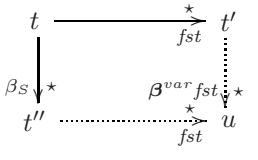
Remark 3. Notice that the hypothesis on μ-closed terms is a necessary restriction considering that the term $(\mu\beta.x)\alpha$ may reduce to x or to $(\lambda y.\mu\gamma.x)\alpha$ which cannot reduce to the same term.

In the following, the terms are always considered to be μ-closed.

Proof. Confluence of $\Lambda\mu$-calculus is proved thanks to some preliminary lemmas. Essentially, confluence of the calculus follows from the confluence of subsystem $\boldsymbol{\beta}fst$ (proposition 1), confluence of subsystem $\boldsymbol{\eta}$ (proposition 2) and the commutation of the two previous systems (proposition 3): thanks to Hindley-Rosen lemma, this ensures confluence of $\Lambda\mu$-calculus. □

Lemmas 2 and 3 are crucial steps. Lemma 2 requires μ-closed terms while commutation $\boldsymbol{\beta}^{var}fst/\boldsymbol{\beta}fst$ is the difficult case because fst does not terminate:

Lemma 2.



Lemma 3. *$\beta_T fst$ (resp. $\boldsymbol{\beta}^{var}fst$) and $\boldsymbol{\beta}fst$ commute.*

Proposition 1. *$\boldsymbol{\beta}fst$-reduction is confluent.*

Proposition 2. *$\boldsymbol{\eta}$-reduction is confluent.*

Proposition 3 is consequence of $\boldsymbol{\eta}$ commuting with β_T, β_S and fst respectively:

Proposition 3. *$\boldsymbol{\eta}$ commutes with $\boldsymbol{\beta}fst$.*

Thanks to lemma 1, confluence of $\lambda\mu\eta$-calculus is an easy consequence of confluence of $\Lambda\mu$-calculus :

	Confluence	Separation	Type System	Subject Reduction	SN
$\lambda\mu$	yes	no	CND	yes	yes
$\lambda\mu\eta$	yes (μ-closed)	no	CND + $\bot$	yes	yes
$\lambda\mu\epsilon$	yes	?	CND + $\bot$	yes	yes
$\Lambda\mu$	yes (μ-closed)	yes	CND + $\bot$ and $\Lambda_{\mathcal{S}}$	yes	yes

Fig. 2. Properties of the four $\lambda\mu$-calculi

Corollary 1. *$\lambda\mu\eta$-calculus is confluent on μ-closed terms.*

Remark 4. A proof of confluence for $\lambda\mu\eta$ can be found in Py's thesis [19] and is outlined in [3]. Our proof of confluence for $\Lambda\mu$-calculus uses some of the ideas of the proof by Py but is simpler, in particular we can avoid a lengthy development where Py uses annotations on terms. The simplification lies in particular in the use of lemma 2. As a result, the proof we outlined here is, to our knowledge, the shortest known proof of confluence for $\lambda\mu\eta$.

The following proposition is actually not a trivial consequence of confluence since even though t, u are μ-closed, the sequence of terms justifying $t =_{\Lambda\mu} u$ may involve non-closed terms for which confluence does not hold in general:

Proposition 4. *If $t, u \in \Sigma_{\Lambda\mu}$ are μ-closed and $t =_{\Lambda\mu} u$ then there exists a $v \in \Sigma_{\Lambda\mu}$ such that $t, u \longrightarrow^{\star}_{\Lambda\mu} v$.*

3.3 Characterizing Canonical Normal Forms in $\Lambda\mu$

Proposition 5. *μ-closed canonical normal forms are exactly the μ-closed $\Lambda\mu$-terms in $\beta\eta fst^{-}$-normal form.*

The following property corresponds to the property of uniqueness of the $\beta\eta$-normal form in λ-calculus.

Proposition 6. *μ-closed canonical normal forms are $\Lambda\mu$-equivalent if, and only if, they are fst-equivalent: if $t, u \in \Sigma_{\Lambda\mu}$ are CNF, then $t =_{\Lambda\mu} u \Leftrightarrow t =_{fst} u$.*

Proof. Simple consequence of proposition 4. □

4 Comparing the Four $\lambda\mu$-Calculi

The four calculi introduced in Section 2 share some properties but differ on others. In this Section, we review some of the known properties of the calculi and we investigate their relationships.

We summarize in figure 2 the properties of the four $\lambda\mu$-calculi considered in this paper. The four calculi satisfy confluence [15,19]. They all have a type system which is in correspondence with classical natural deduction proofs, with or without explicit $\bot$. In the next part of the paper we shall introduce and study $\Lambda_{\mathcal{S}}$, a new type system for $\Lambda\mu$-calculus. Strong normalization is known in the simply typed case for $\lambda\mu\eta$, $\lambda\mu\epsilon$ and $\Lambda\mu$ while Parigot provided [18] a proof of strong normalization for second-order $\lambda\mu$.

4.1 Conservative Extensions

Proposition 7 ($\Lambda\mu$ **is a conservative extension of** $\lambda\mu\eta$)**.** *If* $t, u \in \Sigma_{\lambda\mu}$ *have no free variable, then* $t =_{\Lambda\mu} u \Leftrightarrow t =_{\lambda\mu\eta} u$.

Proof. It is immediate that if $t, u \in \Sigma_{\lambda\mu}$ then $t =_{\lambda\mu\eta} u$ implies $t =_{\Lambda\mu} u$. The converse property requires confluence and proposition 4: by confluence of $\Lambda\mu$, if $t =_{\Lambda\mu} u$ there exists $v \in \Sigma_{\Lambda\mu}$ such that $t \to^\star_{\Lambda\mu} v \leftarrow^\star_{\Lambda\mu} u$. By lemma 1, if moreover $t, u \in \Sigma_{\lambda\mu}$ then there exists $v \in \Sigma_{\lambda\mu}$ such that $t \to^\star_{\lambda\mu\eta} v \leftarrow^\star_{\lambda\mu\eta} u$ and finally $t =_{\lambda\mu\eta} u$. □

Proposition 8 ($\lambda\mu\epsilon$ **is a conservative extension of** $\lambda\mu$)**.** *If* $t, u \in \Sigma_{\lambda\mu}$ *have no free variable, then* $t =_{\lambda\mu\epsilon} u \Leftrightarrow t =_{\lambda\mu} u$.

Proof. Similar to the pervious proof. □

Proposition 9 ($=_{\Lambda\mu}$ **and** $=_{\lambda\mu\epsilon}$ **are incomparable.)**. *There exist* $t, u, v \in \Sigma_{\Lambda\mu}$ *such that* $t =_{\Lambda\mu} u$ *and* $t \neq_{\lambda\mu\epsilon} u$ *and* $t =_{\lambda\mu\epsilon} v$ *and* $t \neq_{\Lambda\mu} v$.

Proof. Let $t \in \Sigma_{\Lambda\mu}$ be a term of the form $\mu\alpha.t'$ in canonical normal form with no two consecutive μ-abstractions. Let $u = \lambda x.\mu\alpha.t'\{(w)x\alpha/(w)\alpha\}$ and $v = \mu\alpha.\mu\beta.t'$. Then one has $t =_{fst} u$ and $t =_\epsilon v$ so that $t =_{\Lambda\mu} u$ and $t =_{\lambda\mu\epsilon} v$.

t is a $\lambda\mu\epsilon$-normal form and so is u (variable x cannot create a $\lambda\mu\epsilon$-redex without contradicting that t is a CNF): they are distinct normal forms of $\lambda\mu\epsilon$-calculus and thus, by confluence of $\lambda\mu\epsilon$, $t \neq_{\lambda\mu\epsilon} u$. On the other hand, t and v are in CNF so that $t =_{\Lambda\mu} v$ if and only if $t =_{fst} v$. But t and v contain a different number of μ-abstraction although *fst*-reduction preserves the number of μ in a term. As a conclusion $t \neq_{fst} v$ and finally $t \neq_{\Lambda\mu} v$. □

4.2 Separability Properties

We showed in a previous work [20] that $\Lambda\mu$-calculus satisfies the separation property: two canonical normal forms are equivalent if and only if they cannot be separated by any context. On the other hand, it is known that $\lambda\mu\eta$ and a fortiori $\lambda\mu$ do not satisfy separability [19,3].

What can we say about separability in $\lambda\mu\epsilon$? Stating separation in $\lambda\mu\epsilon$-calculus would require to consider the classes modulo $\lambda\mu\epsilon$-rules plus η. However, this makes the study very complex since $\lambda\mu\epsilon + \eta$ is not confluent and thus it is difficult to say anything about two terms not being equated by the equational theory. We shall simply consider an example of two terms being observationally equivalent whereas they are not equationally equivalent in $\lambda\mu\epsilon$: $\mu\alpha.0$ and $\mu\alpha.1$ are observationally equivalent. Indeed, they cannot be separated by any context:

$(\mu\alpha.t)u$	$\longrightarrow_\mu$	$\mu\alpha.t$	if $\alpha \notin FV(t)$.
$\mu\gamma.(\mu\alpha.t)\beta$	$\longrightarrow_\rho$	$\mu\gamma.t$	which is α-equivalent to $\mu\alpha.t$.
$\mu\beta.\mu\alpha.t$	$\longrightarrow_\epsilon$	$\mu\beta.\,\lvert t\rvert_\alpha = \mu\beta.t$	which is α-equivalent to $\mu\alpha.t$.

Actually, any term of the form $\mu\alpha.t$ with t a closed term is observationally equivalent to $\mu\alpha.0$.

5 Simply-Typed $\Lambda\mu$-Calculus

In this section, we introduce a type system for $\Lambda\mu$-calculus and prove some properties about it, namely that it can type strictly more terms than Parigot's type system, that it has subject reduction and strong normalization.

One may of course think of typing $\Lambda\mu$ using a type system for standard classical λ-calculi, similar to the one shown in figure 1, by adding rules:

$$\frac{\Gamma \vdash t \ : \bot | \Delta, \alpha : A}{\Gamma \vdash \mu\alpha^A.t \ : A | \Delta} \mu Abs \qquad \frac{\Gamma \vdash t \ : A | \Delta, \alpha : A}{\Gamma \vdash (t)\alpha \ : \bot | \Delta, \alpha : A} \mu App$$

The system uses a typing discipline *à la* Church by writing explicitly the type on the abstracted (stream or term) variables. Considering that we have an expansion rule, we restrict the *fst* rule to apply only on terms of type $A \to B$ in order the achieve subject reduction: $\mu\alpha^{A\to B}.t \ \longrightarrow_{fst} \ \lambda x^A.\mu\beta^B.t\,\{(u)x\beta/(u)\alpha\}$. Such a type system has good properties but it is unsatisfactory in the sense that many $\Lambda\mu$-terms are ruled out of the typed calculus even though they have good computational behaviours. For instance, many terms essential for separation to hold cannot be typed in this way such as $\mu\alpha.\lambda x.t$.

Making Streams First-class Citizens in the Typed Setting. We look for a type system that would reflect in types the stream construction. In particular, since μ is seen as a stream abstraction, one might think of a functional type for streams: if the term t is of type $\mathcal{T}$ when stream α is of stream type $\mathcal{S}$, then $\mu\alpha.t$ would be of the type of a stream functional from $\mathcal{S}$ to $\mathcal{T}$ (that we write $\mathcal{S} \Rightarrow \mathcal{T}$). We can thus think of the following typing rules for μ-abstracted terms:

$$\frac{\Gamma \vdash t \ : \mathcal{T} | \Delta, \alpha : \mathcal{S}}{\Gamma \vdash \mu\alpha^{\mathcal{S}}.t \ : \mathcal{S} \Rightarrow \mathcal{T} | \Delta} Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ : \mathcal{S} \Rightarrow \mathcal{T} | \Delta, \alpha : \mathcal{S}}{\Gamma \vdash (t)\alpha \ : \mathcal{T} | \Delta, \alpha : \mathcal{S}} App_{\mathcal{S}}$$

A Type Mismatch. *fst* reduction does complicate the definition of a type system for $\Lambda\mu$ that would take streams into account: whereas $\mu\alpha^{\mathcal{S}}.t$ is of a stream type, say $\mathcal{S} \Rightarrow \mathcal{T}$, the term resulting from $\mu\alpha.t$ by applying the *fst* rule once (namely $\lambda x^A.\mu\beta^{\mathcal{S}'}.t\,\{(u)x\beta/(u)\alpha\}$) should be of a standard function type $A \to B$ (more precisely $A \to (\mathcal{S}' \Rightarrow \mathcal{T}')$). Moreover, streams are streams of terms and they should be related, not only by the *fst* rule, but also by allowing to apply a term to a stream functional (for instance $(\mu\alpha.t)u$) and conversely, one might want to apply a stream to a λ-abstracted term (for instance $(\lambda x.t)\alpha$). $\Rightarrow$-types and $\to$-types should be related in some way. *fst* gives the key to this connection.

A Relation over Stream Types. *fst* was synthesized in $\Lambda\mu$-calculus (and previously in $\lambda\mu$-calculus by Py [19,3][5]) as the result of an η-expansion followed by a μ-reduction. In the typed case, the η-expansion can occur only on $\to$-type terms. This restriction adapted to $\Lambda\mu$-calculus results in the condition that $\mu\alpha.t$

[5] It had actually been already briefly discussed in 1993 by Parigot [16].

is of a stream type of the form $(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}'$. After an application of *fst*, we have term $\lambda x.\mu\beta.t\,\{(u)x\beta/(u)\alpha\}$ that should be of type $\mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$.

This corresponds to an associativity rule between constructors $\to$ and $\Rightarrow$:

$$(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}' =_{\text{assoc}^{\Rightarrow}_{\to}} \mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$$

5.1 Simply Typed Streams: $\Lambda_{\mathcal{S}}$

Pre-Types and Types. We now define the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus.

Definition 8 ($\Lambda_{\mathcal{S}}$ pre-types). *Pre-types are given by the following grammar:*

$$\begin{array}{ll} \textbf{Term pre-types:} & \mathcal{T}, A, B, \ldots ::= o_i \mid A \to B \mid \mathcal{S} \Rightarrow \mathcal{T} \\ \textbf{Stream pre-types:} & \mathcal{S}, P, Q, \ldots ::= \sigma_i \mid \mathcal{T} \to \mathcal{S} \mid \bot \end{array}$$

o_i and σ_i are respectively term and stream type variables. We keep a $\bot$ constant in the calculus, more for tradition than for real need: $\bot$ type may be regarded as a distinguished stream type variable[6]. We might want to withdraw this $\bot$ in future works, however it will be useful when studying the relationships between $\lambda\mu$-typable and $\Lambda_{\mathcal{S}}$-typable terms.

Definition 9 ($\equiv_{fst}$). *$\equiv_{fst}$ is a congruence relation over pre-types which is the symmetric, reflexive and transitive closure of relation $\succ_{fst}$ defined by*

$$(\mathcal{T} \to \mathcal{S}) \Rightarrow \mathcal{T}' \succ_{fst} \mathcal{T} \to (\mathcal{S} \Rightarrow \mathcal{T}')$$

Types of $\Lambda_{\mathcal{S}}$ are always considered up to this congruence relation:

Definition 10 ($\Lambda_{\mathcal{S}}$ types). *A $\Lambda_{\mathcal{S}}$ type is an equivalence class for $\equiv_{fst}$.*

Typed $\Lambda\mu$-calculus is considered *à la Church*, that is the syntax of typed $\Lambda\mu$-terms is as follows:

Definition 11 (Typed $\Lambda\mu$-calculus). $t ::= x \mid \lambda x^{\mathcal{T}}.t \mid (t)u \mid \mu\alpha^{\mathcal{S}}.t \mid (t)\alpha$.

We show in figure 3 the type system $\Lambda_{\mathcal{S}}$ for $\Lambda\mu$-calculus. In this type system, we deal with pre-types and an explicit conversion rule between two equivalent pre-types.

5.2 Typed Reduction Rules

The *fst* rule is an expansion rule and shall thus be treated with care if one wants subject reduction to hold in $\Lambda_{\mathcal{S}}$. In the typed case, to allow an application of the *fst* rule on t, we will require that the term has a type represented by a pre-type of shape $(\mathcal{T}_1 \to \mathcal{S}) \Rightarrow \mathcal{T}_2$. This requirement is similar to the condition on the η-expansion application in simply typed λ-calculus and is necessary to satisfy subject reduction in the presence of an expansion rule. The η-expansion is usually restricted as $t : A \to B \longrightarrow_{\eta^{\to}_{exp}} \lambda x^A.(t)x : A \to B$.

We require the same sort of constraint on *fst*-rule:

[6] It may alternatively be seen as a variable that cannot be substituted by other types.

$$\frac{}{\Gamma, x:\mathcal{T} \vdash x \ :\mathcal{T}|\Delta}\, Var_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ :\mathcal{T}|\Delta}{\Gamma \vdash t \ :\mathcal{T}'|\Delta} \equiv_{fst} \quad (\text{provided } \mathcal{T} \equiv_{fst} \mathcal{T}')$$

$$\frac{\Gamma, x:\mathcal{T} \vdash t \ :\mathcal{T}'|\Delta}{\Gamma \vdash \lambda x^{\mathcal{T}}.t \ :\mathcal{T} \to \mathcal{T}'|\Delta}\, Abs_{\mathcal{T}} \qquad \frac{\Gamma \vdash t \ :\mathcal{T} \to \mathcal{T}'|\Delta \qquad \Gamma \vdash u \ :\mathcal{T}|\Delta}{\Gamma \vdash (t)u \ :\mathcal{T}'|\Delta}\, App_{\mathcal{T}}$$

$$\frac{\Gamma \vdash t \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash \mu\alpha^{\mathcal{S}}.t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta}\, Abs_{\mathcal{S}} \qquad \frac{\Gamma \vdash t \ :\mathcal{S} \Rightarrow \mathcal{T}|\Delta, \alpha:\mathcal{S}}{\Gamma \vdash (t)\alpha \ :\mathcal{T}|\Delta, \alpha:\mathcal{S}}\, App_{\mathcal{S}}$$

Fig. 3. $\Lambda_{\mathcal{S}}$: a type system for $\Lambda\mu$-calculus

Definition 12 ($fst^{\rightarrow}$ reduction). *Reduction $fst^{\rightarrow}$ is defined as a restriction on fst-reduction on typed $\Lambda\mu$-terms as follows:*

$$\mu\alpha^{A\to\mathcal{S}}.t \longrightarrow_{fst^{\rightarrow}} \lambda x^{A}\mu\beta^{\mathcal{S}}.t\,\{(u)x\beta/(u)\alpha\}$$

One can notice that $fst^{\rightarrow}$ is an intermediate reduction between fst and fst^{-}:

Proposition 10. *Let t, u be two typed $\Lambda\mu$-terms. The following implications hold:*

$$t \longrightarrow_{fst^{-}} u \quad \Rightarrow \quad t \longrightarrow_{fst^{\rightarrow}} u \quad \Rightarrow \quad t \longrightarrow_{fst} u$$

The converse implications do not hold.

5.3 Comments about the Type System $\Lambda_{\mathcal{S}}$

Moving from $\Rightarrow$ to $\parr$. Contrarily to what the notation $\Rightarrow$ may suggest, no duality is involved with this connective. The rule $Abs_{\mathcal{S}}$ would rather suggest the $\Rightarrow$ connective to be related with the $\parr$ connective of linear logic. This is precisely what we evidence in a related work with Pagani [13]: when translating $\Lambda_{\mathcal{S}}$ into (a kind of) polarized proof nets, $\mathcal{T}_1 \to \mathcal{T}_2$ becomes $?\mathcal{T}_1^{\perp} \parr \mathcal{T}_2$ as usual while $\mathcal{S} \Rightarrow \mathcal{T}$ is translated into $\mathcal{S} \parr \mathcal{T}$.

$\equiv_{fst}$ is thus an associativity property of $\parr$ which is perfectly sound logically:

$$(?\mathcal{T}^{\perp} \parr \mathcal{S}) \parr \mathcal{T} \equiv_{fst} ?\mathcal{T}^{\perp} \parr (\mathcal{S} \parr \mathcal{T}).$$

Relation with $\lambda\mu\widehat{\mathsf{tp}}$ Type System. Herbelin et al [1,9] introduced recently a calculus $\lambda\mu\widehat{\mathsf{tp}}$ which is a $\lambda\mu$-calculus with one dynamically bound variable. This allows them to model call-by-value and call-by-name delimited continuations. They noticed that call-by-name $\lambda\mu\widehat{\mathsf{tp}}$ is very close to $\Lambda\mu$-calculus and they introduced independently a type system for this calculus which is very similar to $\Lambda_{\mathcal{S}}$. They have however a different structure for typing judgements: $\Gamma \vdash_{\Sigma} M : A; \Delta$ (Σ, annotating the $\vdash$ is a list of types).

In a current work with Herbelin and Ghilezan, we are investigating further the meta-theory $\lambda\mu\widehat{\mathsf{tp}}$ and interestingly $\lambda\mu\epsilon$ appears also to be connected to $\lambda\mu\widehat{\mathsf{tp}}$ when some critical pair naturally arising with the dynamic variable $\widehat{\mathsf{tp}}$ is oriented in the opposite direction: $\left[\widehat{\mathsf{tp}}\right]\mu\alpha.c \longrightarrow c\left\{\widehat{\mathsf{tp}}/\alpha\right\}$.

5.4 Properties of $\Lambda_{\mathcal{S}}$

$\Lambda_{\mathcal{S}}$ Types Strictly More Terms Than Parigot's $\lambda\mu$. We show that every typable term of Parigot's $\lambda\mu$-calculus can be typed in $\Lambda_{\mathcal{S}}$.

Theorem 3. *Let t a $\lambda\mu$-term in Parigot's syntax. If there exists Γ, Δ and A such that $\Gamma \vdash_{\lambda\mu} t : A|\Delta$, then there exists Γ', Δ' and A' such that $\Gamma' \vdash_{\Lambda_{\mathcal{S}}} t : A'|\Delta'$.*

Definition 13. *We consider $o_\perp$ a special term type variable and we define the following transformations on the types of Parigot's $\lambda\mu$-calculus to $\Lambda_{\mathcal{S}}$ pre-types.*

Term pre-types:	(i) $(o)^{\mathcal{T}} = (o \to \perp) \Rightarrow o_\perp$	(ii) $(A \to B)^{\mathcal{T}} = A^{\mathcal{T}} \to B^{\mathcal{T}}$
Stream pre-types:	(i) $(o)^{\mathcal{S}} = o \to \perp$	(ii) $(A \to B)^{\mathcal{S}} = A^{\mathcal{T}} \to B^{\mathcal{S}}$

Proposition 11. *Given a simple type A, then $A^{\mathcal{T}} \equiv_{fst} A^{\mathcal{S}} \Rightarrow o_\perp$.*

Remark 5. The previous theorem helps to understand more precisely what is the limitation of Parigot's $\lambda\mu$-calculus with respect to the flexibility of $\Lambda\mu$-calculus: images of $\lambda\mu$-terms need never be assigned a type of shape $\mathcal{S}_1 \Rightarrow (\mathcal{S}_2 \Rightarrow A)$.

Type Preservation. Typed $\Lambda\mu$-calculus satisfies subject reduction:

Theorem 4 (Subject Reduction). *Reduction of typed $\Lambda\mu$-terms preserves type: let $t, u \in \Lambda\mu$. If $\Gamma \vdash t : A|\Delta$ and $t \longrightarrow_{\Lambda\mu^\to} u$ then $\Gamma \vdash u : A|\Delta$.*

Strong Normalization. Finally, we prove strong normalization:

Theorem 5 (Typed $\Lambda\mu$-calculus is strongly normalizing). *Let t be a well-typed term in $\Lambda_{\mathcal{S}}$. There is no infinite reduction from t in $\Lambda\mu^\to$.*

The theorem is proved thanks to a method inspired by one of the proofs given by Parigot in [18] for proving strong normalization of simply typed $\lambda\mu$-calculus. It consists in providing a translation of typed $\Lambda\mu$-terms into simply typed λ-calculus and deducing strong normalization of typed $\Lambda\mu$-calculus after strong normalization of simply typed λ-calculus.

We first give a translation of $\Lambda_{\mathcal{S}}$ types into simple types:

Definition 14. *To each pre-type of $\Lambda_{\mathcal{S}}$, we associate a simple type as follows: We first enrich the set of type variables of the simply typed λ-calculus. To each stream-type variable σ, one considers a new simple type variable written σ as well. Moreover, we add a new variable $\sigma_\perp$.*

- $|o_i| = o_i$ *if o_i is a term-type variable.*
- $|\mathcal{T}_1 \to \mathcal{T}_2| = |\mathcal{T}_1| \to |\mathcal{T}_2|$
- $|(\mathcal{T}_1 \to \mathcal{S}) \Rightarrow \mathcal{T}_2| = |\mathcal{T}_1| \to |\mathcal{S} \Rightarrow \mathcal{T}_2|$
- $|\sigma_i \Rightarrow \mathcal{T}| = \sigma_i \to |\mathcal{T}|$
- $|\perp \Rightarrow \mathcal{T}| = \sigma_\perp \to |\mathcal{T}|$

This translation defines actually a translation of $\Lambda_{\mathcal{S}}$ types into simple types since all the pre-types of the same type are sent to the same simple type as is easily checked. We now translate typed $\Lambda\mu$-terms into λ-terms:

Definition 15. *For each stream variable α, one considers new variables of λ-calculus: $\alpha_0, \alpha_1, \ldots, \alpha_n, \ldots$ The translation is then defined as follows:*

$$\begin{aligned}
\lceil x \rceil^{\Lambda s} &= x \\
\lceil \lambda x^A.t \rceil^{\Lambda s} &= \lambda x^{|A|}.\lceil t \rceil^{\Lambda s} \\
\lceil (t)u \rceil^{\Lambda s} &= (\lceil t \rceil^{\Lambda s})\lceil u \rceil^{\Lambda s} \\
\lceil \mu\alpha^{A_1 \to \ldots A_n \to \sigma}.t \rceil^{\Lambda s} &= \lambda\alpha_1{}^{|A_1|}. \ldots \lambda\alpha_n{}^{|A_n|}.\lambda\alpha_{n+1}{}^{|\sigma|}.\lceil t \rceil^{\Lambda s} \\
\lceil (t)\alpha \rceil^{\Lambda s} &= (\lceil t \rceil^{\Lambda s})\alpha_1 \ldots \alpha_{n+1} \text{ if } \alpha \text{ is of type } A_1 \to \ldots A_n \to \sigma
\end{aligned}$$

where σ is either a stream-type variable or $\bot$.

Proposition 12. *If t is a typed $\Lambda\mu$-term, then $\lceil t \rceil^{\Lambda s}$ is a simply typed λ-term.*

Proposition 13 (Simulation). *Given two typed $\Lambda\mu$-terms t and u, the following facts hold:*

- *If $t \longrightarrow_{fst^{\to}} u$ then $\lceil t \rceil^{\Lambda s} = \lceil u \rceil^{\Lambda s}$*
- *If $t \longrightarrow_{\boldsymbol{\beta\eta}} u$ then $\lceil t \rceil^{\Lambda s} \longrightarrow^{+}_{\beta\eta} \lceil u \rceil^{\Lambda s}$*

Proposition 14 ($\boldsymbol{fst^{\to}}$ terminates). *Let t be a typed $\Lambda\mu$-term. there is no infinitely long $fst^{\to}$-derivation from t.*

Proposition 15 ($\lceil \_ \rceil^{\Lambda s}$ is reduction-length increasing). *If $t \longrightarrow^{\star}_{\Lambda\mu^{\to}} u$ with m $\boldsymbol{\beta\eta}$-reduction steps, then there exists a $\beta\eta$-reduction from $\lceil t \rceil^{\Lambda s}$ to $\lceil u \rceil^{\Lambda s}$ in at least m reduction steps.*

We can finally prove strong normalization of typed $\Lambda\mu$-calculus:

Proof. Let us suppose that there exists an infinitely long typed reduction sequence starting at a typed $\Lambda\mu$-term t: $(t_i)_{i\geq 0}$ with $t = t_0$ and $t_i \longrightarrow_{\Lambda\mu^{\to}} t_{i+1}$.

This reduction sequence contains only a finite number of $\boldsymbol{\beta\eta}$-reduction steps by proposition 15: otherwise we would obtain an infinite $\beta\eta$-reduction sequence from $\lceil t \rceil^{\Lambda s}$ in simply typed λ-calculus. Thus there is a integer n_0 such that for all $n \geq n_0$, $t_n \longrightarrow_{fst^{\to}} t_{n+1}$, and thus we would have an infinitely long $fst^{\to}$ reduction sequence which contradicts termination of $fst^{\to}$. □

6 Conclusion

The aim of this paper was two-fold: to develop the meta-theory of $\Lambda\mu$-calculus, an extension of Parigot's $\lambda\mu$-calculus that has a Böhm theorem [20] and to review and compare different versions of call-by-name $\lambda\mu$-calculus that are found in the literature. Indeed, the relationships between those calculi were seldom made clear.

Contributions of the Paper. The contributions of the paper are as follows:

- We proved confluence of $\Lambda\mu$-calculus and obtained a proof of the confluence for $\lambda\mu\eta$ which is simpler than the proof previously known from [19].
- We introduced a type system, $\Lambda_{\mathcal{S}}$, for $\Lambda\mu$-calculus that has subject reduction and strong normalization. $\Lambda_{\mathcal{S}}$ allows more terms to be typed and in particular terms that were needed to obtain Böhm theorem in [20] whereas they were not typable in the usual type system for $\lambda\mu$-calculi.
- We investigated some relationships between call-by-name $\lambda\mu$-calculi. In particular, we proved that $\Lambda\mu$-calculus is a conservative extension of $\lambda\mu\eta$ and that $\lambda\mu\epsilon$-calculus is a conservative extension of $\lambda\mu$.
- On the other hand, the equational theory of $\Lambda\mu$ and $\lambda\mu\epsilon$ cannot be compared: for any $t \in \Sigma_{\Lambda\mu}$, there exist $u, v \in \Sigma_{\Lambda\mu}$ such that $t =_{\Lambda\mu} u$ and $t \neq_{\lambda\mu\epsilon} u$ and $t =_{\lambda\mu\epsilon} v$ and $t \neq_{\Lambda\mu} v$.
- The difference between $\Lambda\mu$ and $\lambda\mu\epsilon$ is also emphasized by the fact that $\Lambda\mu$ has Böhm theorem while in $\lambda\mu\epsilon$ it is not possible to separate $\mu\alpha.0$ and $\mu\alpha.1$.

Future Works. We plan to develop this work in several directions:

- The comparison between $\Lambda\mu$ and $\lambda\mu\epsilon$ could probably be made more precise thanks to $\lambda\mu\widehat{\mathsf{tp}}$ that we are currently investigating with Herbelin and Ghilezan. This should in particular allow to understand more precisely where are the different limits to the expressiveness of the variants of $\lambda\mu$-calculus.
- The logical content of $\Lambda_{\mathcal{S}}$ is still to be made clearer: some results have been obtained in a joined work with Michele Pagani by connecting $\Lambda\mu$ to a sort of polarized proof-nets thanks to $\Lambda_{\mathcal{S}}$.
- The interpretation of $\Lambda\mu$-calculus as a stream calculus was essential in designing $\Lambda_{\mathcal{S}}$. We wish to develop this aspect by studying the relationships between $\Lambda\mu$ and infinitary λ-calculi.
- We wish to extend $\Lambda_{\mathcal{S}}$ in particular in the direction of polymorphism.
- Finally, an important motivation for providing $\Lambda\mu$ with a type system different from the original Parigot type system, was to investigate typed separation. Indeed, there is no hope to obtain a typed separation result with a classical typing of $\Lambda\mu$ so that another type system, allowing more terms to be typed, was needed.

Acknowledgments. The author wishes to thank Michele Pagani, Hugo Herbelin and Silvia Ghilezan for helpful discussions and fruitful comments on a previous version of a part of this work [21].

References

1. Ariola, Z.M., Herbelin, H., Sabry, A.: A type-theoretic foundation of delimited continuations. Higher-order symbolic computation (2007)
2. Böhm, C.: Alcune proprietà delle forme $\beta\eta$-normali nel λK-calcolo. Publicazioni dell'Istituto per le Applicazioni del Calcolo, 696 (1968)

3. David, R., Py, W.: $\lambda\mu$-calculus and Böhm's theorem. Journal of Symbolic Logic (2001)
4. de Groote, P.: On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In: Pfenning, F. (ed.) LPAR 1994. LNCS (LNAI), vol. 822. Springer, Heidelberg (1994)
5. de Groote, P.: An environment machine for the $\lambda\mu$-calculus. MSCS 8 (1998)
6. Dosen, K., Petric, Z.: The maximality of the typed lambda caluclus and of cartesian closed categories. Publication de l'Institut Mathématique 68(82), 1–19 (2000)
7. Dosen, K., Petric, Z.: The typed Böhm theorem. ENTCS 50(2) (2001); In : Proceedings of BOTH 2001
8. Griffin, T.: A formulae-as-types notion of control. In: POPL 1990 (1990)
9. Herbelin, H., Ghilezan, S.: An approach to call-by-name delimited continuations. In: POPL (January 2008)
10. Howard, W.A.: The formulae-as-type notion of construction, 1969. In: Seldin, J.P., Hindley, R. (eds.) To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism, pp. 479–490. Academic Press, New York (1980)
11. Joly, T.: Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systémes de types. PhD thesis, Université Paris VII (2000)
12. Ong, L.: A semantic view of classical proofs. In: LICS 1996 (1996)
13. Pagani, M., Saurin, A.: Stream Associative Nets and Lambda-mu-calculus. Technical Report RR-6431, INRIA (January 2008)
14. Parigot, M.: Free deduction: An analysis of "computations" in classical logic. In: Voronkov, A. (ed.) RCLP 1990 and RCLP 1991. LNCS, vol. 592, pp. 361–380. Springer, Heidelberg (1991)
15. Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624. Springer, Heidelberg (1992)
16. Parigot, M.: Classical proofs as programs. In: Mundici, D., Gottlob, G., Leitsch, A. (eds.) KGC 1993. LNCS, vol. 713, pp. 263–276. Springer, Heidelberg (1993)
17. Parigot, M.: Strong normalization for second order classical natural deduction. In: Vardi, M. (ed.) Eighth Annual Symposium on Logic in Computer Science, pp. 39–46. IEEE, Los Alamitos (June 1993)
18. Parigot, M.: Proofs of strong normalisation for second order classical natural deduction. Journal of Symbolic Logic 62(4), 1461–1479 (1997)
19. Py, W.: Confluence en $\lambda\mu$-calcul. PhD thesis, Université de Savoie (1998)
20. Saurin, A.: Separation with streams in the $\Lambda\mu$-calculus. In: Twentieth Annual Symposium on Logic in Computer Science. IEEE, Los Alamitos (2005)
21. Saurin, A.: Typing streams in the $\Lambda\mu$-calculus. In: LPAR 2007 (2007)
22. Statman, R.: λ-definable functionals and $\beta\eta$ conversion. Archiv für Mathematische Logik und Grundlagenforschung 22, 1–6 (1983)

A Constructive Semantic Approach to Cut Elimination in Type Theories with Axioms

Olivier Hermant[1,⋆] and James Lipton[2]

[1] *Univ. Complutense de Madrid*, Spain
ohermant@fdi.ucm.es
[2] *Wesleyan University*, USA
and visiting Researcher,
Univ. Politécnica de Madrid, Spain
jlipton@wesleyan.edu

Abstract. We give a fully constructive semantic proof of cut elimination for intuitionistic type theory with axioms. The problem here, as with the original Takeuti conjecture, is that the impredicativity of the formal system involved makes it impossible to define a semantics along conventional lines, in the absence, a priori, of cut, or to prove completeness by induction on subformula structure. In addition, unlike semantic proofs by Tait, Takahashi, and Andrews of variants of the Takeuti conjecture, our arguments are constructive.

Our techniques offer also an easier approach than Girard's strong normalization techniques to the problem of extending the cut-elimination result in the presence of axioms. We need only to relativize the Heyting algebras involved in a straightforward way.

1 Introduction

We give a new constructive semantic proof of cut elimination for an intuitionistic formulation of Church's Theory of Types (ICTT) with axioms. The argument extends and modifies techniques of Prawitz, Takahashi, Andrews and [4] which are non-constructive. A discussion of the constructive character of the proof, and the reasons why some older semantic proofs are not constructive can be found in Section 7. We also make use of a simple new technique to handle sets of axioms: relativization of infinite-context Heyting Algebras, as discussed below.

We recall that the central problem in extending the conventional syntactic proof of cut-elimination to certain *impredicative* higher-order logics is that one cannot induct on the natural subformula ordering that places instances $M[t/x]$ below quantified formulae such as $\exists x.M$, because it is not a well-ordering. This can be seen by taking M to be the variable x of type o and taking $t = \exists x.M \wedge A$ for any A, for example.

The problem of extending cut-elimination to higher-order logic (known as Takeuti's conjecture when it was still open) was solved by e.g. Takahashi[21],

⋆ This work has been partially supported by the Spanish projects Merit-Forms-UCM (TIN2005-09207-C03-03) and Promesas-CAM (S-0505/TIC/0407).

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 169–183, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Prawitz[18] and Andrews [1] by extending work by Tait [20] and following the blueprint given by Schütte in [19] where he observed that cut admissibility can be proved semantically by showing completeness of the cut-free fragment with respect to a weaker semantics he called semivaluations, and then showing every semivaluation gives rise to a total valuation extending it.

We generalize this approach by replacing Schütte's semivaluations by a *pair* of semantic mappings into a Heyting Algebra which give an upper and lower bound for the desired model, and show that such a pair can be defined syntactically (and constructively) using sets of contexts of cut-free proofs. The resulting model is easily relativized to extend to non-logical axioms by using a new parameter: an arbitrary set of axioms.

Cut-elimination for many impredicative formal systems (but not the ones considered here) has also been proved constructively using strong normalization techniques following Girard[8,9]. We have chosen, rather, to take the alternative approach, namely that of the Takahashi-Schütte-Andrews tradition because it seems to lend itself more readily to the addition of axioms, a central concern of this paper. Also one of the main interests of the authors in this work is to apply these techniques to formal systems in which rewriting rules are combined with sequent calculus, such as Deduction Modulo, invented by Dowek, Hardin, Kirchner and Werner [5,6]. Cut elimination for various fragments and variants of this system, studied elsewhere by Hermant and Dowek, does *not*, in general, satisfy strong normalization, and it is therefore not obviously amenable to Girard's techniques.

2 The Formal System: A Sketch

For definitions of types, terms and reduction in the intuitionistic formulation of Church's Theory of types, due originally to Miller et al. [13], we refer the reader to [2,1,4], and limit ourselves to recapitulating the rules of inference, in Fig. 1, where λ stands for $\beta\eta$ conversion, and where structural rules, such as contraction and weakening, are implicitly assumed. Types are omitted where clear from context, and we use Church's notation $(\beta\alpha)$ for the arrow type $\alpha \to \beta$ with association to the left. Fig. 1 does not include the cut rule:

$$\frac{\Gamma \vdash B \quad \Gamma, B \vdash A}{\Gamma \vdash A}\ \text{Cut}$$

When we mean a proof within the rules of Fig. 1, we use the symbol $\vdash^*$, and the unadorned $\vdash$ when we allow the cut rule. $\Gamma \vdash A$ will also abbreviate "the sequent $\Gamma \vdash A$ has a proof". In the rest of the paper, we consider a fixed language S for ICTT, i.e. for each type, a set of constants.

3 Global Models

We will make use of the notion of applicative structures, a well-known semantic framework for the simply-typed lambda calculus [7,17,14].

$$\frac{}{\Gamma \vdash \top} \qquad \frac{}{\Gamma, U \vdash U} \qquad \frac{}{\Gamma, \bot \vdash \bot}$$

$$\frac{\Gamma, B, C \vdash A}{\Gamma, B \wedge C \vdash A}\ \wedge_L \qquad \frac{\Gamma \vdash B \quad \Gamma \vdash C}{\Gamma \vdash B \wedge C}\ \wedge_R$$

$$\frac{\Gamma, B \vdash A \quad \Gamma, C \vdash A}{\Gamma, B \vee C \vdash A}\ \vee_L \qquad \frac{\Gamma \vdash B_i}{\Gamma \vdash B_1 \vee B_2}\ \vee_R$$

$$\frac{\Gamma \vdash B \quad \Gamma, C \vdash A}{\Gamma, B \supset C \vdash A}\ \supset_L \qquad \frac{\Gamma, B \vdash C}{\Gamma \vdash B \supset C}\ \supset_R$$

$$\frac{\Gamma, P[t/x] \vdash A}{\Gamma, \forall x.P \vdash A}\ \forall_L \qquad \frac{\Gamma \vdash P}{\Gamma \vdash \forall x.P}\ \forall_R *$$

$$\frac{\Gamma, P \vdash A}{\Gamma, \exists x.P \vdash A}\ \exists_L * \qquad \frac{\Gamma \vdash P[t/x]}{\Gamma \vdash \exists x.P}\ \exists_R$$

$$\frac{\Gamma' \vdash A'}{\Gamma \vdash A}\ \lambda \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash B}\ \bot_R$$

Fig. 1. Higher-order Sequent Rules

Definition 1. *A typed applicative structure* $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const} \rangle$ *consists of an indexed family* $\mathsf{D} = \{\mathsf{D}_\alpha\}$ *of sets* D_α *for each type* α*, an indexed family* App *of functions* $\mathsf{App}_{\alpha,\beta} : \mathsf{D}_{\beta\alpha} \times \mathsf{D}_\alpha \to \mathsf{D}_\beta$ *for each pair* (α, β) *of types, and an (indexed) interpretation function* $\mathsf{Const} = \{\mathsf{Const}_\alpha\}$ *taking constants of each type* α *to elements of* D_α*.*

We will abbreviate the mapping App to the infix operator $\cdot$ when types are clear from context.

So far we have only supplied a structure to interpret the underlying typed λ-calculus. Now we interpret the logic as well, by adjoining a Heyting algebra and some additional structure to handle the logical constants and predicates.

Definition 2. *A Heyting applicative structure, or HAS* $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \omega, \Omega \rangle$ *for* ICTT *is a typed applicative structure with an associated Heyting algebra* Ω *and function* ω *from* D_o *to* Ω *such that for each* f *in* $\mathsf{D}_{o\alpha}$*,* Ω *contains the parametrized meets and joins*

$$\bigwedge \{\omega(\mathsf{App}(f, d)) : d \in \mathsf{D}_\alpha\} \text{ and } \bigvee \{\omega(\mathsf{App}(f, d)) : d \in \mathsf{D}_\alpha\},$$

and the following conditions are satisfied:

$$\begin{aligned}
\omega(\mathsf{App}(\mathsf{Const}(\top_o)) &= \top_\Omega \\
\omega(\mathsf{App}(\mathsf{Const}(\bot_o)) &= \bot_\Omega \\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\wedge_{ooo}), d_1), d_2)) &= \omega(d_1) \wedge \omega(d_2) \\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\vee_{ooo}), d_1), d_2)) &= \omega(d_1) \vee \omega(d_2) \\
\omega(\mathsf{App}(\mathsf{App}(\mathsf{Const}(\supset_{ooo}), d_1), d_2)) &= \omega(d_1) \to \omega(d_2) \\
\omega(\mathsf{App}(\mathsf{Const}(\Sigma_{o(o\alpha)}), f)) &= \bigvee \{\omega(\mathsf{App}(f, d)) : d \in \mathsf{D}_\alpha\} \\
\omega(\mathsf{App}(\mathsf{Const}(\Pi_{o(o\alpha)}), f)) &= \bigwedge \{\omega(\mathsf{App}(f, d)) : d \in \mathsf{D}_\alpha\}
\end{aligned}$$

By supplying an object Ω of truth values we are able to distinguish between denotations of formulae (elements $d \in \mathsf{D}_o$) and their truth-values $\omega(d) \in \Omega$.[1]

An assignment φ is a function from the free variables of the language into D which respects types, and which allows us to give meaning to open terms.

Definition 3. *A global model for* ICTT *is a total assignment-indexed function* $\mathfrak{D} = \{\mathfrak{D}(\ )_\varphi : \varphi \text{ an assignment}\}$ *into an HAS (Heyting applicative structure)* $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \omega, \Omega \rangle$ *which takes (possibly open) terms of type* α *into* D_α *and satisfies the following environmental model conditions and* η*-conversion:*

$$\begin{aligned}
\mathfrak{D}(c)_\varphi &= \mathsf{Const}(c) && \text{for constants } c\\
\mathfrak{D}(x)_\varphi &= \varphi(x) && \text{for variables } x\\
\mathfrak{D}((MN))_\varphi &= \mathsf{App}(\mathfrak{D}(M)_\varphi, \mathfrak{D}(N)_\varphi)\\
\mathfrak{D}(\lambda x_\alpha.M_\beta)_\varphi &\quad \text{is the unique member of } \mathsf{D}_{\beta\alpha} \text{ s.t.}\\
&\quad \text{for every } d \in \mathsf{D}_\alpha\ \mathsf{App}(\mathfrak{D}(\lambda x_\alpha.M_\beta)_\varphi, d) = \mathfrak{D}(M)_{\varphi[d/x]}\\
\mathfrak{D}(M)_\varphi &= \mathfrak{D}(N)_\varphi && M \ \eta\text{-equivalent to } N
\end{aligned}$$

Given a model $\mathfrak{D}$ and an assignment φ, we say that φ satisfies B in $\mathfrak{D}$ if $\omega(\mathfrak{D}(B_o)_\varphi) = \top_\Omega$; this is abbreviated to $\mathfrak{D} \models_\varphi B_o$. We say B_o is valid in $\mathfrak{D}$ (equivalently, $\mathfrak{D} \models B_o$) if $\mathfrak{D} \models_\varphi B_o$ for every assignment φ. We abbreviate the truth-value $\omega(\mathfrak{D}(B_o)_\varphi)$ to $(B_o)^*_\varphi$. We also omit the subscript φ and parenthesis when our intentions are clear. We often use the word *model* just to refer to the mapping $(\_)^*$ from logical formulae to truth values in Ω.

3.1 Soundness of ICTT for Global Models

In the following we extend interpretations to sequents in a natural way.

Definition 4. *We define the meaning of a sequent in a model to be the truth-value in* Ω *given by:*

$$(\Gamma \vdash A)^* := (\bigwedge \Gamma \supset A)^*$$

where $\bigwedge \Gamma$ *signifies the conjunction of the elements of* Γ*.*

Note that $(\bigwedge \Gamma \supset A)^* = \top$ if and only if $\top \leq (\bigwedge \Gamma \supset A)^*$, which is equivalent to $(\bigwedge \Gamma)^* \leq (A)^*$. We will abbreviate $(\bigwedge \Gamma)^*$ to $(\Gamma)^*$, and express the validity of the indicated sequent by $(\Gamma)^* \leq (A)^*$ or, when referring to the environment, by $(\Gamma)^*_\varphi \leq (A)^*_\varphi$ henceforth.

Theorem 1 (Soundness). *If* $\Gamma \vdash A$ *is provable in* ICTT *then* $(\Gamma)^* \leq (A)^*$ *in every global model* $\mathfrak{E}$ *of* ICTT*.*

A proof can be found in [4].

[1] This allows us to assign different truth values to $p_{oo}(A_o)$ and $p_{oo}(B_o)$ even when A and B are provably equivalent and hence have the same truth value. The equivalence of the higher order formulae $p_{oo}(A_o)$ and $p_{oo}(B_o)$ holds neither in ICTT as presented here nor in the λProlog programming language, based on a sub-system of ICTT.

A straightforward proof of completeness of ICTT for global models can be given *under the assumption that cut is admissible for ICTT* along the lines of [22,4], i.e. by choosing Ω to be the Lindenbaum algebra of equivalence classes of formulae and then interpreting each formula as its own equivalence class. Just to show Ω is partially ordered, we need cut.

Since we are not assuming cut holds in ICTT we must proceed differently. We will choose the complete Heyting algebra Ω_{cfk} generated by "relativized cut-free contexts", that is to say, contexts from which formulae can be proved without using cut. A partial valuation will be defined for this cHa, yielding an interpretation that establishes completeness and the admissibility of cut.

4 From Semivaluations to Valuations

In order to apply Schütte's plan [19], we need to extend the definition of a semivaluation in our intuitionistic (and higher-order) setting and lift the notion to Heyting Algebras. We also generalize Schütte's definition in one critical way: the partial information is given in terms of lower and an upper bounds for a model, which gives us an additional degree of freedom in how we approximate the truth value of a formula.

Definition 5. *Let Ω be a Heyting algebra. A global Ω* **semivaluation** $\mathcal{V} = \langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \pi, \nu, \Omega\rangle$ *consists of a typed applicative structure $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}\rangle$ together with a pair of maps $\pi : \mathsf{D}_o \longrightarrow \Omega$ and $\nu : \mathsf{D}_o \longrightarrow \Omega$, called the lower and upper constraints of $\mathcal{V}$, satisfying $\pi(d) \leq \nu(d)$ for any $d \in \mathsf{D}_o$, as well as the following:*

$$\begin{aligned}
\pi(\top_o) &= \top_\Omega \qquad \pi(\bot_o) = \bot_\Omega \\
\pi(\mathsf{Const}(*) \cdot A \cdot B) &\leq \pi(A) *_\Omega \pi(B) \qquad \text{for } * \in \{\wedge, \vee, \supset\} \\
\pi(\mathsf{Const}(\Sigma_{o(o\alpha)}) \cdot f) &\leq \bigvee\{\pi(f \cdot d) : d \in \mathsf{D}_\alpha\} \\
\pi(\mathsf{Const}(\Pi_{o(o\alpha)}) \cdot f_{(o\alpha)}) &\leq \bigwedge\{\pi(f \cdot d) : d \in \mathsf{D}_\alpha\}
\end{aligned}$$

and

$$\begin{aligned}
\nu(\top_o) &= \top_\Omega \qquad \nu(\bot_o) = \bot_\Omega \\
\nu(\mathsf{Const}(*) \cdot A \cdot B) &\geq \nu(A) *_\Omega \nu(B) \qquad \text{for } * \in \{\wedge, \vee, \supset\} \\
\nu(\mathsf{Const}(\Sigma_{o(o\alpha)}) \cdot f) &\geq \bigvee\{\nu(f \cdot d) : d \in \mathsf{D}_\alpha\} \\
\nu(\mathsf{Const}(\Pi_{o(o\alpha)}) \cdot f_{(o\alpha)}) &\geq \bigwedge\{\nu(f \cdot d) : d \in \mathsf{D}_\alpha\}
\end{aligned}$$

and the consistency *or* separation *conditions*

$$\pi(\mathsf{Const}(\supset) \cdot B \cdot C) \wedge \nu(B) \leq \pi(C) \tag{1}$$

$$\pi(B) \rightarrow_\Omega \nu(C) \leq \nu(\mathsf{Const}(\supset) \cdot B \cdot C). \tag{2}$$

Remark 1. The reader should note that some of these requirements are superfluous. For example, the separation conditions and the first condition imply the $\supset$ requirements for both π and ν. If we think of $[\pi(A), \nu(A)]$ as a – by definition

nonempty – interval, one sees that it contains all the potential truth values of A, indeed the semantic "truth value candidates", instead of Girard's reducibility candidates. The circularity mentioned in the introduction will then be avoided by quantifying over all those candidates rather than subformulae.

The definition of environment, and global structure remain the same for semivaluations. As with Heyting applicative structures, in the presence of an environment φ, a semivaluation $\mathcal{V}$ induces an interpretation $\mathfrak{V}_\varphi$ from open terms A to the carriers D as follows:

$$\begin{array}{lll}
\mathfrak{V}(c)_\varphi & = \mathsf{Const}(c) & \text{for constants } c \\
\mathfrak{V}(x)_\varphi & = \varphi(x) & \text{for variables } x \\
\mathfrak{V}(M)_\varphi & = \mathfrak{V}(N)_\varphi & M \text{ eta-equivalent to } N \\
\mathfrak{V}((MN))_\varphi & = \mathsf{App}(\mathfrak{V}(M)_\varphi, \mathfrak{V}(N)_\varphi) & \\
\mathsf{App}(\mathfrak{V}(\lambda x_\alpha.M_\beta)_\varphi, d) & = \mathfrak{V}(M)_{\varphi[x:=d]} & \text{with } \mathfrak{V}(\lambda x_\alpha.M_\beta)_\varphi \text{ the unique such value.}
\end{array}$$

This assignment induces a *pair* of partial, or semi-truth-value assignments $[\![\_]\!]^\pi_\varphi$ and $[\![\_]\!]^\nu_\varphi$ to terms A_o of type o given by

$$\mathcal{V}[\![A]\!]^\pi_\varphi = \pi(\mathfrak{V}(A)_\varphi) \qquad \mathcal{V}[\![A]\!]^\nu_\varphi = \nu(\mathfrak{V}(A)_\varphi)$$

Theorem 2. *Given an Ω-semivaluation $\mathcal{V} = \langle \mathsf{D}, \cdot, \mathsf{Const}, \pi, \nu, \Omega\rangle$, there is a model $\mathfrak{D} = \langle \hat{\mathsf{D}}, \odot, \hat{\mathsf{C}}, \omega, \Omega\rangle$ extending $\mathcal{V}$ in the following sense: for all closed terms A_o*

$$\mathcal{V}[\![A]\!]^\pi \leq \omega(\mathfrak{D}(A)) \leq \mathcal{V}[\![A]\!]^\nu.$$

Furthermore, there is a surjective indexed map $\delta : \hat{\mathsf{D}} \longrightarrow \mathsf{D}$ such that for any $\hat{d} \in \hat{\mathsf{D}}_o$

$$\pi(\delta(\hat{d})) \leq \omega(\hat{d}) \leq \nu(\delta(\hat{d}))$$

Proof. We refer the reader to [4] for the proof.

5 Cut Elimination by Completeness

From Thm. 2, deriving a (cut-free) completeness theorem for ICTT requires a complete Heyting algebra Ω and an Ω-semivaluation. We first give the definition of Ω_{cfk}, the Heyting algebra of cut-free contexts, critically different from the one given in [4], where a tableaux-style construction is used, and extend the usual notion of context-based semantics [16,15] to the notion of infinite contexts, taken themselves as a new free *parameter*.

5.1 The Cut-Free Contexts Heyting Algebra

We first define what is a cut-free context, generalizing Okada [16,15].

Definition 6 (outer value). *Assume given a set of formulae Ξ, possibly infinite, but containing only a finite number of variables. Let A be a closed formula. We let the* outer value *of A be:*

$$[\![A]\!] = \{\Gamma \mid \Xi, \Gamma \vdash^* A\}$$

The contexts Γ considered are always finite. The provability relation $\Xi, \Gamma \vdash^* A$ is with respect to some finite *subset of Ξ, Γ.*

So, an outer value $[\![A]\!]$ is the set of contexts proving A without cut (cut-free contexts). With this, we build Ω_{cfk}. When it is relevant, we stress the dependence on the considered set of axioms Ξ by $\Omega_{\mathsf{cfk}}(\Xi)$.

Definition 7 (Ω_{cfk}). *Let Ξ be a fixed set of formulae. Let $|\Omega|$ be the least set of sets of contexts generated by $[\![A]\!]$ for any formula A, closed under arbitrary (denumerable) intersection, and ordered by inclusion. Then define meets and joins on $|\Omega|$ as follows*

- $\bigwedge$ *= arbitrary intersection, just set-theoretic intersections.*
- $\bigvee$ *= arbitrary pseudo-union, that is to say*

$$\bigvee S = \bigcap \{c \in |\Omega| : c \geq S\}$$

where $c \geq S$ means $\forall s \in S\, c \geq s$

Remark 2. From the definition, it follows that:

- $\top_\Omega$ is the set of all contexts and as well $[\![\top_o]\!]$.
- $\bot_\Omega$ is the intersection of all $[\![A]\!]$ and as well $[\![\bot_o]\!]$. In particular, $\bot_\Omega \neq \emptyset$.
- the suprema can be slightly simplified: $a \vee_\Omega b = \bigcap \{[\![A]\!] \mid a \cup b \subseteq [\![A]\!]\}$, since any $c \in \Omega$ is of the form $\bigcap_{i \in \Lambda} [\![A_i]\!]$. As well, $\bigvee S = \bigcap \{[\![A]\!] \mid [\![A]\!] \geq S\}$.

Taking $a \to b = \bigvee \{x : x \wedge a \leq b\}$, the resulting structure $\Omega = \langle |\Omega|, \bigvee, \bigwedge, \to \rangle$ (also written Ω_{cfk}, when ambiguity may arise) is a complete Heyting algebra. We now check that the $\wedge \bigvee$ distributivity law [22] holds.

We first show one direction: for each member $a = \bigcap_i [\![A_i]\!]$ of Ω, we must have $a \cap \bigvee S \leq \bigvee a \cap S$, where $a \cap S$ means $\{a \cap s : s \in S\}$. Unfolding definitions, the inclusion to prove becomes:

$$a \cap \bigcap \{[\![B]\!] : [\![B]\!] \geq S\} \subseteq \bigcap \{[\![D]\!] : [\![D]\!] \geq a \cap S\} \tag{3}$$

Let Γ be a context member of the left hand side, i.e. such that $\Xi, \Gamma \vdash^* A_i$ for any A_i and $\Xi, \Gamma \vdash^* B$ for every B such that $[\![B]\!] \geq S$. Let D be a formula such that $[\![D]\!] \geq a \cap S$. We must show $\Xi, \Gamma \vdash^* D$ to prove that 3 holds.

Let Δ be a context such that $\Delta \in s$ for some $s \in S$. Since provability in Def. 6 deals with subcontexts, we directly have $\Xi, \Delta, \Gamma \vdash^* A_i$ and by a similar reasoning $\Delta, \Gamma \in s$. By definition of D, we get $\Xi, \Delta, \Gamma \vdash^* D$. Hence $\Delta \vdash^* \Gamma \supset D$, where $\Gamma \supset D$ is a shorthand for $B_1 \supset \cdots B_n \supset C$, and $\Delta \in [\![\Gamma \supset D]\!]$. Since this is valid for any s, we have shown $[\![\Gamma \supset D]\!] \geq S$.

But then, $\Xi, \Gamma \vdash^* \Gamma \supset D$ by assumption on Γ. By Kleene's Lem. 1 below and contraction on the formulae of Γ we have $\Xi, \Gamma \vdash^* D$, which shows that Γ is a member of the right-hand-side of 3, which proves the claim.

The other direction follows, by elementary lattice theory: for any $s \in S$ it is the case that $a \cap \bigvee S \geq a \cap s$. Now take the supremum of $a \cap s$ over all $s \in S$.

To complete the proof, we need Kleene's lemma, for the $\supset_R$ rule.

Lemma 1 (Kleene). *Let $C \equiv_\lambda A \supset B$ be formulae and Γ be a context. If $\Gamma \vdash C$ then $\Gamma, A \vdash B$, and if $\Gamma \vdash^* C$ then $\Gamma, A \vdash^* B$.*

Proof. Standard (see [10]) by induction on the structure of the proof.

5.2 A Semivaluation π and ν

Now, we need to exhibit a Ω-semivaluation to be able to apply Thm. 2. For this, we need the following definition:

Definition 8 (closure). *Let S be a set of contexts, we define its closure by:*

$$cl(S) = \bigcap \{[\![A]\!] \mid S \subseteq [\![A]\!]\}$$

It is the least element of Ω containing S. We also write, for a single context Γ, $cl(\Gamma)$ to mean $cl(\{\Gamma\})$.

Remark 3. $cl(A) \subseteq d$ is equivalent to $A \in d$ for any $d \in \Omega$. Indeed, $A \in cl(A)$ and $cl(A)$ is the l.u.b. of A. $cl(S)$ can also be seen as the *set of contexts admitting cut with all the elements of S* as shown in the following lemma.

Lemma 2. *Let A be a formula. The following formulations are equivalent:*

(i) $cl(A) = \bigcap \{[\![B]\!] \mid A \in [\![B]\!]\}$
(ii) $cl(A) = \{\Gamma \mid \Xi, \Gamma \vdash^* B$ *whenever* $\Xi, A \vdash^* B\}$. *Equivalently,* $\Gamma \in cl(A)$ *iff given any proof* $\Xi, A \vdash^* B$, *a proof of* $\Xi, \Gamma \vdash^* B$ *is derivable.*
(iii) $cl(A) = \{\Gamma \mid \Xi, \Gamma \vdash^* B$ *whenever* $\Xi, \Gamma, A \vdash^* B\}$. *Equivalently,* $\Gamma \in cl(A)$ *iff given any proof* $\Xi, \Gamma, A \vdash^* B$ *a proof of* $\Xi, \Gamma \vdash^* B$ *is derivable.*
(iv) $cl(A) = \{\Gamma \mid \Xi, \Delta, \Gamma \vdash^* B$ *whenever* $\Xi, \Delta, A \vdash^* B\}$. *Equivalently,* $\Gamma \in cl(A)$ *iff given any proof* $\Xi, \Delta, A \vdash^* B$ *a proof of* $\Xi, \Delta, \Gamma \vdash^* B$ *is derivable.*

Cases (ii) – (iv) *can be summarized as follows: Γ admits (Ξ-) cuts with A, hence the terminology "Γ is A-cuttable".*

Proof. (*ii*) unfolds the definition of $[\![B]\!]$ in (*i*). (*iii*) and (*iv*) reformulate (*ii*) with equivalent – thanks to Lem. 1, $\supset_R$ and contraction rules – notions of cuts.

We shall use any of the formulations given above, depending on our need. Now we are ready to give the semivaluation we work with.

Definition 9 (the cut-free context semivaluation). *Let the typed applicative structure $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const} \rangle$ be the open term model: carriers D_α are open terms of type α in normal form, application $A \cdot B$ is $[AB]$, the normal form of AB, and we interpret constants as themselves. For any formula A, define:*

$$\pi(A) = cl(A) \quad \textit{and} \quad \nu(A) = [\![A]\!]$$

The definition just given of a pair of semantic mappings based on cut-free proofs and their contexts, and shown below to give rise to a semivaluation in the sense of Def. 5, is essential to the constructive character of our proof of cut-elimination, avoiding as it does the use of tableau style (Hintikka-set) construction of partial models, as in [1,4], and the infinite tree arguments required.

Lemma 3. $\langle \mathsf{D}, \mathsf{App}, \mathsf{Const}, \pi, \nu, \Omega_{\mathsf{cfk}} \rangle$ *is a semivaluation in the sense of Def. 5.*

Proof. We check the conditions of Def. 5, with respect to the open term model. Each case follows the same pattern: it uses the corresponding rule of inference.

- $cl(A) \subseteq [\![A]\!]$. Immediate since $\{A\} \subseteq [\![A]\!]$ and from Rem. 3.
- $cl(\top_o) = \top_\Omega$. $\top_\Omega$ is the greatest element so we focus on the reverse inclusion. Consider a proof of $\Xi, \top_o \vdash^* A$. The only rule we can use on $\top_o$ besides structural ones and conversion is the axiom. We can replace it:

$$\frac{}{\vdash^* \top_o} \top_R$$

  Hence, $\Xi \vdash^* A$ and, by weakening, $\Xi, \Gamma \vdash^* A$ for any Γ, and $\top_\Omega \subseteq cl(\top_o)$.
- $cl(\bot_o) = \bot_\Omega$. $\bot_\Omega$ is the least element and, by other cases $cl(\bot) \subseteq [\![\bot]\!] = \bot_\Omega$.
- $cl(A \wedge B) \leq cl(A) \cap cl(B)$. This amounts to showing $A \wedge B \in cl(A) \cap cl(B)$. We prove that $A \wedge B$ is A-cuttable. Consider a proof of $\Xi, A \vdash^* C$. We construct the following proof:

$$\dfrac{\dfrac{\Xi, A \vdash^* C}{\Xi, A, B \vdash^* C}\,\text{weak}}{\Xi, A \wedge B \vdash^* C}\,\wedge_L$$

  Hence, $A \wedge B \in cl(A)$. On the same way, $A \wedge B \in cl(B)$.
- $cl(A \vee B) \subseteq cl(A) \vee_\Omega cl(B)$. It suffices to show $A \vee B \in cl(A) \vee_\Omega cl(B)$. Let C be such that $cl(A) \cup cl(B) \subseteq [\![C]\!]$. $A \in [\![C]\!]$, $B \in [\![C]\!]$, and the proof:

$$\frac{\Xi, A \vdash^* C \qquad \Xi, B \vdash^* C}{\Xi, A \vee B \vdash^* C} \vee_L$$

  shows that $A \vee B \in [\![C]\!]$. This holds for any such C, hence for their meet, and $A \vee B \in cl(A) \vee_\Omega cl(B)$.
- $cl(A \supset B) \subseteq cl(A) \to cl(B)$ is a consequence of $cl(A \supset B) \wedge [\![A]\!] \subseteq cl(B)$ (proved below) as mentioned in Rem. 1.
- $cl(\Sigma.f) \subseteq \bigvee\{cl(ft) \mid t \in \mathcal{T}_\alpha\}$ (where α is the suitable type). Equivalently, $\Sigma.f \in \bigvee\{cl((ft)) \mid t \in \mathcal{T}_\alpha\}$. Let t be a variable y of type α that is fresh for f and Ξ. We prove that $\Sigma.f$ is fy-cuttable. Assume we have a proof $\Xi, fy \vdash^* C$. The proof:

$$\frac{\Xi, fy \vdash^* C}{\Xi, \Sigma.f \vdash^* C} \exists_L$$

  justifies the fy-cuttability. Hence $\Sigma.f \in cl(fy)$, and it is in the supremum.
- $cl(\Pi.f) \in \bigwedge\{cl(ft) \mid t \in \mathcal{T}_\alpha\}$. Let t be a term of type α. The proof:

$$\frac{\Xi, ft \vdash^* C}{\Xi, \Pi.f \vdash^* C} \forall_L$$

  shows that $\Pi.f$ is ft-cuttable for any t.
- $[\![\top_o]\!] = \top_\Omega$ and $[\![\bot_o]\!] = \bot_\Omega$ hold both by definition, from Rem. 2.
- $[\![A \wedge B]\!] \supseteq [\![A]\!] \wedge_\Omega [\![B]\!]$. Let Γ such that $\Xi, \Gamma \vdash^* A$ and $\Xi, \Gamma \vdash^* B$. The claim is established by the proof:

$$\frac{\Xi, \Gamma \vdash^* A \qquad \Xi, \Gamma \vdash^* B}{\Xi, \Gamma \vdash^* A \wedge B} \wedge_R$$

- $[\![A \vee B]\!] \supseteq [\![A]\!] \vee_\Omega [\![B]\!]$. We show $[\![A \vee B]\!] \supseteq [\![A]\!]$. Let $\Gamma \in [\![A]\!]$. The proof:

$$\frac{\Xi, \Gamma \vdash^* A}{\Xi, \Gamma \vdash^* A \vee B} \vee_R$$

  shows that $\Gamma \in [\![A \vee B]\!]$. Hence $[\![A \vee B]\!]$ is an upper bound for $[\![A]\!]$ and $[\![B]\!]$.
- $[\![A \supset B]\!] \supseteq [\![A]\!] \to_\Omega [\![B]\!]$ is a consequence of $cl(A) \to [\![B]\!] \subseteq [\![A \supset B]\!]$.
- $[\![\Sigma.f]\!] \supseteq \bigvee \{[\![ft]\!] \mid t \in \mathcal{T}_\alpha\}$. Let t be a term, and $\Gamma \in [\![ft]\!]$. The proof:

$$\frac{\Xi, \Gamma \vdash^* ft}{\Xi, \Gamma \vdash^* \Sigma.f} \exists_R$$

  shows that $[\![\Sigma.f]\!]$ is an upper bound for any $[\![ft]\!]$, hence for their supremum.
- $[\![\Pi.f]\!] \supseteq \bigwedge \{[\![ft]\!] \mid t \in \mathcal{T}_\alpha\}$. Let $\Gamma \in \bigwedge \{[\![ft]\!] \mid t \in \mathcal{T}_\alpha\}$. Let y be a fresh variable with respect to Γ, Ξ and f. In particular, $\Gamma \in [\![fy]\!]$. The proof:

$$\frac{\Xi, \Gamma \vdash^* fy}{\Xi, \Gamma \vdash^* \Pi.f} \forall_R$$

  shows that $\Gamma \in [\![\Pi.f]\!]$.
- $cl(B \supset C) \wedge_\Omega [\![B]\!] \subseteq cl(C)$. Let $\Gamma \in cl(B \supset C) \cap [\![B]\!]$. We must show the C-cuttability of Γ. Consider a proof of $\Xi, C \vdash^* D$. Since $\Gamma \vdash^* B$:

$$\frac{\Xi, \Gamma \vdash^* B \qquad \Xi, \Gamma, C \vdash^* D}{\Xi, \Gamma, B \supset C \vdash^* D} \supset_L$$

  By $B \supset C$-cuttability of Γ we get $\Xi, \Gamma \vdash^* D$.
- $cl(B) \to_\Omega [\![C]\!] \subseteq [\![B \supset C]\!]$. Let $\Gamma \in cl(B) \to [\![C]\!]$ and show $\Xi, \Gamma \vdash^* B \supset C$. Since $\Gamma \in cl(B) \to [\![C]\!]$, we have $cl(\Gamma) \cap cl(B) \subseteq [\![C]\!]$. Furthermore, $\Gamma \in cl(\Gamma)$ and $B \in cl(B)$, therefore Γ, B belongs to both. So $\Gamma, B \in [\![C]\!]$, and we derive the desired proof:

$$\frac{\Xi, \Gamma, B \vdash^* C}{\Xi, \Gamma \vdash^* B \supset C} \supset_R$$

5.3 Completeness and Cut Elimination of ICTT

We now have all the results needed to establish completeness.

Theorem 3 (cut-free completeness of ICTT). *Let Γ be a context and A be a formula such that for any global model $\Gamma^* \leq A^*$. Then $\Gamma \vdash A$ has a cut-free proof.*

Proof. Calling ε the empty context, we apply Thm. 2 with the Heyting algebra $\Omega_{\mathsf{cfk}}(\varepsilon)$ given in Def. 7 and the semivaluation π, ν of Def. 9. We get, from Rem. 3, by Thm. 2 and by hypothesis that:

$$\Gamma \in cl(\Gamma) \subseteq \Gamma^* \subseteq A^* \subseteq [\![A]\!]$$

Hence, $\Gamma \vdash^* A$. An alternative proof involves $\Omega_{\mathsf{cfk}}(\Gamma)$: any context is trivially Γ-cuttable, so $\varepsilon \in cl(\Gamma) = \top$. With the same inclusions as above (but the first) we get that $\Gamma, \varepsilon \vdash^* A$. The interested reader may in fact prove Thm. 3 as many different ways than elements in $\mathfrak{P}(\Gamma)$, the powerset of Γ.

As an immediate corollary, we have:

Corollary 1 (constructive cut elimination for ICTT). *Let Γ be a context and A be a formula. If $\Gamma \vdash A$ in ICTT, then it has a proof without cut.*

Proof. By soundness and cut-free completeness, both of which were proved constructively.

6 Adding Non-logical Axioms

Now, we allow a more liberal notion of proof, with *non-logical axioms*.

Definition 10. *A non-logical axiom is a closed sequent $A \vdash B$. Assuming such and axiom $A \vdash B$, an axiomatic cut is the following implicit cut rule*

$$\frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma \vdash C}$$

A proof with non-logical axioms is a proof whose leaves are either a proper axiom rule, or a non-logical axiom and allowing the use of axiomatic cuts.

In the sequel, we fix a set (potentially infinite) of axioms, and consider proof system is ICTT with those non-logical axioms.

The constraint for $A \vdash B$ to be closed is not a theoretical limitation: it suffices to quantify over the free variables. In particular, we capture axiom schemes.

The two new rules overlap, since an axiomatic cut is simulated with a non-logical axiom and two (usual) cuts. Conversely, we can simulate the non logical axiom rules, even in a cut-free setting, so we often consider only axiomatic cuts:

$$\frac{\dfrac{}{\Gamma, A \vdash A} \qquad \dfrac{}{\Gamma, B \vdash B}}{\Gamma, A \vdash B}\ \text{axiomatic cut}$$

We show in this section that we still have, by the same means, cut elimination in ICTT with non-logical axioms, but that we can not, in the general setting, eliminate axiomatic cuts. First, we need another, unsurprising, notion of model:

Definition 11 (model for axioms). *A global model for ICTT (Def. 3) is a model of the non-logical axioms $A_i \vdash B_i, i \in \Lambda$ if and only if for any i, $A_i^* \leq B_i^*$.*

In the sequel, we will only be interested in such models.

Theorem 4 (Soundness of ICTT with non-logical axioms). *If $\Gamma \vdash A$ in ICTT with non-logical axioms, then $\Gamma^* \leq A^*$ in any global model of the non-logical axioms.*

Proof. We replace axiomatic cuts by axioms and cuts. Then the proof is done by the very same induction as the one of Thm. 1. The only additional case is a non-logical axiom $A \vdash B$, trivial from the assumption on the model.

Now we work towards a proof of a cut-free completeness theorem for ICTT with non-logical axioms. Cut-free means free of cuts, but not of axiomatic cuts, which we will not be able to remove.

6.1 Completeness and Cut Elimination in Presence of Axioms

Given the non logical axioms $A_i \vdash B_i$, let Ξ be the set of all the $A_i \supset B_i$. We show that the valuation in $\Omega_{\mathsf{cfk}}(\Xi)$ given by the Ω-semivaluation of Def. 9 is a model of the non-logical axioms. So in Lem. 4, provability refers to pure ICTT.

Lemma 4. *The valuation given by Theorem 2 with $cl(\_), [\![\_]\!]$ as an $\Omega_{\mathsf{cfk}}(\Xi)$-semivaluation is a model of the non-logical axioms.*

Proof. Let $A \vdash B$ be an axiom. $A^* \subseteq [\![A]\!]$ and $cl(B) \subseteq B^*$ by Thm. 2, so we show $[\![A]\!] \subseteq cl(B)$. This is implied by the fact that Γ is B-cuttable whenever $\Xi, \Gamma \vdash^* A$. Given a proof $\Xi, \Gamma, B \vdash^* C$, the following proof shows this claim:

$$\dfrac{\dfrac{\Xi, \Gamma, B \vdash^* C \qquad \Xi, \Gamma \vdash^* A}{\Xi, \Gamma, A \supset B \vdash^* C}\supset_L}{\Xi, \Gamma \vdash^* C}\text{contraction}$$

Before we prove the completeness theorem, we have to switch from proofs of $\Xi, \Gamma \vdash^* A$ in ICTT to proofs of $\Gamma \vdash^* A$ in ICTT with axiomatic cuts.

Lemma 5. *Assume we have a proof π of the sequent $\Gamma, A \supset B \vdash C$ in* ICTT*, possibly using axiomatic cuts. We can transform it into a proof of the sequent $\Gamma \vdash C$ in* ICTT *with additional cuts on the non logical axiom $A \vdash B$. If the initial proof is free of cuts, then so is the resulting proof (save axiomatic cuts).*

Proof. We can omit (by simulating) non logical axiom rules. We track the decomposition of $A \supset B$, and replace it by an axiomatic cut rule, that is the exact premises of the $\supset_L$ rule. We assume to have a proof of the sequent $\Gamma, D_1, \ldots, D_n \vdash C$, where $D_i \equiv_\lambda A \supset B$ and prove the result by induction over the structure of π. All cases are a trivial use of induction hypothesis, save:

- an axiom rule with D_1 the active formula. We build the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\text{axiom} \qquad \dfrac{}{\Gamma, B \vdash B}\text{axiom}}{\Gamma, A \vdash B}\text{axiomatic cut}}{\Gamma \vdash A \supset B}\supset_R}{\Gamma \vdash D_1}\lambda$$

- a $\supset$-l rule on $D_i = A' \supset B'$. We have the proof:

$$\dfrac{\dfrac{\pi_1}{\Gamma, D_2, \ldots, D_n \vdash A'} \qquad \dfrac{\pi_2}{\Gamma, B', D_2, \ldots, D_n \vdash C}}{\Gamma, D_1, D_2, \ldots, D_n \vdash C}\supset_R$$

  Applying induction hypothesis to get π_1' and π_2', we form the proof:

$$\dfrac{\dfrac{\dfrac{\pi_1'}{\Gamma \vdash A'}\lambda}{\Gamma \vdash A} \qquad \dfrac{\dfrac{\pi_2'}{\Gamma, B' \vdash C}\lambda}{\Gamma, B \vdash C}}{\Gamma \vdash C}\text{axiomatic cut}$$

Observe that we do not introduce any cut save axiomatic ones.

Theorem 5 (cut-free completeness of ICTT with non-logical axioms). *Let Γ be a context and A be a formula such that $\Gamma^* \leq A^*$ for any global model of the non logical axioms. Then $\Gamma \vdash A$ has a cut-free proof.*

Proof. Considering $\Omega_{\mathsf{cfk}}(\Xi)$ in Thm. 3, we get $\Xi, \Gamma \vdash^* A$ in ICTT. Applying Lem. 5 a finite number of times (provability is always with respect to a finite subset, from Def. 6), we get a cut-free proof of $\Gamma \vdash^* A$.

As an immediate corollary, we have:

Corollary 2 (constructive cut elimination for ICTT). *Let Γ be a context and A be a formula. If $\Gamma \vdash A$ has a proof in ICTT with non logical axioms, then it has a proof without cut.*

Proof. By soundness and cut-free completeness, both of which were proved constructively.

7 On the Constructivity of the Proof of Cut Admissibility

Our proof, unlike [21,1] for the classical case or [4] for the intuitionistic case, makes no appeal to the excluded middle. The works cited (and our work as well) start directly, or indirectly from Schütte's observation [19] that cut admissibility can be proved semantically by showing completeness of the cut-free fragment with respect to semivaluations, and then showing every semivaluation gives rise to a total valuation extending it.

There are a number of pitfalls to avoid in finding a constructively valid proof based on this kind of argument, both in the way a semivaluation is produced and how one passes to a valuation.

Andrews shows [1] that any abstract consistency property gives rise to a semivaluation, but then builds one in a way that requires deciding whether or not a refutation exists of a given finite set of sentences. In particular, he needs to show (Thm. 3.5 in [1]) that *any finite set S satisfying an Abstract Consistency Property is consistent.* The proof actually establishes $\neg\neg$[Th. 3.5]. Furthermore, when showing that his cut-free proof system defines an Abstract Consistency Property (Sec. 4.10.2) he ends up proving the contrapositives of the defining properties of an ACP.

One can also exhibit a semivaluation by developing a tableau refutation of a formula (a Hintikka set) as is done in [4] but some care must be taken in the way the steps are formalized so as not to appeal to the fan theorem to produce an open path. No discussion of how this might be done appears in [4].

The proof given in this paper appeals to the strengthened version of Schütte's lemma in [4] which uses the more liberal definition of semivaluation *pairs*, (rather than semivaluations) which provide an upper and lower bound for the truth values of the valuation eventually produced by Takahashi's V-complex construction.

As we have shown, it is possible to give an instance of such a pair (namely $cl(\_)$ and $[\![\_]\!]$) without using tableaux and prove they satisfy the semivaluation axioms without appeal to the excluded middle.

Constructive Completeness. Producing a constructive proof of completeness is itself problematic, as pointed out by Gödel and discussed in [12,23] if a sufficiently restrictive definition of validity is assumed, e.g. conventional Kripke models. However, there are a number of ways to liberalize the definition of validity to "save" constructive completeness [24,3,22,11], in particular by allowing truth-values in a sufficiently broad class of structures. In our case these structures include complete Heyting Algebras *in which we cannot decide whether or not any given element is distinct from* $\top$ or even, for that matter, if the structure itself collapses to a one-element set. This appears to be a natural Heyting-valued counterpart to Veldman's exploding nodes [24].

In [22] completeness for an intuitionistic system *with* cut is shown constructively by mapping each formula to its own equivalence class in the Lindenbaum cHa. We cannot use this semantics here since cut is required to show that the target structure is partially ordered.

The semantics used in this paper can be seen as a cut-free variant of the Lindenbaum algebra, in which formulas are mapped to the sets of contexts that prove them without cut. Here too, one is not required to decide the provability of formulae in order to show model existence, in contrast with the $\top, \bot$-valued semantics of [1,21].

References

1. Andrews, P.: Resolution in type theory. Journal of Symbolic Logic 36(3) (1971)
2. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)
3. de Swart, H.C.M̃.: Another intuitionistic completeness proof. Journal of Symbolic Logic 41, 644–662 (1976)
4. DeMarco, M., Lipton, J.: Completeness and cut elimination in the intuitionistic theory of types. Journal of Logic and Computation, 821–854 (November 2005)
5. Dowek, G., Hardin, T., Kirchner, C.: Theorem proving modulo. Journal of Automated Reasoning 31, 33–72 (2003)
6. Dowek, G., Werner, B.: Proof normalization modulo. The Journal of Symbolic Logic 68(4), 1289–1316 (2003)
7. Friedman, H.: Equality between functionals. In: Parikh, R. (ed.) Logic Colloquium. Lecture Notes in Mathematics, vol. 453, pp. 22–37. Springer, Heidelberg (1975)
8. Girard, J.Y.: Une extension de l'interprétation de Gödel à l'analyse et son application à l'élimination de coupures dans l'analyse et la théorie des types. In: Fenstad, J.E. (ed.) Proceedings of the second Scandinavian proof theory symposium. North-Holland, Amsterdam (1971)
9. Girard, J.Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge University Press, Cambridge (1998)
10. Kleene, S.C.: Permutability of inferences in Gentzen's calculi LK and LJ. Memoirs of the American Mathematical Society 10, 1–26, 27–68 (1952)
11. Kreisel, G.: A remark on free choice sequences and the topological completeness proofs. Journal of Symbolic Logic 23, 369–388 (1958)
12. Kreisel, G.: On weak completeness of intuitionistic predicate logic. Journal of Symbolic Logic 27, 139–158 (1962)

13. Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic 51(1-2), 125–157 (1991)
14. Mitchell, J.: Foundations for Programming Languages. MIT Press, Cambridge (1996)
15. Okada, M.: Phase semantic cut-elimination and normalization proofs of first- and higher-order linear logic. Theoretical Computer Science 227, 333–396 (1999)
16. Okada, M.: A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. Theoretical Computer Science 281, 471–498 (2002)
17. Plotkin, G.: Lambda definability in the full type hierarchy. In: Seldin, J.P., Hindley, J.R. (eds.) To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism. Academic Press, New York (1980)
18. Prawitz, D.: Hauptsatz for higher order logic. The Journal of Symbolic Logic 33(3), 452–457 (1968)
19. Schütte, K.: Syntactical and semantical properties of simple type theory. Journal of Symbolic Logic 25, 305–326 (1960)
20. Tait, W.: A non-constructive proof of Gentzen's Hauptsatz for second-order predicate logic. Bulletin of the American Mathematical Society 72, 980–983 (1966)
21. Takahashi, M.-o.: A proof of cut-elimination in simple type theory. J. Math. Soc. Japan 19(4) (1967)
22. Troelstra, A.S., van Dalen, D.: Constructivism in Mathematics: An Introduction, vol. 2. Elsevier Science Publishers, Amsterdam (1988)
23. van Dalen, D.: Lectures on Intuitionism. Lecture Notes in Mathematics, vol. 337, pp. 1–94. Springer, Heidelberg (1973)
24. Veldman, W.: An intuitionistic completeness theorem for intuitionistic predicate logic. Journal of Symbolic Logic 41, 159–166 (1976)

Proving Infinitude of Prime Numbers Using Binomial Coefficients

Phuong Nguyen

University of Toronto

Abstract. We study the problem of proving in weak theories of Bounded Arithmetic the theorem that there are arbitrarily large prime numbers. We show that the theorem can be proved by some "minimal" reasoning (i.e., in the theory $\mathbf{I\Delta}_0$) using concepts such as (the logarithm) of a binomial coefficient. In fact we prove Bertrand's Postulate (that there is at least a prime number between n and $2n$, for all $n > 1$) and the fact that the number of prime numbers between n and $2n$ is of order $\Theta(n/\ln(n))$. The proofs that we formalize are much simpler than several existing formalizations, and our theory turns out to be a sub-theory of a recent theory proposed by Woods and Cornaros that extends $\mathbf{I\Delta}_0$ by a special counting function.

1 Introduction

A long standing problem in proof complexity theory is whether the fact that there are infinitely many prime numbers is provable in the theory $\mathbf{I\Delta}_0$, the theory over the vocabulary $0, 1, +, \cdot, <$ that is axiomatized by basic properties of this vocabulary and induction axioms for all bounded formulas. The problem remains open even when we replace $\mathbf{I\Delta}_0$ by $\mathbf{I\Delta}_0(\pi)$, a theory that extends $\mathbf{I\Delta}_0$ by adding the function $\pi(n)$ which is the number of prime numbers less than or equal to n [Woo81]. ($\mathbf{I\Delta}_0(\pi)$ is also called $\mathbf{I\Delta}_0(\pi) + \mathit{def}(\pi)$ in the literature.) The motivation for the latter is: suppose that we are able to count the number of primes, then is it possible to prove the infinitude of primes using some "minimal" reasoning?

These problems belong to the area recently named Bounded Reverse Mathematics [Coo07] whose purpose is to formalize and prove (the discrete versions of) mathematical theorems in weak theories of Bounded Arithmetic. A related problem [PWW88] is whether a weak form of the Pigeonhole Principle is provable in $\mathbf{I\Delta}_0$, or equivalently, whether it has polynomial-size constant-depth Frege proofs.

Recently some progress has been made in [WC07] where it is shown that $\mathbf{I\Delta}_0(\xi)$ (called $\mathbf{I\Delta}_0(\xi) + \mathit{def}(\xi)$ in [WC07]) proves the infinitude of primes. Here $\mathbf{I\Delta}_0(\xi)$ extends $\mathbf{I\Delta}_0$ by the function ξ that counts some definable sets of prime numbers. The function π can be defined using ξ, so $\mathbf{I\Delta}_0(\xi)$ is an extension of $\mathbf{I\Delta}_0(\pi)$. It is unlikely that ξ can be defined in $\mathbf{I\Delta}_0(\pi)$.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 184–198, 2008.
© Springer-Verlag Berlin Heidelberg 2008

In an earlier paper [Cor95] it is shown that the infinitude of primes is also provable in $\mathbf{I\Delta}_0(\pi, K)$, the theory that extends $\mathbf{I\Delta}_0(\pi)$ by a defining axiom for the function

$$K(n) = \sum_{i=1}^{n} \ln(i)$$

It is not clear whether $\mathbf{I\Delta}_0(\xi)$ extends $\mathbf{I\Delta}_0(\pi, K)$, or vice versa.

In this paper we show that the infinitude of prime numbers is provable in $\mathbf{I\Delta}_0(\pi, lbc)$, the theory obtained from $\mathbf{I\Delta}_0(\pi)$ by adding a defining axiom for the function

$$lbc(n) = \ln(\frac{(2n)!}{n!n!})$$

(*lbc* stands for *logarithm of binomial coefficient*). We also show that the function *lbc* is definable in $\mathbf{I\Delta}_0(\xi)$. Together with the fact proved in [WC07] that π is definable in $\mathbf{I\Delta}_0(\xi)$, this implies that $\mathbf{I\Delta}_0(\pi, lbc)$ is a sub-theory of $\mathbf{I\Delta}_0(\xi)$. So our results strengthen the results from [WC07]. On the other hand, we do not know whether our theory extends that of [Cor95], or vice versa.

Note that the function ξ [WC07] is a counting function that is more general than π, while both K [Cor95] and our function *lbc* are not. Also, if we add to $\mathbf{I\Delta}_0$ a counting function and its defining axiom for every $\mathbf{\Delta}_0$-definable set, then the resulting theory, here we called $\mathbf{I\Delta}_0(count)$, extends all $\mathbf{I\Delta}_0(\xi)$, $\mathbf{I\Delta}_0(\pi, K)$ and $\mathbf{I\Delta}_0(\pi, lbc)$. It has been shown [CD94] that $\mathbf{I\Delta}_0(count)$ proves the Prime Number Theorem (that there are $\Theta(n/\ln(n))$ primes less than n). It is easy to see that $\mathbf{I\Delta}_0(count)$ is equivalent to the number part of the theory $\mathbf{VTC}^0$ [NC05, CN06], a two-sorted theory that is associated with the two-sorted complexity class $\mathbf{TC}^0$.

1.1 Existing Formalizations

Our formalization is based on [Ngu08a, Chapter 8]. At high level, the proof that we choose to formalize is essentially the same as that of [WC07]. However, we explicitly use the binomial coefficients mentioned above, so our formalization is simpler. In fact, the axiom that we need to define *lbc* is provable (in $\mathbf{I\Delta}_0$) from the defining axiom for the function ξ introduced in [WC07]. Moreover, the function ξ seems to be indispensable for the formalization in [WC07], because it is needed in proving (the approximate version of) the asymptotic identity

$$(\psi(x) - \psi(\frac{x}{2}) + \psi(\frac{x}{3}) - \psi(\frac{x}{4}) + \ldots) = x\ln(2)$$

where

$$\psi(x) = \sum_{i \leq x} \Lambda(i)$$

and $\Lambda(x)$ is the von Mangoldt function,

$$\Lambda(x) = \begin{cases} \ln(p) & \text{if } x = p^j \text{ for some prime } p \text{ and some } j \geq 1 \\ 0 & \text{otherwise} \end{cases}$$

1.2 Our Formalizations

The proofs that we formalized are simple proofs which rely on different (approximate) representations of

$$\ln(\frac{(2n)!}{n!n!}) \tag{1}$$

One way of computing (1) is to use the fact that

$$\sum_{i=1}^{n} \ln(i) = n\ln(n) - n + \mathcal{O}(\ln(n)) \tag{2}$$

This produces

$$\ln(\frac{(2n)!}{n!n!}) = \sum_{i=1}^{2n} \ln(i) - 2\sum_{i=1}^{n} \ln(i) = 2n\ln(2) + \mathcal{O}(\ln(n)) \tag{3}$$

Another expression for (1) is

$$\sum_{p\leq 2n} \left(\ln(p) \sum_{1\leq j\wedge p^j\leq 2n} (\lfloor 2n/p^j \rfloor - 2\lfloor n/p^j \rfloor) \right) \tag{4}$$

This expression reveals useful information about the prime numbers that are $\leq 2n$. For example, it gives us

$$\ln(\frac{(2n)!}{n!n!}) \leq \pi(2n)\ln(2n)$$

and so a lower bound for $\pi(2n)$ follows using (3). Moser's simple proof of Bertrand's Postulate that we formalize also stems from (4) (see Lemma 18).

In our formalizations, the function *lbc* is defined based on the expression (4). The obstacle that prevents us from resolving Woods' conjecture is the inability to compute in $\mathbf{I\Delta}_0(\pi)$ this summation.

Of course we cannot compute the function $\ln(x)$ precisely, so as in [Woo81] we use an approximation to it. Our approximation and much of the formalizations are from [Ngu08a, Chapter 8]. The approximation to $\ln(x)$, denoted by $\ln(x, m)$ for a parameter m, is essentially the same as the approximation given in [Woo81]. Here we give a more detailed and direct proof of our version of (2).

1.3 Organization

The paper is organized as follows. In Section 2 we recall $\mathbf{I\Delta}_0$ and some important properties. In Section 2.2 we define in $\mathbf{I\Delta}_0$ an approximation to $\ln(x)$. The function *lbc* is defined in Section 2.6, and in Section 2.7 we show that it is definable in $\mathbf{I\Delta}_0(\xi)$. The $\mathbf{I\Delta}_0(\pi, lbc)$-proof of a lower bound for $\pi(n)$ is given in Section 3. The lower bound for $\pi(2n) - \pi(n)$ and Bertrand's Postulate are proved in Section 4.

2 The Theories $\mathbf{I\Delta_0}$, $\mathbf{I\Delta_0}(\pi)$, and $\mathbf{I\Delta_0}(\pi, lbc)$

The language of $\mathbf{I\Delta_0}$ is

$$\{0, 1, +, \cdot, <, =\}$$

The theory $\mathbf{I\Delta_0}$ is axiomatized by some basic defining axioms for the symbols in the language (see [HP93, Kra95, CN06]) and induction axiom scheme for bounded formulas. $\overline{\mathbf{I\Delta}}_0$ denotes the universal conservative extension of $\mathbf{I\Delta_0}$ obtained by adding Skolem functions that eliminate quantifiers in the axioms of $\mathbf{I\Delta_0}$. (We do not need the fact that $\overline{\mathbf{I\Delta}}_0$ is a universal theory here.)

(Instead of $\mathbf{I\Delta_0}$ and its extensions, we can use the two-sorted theory $\mathbf{V}^0$ [CN06] and its corresponding extensions, because $\mathbf{V}^0$ is conservative over $\mathbf{I\Delta_0}$ and the same can be shown for their respective extensions. Care should be taken, however, when we look at the associated complexity classes: $\mathbf{V}^0$ is associated with the two-sorted class $\mathbf{AC}^0$ where sets are presented by binary strings and numbers by unary strings; on the other hand, $\mathbf{I\Delta_0}$ is associated with the Linear Time Hierarchy, because here numbers are written in binary.)

The following theorem is from [Ben62, HP93, Bus98, CN06]:

Theorem 1. *The relation (on numbers) $y = z^x$ can be represented by a $\mathbf{\Delta}_0$ formula.*

Corollary 2. *The function $|x|$ (or also $\log(x)$), where $|0| = 0$ and $|x| = \lfloor \log_2(x) \rfloor$ if $x \geq 1$, is definable in $\mathbf{I\Delta_0}$.*

The following theorem is from [Woo81]:

Theorem 3. *For a bounded $\mathbf{\Delta}_0$-sequence $x_1, x_2, \ldots, x_\ell$ where $\ell \leq (\log(a))^d$ for some a and some constant $d \in \mathbb{N}$, the function*

$$\sum_{1 \leq i \leq \ell} x_i$$

is definable in $\mathbf{I\Delta_0}$ and it is provable in $\overline{\mathbf{I\Delta}}_0$ that

$$\sum_{1 \leq i \leq \ell+1} x_i = \sum_{1 \leq i \leq \ell} x_i + x_{\ell+1}$$

2.1 Rational Numbers in $\mathbf{I\Delta_0}$

We will approximate the natural logarithm function by rational numbers. Here we only need nonnegative numbers which can be defined in $\mathbf{I\Delta_0}$ by pairs $\langle x, y \rangle$, where

$$\langle x, y \rangle =_{\text{def}} (x + y)(x + y + 1) + 2y$$

For readability we will write $\frac{x}{y}$ for $\langle x, y \rangle$. Equality, inequality, addition and multiplication for rational numbers are defined in the standard way, and these are preserved under the embedding $x \mapsto \frac{x}{1}$. For example, $=_{\mathbb{Q}}$ and $\leq_{\mathbb{Q}}$ are defined as:

$$\frac{x}{y} =_{\mathbb{Q}} \frac{x'}{y'} \equiv xy' = x'y, \qquad \text{and} \qquad \frac{x}{y} \leq_{\mathbb{Q}} \frac{x'}{y'} \equiv xy' \leq x'y$$

Then it can be shown that

$$\mathbf{I\Delta}_0 \vdash \lfloor x/y \rfloor \leq_{\mathbb{Q}} \frac{x}{y} <_{\mathbb{Q}} \lfloor x/y \rfloor + 1$$

(here $\lfloor x/y \rfloor = max\{z \ : \ zy \leq x\}$, and $r <_{\mathbb{Q}} s \equiv (r \leq_{\mathbb{Q}} s \wedge r \neq_{\mathbb{Q}} s)$). In the following discussion, we will simply omit the subscript $\mathbb{Q}$ from $=_{\mathbb{Q}}$, $\leq_{\mathbb{Q}}$, etc.; the exact meaning will be clear from the context.

For a rational number $\frac{r}{s} \geq 1$, define

$$|\frac{r}{s}| = max\{i \ : \ s2^i \leq r\}$$

2.2 Approximating $\ln(x)$ in $\mathbf{I\Delta_0}$

We will now define in $\mathbf{I\Delta}_0$ a function $\ln(x, m)$ which approximates $\ln(x)$ up to $\mathcal{O}(|x|/m)$, for $x \in \mathbb{N}$, where m is a polynomial in $|a|$. Following [Woo81] we will first define $\ln(x, m)$ that approximates $\ln(x)$ upto $1/m$ for $1 \leq x \leq 2$. Then for $x > 2$ define

$$\ln(x, m) = |x| \ln(2, m) + \ln(\frac{x}{2^{|x|}}) \tag{5}$$

It is easy to see that for any $x > 1$, $\ln(x, m)$ approximates $\ln(x)$ upto $\mathcal{O}(|x|/m)$.

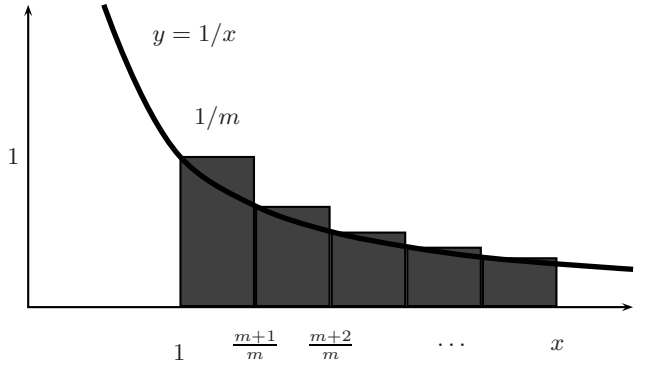


Fig. 1. Defining $\ln(x, m)$ for $1 \leq x \leq 2$: the shaded area is (6)

Our definition of $\ln(x, m)$ for $1 \leq x \leq 2$ is essentially the same as the definition of $\log^+$ of [Woo81]. Note that

$$\ln(x) = \int_1^x \frac{1}{y} dy$$

Our approximation will be roughly (the shaded area in Figure 1):

$$\sum_{m \leq k < \lceil mx \rceil} \frac{1}{m} \frac{1}{k/m} = \sum_{m \leq k < \lceil mx \rceil} \frac{1}{k} \tag{6}$$

We will not compute this summation precisely (since we want to avoid computing the common denominator). Instead we approximate $\frac{1}{k}$ by $\frac{\lfloor b/k \rfloor}{b}$ for some b determined below. Thus

$$\ln(x,m) = \frac{\sum_{m \le k < \lceil mx \rceil} \lfloor b/k \rfloor}{b} \tag{7}$$

The summation in (7) can be carried out in $\mathbf{I\Delta}_0$ by Theorem 3.

Notice that (6) is an upper bound for $\ln(x)$ with an error (the total area of the shaded region above the line $xy = 1$) at most $1/m$, and (7) is a lower bound for (6) with an error at most mx/b. So to get an $1/m$-approximation to $\ln(x)$ it suffices to take $b = m^3$.

Notation. Throughout this paper, fix some a sufficiently large and m a power of 2, $m = \mathit{polylog}(a) = 2^h$. (In particular, $m > |a|^2$.) We use $\|\cdot\|$ for absolute value, e.g., $\|t_1 - t_2\| \le s$ is an abbreviation for $t_1 \le t_2 + s \wedge t_2 \le t_1 + s$.

Definition 4 ($\ln(x,m)$ or just $\ln(x)$). *Let a, m be as in the above Notation. For $1 \le x \le 2$, $\ln(x,m)$ is defined as in (7) with $b = m^3$. For $x > 2$, $\ln(x,m)$ is defined as in (5).*

Lemma 5 (Provable in $\overline{\mathbf{I\Delta}}_0$). a) $x \le y \supset \ln(x,m) \le \ln(y,m)$.
b) $\|\ln(xy,m) - (\ln(x,m) + \ln(y,m))\| = \mathcal{O}(\frac{|x|+|y|}{m})$

For a proof see [Ngu08b, Lemma 2.5]

2.3 Defining $\sum \ln(i)$ in $\mathbf{I\Delta}_0$

The fact that $\sum_{1 \le i \le n} \ln(i)$ is definable in $\mathbf{I\Delta}_0$ is from [Woo81]. We reprove it here (for our definition of $\ln(x)$) in order to roughly estimate the sum.

Theorem 6. a) *The following function is definable in $\mathbf{I\Delta}_0$:*

$$\sum_{i=1}^{n} \ln(i) \tag{8}$$

b) *Let*

$$S = \sum_{i=1}^{m} \ln(i), \qquad T = \sum_{t=1}^{m} \ln(\frac{m+t}{m}), \qquad T_n = \sum_{i=2^{|n-1|}+1}^{n} \ln(\frac{i}{2^{|n-1|}}) \tag{9}$$

Then S, T, T_n are definable in $\mathbf{I\Delta}_0$, and it is provable in $\overline{\mathbf{I\Delta}}_0$ that (let $\ell = |n-1|$)

$$\sum_{i=1}^{n} \ln(i) = S + (n\ell - 2^{\ell+1} - (h-2)2^h)\ln(2) + (2^{\ell-h} - 1)T + T_n \tag{10}$$

c) *It is provable in $\overline{\mathbf{I\Delta}}_0$ that*

$$\sum_{i=1}^{n+1} \ln(i) = \sum_{i=1}^{n} \ln(i) + \ln(n+1) \tag{11}$$

For a proof see [Ngu08b, Theorem 2.6]

2.4 $\mathbf{I\Delta_0(\pi)}$ and Defining $\sum \ln(p)$ in $\mathbf{I\Delta_0(\pi)}$

Notation. Throughout this paper, the index p is used for prime numbers. $\mathcal{P}$ denotes the set of prime numbers. Note that the relation $x \in \mathcal{P}$ is represented by a $\mathbf{\Delta}_0$ formula.

Let

$$\pi(n) = \#\{p \leq n \ : \ p \in \mathcal{P}\}$$

$\mathbf{I\Delta}_0(\pi)$ extends $\mathbf{I\Delta}_0$ by π and the following defining axioms for it:

$$\pi(0) = 0$$

$$\pi(n+1) = \begin{cases} \pi(n) & \text{if } n+1 \notin \mathcal{P} \\ \pi(n)+1 & \text{otherwise} \end{cases}$$

Chebyshev's function

$$\vartheta(x) = \sum_{p \leq x} \ln(p) \tag{12}$$

plays an important role. Here we use

$$\vartheta(x,m) = \sum_{p \leq x} \ln(p,m) \tag{13}$$

and will simply write $\vartheta(x)$ for $\vartheta(x,m)$. We use the following defining axioms for ϑ:

$$\vartheta(1) = 0, \qquad \vartheta(n+1) = \begin{cases} \vartheta(n) + \ln(n+1) & \text{if } n+1 \in \mathcal{P} \\ \vartheta(n) & \text{otherwise} \end{cases} \tag{14}$$

Theorem 7. *The function $\vartheta(x)$ with defining axioms* (14) *is definable in $\mathbf{I\Delta}_0(\pi)$.*

For a proof see [Ngu08b, Theorem 2.7]

2.5 Unique Prime Factorization

The Fundamental Theorem of Arithmetic (or Unique Prime Factorization Theorem) states that any natural number $n > 1$ can be written uniquely as

$$n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$$

where $p_1 < p_2 < \ldots < p_k$ are prime numbers, and $e_i \geq 1$.

In $\mathbf{I\Delta}_0$ we can prove the existence and uniqueness of the sequence

$$(p_1, e_1), (p_2, e_2), \ldots, (p_k, e_k)$$

that contains all prime divisors of n, and $e_i \geq 1, p_i^{e_i} \mid n, p_i^{e_i+1} \nmid n$. Note that the sequence can be encoded by a binary string of length $\mathcal{O}(|n|)$. Also, the product

$$\prod_{i=1}^{k} p_i^{e_i}$$

for such sequence can be defined and proved to be n in $\mathbf{I\Delta}_0$.

Here we use the following function which is provably total in $\mathbf{I\Delta}_0$ (**ex** stands for exponent):

$$\mathbf{ex}(p,n) = max\{j \ : \ p^j|n\} \tag{15}$$

Our version of the Fundamental Theorem of Arithmetic is as follows:

Lemma 8. *The sum*

$$\sum_{p|n} \mathbf{ex}(p,n)\ln(p,m)$$

is definable in $\mathbf{I\Delta}_0$*, and it is provable in* $\overline{\mathbf{I\Delta}}_0$ *that*

$$\|\ln(n,m) - \sum_{p|n} \mathbf{ex}(p,n)\ln(p,m)\| = \mathcal{O}(\frac{|n|}{m})$$

For a proof see [Ngu08b, Lemma 2.9]

2.6 The Function *lbc*

Note that

$$n! = \prod_{p\le n} p^{e_p} \qquad \text{where} \ \ e_p = \sum_{1\le j\wedge p^j\le n} \lfloor n/p^j \rfloor \tag{16}$$

We use the function **exfac** for e_p above.

Corollary 9. *The following function is provably total in* $\mathbf{I\Delta}_0$*:*

$$\mathbf{exfac}(p,n) = \sum_{1\le j\wedge p^j\le n} \lfloor n/p^j \rfloor$$

Also, $\overline{\mathbf{I\Delta}}_0$ *proves that*

$$\mathbf{exfac}(p,1) = 0, \qquad \textit{and} \qquad \mathbf{exfac}(p,n) = \mathbf{ex}(p,n) + \mathbf{exfac}(p,n-1) \tag{17}$$

Proof. The fact that $\mathbf{exfac}(p,n)$ is provably total in $\mathbf{I\Delta}_0$ follows from Theorem 3 and the fact that the sum in the definition of $\mathbf{exfac}(p,n)$ has length $\le |n|$. The second property in (17) is proved by induction on n. □

Lemma 10 (Provable in $\overline{\mathbf{I\Delta}}_0$)

$$0 \le \mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n) \le \frac{\ln(2n)}{\ln(p)} + \mathcal{O}(\frac{|n|}{m})$$

For a proof see [Ngu08b, Lemma 2.11].

Note that from (16) we have

$$\frac{(2n)!}{n!n!} = \prod_{p\le 2n} p^{e'_p} \qquad \text{where} \ \ e'_p = \sum_{1\le j\wedge p^j\le 2n} (\lfloor 2n/p^j \rfloor - 2\lfloor n/p^j \rfloor) \tag{18}$$

Now we introduce the following functions (*lbc* stands for *logarithm of binomial coefficient*):

$$lbc(n) = \ln(\frac{(2n)!}{n!n!}) = \sum_{p \leq 2n} e'_p \ln(p) = \sum_{p \leq 2n} (\mathbf{exfac}(p, 2n) - 2\mathbf{exfac}(p, n)) \ln(p)$$

Recall that $\mathcal{P}$ denotes the set of prime numbers. The function *lbc* is formally defined as follows.

Definition 11. *Let* lbc' *be the function with the following defining axioms*

$$lbc'(n, 1) = 0$$

$$lbc'(n, k+1) = \begin{cases} lbc'(n, k) & \text{if } k+1 \notin \mathcal{P} \\ lbc'(n, k) + (\mathbf{exfac}(p, 2n) - 2\mathbf{exfac}(p, n)) \ln(p) & \text{if } k+1 = p \in \mathcal{P} \end{cases}$$

Let $lbc(n) = lbc'(n, 2n)$.

Theorem 12. *It is provable in* $\overline{\mathbf{I\Delta}}_0(lbc)$ *that*

$$lbc(n) = \sum_{i=1}^{2n} \ln(i) - 2\sum_{i=1}^{n} \ln(i) + \mathcal{O}(\frac{n|n|}{m})$$

For a proof see [Ngu08b, Theorem 2.13]

2.7 Defining *lbc* in $\mathbf{I\Delta}_0(\xi)$

The theory $\mathbf{I\Delta}_0(\xi) + def(\xi)$ [WC07] is obtained from $\mathbf{I\Delta}_0$ by augmenting the function ξ and its defining axioms. The function $\xi(x) = \xi(x, y, e)$ [WC07] is

$$\xi(x) = \#\{p \ : \ p \in \mathcal{P}, p \leq x, \text{and } \lfloor y/p^e \rfloor \text{ is odd}\}$$

and has defining axioms (suppressing y, e):

$$\xi(0) = 0$$

$$\xi(x+1) = \begin{cases} \xi(x) + 1 & \text{if } x+1 \in \mathcal{P} \text{ and } \lfloor y/(x+1)^e \rfloor \text{ is odd} \\ \xi(x) & \text{otherwise} \end{cases}$$

Here we show that our function *lbc* is definable in $\mathbf{I\Delta}_0(\xi) + def(\xi)$. As a result, the lower bounds for $\pi(n)$ and $\pi(2n) - \pi(n)$ that we prove in the following sections are also theorems of $\mathbf{I\Delta}_0(\xi) + def(\xi)$. Thus we obtain alternative proofs for the results from [WC07].

Theorem 13. *The function lbc with defining axioms given in Definition 11 is definable in* $\mathbf{I\Delta}_0(\xi) + def(\xi)$.

Proof. We show how to compute $lbc'(n,k)$ in $\mathbf{I\Delta}_0(\xi)$. Note that

$$lbc'(n,k) = \sum_{p\le k}(\mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n))\ln(p)$$

and by Lemma 10,

$$0 \le \mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n) \le \frac{\ln(2n)}{\ln(p)} + \mathcal{O}(\frac{|n|}{m})$$

By definition,

$$\mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n) = \sum_{p^j\le 2n}\lfloor 2n/p^j\rfloor - 2\sum_{p^j\le 2n}\lfloor n/p^j\rfloor$$

So, since the summations have length $\le |n|$, it is provable in $\overline{\mathbf{I\Delta}}_0$ that

$$\mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n) = \sum_{p^j\le 2n}(\lfloor 2n/p^j\rfloor - 2\lfloor n/p^j\rfloor)$$

In other words,

$$\mathbf{exfac}(p,2n) - 2\mathbf{exfac}(p,n) = \#\{j \le \frac{\ln(2n)}{\ln(p)} \ : \ \lfloor 2n/p^j\rfloor \text{ is odd}\}$$

As a result,

$$lbc'(n,k) = \sum_{j\le \ln(2n)}\left(\sum_{p\le k\wedge \lfloor 2n/p^j\rfloor \text{ is odd}}\ln(p)\right)$$

The summation in brackets can be computed in $\mathbf{I\Delta}_0(\xi)$ using the counting function ξ just as described in Theorems 2.6 and 2.7 of [Ngu08b]. □

3 A Lower Bound for $\pi(n)$ in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$

Note that $\pi(2n-1) = \pi(2n)$ for $n \ge 2$. So it suffices to give a lower bound for $\pi(2n)$. We choose a simple proof for the $\Omega(n/\ln(n))$ lower bound for $\pi(2n)$ and point out that this proof can be formalized using the function lbc introduced above. From this lower bound for $\pi(n)$ we can derive in $\mathbf{I\Delta}_0(\pi, lbc')$ the fact that there are infinitely many prime numbers.

The idea is to compute an upper bound and a lower bound for $\frac{(2n)!}{n!n!}$; by comparing these bounds we can derive a lower bound for $\pi(2n)$. In our formalization, we will use $lbc(n)$ instead of $\frac{(2n)!}{n!n!}$.

Lemma 14 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$)

$$lbc(n) \le \pi(2n)(\ln(2n) + \mathcal{O}(\frac{|n|}{m}))$$

Proof. We prove by induction on $k \leq 2n$ that $lbc'(n,k) \leq \pi(k)\ln(2n)$ using the defining axioms for lbc' (Definition 11) and Lemma 10. □

Lemma 15 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$). *For $n > m$:*

$$lbc(n) = 2n\ln(2) + c(m) + \mathcal{O}(\frac{n|n|}{m}) \tag{19}$$

for some constant $c(m)$ depends only on m.

Proof. By (10) in Theorem 6 we have

$$\sum_{i=1}^{2n} \ln(i) - \sum_{i=1}^{n} \ln(i) = (2n + (h-2)2^{h+1})\ln(2) + T - S$$

where T, S depend only on m (recall also that $m = 2^h$). Now the lemma follows from Theorem 12. □

Corollary 16 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$)

$$\pi(n) = \Omega(n/\ln(n)) \tag{20}$$

It follows that the existence of arbitrarily large prime numbers is provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc)$.

4 Bertrand's Postulate and a Lower Bound for $\pi(2n) - \pi(n)$

We will prove Bertrand's Postulate (that $\pi(2n) - \pi(n) \geq 1$ for all n) and a lower bound for the number of prime numbers between n and $2n$: $\pi(2n) - \pi(n) = \Omega(n/\ln(n))$. For the latter, we follow the proof from [Mos49]. First we outline the proof of the lower bound for $\pi(2n) - \pi(n)$; the formalizations are given in Section 4.1.

Recall Chebyshev's function $\vartheta(x)$ from (12).

Theorem 17. *For $n \geq 1$, $\vartheta(n) < 2n\ln(2)$.*

Proof. First, because

$$\frac{(2k+1)!}{k!(k+1)!}$$

appears twice in the binomial expansion of 2^{2k+1}, we have

$$\frac{(2k+1)!}{k!(k+1)!} \leq \frac{1}{2}2^{2k+1} = 2^{2k} \tag{21}$$

Also, all primes p where $k+1 < p \leq 2k+1$ divide $\frac{(2k+1)!}{k!(k+1)!}$. Hence

$$\prod_{k+1<p\leq 2k+1} p \leq \frac{(2k+1)!}{k!(k+1)!} \tag{22}$$

Consequently,

$$\vartheta(2k+1) - \vartheta(k+1) = \sum_{k+1<p\leq 2k+1} \ln(p) \leq \ln \frac{(2k+1)!}{k!(k+1)!} \leq \ln(2^{2k}) = 2k\ln(2) \tag{23}$$

Now we prove the theorem by induction on n. The base cases ($n = 1$ and $n = 2$) are trivial. For the induction step, the case where n is even is also obvious, since then $\vartheta(n) = \vartheta(n-1)$. So suppose that $n = 2k+1$. Using (23) and the induction hypothesis (for $n = k+1$) we have $\vartheta(2k+1) < 2k\ln(2) + 2(k+1)\ln(2) = 2(2k+1)\ln(2)$. □

Note that this theorem gives a $\mathcal{O}(n/\ln(n))$ upper bound for $\pi(n)$, but we do not need this fact here.

Lemma 18

$$\frac{(2n)!}{n!n!} \leq (2n)^{\sqrt{2n}} \left(\prod_{\sqrt{2n}<p\leq 2n/3} p \right) \left(\prod_{n<p<2n} p \right) \tag{24}$$

Proof. From (18), by noting that

$$e'_p \begin{cases} = 1 & \text{if } n < p < 2n \\ = 0 & \text{if } 2n/3 < p \leq n \\ \leq 1 & \text{if } \lceil\sqrt{2n}\rceil \leq p \leq \lfloor 2n/3 \rfloor \\ \leq \frac{\ln(2n)}{\ln(p)} & \text{if } p < \sqrt{2n} \end{cases}$$

□

Corollary 19. $\pi(2n) - \pi(n) = \Omega(n/\ln(n))$.

Proof. Note that

$$\frac{(2n)!}{n!n!} \geq \frac{2^{2n}}{2n+1}$$

(because $\frac{(2n)!}{n!n!}$ is the largest coefficient in $(1+1)^{2n}$). Therefore

$$\ln(\frac{(2n)!}{n!n!}) \geq 2n\ln(2) - \ln(2n+1)$$

Also,

$$\ln \left(\prod_{\sqrt{2n}<p\leq 2n/3} p \right) \leq \ln \left(\prod_{p\leq 2n/3} p \right) = \vartheta(2n/3)$$

so by Theorem 17,

$$\ln \left(\prod_{\sqrt{2n}<p\leq 2n/3} p \right) < 4n\ln(2)/3$$

In addition,

$$\ln\left(\prod_{n<p<2n} p\right) < (\pi(2n) - \pi(n))\ln(2n)$$

As a result, by taking logarithm of both sides of (24) we have

$$2n\ln(2) - \ln(2n+1) < \sqrt{2n}\ln(2n) + 4n\ln(2)/3 + (\pi(2n) - \pi(n))\ln(2n)$$

From this the conclusion follows easily. □

4.1 Formalization in $\overline{\mathbf{I\Delta_0}}(\pi, lbc')$

Recall (Section 2.4) that our version of Chebyshev's function, $\vartheta(x, m)$, or simply $\vartheta(x)$, is definable in $\mathbf{I\Delta}_0(\pi)$. Following Theorem 17 we prove:

Theorem 20 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$). *For some constant $c'(m)$,*

$$\vartheta(n, m) \leq 2n\ln(2) + |n|c'(m) + \mathcal{O}(\frac{n|n|}{m})$$

Proof. Note that

$$\ln(\frac{(2k+1)!}{k!(k+1)!}) = lbc(k+1) - \ln(2)$$

Using Lemma 8 and from the definition of lbc (Definition 11), we can prove in $\overline{\mathbf{I\Delta}}_0(\pi, lbc)$ that

$$\ln(2) + \sum_{k+1<p\leq 2k+1} \ln(p) \leq lbc(k+1)$$

(By proving by induction on $j \leq 2k$ that

$$\ln(2) + \sum_{k+1<p\leq j} \ln(p) \leq lbc'(k+1, j)$$

We will have to consider two cases: either $k+1$ is a power of 2, or not.)

As a result, by Lemma 15 we have

$$\sum_{k+1<p\leq 2k+1} \ln(p) \leq lbc(k+1) - \ln(2) = 2k\ln(2) + (c(m) + \ln(2)) + \mathcal{O}(\frac{k|k|}{m})$$

That is, for $c'(m) = c(m) + \ln(2)$,

$$\vartheta(2k+1) - \vartheta(k+1) \leq 2k\ln(2) + c'(m) + \mathcal{O}(\frac{k|k|}{m})$$

Now we can prove by strong induction on k that

$$\vartheta(k) \leq 2k\ln(2) + |k|c'(m) + \mathcal{O}(\frac{k|k|}{m})$$

(using the fact that $|2k+1| = |k| + 1$). □

Following Lemma 18 we have:

Lemma 21 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc,)$)

$$lbc(n) \leq \lfloor \sqrt{2n} \rfloor \ln(2n) + \vartheta(\frac{2n}{3}) + \sum_{n<p<2n} \ln(p)$$

Proof. The proof is similar to the proof of Lemma 14. □

Corollary 22 (Provable in $\overline{\mathbf{I\Delta}}_0(\pi, lbc')$)

$$\pi(2n) - \pi(n) = \Omega(\frac{n}{\ln(n)})$$

Proof. By Lemma 15, Theorem 20 and the above lemma we have

$$2n\ln(2) + c(m) + \mathcal{O}(\frac{n|n|}{m}) \leq \lfloor \sqrt{2n} \rfloor \ln(2n) + \left(\frac{4n\ln(2)}{3} + |\frac{2n}{3}|c'(m) + \mathcal{O}(\frac{n|n|}{m}) \right) + \sum_{n<p<2n} \ln(p)$$

It follows that for $n > m^2, m > |n|^2$:

$$\sum_{n<p<2n} \ln(p) \geq \frac{2\ln(2)}{3} n - \mathcal{O}(\frac{n|n|}{m})$$

The conclusion follows from the fact provable in $\overline{\mathbf{I\Delta}}_0(\pi)$ that the LHS is at most $(\pi(2n) - \pi(n))\ln(2n)$. □

Corollary 23 (Provable in $\mathbf{I\Delta}_0(\pi, lbc')$). *For all n, $\pi(2n) - \pi(n) \geq 1$.*

Proof. The previous corollary shows that for some standard threshold $n_0 \in \mathbb{N}$, $\pi(2n) - \pi(n) > 0$ for all $n \geq n_0$. The fact that $\pi(2n) - \pi(n) \geq 1$ for $n < n_0$ is true in $\mathbb{N}$, and hence is provable in $\mathbf{I\Delta}_0$. □

5 Conclusion

Sylvester's Theorem asserts that for $1 \leq x \leq y$, some number among

$$y + 1, y + 2, \ldots, y + x$$

has a prime divisor $p > x$. In [Woo81] it is shown that Sylvester's Theorem can be proved in $\mathbf{I\Delta}_0 + PHP(\mathbf{\Delta}_0)$. ($PHP(\mathbf{\Delta}_0)$ is the axiom scheme which asserts that the Pigeonhole Principle, where the mappings between "pigeons" and "holes" are described by $\mathbf{\Delta}_0$ formula, is true.) Here, as well as in [Cor95, WC07], we have a $\Omega(n/\ln(n))$ lower bound for $\pi(2n) - \pi(n)$, the number of prime numbers between n and $2n$. Such lower bound does not seem to follow from the proof in

[Woo81]. However, it is not clear whether $PHP(\mathbf{\Delta}_0)$ is provable in $\mathbf{I\Delta}_0(\pi, lbc,)$ or even $\mathbf{I\Delta}_0(\xi) + def(\xi)$.

Also, as far as we know, the axiom for *lbc* considered here (or even the axiom for ξ considered in [WC07]) and the axiom for K [Cor95] are incomparable over $\mathbf{I\Delta}_0(\pi)$. It is an interesting problem to see whether one follows from the other in $\mathbf{I\Delta}_0$.

Acknowledgments. I would like to thank Steve Cook and the referees for their helpful comments.

References

[Ben62] Bennett, J.: On Spectra. PhD thesis, Princeton University, Department of Mathematics (1962)

[Bus98] Buss, S.: First–Order Proof Theory of Arithmetic. In: Buss, S. (ed.) Handbook of Proof Theory, pp. 79–147. Elsevier, Amsterdam (1998)

[CD94] Cornaros, C., Dimitracopoulos, C.: The Prime Number Theorem and Fragments of **PA**. Archive for Mathematical Logic 33, 265–281 (1994)

[CN06] Cook, S., Nguyen, P.: Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations (Book in progress, 2006)

[Coo07] Cook, S.: Bounded Reverse Mathematics. In: Plenary Lecture for CiE 2007 (2007)

[Cor95] Cornaros, C.: On Grzegorczyk Induction. Annals of Pure and Applied Logic 74, 1–21 (1995)

[HP93] Hájek, P., Pudlák, P.: Metamathematics of First-Order Arithmetic. Springer, Heidelberg (1993)

[Kra95] Krajíček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Cambridge University Press, Cambridge (1995)

[Mos49] Moser, L.: A theorem on the distribution of primes. American Mathematical Monthly 56(9), 624–625 (1949)

[NC05] Nguyen, P., Cook, S.: Theory for $\mathbf{TC}^0$and Other Small Complexity Classes. Logical Methods in Computer Science 2 (2005)

[Ngu08a] Nguyen, P.: Bounded Reverse Mathematics. PhD thesis, University of Toronto (2008), `http://www.cs.toronto.edu/~pnguyen/`

[Ngu08b] Nguyen, P.: Proving Infinitude of Prime Numbers Using Binomial Coefficients (submitted, 2008), `http://www.cs.toronto.edu/~pnguyen/`

[PWW88] Paris, J.B., Wilkie, A.J., Woods, A.R.: Provability of the pigeonhole principle and the existence of infinitely many primes. Journal of Symbolic Logic 53(4), 1235–1244 (1988)

[WC07] Woods, A., Cornaros, C.: On bounded arithmetic augmented by the ability to count certain sets of primes (unpublished, 2007)

[Woo81] Woods, A.: Some Problems in Logic and Number Theory and Their Connections. PhD thesis, University of Manchester (1981)

A Tight Karp-Lipton Collapse Result in Bounded Arithmetic

Olaf Beyersdorff[1] and Sebastian Müller[2,⋆]

[1] Institut für Theoretische Informatik, Leibniz Universität Hannover, Germany
beyersdorff@thi.uni-hannover.de
[2] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
smueller@informatik.hu-berlin.de

Abstract. Cook and Krajíček [9] have obtained the following Karp-Lipton result in bounded arithmetic: if the theory PV proves $\mathsf{NP} \subseteq \mathsf{P}/poly$, then PH collapses to BH, and this collapse is provable in PV. Here we show the converse implication, thus answering an open question from [9]. We obtain this result by formalizing in PV a hard/easy argument of Buhrman, Chang, and Fortnow [3].

In addition, we continue the investigation of propositional proof systems using advice, initiated by Cook and Krajíček [9]. In particular, we obtain several optimal and even p-optimal proof systems using advice. We further show that these p-optimal systems are equivalent to natural extensions of Frege systems.

Keywords: Karp-Lipton Theorem, Advice, Optimal Propositional Proof Systems, Bounded Arithmetic, Extended Frege.

1 Introduction

The classical Karp-Lipton Theorem states that $\mathsf{NP} \subseteq \mathsf{P}/poly$ implies a collapse of the polynomial hierarchy PH to its second level [15]. Subsequently, these collapse consequences have been improved by Köbler and Watanabe [16] to $\mathsf{ZPP}^{\mathsf{NP}}$ and by Sengupta and Cai to $\mathsf{S}_2^{\mathsf{p}}$ (cf. [4]). This currently forms the strongest known collapse result of this kind.

Recently, Cook and Krajíček [9] have considered the question which collapse consequences can be obtained if the assumption $\mathsf{NP} \subseteq \mathsf{P}/poly$ is provable in some weak arithmetic theory. This assumption seems to be stronger than in the classical Karp-Lipton results, because in addition to the inclusion $\mathsf{NP} \subseteq \mathsf{P}/poly$ we require an easy proof for it. In particular, Cook and Krajíček showed that if $\mathsf{NP} \subseteq \mathsf{P}/poly$ is provable in PV, then PH collapses to the Boolean hierarchy BH, and this collapse is provable in PV. For stronger theories, the collapse consequences become weaker. Namely, if PV is replaced by S_2^1, then $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}[O(\log n)]}$, and for S_2^2 one gets $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}}$ [9]. Still all these consequences are presumably stronger than in Sengupta's result above, because $\mathsf{P}^{\mathsf{NP}} \subseteq \mathsf{S}_2^{\mathsf{p}}$.

⋆ Supported by DFG grants KO 1053/5-1 and KO 1053/5-2.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 199–214, 2008.
© Springer-Verlag Berlin Heidelberg 2008

In [9] Cook and Krajíček ask whether under the above assumptions, their collapse consequences for PH are optimal in the sense that also the converse implications hold. In this paper we give an affirmative answer to this question for the theory PV. Thus PV proves $\mathsf{NP} \subseteq \mathsf{P}/poly$ if and only if PV proves $\mathsf{PH} \subseteq \mathsf{BH}$. To show this result we use the assertion $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$ as an intermediate assumption. Surprisingly, Cook and Krajíček [9] have shown that provability of this assumption in PV is equivalent to the provability of $\mathsf{NP} \subseteq \mathsf{P}/poly$ in PV. While such a trade-off between nondeterminism and advice seems rather unlikely to hold unconditionally, Buhrman, Chang, and Fortnow [3] proved that $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$ holds if and only if PH collapses to BH. Their proof in [3] refines the hard/easy argument of Kadin [14]. We formalize this technique in PV and thus obtain that $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$ is provable in PV if and only if PV proves $\mathsf{PH} \subseteq \mathsf{BH}$. Combined with the mentioned results from [9], this implies that $PV \vdash \mathsf{PH} \subseteq \mathsf{BH}$ is equivalent to $PV \vdash \mathsf{NP} \subseteq \mathsf{P}/poly$.

Assumptions of the form $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$ play a dominant role in the above Karp-Lipton results. These hypotheses essentially ask whether advice is helpful to decide propositional tautologies. Motivated by this observation, Cook and Krajíček [9] started to investigate propositional proof systems taking advice. In the second part of this paper we continue this line of research. We give a quite general definition of functional propositional proof systems with advice. Of particular interest are those systems where the advice depends on the proof (input advice) or on the proven formula (output advice).

In our investigation we focus on the question whether there exist optimal proof systems for different advice measures. While the existence of optimal propositional proof systems without advice is a long-standing open question, posed by Krajíček and Pudlák [18], we obtain optimal proof systems with input advice for each advice class. Such a result was already obtained by Cook and Krajíček [9], who prove that there is a system with one bit of input advice which is optimal for all systems using up to logarithmically many advice bits. We extend the proof method from [9] to obtain even p-optimal systems with input advice within each class of systems with super-logarithmic advice function.

These optimality results only leave open the question whether the classes of proof systems with constant advice contain p-optimal systems. We prove that for each constant k, there is a proof system which p-simulates all systems with k advice bits, but itself uses $k+1$ bits of advice. We also use a technique of Sadowski [20] to show that the existence of p-optimal proof systems for SAT_2 implies the existence of p-optimal propositional proof systems using k advice bits for each constant k.

In contrast to these optimality results for input advice, we show that we cannot expect similar results for proof systems with output advice, unless $\mathsf{PH} \subseteq \mathsf{BH}$ already implies $\mathsf{PH} \subseteq \mathsf{D}^{\mathsf{P}}$.

Finally, we consider classical proof systems like Frege systems using advice. We show that our optimal and p-optimal proof systems with advice are p-equivalent to extensions of Frege systems, thus demonstrating that these p-optimal proof systems admit a robust and meaningful definition.

Due to space constraints, a number of proofs is omitted or only briefly sketched in this extended abstract.

2 Preliminaries

Let $\Sigma = \{0,1\}$. Σ^n denotes the set of strings of length n, and $(\Sigma^n)^k$ the set of k-tuples of Σ^n. Let $\pi_i : (\Sigma^*)^k \to \Sigma^*$ be the projection to the i^{th} string, and let $\pi_i^* : \Sigma^* \to \{0,1\}$ be the projection to the i^{th} bit of a string. Let π^*_{-i} and π_{-i} be projections deleting the i^{th} string from a tuple or the i^{th} bit from a string, respectively. Although we enumerate the bits of a string starting with 0, we will speak of the first bit, the second bit, etc. of a string, and thus for example $\pi_1^*(a_0a_1a_2) = a_0$ and $\pi^*_{-1}(a_0a_1a_2) = a_1a_2$.

Let $\langle\cdot\rangle$ be a polynomial-time computable function, mapping tuples of strings to strings. Its inverse will be denoted by *enc*.

Complexity Classes. We assume familiarity with standard complexity classes (cf. [1]). In particular, we will need the *Boolean hierarchy* BH which is the closure of NP under the Boolean operations $\cup$, $\cap$, and $^-$. The levels of BH are denoted BH_k and are inductively defined by $\mathsf{BH}_1 = \mathsf{NP}$ and $\mathsf{BH}_{k+1} = \{L_1 \setminus L_2 \mid L_1 \in \mathsf{NP} \text{ and } L_2 \in \mathsf{BH}_k\}$. The second level BH_2 is also denoted by $\mathsf{D^P}$. The Boolean hierarchy coincides with $\mathsf{P}^{\mathsf{NP}[O(1)]}$, consisting of all languages which can be solved in polynomial time with constantly many queries to an NP-oracle. For each level BH_k it is known that k non-adaptive queries to an NP-oracle suffice, i.e., $\mathsf{BH}_k \subseteq \mathsf{P}_{tt}^{\mathsf{NP}[k]}$ (cf. [2]).

Complete problems BL_k for BH_k are inductively given by $\mathrm{BL}_1 = \mathrm{SAT}$ and

$$\mathrm{BL}_{2k} = \{\langle x_1, \ldots, x_{2k}\rangle \mid \langle x_1, \ldots, x_{2k-1}\rangle \in \mathrm{BL}_{2k-1} \text{ and } x_{2k} \in \overline{\mathrm{SAT}}\}$$
$$\mathrm{BL}_{2k+1} = \{\langle x_1, \ldots, x_{2k+1}\rangle \mid \langle x_1, \ldots, x_{2k}\rangle \in \mathrm{BL}_{2k} \text{ or } x_{2k+1} \in \mathrm{SAT}\} \ .$$

Observe that $\langle x_1, \ldots, x_k\rangle \in \mathrm{BL}_k$ if and only if there exists an $i \leq k$, such that x_i is satisfiable and the largest such i is odd.

Complexity classes with *advice* were first considered by Karp and Lipton [15]. For each function $k : \mathbb{N} \to \Sigma^*$ and each language L we let $L/k = \{x \mid \langle x, k(|x|)\rangle \in L\}$. If C is a complexity class and F is a class of functions, then $\mathsf{C}/F = \{L/k \mid L \in \mathsf{C}, k \in F\}$.

Propositional Proof Systems. Propositional proof systems were defined in a general way by Cook and Reckhow [11] as polynomial-time computable functions P which have as their range the set of all tautologies. A string π with $P(\pi) = \varphi$ is called a P-proof of the tautology φ. Equivalently, propositional proof systems can be defined as polynomial-time computable relations $P(\pi, \varphi)$ such that φ is a tautology if and only if $(\exists\pi)P(\pi, \varphi)$ holds. A propositional proof system P is *polynomially bounded* if all tautologies have polynomial size P-proofs.

Proof systems are compared according to their strength by simulations introduced in [11] and [18]. A proof system S *simulates* a proof system P (denoted by $P \leq S$) if there exists a polynomial p such that for all tautologies φ and

P-proofs π of φ there is an S-proof π' of φ with $|\pi'| \leq p(|\pi|)$. If such a proof π' can even be computed from π in polynomial time we say that S *p-simulates* P and denote this by $P \leq_p S$. If the systems P and S mutually (p-)simulate each other, they are called *(p-)equivalent*. A proof system is called *(p-)optimal* if it (p-)simulates all proof systems.

A prominent class of propositional proof systems is formed by *extended Frege systems EF* which are usual textbook proof systems based on axioms and rules, augmented by the possibility to abbreviate complex formulas by propositional variables to reduce the proof size (cf. [11,17]).

3 Representing Complexity Classes by Bounded Formulas

The relations between computational complexity and bounded arithmetic are rich and varied, and we refer to [17,10] for background information. Here we will use the two-sorted formulation of arithmetic theories [8,10]. In this setting we have two sorts: numbers and finite sets of numbers, which are interpreted as strings. Number variables will be denoted by lower case letter $x, y, n, \ldots$ and string variables by upper case letters $X, Y, \ldots$ The two-sorted vocabulary includes the symbols $+, \cdot, \leq, 0, 1$, and the function $|X|$ for the length of strings.

Our central arithmetic theory will be the theory VPV, which is the two-sorted analogue of Cook's PV [7]. In addition to the above symbols, the language of VPV contains names for all polynomial-time computable functions (where the running time is measured in terms of the length of the inputs with numbers coded in unary). The theory VPV is axiomatized by definitions for all these functions as well as by the number induction scheme for open formulas.

Bounded quantifiers for strings are of the form $(\forall X \leq t)\varphi$ and $(\exists X \leq t)\varphi$, abbreviating $(\forall X)(|X| \leq t \rightarrow \varphi)$ and $(\exists X)(|X| \leq t \wedge \varphi)$, respectively (where t is a number term not containing X). We use similar abbreviations for $=$ instead of $\leq$. By counting alternations of quantifiers, a hierarchy Σ_i^B, Π_i^B of bounded formulas is defined. The first level Σ_1^B contains formulas of the type $(\exists X_1 \leq t_1)\ldots(\exists X_k \leq t_k)\varphi$ with only bounded number quantifiers occurring in φ. Similarly, Π_1^B*-formulas* are of the form $(\forall X_1 \leq t_1)\ldots(\forall X_k \leq t_k)\varphi$.

As we want to investigate the provability of various complexity-theoretic assumptions in arithmetic theories, we need to formalize complexity classes within bounded arithmetic. To this end we associate with each complexity class C a class of arithmetic formulas $\mathcal{F}_\mathsf{C}$. The formulas $\mathcal{F}_\mathsf{C}$ describe C, in the sense that for each $A \subseteq \Sigma^*$ we have $A \in \mathsf{C}$ if and only if A is definable by an $\mathcal{F}_\mathsf{C}$-formula $\varphi(X)$ with a free string variable X.

It is well known that Σ_1^B-formulas describe NP-sets in this sense, and this connection extends to the formula classes Σ_i^B and Π_i^B and the respective levels $\mathsf{\Sigma_i^p}$ and $\mathsf{\Pi_i^p}$ of the polynomial hierarchy. Given this connection, we can model the levels BH_k of the Boolean hierarchy by formulas of the type

$$\varphi_1(X) \wedge \neg(\varphi_2(X) \wedge \ldots \neg(\varphi_{k-1}(X) \wedge \neg\varphi_k(X))\ldots) \tag{1}$$

with Σ_1^B-formulas $\varphi_1, \ldots, \varphi_k$.

Another way to speak about complexity classes in arithmetic theories is to consider complete problems for the respective classes. For the satisfiability problem SAT we can build an open formula $Sat(T, X)$, stating that T codes a satisfying assignment for the propositional formula coded by X. In VPV we can prove that $(\exists T \leq |X|)Sat(T, X)$ is NP-complete, in the sense, that every Σ_1^B-formula φ is provably equivalent to $(\exists T \leq |X|)Sat(T, F_\varphi(X))$ for some polynomial-time computable function F_φ.

Using this fact, we can express the classes BH_k in VPV equivalently as:

Lemma 1. *For every formula φ describing a language from BH_k as in* (1) *there is a polynomial-time computable function $F : \Sigma^* \to (\Sigma^*)^k$ such that VPV proves the equivalence of φ and*

$$\begin{aligned}(\exists T_1, T_3, \ldots, T_{2\cdot\lfloor k/2\rfloor+1} \leq t)(\forall T_2, T_4, \ldots, T_{2\cdot\lfloor k/2\rfloor} \leq t)\\ (\ldots((Sat(T_1, \pi_1(F(X))) \wedge \neg Sat(T_2, \pi_2(F(X))))\\ \vee Sat(T_3, \pi_3(F(X)))) \wedge \cdots \wedge_k \neg^{k+1} Sat(T_k, \pi_k(F(X))))\end{aligned} \tag{2}$$

where $\wedge_k = \wedge$ if k is even and $\vee$ otherwise, $\neg^k = \neg \ldots \neg$ (k-times), and t is a number term bounding $|F(X)|$. We will abbreviate (2) *by $BL_k(F(X))$.*

Similarly, we can define the class $\mathsf{P}_{tt}^{\mathsf{NP}[k]}$ by all formulas of the type

$$\begin{aligned}(\exists T_1 \ldots T_k \leq t)(Sat(T_1, F_1(X)) \wedge \cdots \wedge Sat(T_k, F_k(X)) \wedge \varphi_1(X)) \vee \cdots \vee\\ (\forall T_1 \ldots T_k \leq t)(\neg Sat(T_1, F_1(X)) \wedge \cdots \wedge \neg Sat(T_k, F_k(X)) \wedge \varphi_{2^k}(X))\end{aligned} \tag{3}$$

where $\varphi_1, \ldots, \varphi_{2^k}$ are open formulas, $F_1, \ldots, F_k$ are polynomial-time computable functions, and t is a term bounding $|F_i(X)|$ for $i = 1, \ldots, k$. In (3), every combination of negated and unnegated Sat-formulas appears in the disjunction.

With these arithmetic representations we can prove inclusions between complexity classes in arithmetic theories. Let A and B are complexity classes represented by the formula classes $\mathcal{A}$ and $\mathcal{B}$, respectively. Then we use $VPV \vdash \mathcal{A} \subseteq \mathcal{B}$ to abbreviate that for every formula $\varphi_\mathcal{A} \in \mathcal{A}$ there exists a formula $\varphi_\mathcal{B} \in \mathcal{B}$, such that $VPV \vdash \varphi_\mathcal{A}(X) \leftrightarrow \varphi_\mathcal{B}(X)$.

In the following, we will use the same notation for complexity classes and their respective representations. Hence we can write statements like $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}$, with the precise meaning explained above. For example, using Lemma 1 it is straightforward to verify:

Lemma 2. *For every number k we have $VPV \vdash \mathsf{BH}_k \subseteq \mathsf{P}_{tt}^{\mathsf{NP}[k]}$.*

Finally, we will consider complexity classes that take advice. Let $\mathcal{A}$ be a class of formulas. Then $VPV \vdash \mathcal{A} \subseteq \mathsf{NP}/k$ abbreviates that, for every $\varphi \in \mathcal{A}$ there exist Σ_1^B-formulas $\varphi_1, \ldots, \varphi_{2^k}$, such that

$$VPV \vdash (\forall n) \bigvee_{1 \leq i \leq 2^k} (\forall X)\,(|X| = n \to (\varphi(X) \leftrightarrow \varphi_i(X)))\ . \tag{4}$$

Similarly, using the self-reducibility of SAT, we can formalize the assertion $VPV \vdash \mathsf{NP} \subseteq \mathsf{P}/poly$ as

$$VPV \vdash (\forall n)(\exists C \leq t(n))(\forall X \leq n)(\forall T \leq n)(Sat(T, X) \to Sat(C(X), X))$$

where t is a number term and $C(X)$ is a term expressing the output of the circuit C on input X (cf.[9]).

4 The Karp-Lipton Collapse Result in *VPV*

In this section we will prove that the Karp-Lipton collapse $\mathsf{PH} \subseteq \mathsf{BH}$ from [9] is optimal in VPV, in the sense that $VPV \vdash \mathsf{NP} \subseteq \mathsf{P}/poly$ is equivalent to $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}$. For this we will use the following complexity-theoretic result.

Theorem 3 (Buhrman, Chang, Fortnow [3]). *For every constant k we have $\mathsf{coNP} \subseteq \mathsf{NP}/k$ if and only if $\mathsf{PH} \subseteq \mathsf{BH}_{2^k}$.*

While the forward implication of Theorem 3 is comparatively easy, and was shown to hold relative to VPV by Cook and Krajíček [9], the backward implication was proven in [3] by a sophisticated hard/easy argument. In the sequel, we will formalize this argument in VPV, thereby answering a question of Cook and Krajíček [9], who asked whether $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}$ already implies $VPV \vdash \mathsf{coNP} \subseteq \mathsf{NP}/O(1)$.

Assuming $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}$, we claim that there is some constant k such that $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}_k$. This follows, because $\mathsf{PH} \subseteq \mathsf{BH}$ implies $\mathsf{PH} = \mathsf{BH} = \Sigma_2^{\mathsf{p}}$. Therefore every problem in PH can be reduced to a fixed Σ_2^{p}-complete problem. Since this problem is contained in some level BH_k of BH, it can be reduced to an appropriate BH_k-complete problem as well. Thus $\mathsf{PH} \subseteq \mathsf{BH}_k$.

Therefore, BH_k is provably closed under complement in VPV, i.e., there exists a polynomial-time computable function h such that

$$VPV \vdash BL_k(X_1, \ldots, X_k) \leftrightarrow \neg BL_k(h(X_1, \ldots, X_k)) \ . \tag{5}$$

Given h, we define the notion of a *hard sequence*. This concept was defined in [6] as a generalization of the notion of hard strings from [14]. Hard strings were first used to show that $\mathsf{BH} \subseteq \mathsf{D}^{\mathsf{p}}$ implies a collapse of PH [14].

Definition 4. *Let h be a function as in (5). A sequence $\bar{x} = (x_1, \ldots, x_r)$ of strings is a* hard sequence *of order r for length n, if for all $i \leq r$, x_i is an unsatisfiable formula of length n, and for all $(k-r)$-tuples $\bar{u}$ of formulas of length n, the formula $\pi_{k-r+i}(h(\bar{u}, \bar{x}))$ is unsatisfiable.*

A hard sequence $\bar{x}$ of order r for length n is not extendable *if, for every unsatisfiable formula x of length n the sequence $x^\frown\bar{x}$ is not hard. Finally, a* maximal hard sequence *is a hard sequence of maximal order. Maximal hard sequences are obviously not extendable. Note that the empty sequence is a hard sequence for every length.*

To use this definition in VPV, we we note that the notion of a maximal hard sequence can be formalized by a bounded predicate *MaxHS*. Maximal hard sequences allow us to define the unsatisfiability of propositional formulas by a Σ_1^B-formula, as stated in the following lemma.

Lemma 5. *Assume that h is a polynomial-time computable function which for some constant k satisfies* (5)*. Then VPV proves the formula*

$$(\forall n)(\forall X = n)(\forall r \leq k)(\forall H \in (\Sigma^n)^{k-r-1})\ (MaxHS(H) \rightarrow$$
$$[(\forall T \leq n)\neg Sat(T, X) \leftrightarrow (\exists T \leq n)(\exists \bar{U} \in (\Sigma^n)^r) Sat(T, \pi_{r+1}(h(\bar{U}, X, H)))])\ .$$

By the preceding lemma, given maximal hard sequences, we can describe Π_1^B-formulas by Σ_1^B-formulas. Most part of the proof of the next theorem will go into the construction of such sequences. It will turn out, that, assuming $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}_{2^k}$, we can construct 2^k Σ_1^B-formulas, whose disjunction decides the elements of a maximal hard sequence as in (4).

Theorem 6. *If $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}_{2^k}$, then $VPV \vdash \mathsf{coNP} \subseteq \mathsf{NP}/k$.*

Proof. Assuming $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}_{2^k}$, there exists a polynomial-time computable function h, such that for tuples $\bar{X} = (X_1, \ldots, X_{2^k})$ we have $VPV \vdash BL_{2^k}(\bar{X}) \leftrightarrow \neg BL_{2^k}(h(\bar{X}))$. Thus, by Lemma 5, given a maximal hard sequence for length n, we can define $(\forall T \leq n)\neg Sat(T, X)$ by a Σ_1^B-formula. Therefore, our aim is to construct such a sequence using k bits of advice.

To this end, for $i > 0$ let $HardSeqBits(1^n, i)$ hold, if and only if the i^{th} bit of the encoding of the lexically shortest maximal hard sequence for length n is 1. *HardSeqBits* can be defined by a bounded predicate.

By the assumption $VPV \vdash \mathsf{PH} \subseteq \mathsf{BH}_{2^k}$ and Lemma 2, there is a formula ψ as in (3), with appropriate polynomial-time computable functions $F_1, \ldots, F_{2^k}$ and open formulas $\varphi_1, \ldots, \varphi_{2^{2^k}}$, such that the predicate $HardSeqBits(X)$ is VPV-provably equivalent to ψ. Without loss of generality, we may assume, that $|F_i(1^n, a)| = |F_j(1^n, b)|$ for all i, j and a, b.

Using ψ we can prove $VPV \vdash HardSeqBits \in \mathsf{NP}/k$ (we omit the details due to space constraints). This means that we can construct Σ_1^B-formulas $\psi^z_{HSB}(X)$ of the form $(\exists Y \leq t)\varphi^z_{HSB}(X, Y)$ with open formulas φ^z_{HSB} for $z = 0, \ldots, 2^k - 1$ such that

$$VPV \vdash (\forall n) \bigvee_{0 \leq z < 2^k} (\forall X = n)\,(HardSeqBits(X) \leftrightarrow (\exists Y \leq t)\varphi^z_{HSB}(X, Y))\ .$$

In this formula, z is the order of a maximal hard sequence for length n. Observe that z, acting as the advice, can be non-uniformly obtained from n.

Provided the right z, there is a Σ_1^B-formula $EasyUnSat_z(X)$ that, for every X of length n, is VPV-equivalent to $(\forall T \leq n)\neg Sat(T, X)$. This formula $EasyUnSat_z(X)$ is defined as

$$(\exists C \leq t')\ (\forall i \leq |C|)(\exists Y \leq t)[(\pi^*_{i+1}(C) = 1 \leftrightarrow \varphi^z_{HSB}(1^{|X|}, i, Y))$$
$$\wedge\ (\exists T \leq |X|)(\exists \bar{U} \in (\Sigma^n)^{2^k - 1 - |enc(C)|})$$
$$Sat(T, \pi_{2^k - |enc(C)|}(h(\bar{U}, X, enc(C))))]$$

for an appropriate number term t'. Now, by line 1 of this formula, C is the encoding of some maximal hard sequence. As in Lemma 5, C is used to define $\neg Sat$ by a Σ_1^B-formula (lines 2 and 3). Thus, we have

$$VPV \vdash (\forall n) \bigvee_{0 \le z < 2^k} (\forall X = n)[(\forall T \le n) \neg Sat(T, X) \leftrightarrow EasyUnSat_z(X)] \ .$$

This concludes the proof. □

With this result we can now prove the optimality of the following Karp-Lipton collapse result of Cook and Krajíček [9]:

Theorem 7 (Cook and Krajíček [9]). *If VPV proves $\mathsf{NP} \subseteq \mathsf{P}/poly$, then $\mathsf{PH} \subseteq \mathsf{BH}$, and this collapse is provable in VPV.*

To show the converse implication, we use the following surprising trade-off between advice and nondeterminism in VPV:

Theorem 8 (Cook and Krajíček [9]). *$VPV \vdash \mathsf{NP} \subseteq \mathsf{P}/poly$ if and only if $VPV \vdash \mathsf{coNP} \subseteq \mathsf{NP}/O(1)$.*

We remark that the proof of Theorem 8 uses strong witnessing arguments in form of the Herbrand Theorem and the KPT witnessing theorem [19]. Thus it seems unlikely, that a similar result holds without assuming provability of $\mathsf{NP} \subseteq \mathsf{P}/poly$ and $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$ in some weak arithmetic theory. Theorem 7 can be obtained as a consequence of Theorem 8 and a complexity-theoretic proof of $\mathsf{coNP} \subseteq \mathsf{NP}/O(1) \Rightarrow \mathsf{PH} \subseteq \mathsf{BH}$ (cf. [3,9]).

Combining Theorems 6, 7, and 8 we can now state the optimality of the Karp-Lipton collapse $\mathsf{PH} \subseteq \mathsf{BH}$ in VPV.

Corollary 9. *The theory VPV proves $\mathsf{NP} \subseteq \mathsf{P}/poly$ if and only if VPV proves that the polynomial hierarchy collapses to the Boolean hierarchy.*

The backward direction of this result can also be obtained in a less direct way using a recent result of Jeřábek [13]. The argument goes as follows:[1] by results of Zambella [21], $PV \vdash \mathsf{PH} = \mathsf{BH}$ implies $PV = S_2$. The latter, however, implies $PV \vdash \mathsf{NP} \subseteq \mathsf{P}/poly$ by a result of Jeřábek [13].

5 Propositional Proof Systems with Advice

Cook and Krajíček [9] defined propositional proof systems with advice, both in the functional and in the relational setting for proof systems. For both models, different concepts of proof systems with advice arise that not only differ in the amount of advice, but also in the way the advice is used by the proof system.

Our general model of computation for functional proof systems with advice is a Turing transducer with several tapes: an input tape containing the proof, possibly several work tapes for the computation of the machine, an output tape

[1] We are grateful to an anonymous referee for supplying this alternative argument.

where we output the proven formula, and an advice tape containing the advice. We start with a quite general definition for functional proof systems with advice which subsumes the definitions given in [9].

Definition 10. *Let $k : \mathbb{N} \to \mathbb{N}$ be a function on natural numbers. A* general functional propositional proof system with k bits of advice*, abbreviated general fpps/k, consists of two functions f and ℓ such that*

1. *$\ell : \Sigma^* \to \{1^n \mid n \geq 0\}$ is computable in polynomial time.*
2. *$f : \Sigma^* \to$ TAUT is a surjective polynomial-time computable function which on input π uses $k(|\ell(\pi)|)$ bits of advice depending only on $|\ell(\pi)|$.*

Let us give some explanation for this definition. For each length n there is a unique advice string of length $k(n)$. Which of these strings is used at a particular computation of f is determined by the function ℓ which computes from the input π the relevant advice length. In the functional definition of propositional proof systems, there are two natural options for this function ℓ: the advice may depend on the length of the input (i.e. the proof) or the length of the output (i.e. the proven formula).

Definition 11. *Let (f, ℓ) be a general fpps/k using advice function $k(n)$.*

1. *We say that f has* input advice *if for all inputs π we have $\ell(\pi) = 1^{|\pi|}$, i.e., the proof system f uses $k(|\pi|)$ bits of advice.*
2. *f has* output advice *if for all inputs π, the length of the output $f(\pi)$ does not depend on the advice (i.e., the content of the advice tape) and we have $\ell(\pi) = 1^{|f(\pi)|}$, i.e., the proof system f uses $k(|f(\pi)|)$ bits of advice.*

We remark that Cook and Krajíček [9] defined a more restrictive concept of proof systems with output advice, which they called length-determined functional proof systems.

The notions of (p-)simulations and (p-)optimality are easily generalized to proof systems with advice. For p-simulations we will use polynomial-time computable functions without advice (unless stated otherwise). We say that a proof system f is (p-)optimal for some class $\mathcal{F}$ of advice systems if f (p-)simulates every system in $\mathcal{F}$ and $f \in \mathcal{F}$.

In the next proposition we observe that *fpps/k* with input advice are already as strong as any general *fpps/k* (Definition 10).

Proposition 12. *Let $k : \mathbb{N} \to \mathbb{N}$ be a monotone function and let (f, ℓ) be a general fpps/k with advice function k. Then there exists a functional proof system f' with k bits of input advice such that f and f' are p-equivalent.*

In the relational setting for propositional proof systems, advice can be easily implemented as follows:

Definition 13 (Cook, Krajíček [9]). *A* propositional proof system with $k(n)$ bits of advice*, abbreviated pps/k, is a relation P such that for all $x \in \Sigma^*$ we have $x \in$ TAUT if and only if $(\exists y)P(y, x)$, and P is can be decided by a polynomial-time (in $|x| + |y|$) algorithm which uses $k(|x|)$ bits of advice.*

It is easy to see that, as in the classical case without advice, relational proof systems with advice and functional proof systems with output advice are two formulations of the same concept:

Proposition 14. *Let $k : \mathbb{N} \to \mathbb{N}$ be a function. Then every fpps/k with output advice is p-equivalent to some pps/k. Conversely, every pps/k is p-equivalent to an fpps/k with output advice.*

As in the classical theorem of Cook and Reckhow [11], we get the following equivalence:

Theorem 15. *Let k be any function. Then there exists a polynomially bounded fpps/k with output advice if and only if $\mathsf{coNP} \subseteq \mathsf{NP}/k$.*

6 Optimal Proof Systems with Advice

In this section we will investigate the question whether there exist optimal or p-optimal propositional proof systems with advice. A strong positive result was shown by Cook and Krajíček [9].

Theorem 16 (Cook, Krajíček [9]). *There exists a functional propositional proof system P with one bit of input advice which p-simulates all functional propositional proof systems with $k(n)$ bits of input advice for $k(n) = O(\log n)$. The p-simulation is computed by a polynomial-time algorithm using $k(n)$ bits of advice.*

In terms of simulations rather than p-simulations this result yields:

Corollary 17. *The class of all general fpps/$O(\log n)$ contains an optimal functional proof system with one bit of input advice.*

In the next definition we single out a large class of natural advice functions with at least logarithmic growth rate.

Definition 18. *A function k is* polynomially monotone *if k is computable in polynomial time and there exists a polynomial p, such that for each $x, y \in \Sigma^*$, $|y| \geq p(|x|)$ implies $|k(y)| > |k(x)|$.*

Polylogarithmic functions and polynomials are examples for polynomially monotone functions. If we consider proof systems with polynomially monotone advice functions, then we obtain p-optimal proof systems within each such class. This is the content of the next theorem which we prove by the same technique as was used for Theorem 16.

Theorem 19. *Let $k(n)$ be a polynomially monotone function. Then the class of all general fpps/k contains a p-optimal proof system.*

Proof. Let k be a function as above. Since k is polynomially monotone we can find a polynomial-time computable function $\ell : \Sigma^* \to 1^*$ such that for each $x \in \Sigma^*$ we have $k(|\ell(x)|) \geq k(|x|)+1$. Let $\|\cdot\|$ be an encoding of deterministic Turing

transducers by natural numbers. Without loss of generality we may assume that every machine M has running time $|x|^{\|M\|}$. Further, we need a polynomial-time computable function $\langle \cdot, \cdot, \cdot \rangle$ mapping triples of $\mathbb{N}$ bijectively to $\mathbb{N}$.

We will define a functional proof system (P, ℓ) using advice function k, which is p-optimal for the class of all general *fpps*$/k$. Let Q be a system from the class of all general *fpps*$/k$. By Proposition 12 we may assume that Q has input advice. First we will define a polynomial-time computable function f_Q translating Q-proofs into P-proofs and then we will describe how P works. We set $f_Q(\pi) = \pi 1^m$ where m is determined from the equation $m + |\pi| = \langle |\pi|, \|Q\|, |\pi|^{\|Q\|} \rangle$.

Now we define the system P: upon input x we first compute the unique numbers m_1, m_2, m_3 such that $|x| = \langle m_1, m_2, m_3 \rangle$. Let $\pi = x_1 \dots x_{m_1}$ be the first m_1 bits of x. Then we determine the machine Q from the encoding $m_2 = \|Q\|$. By the construction of ℓ, the system P receives at least one more bit of advice than Q. We can therefore use the first advice bit of P to certify that Q is indeed a correct propositional proof system when it is supplied with the last $k(|\pi|)$ advice bits of P. Therefore, if the first advice bit of P is 1, P simulates Q on input π for m_3 steps, where it passes the last $k(|\pi|)$ advice bits of P to Q. Otherwise, if the first advice bit of P is 0, P outputs $\top$. Apparently, P is correct and p-simulates every *fpps*$/k$ Q with input advice via the polynomial-time computable function f_Q. Thus, by Proposition 12, P also p-simulates every general *fpps*$/k$. □

In a similar way we get:

Proposition 20. *For each constant $k \geq 0$ there exists an fpps with $k+1$ bits of input advice that p-simulates every fpps with k bits of input advice.*

Proof. (Sketch) The proof uses the same construction as in the proof of Theorem 19 with the following difference in the usage of advice: the last k advice bits of the new *fpps*$/(k+1)$ P are the advice bits for the machine Q which we simulate, if the first of the $k+1$ advice bits certifies that Q is correct, i.e., it only produces tautologies. □

Regarding the two previous results there remains the question whether we also have a p-optimal system within the class of all general *fpps*$/k$ for constant k. Going back to the proof of Proposition 20, we observe that the proof system with $k+1$ advice bits, which simulates each with k bits, does not really need the full power of these $k+1$ bits, but in fact only needs $2^k + 1$ different advice strings. Assuming the existence of a p-optimal proof system for SAT_2 (the canonical complete problem for Σ_2^{p}), we can manage to reduce the amount of the necessary advice to exactly k bits, thus obtaining a p-optimal system within the class of all general *fpps*$/k$.

Theorem 21. *Assume that there exists a p-optimal proof system for* SAT_2. *Then for each constant $k \geq 1$ the class of all general fpps$/k$ contains a p-optimal proof system.*

Proof. Similarly as in Sadowski's characterization of the existence of p-optimal propositional proof systems [20], we can prove:

There exists a p-optimal proof system for SAT_2 *if and only if there exists a recursive enumeration* M_i, $i \in \mathbb{N}$, *of deterministic polynomial-time Turing machines such that*

1. *for every* $i \in \mathbb{N}$ *we have* $L(M_i) \subseteq \mathrm{SAT}_2$ *and*
2. *for every polynomial-time decidable subset* $L \subseteq \mathrm{SAT}_2$ *there exists an index* i *such that* $L \subseteq L(M_i)$.

Assume now that M_i is an enumeration of the easy subsets of SAT_2 as above. For every proof system Q with k bits of input advice we construct a sequence of propositional formulas

$$Prf^Q_{m,n,k}(\pi, \varphi, a) \ ,$$

asserting that the computation of Q at input π of length m leads to the output φ of length n under the k advice bits of a. We also choose a propositional formula $Taut_n(\varphi)$ stating that the formula encoded by φ is a propositional tautology. As Q is an *fpps*$/k$, the formulas

$$Correct^Q_{m,n,k} = (\exists a)(\forall \pi, \varphi)\left(Prf^Q_{m,n,k}(\pi, \varphi, a) \rightarrow Taut_n(\varphi)\right)$$

are quantified Boolean formulas from SAT_2 for every $n, m \geq 0$. Because these formulas can be constructed in polynomial time from Q, there exists an index $i \in \mathbb{N}$ such that M_i accepts the set $\{Correct^Q_{m,n,k} \mid m, n \geq 0\}$.

Now we construct a p-optimal system P with k bits of input advice as follows: at input x we compute the unique numbers $m_1, \ldots, m_4$ such that $|x| = \langle m_1, \ldots, m_4 \rangle$. As in the proof of Theorem 19, we set $\pi = x_1 \ldots x_{m_1}$ and $\|Q\| = m_2$. The system P then simulates $Q(\pi)$ with its own k advice bits for m_3 steps. If the simulation does not terminate, then P outputs $\top$. Otherwise, let φ be the output of this simulation. But before also P can output φ, we have to check the correctness of Q for the respective input and output length. To do this, P simulates the machine M_{m_4} on input $Correct^Q_{m_1,|\varphi|,k}$. If M_{m_4} accepts, then we output φ, and $\top$ otherwise.

The advice which P receives is the correct advice for Q, in case that M_{m_4} certifies that such advice indeed exists. Therefore P is a correct *fpps*$/k$. To show the p-optimality of P, let Q be an *fpps*$/k$ with input advice and let M_i be the machine accepting $\{Correct^Q_{m,n,k} \mid m, n \geq 0\}$. Then the system Q is p-simulated by P via the mapping $\pi \mapsto \pi 1^m$ where $m = \langle |\pi|, \|Q\|, |\pi|^{\|Q\|}, i \rangle - |\pi|$. □

All the optimal and p-optimal proof systems that we have so far constructed were using input advice. It is a natural question whether we can improve these constructions to obtain proof systems with output advice that still have the same optimality conditions. Our next result shows that this is rather unlikely, as otherwise collapse assumptions of presumably different strength would be equivalent. This result indicates that input advice for propositional proof systems is indeed a more powerful concept than output advice.

Theorem 22. *Let* $k \geq 1$ *be a constant and assume that there exists an fpps*$/k$ *with output advice that simulates every fpps*$/1$. *Then the following conditions are equivalent:*

1. *The polynomial hierarchy collapses to* BH_{2^k}.
2. *The polynomial hierarchy collapses to* BH.
3. $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n)$.
4. $\mathsf{coNP} \subseteq \mathsf{NP}/k$.

Proof. The equivalence of 1 and 4 was shown by Buhrman, Chang, and Fortnow (Theorem 3), and clearly, item 1 implies item 2. It therefore remains to prove the implications $2 \Rightarrow 3$ and $3 \Rightarrow 4$.

For the implication $2 \Rightarrow 3$, let us assume $\mathsf{PH} \subseteq \mathsf{BH}$. We choose a Σ_2^{p}-complete problem L, which by assumption is contained in $\mathsf{BH}_{k'}$ for some number k'. By Theorem 3 this implies $\mathsf{coNP} \subseteq \mathsf{NP}/k'$ and hence $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n)$.

For the final implication $3 \Rightarrow 4$, we assume $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n)$. By Theorem 15 this guarantees the existence of a polynomially bounded system P with $O(\log n)$ bits of output advice. By Theorem 16, P is simulated by a proof system P' with only one bit of input advice. Hence also P' is polynomially bounded. Now we use the hypothesis of the existence of a functional proof system Q with k bits of output advice which simulates all *fpps*/1. In particular, $P' \leq Q$ and therefore Q is a polynomially bounded *fpps*/k with output advice. Using again Theorem 15 we obtain $\mathsf{coNP} \subseteq \mathsf{NP}/k$. □

With respect to the optimal proof system from Corollary 17 we obtain:

Corollary 23. *The optimal fpps*/1 *from Corollary 17 is not equivalent to an fpps*/1 *with output advice, unless* $\mathsf{PH} \subseteq \mathsf{BH}$ *implies* $\mathsf{PH} \subseteq \mathsf{D}^{\mathsf{p}}$.

7 Classical Proof Systems with Advice

Let us now outline how one can define classical proof systems that use advice. A priori it is not clear how systems like resolution or Frege can sensibly use advice, but a canonical way to implement advice into them is to enhance these systems by further axioms which can be decided in polynomial time with advice. Cook and Krajíček [9] have defined the notion of extended Frege systems using advice. We give a more general definition.

Definition 24. *Let* Φ *be a set of tautologies that can be decided in polynomial time with* $k(n)$ *bits of advice. We define the system* $EF + \Phi/k$ *as follows. An* $EF+\Phi/k$*-proof of a formula* φ *is an EF-proof of an implication* $\psi \to \varphi$*, where* ψ *is a simple substitution instance of a formula from* Φ *(where simple substitutions only replace some of the variables by constants).*

If π is an $EF+\Phi/k$-proof of a formula φ, then the advice used for the verification of π neither depends on $|\pi|$ nor on $|\varphi|$, but on the length of the substitution instance ψ from Φ, which is used in π. As $|\psi|$ can be easily determined from π, $EF + \Phi/k$ are systems of the type *fpps*/k (in fact, this was the motivation for our general Definition 10).

If we require that the length of ψ in the implication $\psi \to \varphi$ is determined by the length of the proven formula φ, then the advice only depends on the output

and hence we get an *fpps*/k with output advice. This is the case for a collection of extensions of *EF* defined by Cook and Krajíček [9], which are motivated by the proof of Theorem 8. Cook and Krajíček proved that these systems are polynomially bounded if *VPV* proves $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$.

Our next result shows that the optimal proof systems constructed in Sect. 6 are equivalent to natural extensions of extended Frege systems with advice.

Theorem 25. *1. Let $k(n)$ be a polynomially monotone function. Then there exists a set $\Phi \in \mathsf{P}/k(n)$ such that $EF + \Phi/k$ is p-optimal for the class of all general fpps/$k(n)$.*
2. For every constant $k \geq 1$ there exists a set $\Phi \in \mathsf{P}/k$ such that $EF + \Phi/k$ p-simulates every general fpps/$k - 1$.
3. In contrast, none of the extensions of EF as defined in [9] simulates every general fpps/1, unless items 1 to 4 from Theorem 22 are equivalent.

Comparing the definition of *EF* with advice from [9] with our Definition 24, we remark that both definitions are parametrized by a set of tautologies Φ, and hence they both lead to a whole class of proof systems rather than *the* extended Frege system with advice. The drawback of our Definition 24 is, that even in the base case, where no advice is used, we do not get *EF*, but again all extensions $EF + \Phi$ with polynomial-time computable $\Phi \subseteq$ TAUT. It is known that each advice-free propositional proof system is p-simulated by such an extension of *EF* [17]. In contrast, Cook and Krajíček's extended Frege systems with advice lead exactly to *EF*, if no advice is used. On the other hand, these systems appear to be strictly weaker than the systems from Definition 24, as indicated by item 3 of Theorem 25.

8 Discussion and Open Problems

In this paper we have shown that $\mathsf{PH} \subseteq \mathsf{BH}$ is the optimal Karp-Lipton collapse within the theory *PV*. It remains as an open problem whether also $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}[O(\log n)]}$ and $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}}$ are optimal within S_2^1 and S_2^2, respectively (cf. [9]). For S_2^1 this corresponds to the problem whether $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n)$ is equivalent to $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}[O(\log n)]}$. Buhrman, Chang, and Fortnow [3] conjecture $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n) \iff \mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}}$ (cf. also [12]). This seems unlikely, as Cook and Krajíček [9] noted that $\mathsf{coNP} \subseteq \mathsf{NP}/O(\log n)$ implies $\mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}[O(\log n)]}$. However, it does not seem possible to extend the technique from [3] to prove the converse implication. Is even $\mathsf{coNP} \subseteq \mathsf{NP}/\mathit{poly} \iff \mathsf{PH} \subseteq \mathsf{P}^{\mathsf{NP}}$ true, possibly with the stronger hypothesis that both inclusions are provable in S_2^2? Currently, $\mathsf{coNP} \subseteq \mathsf{NP}/\mathit{poly}$ is only known to imply $\mathsf{PH} \subseteq \mathsf{S}_2^{\mathsf{NP}}$ [5].

With respect to the proof systems with advice we remark that all advice information we have used for our optimal systems in Sects. 6 and 7 can be decided in coNP. It would be interesting to know whether we can obtain stronger proof systems by using more complicated advice.

Acknowledgements

We are grateful to Jan Krajíček and the anonymous referees for helpful comments and detailed suggestions on how to improve this paper.

References

1. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural Complexity I. Springer, Heidelberg (1988)
2. Beigel, R.: Bounded queries to SAT and the Boolean hierarchy. Theoretical Computer Science 84, 199–223 (1991)
3. Buhrman, H., Chang, R., Fortnow, L.: One bit of advice. In: Proc. 20th Symposium on Theoretical Aspects of Computer Science, pp. 547–558 (2003)
4. Cai, J.-Y.: $S_2^p \subseteq ZPP^{NP}$. Journal of Computer and System Sciences 73(1), 25–35 (2007)
5. Cai, J.-Y., Chakaravarthy, V.T., Hemaspaandra, L.A., Ogihara, M.: Competing provers yield improved Karp-Lipton collapse results. Information and Computation 198(1), 1–23 (2005)
6. Chang, R., Kadin, J.: The Boolean hierarchy and the polynomial hierarchy: A closer connection. SIAM Journal on Computing 25(2), 340–354 (1996)
7. Cook, S.A.: Feasibly constructive proofs and the propositional calculus. In: Proc. 7th Annual ACM Symposium on Theory of Computing, pp. 83–97 (1975)
8. Cook, S.A.: Theories for complexity classes and their propositional translations. In: Krajíček, J. (ed.) Complexity of Computations and Proofs, pp. 175–227. Quaderni di Matematica(2005)
9. Cook, S.A., Krajíček, J.: Consequences of the provability of $NP \subseteq P/poly$. The Journal of Symbolic Logic 72(4), 1353–1371 (2007)
10. Cook, S.A., Nguyen, P.: Foundations of proof complexity: Bounded arithmetic and propositional translations (Book in progress), `http://www.cs.toronto.edu/~sacook`
11. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44, 36–50 (1979)
12. Fortnow, L., Klivans, A.R.: NP with small advice. In: Proc. 20th Annual IEEE Conference on Computational Complexity, pp. 228–234 (2005)
13. Jeřábek, E.: Approximate counting by hashing in bounded arithmetic (preprint, 2007)
14. Kadin, J.: The polynomial time hierarchy collapses if the Boolean hierarchy collapses. SIAM Journal on Computing 17(6), 1263–1282 (1988)
15. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 302–309. ACM Press, New York (1980)
16. Köbler, J., Watanabe, O.: New collapse consequences of NP having small circuits. SIAM Journal on Computing 28(1), 311–324 (1998)
17. Krajíček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Encyclopedia of Mathematics and Its Applications, vol. 60. Cambridge University Press, Cambridge (1995)
18. Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. The Journal of Symbolic Logic 54, 1063–1079 (1989)

19. Krajíček, J., Pudlák, P., Takeuti, G.: Bounded arithmetic and the polynomial hierarchy. Annals of Pure and Applied Logic 52, 143–153 (1991)
20. Sadowski, Z.: On an optimal propositional proof system and the structure of easy subsets of TAUT. Theoretical Computer Science 288(1), 181–193 (2002)
21. Zambella, D.: Notes on polynomially bounded arithmetic. The Journal of Symbolic Logic 61(3), 942–966 (1996)

A Calculus of Realizers for EM_1 Arithmetic (Extended Abstract)

Stefano Berardi and Ugo de'Liguoro

Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149 Torino, Italy
stefano@di.unito.it, deliguoro@di.unito.it

Abstract. We propose a realizability interpretation of a system for quantifier free arithmetic which is equivalent to the fragment of classical arithmetic without nested quantifiers, which we call EM_1-arithmetic. We interpret classical proofs as interactive learning strategies, namely as processes going through several stages of knowledge and learning by interacting with the "environment" and with each other. With respect to known constructive interpretations of classical arithmetic, the present one differs under many respects: for instance, the interpretation is compositional in a strict sense; in particular the interpretation of (the analogous of) the cut rule is the plain composition of functionals. As an additional remark, any two quantifier-free formulas provably equivalent in classical arithmetic have the same realizer.

1 Introduction

We propose a new notion of realizability (see e.g. [16] vol. I p. 195 for an introduction), in which the classical principle EM_1 (which is Excluded Middle restricted to Σ^0_1 formulas, see [1]) is treated by means of realizers that depend on certain growing pieces of knowledge, obtained by trial and error. In our approach the witness hidden in a proof of a Σ^0_1 statement (with parameters) can be computed in the limit (see [6,4,8]), and in this sense it is "learnt". The essential difference with Gold's idea is that the realizer embodies a learning strategy which is the actual content of the proof, and which is often an ingenuous method.

The first step of the construction is the introduction of an oracle χ_{P} (a predicate symbol) and of a Skolem function φ_{P} relative to each primitive recursive predicate P, in such a way that $\exists y.\, \mathsf{P}(x,y) \Leftrightarrow \chi_{\mathsf{P}}(x) \Leftrightarrow \mathsf{P}(x, \varphi_{\mathsf{P}}(x))$. The existence of these oracles intuitionistically implies EM_1, and also that atomic formulas of our system represent 1-quantifier formulas of arithmetic. Then we introduce a set $\mathbb{S}$ of the states of knowledge, which are finite sequences recording finitely many values of χ_{P} and φ_{P}, and ordered by prefix (by "increasing knowledge"). A formula containing such χ_{P} and φ_{P} is not decidable in general, however it can be evaluated w.r.t. a finite state s of knowledge, by assigning dummy values to all values of χ_{P} and φ_{P} which are unknown. A formula is valid if for any state s of knowledge we can effectively find some $s' \geq s$ in which the formula is true.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 215–229, 2008.
© Springer-Verlag Berlin Heidelberg 2008

The soundness property we expect from the semantic interpretation states that all derivable formulas are valid.

More precisely, since the meaning of a predicate or a term might depend on a state of knowledge $s \in \mathbb{S}$, numbers and truth values are lifted to *numbers* and *truth values* indexed over $\mathbb{S}$. We see ordinary numbers and truth values as limits, and we ask that an indexed object is convergent in every sequence $s_0 \leq s_1 \leq s_2 \leq \ldots s_n \leq \ldots$ weakly increasing w.r.t. prefix. Using the state we provide an effective semantics for all terms of a simply typed λ-calculus closed under primitive recursion, and extended by the non-effective maps χ_{P} and φ_{P}. This λ-calculus represents the terms and the atomic formulas of our fragment of arithmetic.

A tentative definition for a realizer of an atomic formula is that of a mapping sending any state s into some extension $s' \geq s$ in which the predicate takes the meaning "true" for all $s'' \geq s'$. The key problem here is that the realizer has to be effective, while there are no uniform and effective means to decide when the value of a term or the truth value of a formula has became stable.

The next idea is that the correct state can be learned. The outcome of the realizer is not a state, rather a process such that, as soon as the realizer becomes aware that something has gone wrong, so that the predicate is not true any more, it is able to extend the present state of knowledge looking for a larger one, where the predicate becomes true anew. Since we classically know that the truth value of the predicate eventually stabilizes, it will be eventually true forever, even if we shall never be able to say when and where.

There is still something missing here: we want a compositional interpretation of proofs, validating logical laws like the modus ponens and the cut rule. This rises the issue of the interaction of different parts of a proof, namely of how to compose two or more realizers. To solve this problem realizers use *continuations*: the process extending the states of knowledge in order to make some predicate true is passed along in the composition of realizers, and used by them as the last step in the computation of the new state. In this way, at the price of having realizers of type two, the combination of the realizers of several premises of an inference rule is (pointwise) composition, which is unsensible to the order: this does not mean, of course, that say $F \circ G$ and $G \circ F$ compute the same value, rather that any choice will be a realizer of the conclusion.

The paper is organized as follows. In §2 we introduce a version of EM_1-Arithmetic without nested quantifiers. Terms are expressed in a simply typed λ-calculus with primitive recursion of level 1. All formulas are quantifier-free, while formulas with non-nested quantifiers are represented through oracles $\chi_{\mathsf{P}}(\boldsymbol{x})$. EM_1 (Excluded Middle for Σ^0_1-formulas) is the following axiom schema:

$$(\mathsf{EM}_1) \qquad \forall \boldsymbol{x}(\exists y\, P(\boldsymbol{x}, y) \vee \forall y\, \neg P(\boldsymbol{x}, y))$$

where $P(\boldsymbol{x}, y)$ is a primitive recursive predicate. It is represented in our formalism without quantifiers through the oracle constants χ_{P}. In section §3 we lift the standard interpretation of numbers, boolean and functions to indexed numbers, booleans and functions introducing the notion of synchronous and convergent

functional. §4 is the core of the paper: we introduce a new notion of realizers for classical proofs, representing the construction hidden in the classical principle EM_1. All other constructions are represented by terms of the system. In the full version of the paper [2] we test our notion of realizer against the example of the Minimum Principle.

Related Works. A primary source of the present research is Coquand's semantics of evidence for classical arithmetic [3], where the role of realizers is taken by the strategies, the state of knowledge is the state of a play, and computation is the interaction of strategies through a dialogue.

The idea of lifting to truth values and numbers depending from a state and converging (in the sense of stabilization) w.r.t. this state comes from Gold's recursiveness in the limit [6] and Hayashi's Limit Computable Mathematics [8]. Our main contribution is to frame these ideas in the longstanding tradition of realizability interpretation of constructive logic.

The investigation of the computational content of classical proofs via continuations is well known and widely documented in the literature. It is impossible to provide a reasonably complete list of the numerous contributions to this topic; see e.g. [5,10] for some basic ideas behind the use of continuations in the interpretation of classical principles; continuations and CPS translation have been used to extend the formula as types paradigm to classical logic in [7,13]; these ideas are found also in the μ-calculus of [14] and in related systems. Our improvement is the compositional property of our approach in the strict sense of functional composition of the realizers, which allows for a clean reading of the use of continuations as mappings that "force" the stabilization of predicates and terms.

As pointed out to us by an anonymous referee, our work reminds Hilbert's ε-calculus (see [9,11,12]). Indeed our φ_{P} is just $\varepsilon_y(\mathsf{P})$ with the proviso that P is quantifier free (and primitive recursive). The essential difference lays in the way we produce the "solving substitution" [11], again similar to our states of knowledge: while in Hilbert's approach one interprets formulas and looks for the substitution of a numeral by blind search (exactly as with Gold's limits and with iterated limits in [15,4]), we interpret proofs into realizers, embodying (possibly cleaver) search strategies.

2 EM_1 Arithmetic of Primitive Recursive Functions

Let Type be the set of simple types with atoms Nat and Bool and the arrow as type constructor. As usual external parentheses will be omitted and the arrow associates to the right: $T_0 \Rightarrow T_1 \Rightarrow T_2$ reads as $T_0 \Rightarrow (T_1 \Rightarrow T_2)$; we also write $T^k \Rightarrow T'$ for $T \Rightarrow \cdots \Rightarrow T \Rightarrow T'$ with k occurrences of T to the left of T'. The symbol $\equiv$ is used for syntactical identity.

Definition 1 (Term Languages $\mathcal{L}_0$ and $\mathcal{L}_1$). *Let $\mathcal{L}_0$ be the language of simply typed λ-calculus with types in* Type*, whose constants are:*

- **zero, successor:** $0 : \mathsf{Nat}$, $\mathsf{succ} : \mathsf{Nat} \Rightarrow \mathsf{Nat}$, **equality:** $\mathsf{eq} : \mathsf{Nat}^2 \Rightarrow \mathsf{Bool}$, **booleans:** $\mathsf{true}, \mathsf{false} : \mathsf{Bool}$

– **if-then-else:** $\mathsf{if}_T : \mathsf{Bool} \Rightarrow T \Rightarrow T \Rightarrow T$, *where either* $T \equiv \mathsf{Nat}$ *or* $T \equiv \mathsf{Bool}$
– **primitive recursion:** $\mathsf{PR} : \mathsf{Nat} \Rightarrow (\mathsf{Nat}^2 \Rightarrow \mathsf{Nat}) \Rightarrow \mathsf{Nat} \Rightarrow \mathsf{Nat}$.

The language $\mathcal{L}_1$ *is obtained by adding to* $\mathcal{L}_0$ *a pair of constants* $\varphi_{\mathsf{P}} : \mathsf{Nat}^k \Rightarrow \mathsf{Nat}$ *and* $\chi_{\mathsf{P}} : \mathsf{Nat}^k \Rightarrow \mathsf{Bool}$ *for each closed term* $\mathsf{P} : \mathsf{Nat}^{k+1} \Rightarrow \mathsf{Bool}$ *of* $\mathcal{L}_0$*, and then closing under term formation rules.*

Term application associates to the left: $M\,N\,P$ reads as $(M\,N)P$; the abstraction $\lambda x^T.\,M$ will be written $\lambda x.\,M$ when T is clear from the context. We abbreviate $\mathsf{n} \equiv \mathsf{succ}^n\,\mathsf{0}$ (n-times applications of succ to $\mathsf{0}$), which is the numeral for $n \in \mathbb{N}$. We write $M[x_1, \ldots, x_n]$ to mean that $FV(M) \subseteq \{x_1, \ldots, x_n\}$, and $M[N_1, \ldots, N_n]$ for $M[N_1/x_1 \ldots, N_n/x_n]$, that is the result of the simultaneous substitution of x_i by N_i for all i (which are supposed to be of the same type), avoiding variable clashes.

Definition 2 (Equational Theory for $\mathcal{L}_0$). *The theory* $\mathcal{T}_0$ *is the equational theory of terms in* $\mathcal{L}_0$ *whose formulas are typed equations* $M = N : T$ *with both* M *and* N *of type* T*. Axioms and inference rules of* $\mathcal{T}_0$ *are the axioms of equality,* β *and* η *from the* λ*-calculus, plus:*

– $\mathsf{eq}\,\mathsf{0}\,\mathsf{0} = \mathsf{true}$, $\mathsf{eq}\,(\mathsf{succ}\,x)\,\mathsf{0} = \mathsf{false}$, $\mathsf{eq}\,\mathsf{0}\,(\mathsf{succ}\,x) = \mathsf{false}$, $\mathsf{eq}\,(\mathsf{succ}\,x)(\mathsf{succ}\,y) = \mathsf{eq}\,x\,y : \mathsf{Bool}$,
– $\mathsf{if}_T\,\mathsf{true}\,M\,N = M : T$, $\mathsf{if}_T\,\mathsf{false}\,M\,N = N : T$
– $\mathsf{PR}\,M\,N\,\mathsf{0} = M : \mathsf{Nat}$, $\mathsf{PR}\,M\,N\,(\mathsf{succ}\,x) = N\,x(\mathsf{PR}\,M\,N\,x) : \mathsf{Nat}$.

By $\mathcal{T}_0 \vdash M = N : T$ *we mean that the equation* $M = N : T$ *is derivable in* $\mathcal{T}_0$*.*

We explicitly exclude the function symbols $\varphi_{\mathsf{P}}, \chi_{\mathsf{P}}$ from $\mathcal{T}_0$: they will denote non-computable maps. The primitive recursor to define $k+1$-ary functions: $\mathsf{PR}_k : (\mathsf{Nat}^k \Rightarrow \mathsf{Nat}) \Rightarrow (\mathsf{Nat}^{k+2} \Rightarrow \mathsf{Nat}) \Rightarrow \mathsf{Nat}^{k+1} \Rightarrow \mathsf{Nat}$ is definable from the unary PR by: $\mathsf{PR}_k \equiv \lambda g\,h\,x_1\,\ldots x_k.\,\mathsf{PR}(g\,x_1\,\ldots x_k)(h x_1\,\ldots x_k)$.

Although $\mathcal{T}_0$ is an equational theory, it is the theory of the convertibility relation associated to a notion of reduction which is confluent and strongly normalizing. In particular it is decidable whether $\mathcal{T}_0 \vdash M = N : T$. Indeed $\mathcal{T}_0$ is a fragment of Gödel system **T**, where the essential limitation consists in the restriction of the recursor PR whose functional arguments are of type one. By this the presence of abstraction of variables at any type has no effect w.r.t. function definability. A k-ary function over natural numbers f is *definable* in $\mathcal{T}_0$ if there exists a closed term (a combinator) $\mathsf{f} : \mathsf{Nat}^k \Rightarrow \mathsf{Nat} \in \mathcal{L}_0$ such that for all $n_1, \ldots, n_k, m \in \mathbb{N}$, $\mathcal{T}_0 \vdash \mathsf{f}\,\mathsf{n}_1 \cdots \mathsf{n}_k = \mathsf{m} : \mathsf{Nat}$ if and only if $f(n_1, \ldots, n_k) = m$.

Proposition 1. *The number theoretic functions definable in* $\mathcal{T}_0$ *are exactly the primitive recursive functions (in particular, are computable).*

We find useful having in $\mathcal{T}_0$ an operator for (weighted) total recursion. Let *ifz* be the primitive recursive function such that $ifz(0, n) = n$ and $ifz(m+1, n) = 0$. For any $w : \mathbb{N} \to \mathbb{N}$, let us abbreviate by $x \prec_w y$ the (primitive recursive) function giving 0 if $w(x) < w(y)$, 1 otherwise. We say that the function $f : \mathbb{N} \to \mathbb{N}$ is

defined by *weighted well-founded recursion* in terms of the functions $w : \mathbb{N} \to \mathbb{N}$ (weight), $g : \mathbb{N}^2 \to \mathbb{N}$, $h_1 : \mathbb{N} \to \mathbb{N}$ if:

$$f(n) = g(n, \mathit{ifz}(h_1(n) \prec_w n, f(h_1(n))))$$

This definition schema generalizes to any list of maps $h_1, \ldots, h_k : \mathbb{N} \to \mathbb{N}$, and to any list $\boldsymbol{m} = m_i, \ldots, m_h$ of parameters. Weighted total recursion can be expressed in $\mathcal{T}_0$ by a combinator WR (see [2]).

We will now define a formal theory PRA-$\exists$ of arithmetic in the formalism of $\mathcal{L}_1$. We rephrase some basic notions of logic. *Terms*, ranged over by $\mathsf{t}, \mathsf{r}, \ldots$ (possibly with primes and indexes) are terms of type Nat in $\mathcal{L}_1$ with free variables of type Nat, and *formulas*, ranged over by P, $\mathsf{Q}, \mathsf{R}, \ldots$ (possibly with primes and indexes) are terms in $\mathcal{L}_1$ of type Bool, with free variables of type Nat. The atomic formulas of the shape $\mathsf{eq\,t\,r}$ and $\mathsf{lt\,t\,r}$ are written $\mathsf{t} = \mathsf{r}$ and $\mathsf{t} < \mathsf{r}$ respectively. Connectives are definable in $\mathcal{L}_1$, and we write them in the usual way; in particular we use the ordinary infix notation for binary connectives. A *propositional formula* is a term $E[z_1, \ldots, z_n]$ of type Bool built out of variables $\boldsymbol{z}$ of type Bool and connectives (hence it is in $\mathcal{L}_0$); a propositional formula E is *tautological consequence* of $E_1, \ldots, E_k$ if any instantiation of the boolean variables by true or false in the implication $(E_1 \wedge \cdots \wedge E_k) \Longrightarrow E$ is provably equal to true in $\mathcal{T}_0$.

The theory PRA-$\exists$ defined below is formally quantifier free: as a matter of fact the meaning of χ_{P} and φ_{P} induced by the axioms (χ) and (φ) is that of the oracle and of a Skolem function for the predicate $\exists y.\, \mathsf{P}[\boldsymbol{x}, y]$ where P is in $\mathcal{L}_0$ (hence primitive recursive). We stress that we cannot have Skolem functions in P: this limitation accounts for the fact that PRA-$\exists$ is 1-quantifier arithmetic, and not the entire arithmetic.

Definition 3. PRA-$\exists$ *is the theory whose theorems are the formulas derivable by the following axioms and rules:*

– **Post rules:**

$$\frac{\mathsf{P}_1 \quad \cdots \quad \mathsf{P}_k}{\mathsf{Q}}\,\mathsf{Post}$$

consisting of the axioms of equality; an axiom for each equation $\mathsf{t} = \mathsf{r} : \mathsf{Nat}$ *derivable in* $\mathcal{T}_0$*; all rules with* $E[z_1, \ldots, z_n]$ *tautological consequence of* $E_1[z_1, \ldots, z_n], \ldots, E_k[z_1, \ldots, z_n]$*:*

$$\frac{}{\mathsf{t} = \mathsf{r}}\,\mathcal{T}_0 \qquad \frac{E_1[\mathsf{P}_1, \ldots, \mathsf{P}_n] \quad E_k[\mathsf{P}_1, \ldots, \mathsf{P}_n]}{E[\mathsf{P}_1, \ldots, \mathsf{P}_n]}\,\mathsf{Taut}$$

– **Skolem Axioms:** *for each formula* $\mathsf{P}[\boldsymbol{x}, y]$ *in* $\mathcal{L}_0$*, the axioms:*

$$\frac{}{\mathsf{P}[\boldsymbol{x}, y] \Longrightarrow \chi_{\mathsf{P}}\,\boldsymbol{x}}\,\chi \qquad \frac{}{\chi_{\mathsf{P}}\,\boldsymbol{x} \Longrightarrow \mathsf{P}[\boldsymbol{x}, \varphi_{\mathsf{P}}\boldsymbol{x}]}\,\varphi$$

– **Well Founded Induction:**

$$\frac{\mathsf{ifz}(\mathsf{t}_1\, x \prec_{\mathsf{w}} x)\mathsf{P}[\boldsymbol{z}, \mathsf{t}_1\, x] \wedge \cdots \wedge \mathsf{ifz}(\mathsf{t}_k\, x \prec_{\mathsf{w}} x)\mathsf{P}[\boldsymbol{z}, \mathsf{t}_k\, x] \Longrightarrow \mathsf{P}[\boldsymbol{z}, x]}{\mathsf{P}[\boldsymbol{z}, \mathsf{r}\,]}\,\mathsf{WF\ Ind}$$

where x is not free in the conclusion, and $\mathsf{w}, \mathsf{t}_1, \ldots, \mathsf{t}_k : \mathsf{Nat} \Rightarrow \mathsf{Nat}$, $\mathsf{r} : \mathsf{Nat}$ *and* $\mathsf{t}_1\, x \prec_{\mathsf{w}} x \equiv \mathsf{w}(\mathsf{t}_1\, x) < \mathsf{w}\, x$.

The theory PRA-$\exists$, when restricted to the language $\mathcal{L}_0$ is a variant of system PRA in [16] for primitive recursive arithmetic.

3 States of Knowledge, Synchronous and Convergent Functionals

In this section we introduce the notion of state of knowledge, then we define hereditary synchronous functionals, whose computation all takes place in the same state of knowledge, then hereditary convergent functionals. These latter will be used to interpret terms of $\mathcal{L}_1$.

Let $\perp$ denote a divergent computation, and define $A_\perp = A \cup \{\perp\}$ for any set of values A. An element $a \in A_\perp$ is *total* if $a \neq \perp$; a map $\tau : A_\perp \to B_\perp$ is *total* if τ sends total elements into total ones. If $f : A^n \to B$, we extend f to $f_\perp : A_\perp^n \to B_\perp$ by $f_\perp(\boldsymbol{a}) = \perp$ if $a_i = \perp$ for some i, and $f_\perp(\boldsymbol{a}) = f(\boldsymbol{a})$ otherwise.

Definition 4 (States of Knowledge). *A* state of knowledge*, shortly a* state*, is a finite list of triples $\langle \mathsf{P}, \boldsymbol{n}, m\rangle$ (with possibly different P, $\boldsymbol{n}$ and m) such that $\mathsf{P} : \mathsf{Nat}^{k+1} \to \mathsf{Bool}$ is a predicate of $\mathcal{L}_0$ and $\boldsymbol{n} = n_1, \ldots, n_k \in \mathbb{N}$, $m \in \mathbb{N}$, and $\mathcal{T}_0 \vdash \mathsf{P}[\boldsymbol{n}, \mathsf{m}] = \mathsf{true}$. We call the empty list $\langle\rangle$ the initial state. Let $\mathbb{S}$ denote the set of states, partially ordered by the prefix ordering $\leq$.*

A triple $\langle \mathsf{P}, \boldsymbol{n}, m\rangle$ stays for the equations $\chi_{\mathsf{P}}(\boldsymbol{n}) = \mathsf{true}$ and $\varphi_{\mathsf{P}}(\boldsymbol{n}) = m$. A state represents a finite set of such equations, the initial state is the empty set. We now define synchronous maps F as mapping of indexed objects such that both the argument and the value of F are evaluated at the same state s. Let us write $\lambda\_.a$ for the function constantly equal to a.

Definition 5 (Synchronous Functions). *Let A and B be any sets:*

1. *$F : (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ is* synchronous *if $F(\tau, s) = F(\lambda\_.\tau(s), s)$ for all τ and s;*
2. *given $f : A \to B$ the* synchronous extension *$f^\dagger : (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ of f is defined by $f^\dagger(\tau, s) = f(\tau(s))$.*

For any τ and s we have $f^\dagger(\tau, s) = f(\tau(s)) = f^\dagger(\lambda\_.\tau(s), s)$, hence $f^\dagger$ is synchronous. We will now interpret terms and formulas of $\mathcal{L}_0$ as synchronous and convergent functions. To this aim, we have to extend the notion of synchronicity to higher types.

We recall that an embedding-projection pair $(\epsilon, \pi) : X < Y$ (e-p pair for short) of X into Y is a pair of mappings $\epsilon : X \to Y$ and $\pi : Y \to X$ s.t. $\pi \circ \epsilon = \mathsf{Id}_X$, where composition will be also written as $\pi\epsilon$. The composition, $\epsilon\pi : Y \to Y$, called *retraction*, is idempotent so that $\epsilon(X)$ (the image of X under ϵ) is the set of fixed points of $\epsilon\pi$, and $\pi : \epsilon(X) \to X$ is bijective.

Definition 6 (Synchronous Retraction). *For any sets A and B we define the mappings:*

$$\epsilon_{A,B} : (\mathbb{S}_\perp \to (A \to B)) \to ((\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B))$$
$$\pi_{A,B} : ((\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)) \to (\mathbb{S}_\perp \to (A \to B))$$

by

$$\epsilon_{A,B}(\alpha, \tau, s) = \alpha(s, \tau(s)) \quad \textit{and} \quad \pi_{A,B}(F, s, a) = F(\lambda\_.a, s)$$

where $\alpha : \mathbb{S}_\perp \to (A \to B)$, $\tau : \mathbb{S}_\perp \to A$, $s \in \mathbb{S}_\perp$, $F : (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ and $a \in A$.

The name of *synchronous retraction* is justified by the fact that the fixed points of the retraction are maps $: (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ bijective to the elements of $\mathbb{S}_\perp \to (A \to B)$, therefore are maps depending on a single state. They coincide with the synchronous maps:

Lemma 1. *If $\epsilon_{A,B}$ and $\pi_{A,B}$ are as in Definition 6 then:*

1. *$(\epsilon_{A,B}, \pi_{A,B})$ is an e-p pair;*
2. *$F : (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ is synchronous if and only if $F \in \epsilon_{A,B}(\mathbb{S}_\perp \to (A \to B))$.*
3. *If $f : A \to B$, the* synchronous extension *$f^\dagger : (\mathbb{S}_\perp \to A) \to (\mathbb{S}_\perp \to B)$ of f is equal to $\epsilon_{A,B}(\lambda\_.f)$.*

The next step is to state some well known properties of embedding-projection pairs, and can be rephrased by stating that they are closed under composition and indeed they do form a category; moreover since the category of sets is cartesian closed, then the category of embedding-projection pairs over sets is such. Observe that retractions are covariant w.r.t. the arrow, which is also known as their characteristic property.

Lemma 2

1. *If $(\epsilon, \pi) : A < B$ and $(\epsilon', \pi') : B < C$ are e-p pairs, then: $(\epsilon'\epsilon, \pi\pi') : A < C$ is such;*
2. *if $(\epsilon_1, \pi_1) : A_1 < B_1$ and $(\epsilon_2, \pi_2) : A_2 < B_2$ are e-p pairs, then $(\epsilon_1, \pi_1) \to (\epsilon_2, \pi_2) : A_1 \to A_2 < B_1 \to B_2$ is such where $(\epsilon_1, \pi_1) \to (\epsilon_2, \pi_2)$ is the pair $(\lambda f.\, \epsilon_2 \circ f \circ \pi_1, \lambda g.\, \pi_2 \circ g \circ \epsilon_1)$.*

Let St be a new ground type; then we define the interpretation $[\![T]\!]$ of the simple type T (of the extended type language) set theoretically by: $[\![\mathsf{Nat}]\!] = \mathbb{N}_\perp$, $[\![\mathsf{Bool}]\!] = \mathbb{B}_\perp$, $[\![\mathsf{St}]\!] = \mathbb{S}_\perp$, and $[\![T \Rightarrow T']\!] = [\![T]\!] \to [\![T']\!]$, namely the full function space. By T^{St} we denote the result of replacing in T each occurrence of Nat by $\mathsf{St} \Rightarrow \mathsf{Nat}$ and of Bool by $\mathsf{St} \Rightarrow \mathsf{Bool}$ respectively.

Lemma 3. *For each type T there is an e-p pair $(\epsilon_T, \pi_T) : \mathbb{S}_\perp \to [\![T]\!] < [\![T^{\mathsf{St}}]\!]$.*

We are now in place to characterize those functionals of $[\![T^{\mathsf{St}}]\!]$ depending on a single state, i.e. in the image of $\mathbb{S}_\perp \to [\![T]\!]$, as fixed points of the retraction $(\epsilon_T, \pi_T) : \mathbb{S}_\perp \to [\![T]\!] < [\![T^{\mathsf{St}}]\!]$. The elements $a \in [\![T]\!]$ can be raised to elements of $[\![T^{\mathsf{St}}]\!]$ by applying ϵ_T to $\lambda\_.a$.

Definition 7 (Hereditarily Synchronous Functionals). *Let T be any type:*

1. *$f \in [\![T^{\mathsf{St}}]\!]$ is* hereditarily synchronous*, or* h. sync. *for short, if $f \in \epsilon_T(\mathbb{S}_\perp \to [\![T]\!])$;*
2. *the* canonical injection *$\_^o : [\![T]\!] \to [\![T^{\mathsf{St}}]\!]$ is defined: $f^o = \epsilon_T(\lambda\_.f)$. We also set $[\![T]\!]^o = \{f^o \mid f \in [\![T]\!]\}$.*

H. sync. functionals are closed under application. Indeed, if $f : \mathbb{S}_\perp \to [\![T_1 \Rightarrow T_2]\!]$ then $\epsilon_{T_1 \Rightarrow T_2}(f) = \epsilon_{T_2} \circ \epsilon_{[T_1],[T_2]}(f) \circ \pi_{T_1}$, hence for any $a \in [\![T_1^{\mathsf{St}}]\!]$ we have $\epsilon_{T_1 \Rightarrow T_2}(f)(a) = b$ for some $b \in \epsilon_{T_2}(\mathbb{S}_\perp \to [\![T_2]\!]) \subseteq [\![T_2^{\mathsf{St}}]\!]$. It follows that, if $g \in [\![(T_1 \Rightarrow T_2)^{\mathsf{St}}]\!]$ is hereditarily synchronous, then $g(a)$ is such for any $a \in [\![T_1^{\mathsf{St}}]\!]$.

If $f \in [\![T]\!]$ then $f^o \in [\![T^{\mathsf{St}}]\!]$ is a map ignoring its input state s, but forcing its argument to use s as the only state. Indeed, by unraveling definitions we obtain:

$$f^o(\tau_1, \ldots, \tau_n, s) = f(\pi_{T_1}(\tau_1)(s), \ldots, \pi_{T_n}(\tau_n)(s)).$$

We can now describe h.sync. functionals by an equation. Let $T = T_1, \ldots, T_n \Rightarrow o$ for either $o = \mathsf{Nat}$ or $o = \mathsf{Bool}$, and consider $F \in [\![T^{\mathsf{St}}]\!]$ and $\tau_i \in [\![T_i^{\mathsf{St}}]\!]$ for $i = 1, \ldots, n$. By unraveling definitions, F is hereditarily synchronous if and only if for all $s \in \mathbb{S}_\perp$

$$F(\tau_1, \ldots, \tau_n, s) = F((\pi_{T_1}(\tau_1)(s))^o, \ldots, (\pi_{T_n}(\tau_n)(s))^o, s).$$

The operator $\_^o$ applied to the arguments of F forces them to reject their input state, and to use only the same input state s of F. This implies that the behavior of any functional over $[\![T^{\mathsf{St}}]\!]$ is fully determined by its behavior over $[\![T]\!]^o$.

Proposition 2. *Let $F, G \in [\![(T_1, \ldots, T_n \Rightarrow U)^{\mathsf{St}}]\!]$, then $F = G$ if and only if $F(a_1^o, \ldots, a_n^o) = G(a_1^o, \ldots, a_n^o)$ for all $a_1 \in [\![T_1]\!], \ldots, a_n \in [\![T_n]\!]$.*

In the sequel we denote by $\{s_i\}_{i<\omega}^{\leq}$ a weakly increasing chain of states $s_0 \leq s_1 \leq s_2 \leq \cdots$.

Definition 8 (Convergence). *Let A be either $\mathbb{B}$ or $\mathbb{N}$. A function $\tau : \mathbb{S}_\perp \to A_\perp$* converges*, written $\tau\Downarrow$, if*

$$\forall \{s_i\}_{i<\omega}^{\leq}\ \exists j\ \forall k \geq j.\ \tau(s_j) = \tau(s_k) \neq \perp.$$

The concept in Definition 8 is a classical one; we can weaken this concept to an intuitionistic notion of convergence: $\tau\Downarrow$ iff for every *recursive* $\{s_i\}_{i<\omega}^{\leq}$ there exists $i < \omega$ such that $\tau(s_i) = \tau(s_{i+1})$ and are both total objects of A. For the sake of simplicity we use a classical meta-theory. Note also that two limits taken along different sequences $s_0 \leq s_1 \leq s_2 \leq \ldots$ can be different: this expresses the fact that our interpretation is non-deterministic.

Lemma 4. *Let A be either $\mathbb{B}$ or $\mathbb{N}$:*

1. *let $a, b : \mathbb{S}_\perp \to A_\perp$, and define $\langle a, b\rangle \in \mathbb{S}_\perp \to A_\perp \times A_\perp$ by $\langle a, b\rangle(s) = (a(s), b(s))$: if both $a\Downarrow$ and $b\Downarrow$ then $\langle a, b\rangle\Downarrow$;*

2. *if* $f : (\mathbb{S}_\perp \to A_\perp) \to (\mathbb{S}_\perp \to A_\perp)$ *is h. sync. such that* $f(a)\Downarrow$ *for all constant* a*, then* $f(b)\Downarrow$ *for all convergent* b*;*
3. *if* $f : (\mathbb{S}_\perp \to A_\perp)^k \to (\mathbb{S}_\perp \to A_\perp)$ *is h. sync. that yields* $f(\boldsymbol{\tau})\Downarrow$ *for any* k*-tuple of constant* $\boldsymbol{\tau} \in (\mathbb{S}_\perp \to A_\perp)^k$*, then* $f(\boldsymbol{\sigma})\Downarrow$ *for any* k*-tuple of convergent* $\boldsymbol{\sigma} \in (\mathbb{S}_\perp \to A_\perp)^k$*.*

We hereditarily extend the notion of total element to all finite types. By this we define for each type T a PER $\sim_T$ such that two functionals are related if and only if they are h. sync. and hereditarily convergent, sending related arguments into related values.

Definition 9 (Hereditary Convergence and Equivalence). *For any type* T *we define simultaneously a predicate* $\Downarrow^T \subseteq [\![T^{\mathsf{St}}]\!]$ *of* hereditarily convergent *objects of type* T*, and a partial equivalence relation* $\sim_T$ *over* $[\![T^{\mathsf{St}}]\!]$ *as follows:*

- *if either* $T \equiv \mathsf{Bool}$ *or* $T \equiv \mathsf{Nat}$ *then* $\tau \Downarrow^T$ *iff* $\tau\Downarrow$*;* $\tau \sim_T \tau'$ *if and for any* $\{s_i\}_{i<\omega}^{\leq}$ *they are definitely equal over it, that is:* $\exists i\, \forall j \geq i.\, \tau(s_j) = \tau'(s_j) \neq \perp$*.*
- *If* $T \equiv T_1 \Rightarrow T_2$ *then* $f \Downarrow^T$ *iff* f *is h. sync and* $f(a)\Downarrow^{T_2}$ *for all* a *such that* $a\Downarrow^{T_1}$*;* $f \sim_T g$ *iff* $f\Downarrow^T$*,* $g\Downarrow^T$ *and* $f(a) \sim_{T_2} g(b)$ *if* $a \sim_{T_1} b$*.*

Lemma 5. *The mapping* $\_^o$ *is functorial w.r.t. the quotient of* $[\![T^{\mathsf{St}}]\!]$ *under* $\sim_T$*:* $\mathsf{Id}^o_T \sim \mathsf{Id}_{T^{\mathsf{St}}}$*; moreover* $f(a_1, \ldots, a_n)^o = f^o(a_1^o, \ldots, a_1^o)$*, and* $(f \circ g)^o = f^o \circ g^o$*.*

We can now interpret each term $M : T$ of $\mathcal{L}_1$ into our non standard model by an element of $[\![T^{\mathsf{St}}]\!]$. If $f : \mathbb{N}^k \to \mathbb{N}$ then $f_\perp$ denotes the strict extension of f to $\mathbb{N}_\perp$, and similarly for boolean functions. The values of $[\![\chi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s)$ and $[\![\chi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s)$ are determined by the information stored in the state s, if any is available, and are a dummy value otherwise.

Definition 10 (Term Interpretation for $\mathcal{L}_1$)
An environment *is a map* ρ *sending any variable* $x : T$ *into an element* $\rho(x) \in [\![T^{\mathsf{St}}]\!]$*. The term interpretation map* $[\![M]\!]_\rho$ *for* $M \in \mathcal{L}_1$ *is defined:*

- $[\![x]\!]_\rho = \rho(x)$*;* $[\![\mathsf{c}]\!]_\rho = c^o_\perp$ *where* c *is any constant among* $0, \mathsf{succ}, \mathsf{eq}, \mathsf{true}, \mathsf{false}, \mathsf{if}, \mathsf{PR}$ *and* c *is its standard interpretation in* $[\![T]\!]$*;* $[\![MN]\!]_\rho = [\![M]\!]_\rho[\![N]\!]_\rho$*;* $[\![\lambda x^T.M]\!]_\rho = \lambda a \in [\![T^{\mathsf{St}}]\!].[\![M]\!]_{\rho'}$*, where* $\rho'(x) = a$*,* $\rho'(y) = \rho(y)$ *if* $y \not\equiv x$*;*
- $[\![\chi_{\mathsf{P}}]\!]_\rho : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to (\mathbb{S}_\perp \to \mathbb{B}_\perp)$*, where* $\mathsf{P} : \mathsf{Nat}^{k+1} \Rightarrow \mathsf{Bool}$*, is such that for all* $\boldsymbol{\tau} \in (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k$ *and* $s \in \mathbb{S}_\perp$*,* $[\![\chi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s) = \perp$ *if either* $\tau_i(s) = \perp$ *for some* $\tau_i \in \boldsymbol{\tau}$ *or* $s = \perp$*; else* $[\![\chi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s) = tt$ *if there exists* m *such that* $\langle \mathsf{P}, \boldsymbol{\tau}(s), m\rangle \in s$*;* $[\![\chi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s) = ff$ *otherwise;*
- $[\![\varphi_{\mathsf{P}}]\!]_\rho : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to (\mathbb{S}_\perp \to \mathbb{N}_\perp)$*, where* $\mathsf{P} : \mathsf{Nat}^{k+1} \Rightarrow \mathsf{Bool}$*, is such that for all* $\boldsymbol{\tau} \in (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k$ *and* $s \in \mathbb{S}_\perp$*,* $[\![\varphi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s) = \perp$ *if either* $\tau_i(s) = \perp$ *for some* $\tau_i \in \boldsymbol{\tau}$ *or* $s = \perp$*; else* $[\![\varphi_{\mathsf{P}}]\!]_\rho(\boldsymbol{\tau}, s) = m$ *where* $\langle \mathsf{P}, \boldsymbol{\tau}(s), m\rangle \in s$ *is the first triple in* s *whose first entries are* P *and* $\boldsymbol{\tau}(s)$*, if any;* $m = 0$ *else.*

The definition above has, as a consequence, the following interpretation for the derived combinator WR. If $f = [\![\mathsf{WR}]\!]_\rho(g, w, h) : (\mathbb{S}_\perp \to \mathbb{N}_\perp) \to (\mathbb{S}_\perp \to \mathbb{N}_\perp)$, then $f(\tau)$ is (using some terms of $\mathcal{L}_0$ in place of their interpretations for brevity):

$$g(\tau, [\![\mathsf{ifz}]\!]([\![\mathsf{lt}]\!]\,(w(h(\tau)), w(\tau)), f(h(\tau))))$$

An environment ρ is convergent, written $\rho\Downarrow$, if $\rho(x)\Downarrow^T$ for all $x : T$. We then summarize the main properties of the construction developed so far.

Theorem 1 (Soundness of the Interpretation). *Let $M, N : T$ be any terms of $\mathcal{L}_1$:*

1. *$[\![M]\!]_\rho \in [\![T^{\mathsf{St}}]\!]$ for any ρ, and if $\rho\Downarrow$ then $[\![M]\!]_\rho\Downarrow^T$;*
2. *if $M \in \mathcal{L}_0$ then $[\![M]\!]_\rho \in [\![T]\!]^o$ for any ρ such that $\rho(x) \in [\![T']\!]^o$ for all $x : T'$; furthermore $[\![M]\!]_\rho$ is computable;*
3. *$[\![M]\!]_\rho$ is hereditarily convergent if all $\rho(x)$ are such;*
4. *if $\mathcal{T}_0 \vdash M = N$ (hence both $M, N \in \mathcal{L}_0$), then $[\![M]\!]_\rho \sim [\![N]\!]_\rho$;*
5. *if $\vdash \mathsf{P}$ is derivable in the theory PRA-$\exists$ then $[\![P]\!]_\rho \sim_{\mathsf{Bool}} tt^o$ for any convergent ρ.*

4 The Realizability Interpretation

If $\mathsf{P}[\mathsf{t}] \in \mathcal{L}_1$ is a closed predicate, then $[\![\mathsf{P}[\mathsf{t}]]\!]$ is a boolean depending on a state, i.e., a map $: \mathbb{S}_\perp \to \mathbb{B}_\perp$. If $\mathsf{P}[x] \in \mathcal{L}_0$ (i.e., if $\mathsf{P}[x]$ is a primitive recursive predicate) then $[\![\mathsf{P}[\mathsf{t}]]\!](s)$ is equal to the truth value of $\mathsf{P}[x]$ on $[\![\mathsf{t}]\!](s) \in \mathbb{N}$. Our goal is to extract from a proof Π_1 of a primitive recursive property $\mathsf{P}[\mathsf{t}] \in \mathcal{L}_0$ for a term $t \in \mathcal{L}_1$, some state $s \in \mathbb{S}_\perp$ such that $n = [\![\mathsf{t}]\!](s) \in \mathbb{N}$ satisfies $\mathsf{P}[x]$: in other words, we want to extract from the proof of $\mathsf{P}[\mathsf{t}]$ some witness n for $\mathsf{P}[x]$. This is by no means immediate: even in the case in which $\mathsf{P}[\mathsf{t}]$ is provable, we cannot guarantee that for all $s \in \mathbb{S}_\perp$ we have $[\![\mathsf{P}[\mathsf{t}]]\!](s) = tt$. We will show that we can turn any proof Π_1 of $\mathsf{P}[\mathsf{t}]$ into a *realizer* picking, given any $s \in \mathbb{S}_\perp$, some $s' \geq s$ (some extension of the state s in the prefix ordering) such that $[\![\mathsf{P}[\mathsf{t}]]\!](s') = tt$ (i.e., such that $\mathsf{P}[n']$ for $n' = [\![\mathsf{t}]\!](s')$). The realizer is the part of the constructive content of the classical proof Π_1 which is not included in the term t.

As a first approximation, the realizer of $\mathsf{P}[\mathsf{t}]$ associated to Π_1 could be some map $\kappa_1 : \mathbb{S}_\perp \to \mathbb{S}_\perp$ such that for all $s \in \mathbb{S}_\perp$, if $s' = \kappa_1(s)$ then $s' \geq s$ and $[\![\mathsf{P}[\mathsf{t}]]\!](s') = tt$. κ_1, however, is not enough when Π_1 is included in some proof-context $\Pi_2[\Pi_1]$ and when $\Pi_2[.]$ corresponds to some other construction $\kappa_2 : \mathbb{S}_\perp \to \mathbb{S}_\perp$. In this case the construction associated to the whole proof $\Pi_2[\Pi_1]$ would be $s'' = \kappa_2(\kappa_1(s)) \geq s$, and since s'' is not a value of κ_1 we cannot guarantee that $[\![\mathsf{P}[\mathsf{t}]]\!](s'') = tt$. We overcome this problem by requiring that the realizer associated to Π_1 is some F taking a state s and a map κ_2 associated to some proof context $\Pi_2[.]$ including Π_1. We ask that F extends s to some $F(\kappa_2, s) \geq s$ in which the conclusions of both Π_1 and $\Pi_2[.]$ are true. The map κ_2 is used by F as a *continuation*, that is, as a function representing the part of the program to be executed after F. Two further constraints on realizers (see the definition below) are: κ_2 is applied as the last step of the computation of

F (i.e., $F(\kappa_2, s) = \kappa_2(s''')$ for some $s''' \geq s$), and κ_2 is applied by F only to extensions of the original state s.

We interpret a realizer $F : (\mathbb{S}_\perp \to \mathbb{S}_\perp) \to (\mathbb{S}_\perp \to \mathbb{S}_\perp)$ as a "process". We think of any $\kappa : \mathbb{S}_\perp \to \mathbb{S}_\perp$ such that $\kappa(s) \geq s$ as *the combined action of all processes outside* F. Remark that the type of a set of processes is *a subtype of the type of processes*: in this way we can represent, within the simply typed lambda calculus, the fact that a process can interact with a set of processes. $\mathtt{id} = \mathtt{id}_{\mathbb{S}_\perp}$ represents the empty action made by the empty set of processes. $F(\mathtt{id}, \langle\rangle)$ is the canonical evaluation of a process, w.r.t. an empty set of other processes and the initial state. If κ represents a set $\{F_1, \ldots, F_n\}$ of processes, and F is a process, then $F(\kappa)$ represents the set of processes $\{F, F_1, \ldots, F_n\}$. The compound process whose components are $\{F_1, \ldots, F_n\}$ is $F_1 \circ \ldots \circ F_n$. We think of the composition of realizers as an arbitrary sequentialization of the parallel composition of "processes".

Definition 11 (Realizer sets and Realizers). *Let us abbreviate* $\mathtt{St} = (\mathbb{S}_\perp \to \mathbb{S}_\perp) \to (\mathbb{S}_\perp \to \mathbb{S}_\perp)$.

1. *A* realizer set *is a total function* $\kappa : \mathbb{S}_\perp \to \mathbb{S}_\perp$ *such that for all* $s \in \mathbb{S}$, $s \leq \kappa(s)$*;*
2. *a* realizer *is any* $F \in \mathtt{St}$ *such that*
   (a) $F(\kappa) = \kappa \circ F'(\kappa)$ *for some* $F' \in \mathtt{St}$ *sending realizer sets into realizer sets and*
   (b) *if* $s \neq \perp$ *and* $\kappa_1(s') = \kappa_2(s')$ *for all* $s' \geq s$*, then* $F(\kappa_1, s) = F(\kappa_2, s)$*;*
3. *a* realizing map *is a function* $\Phi \in (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$ *mapping indexed numbers realizers.*
4. *The* k*-ary pointwise identity is the realizer map* $I_k : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$ *defined by* $I_k(\boldsymbol{\tau}) = \mathtt{id} : \mathtt{St}$*, for all* $\boldsymbol{\tau} \in (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k$*.*

We call $\mathtt{Id} = \mathtt{id}_{\mathbb{S}_\perp \to \mathbb{S}_\perp}$ the trivial realizer *and* I_k the trivial realizer map.

We list below a few basic properties of realizers. For instance (Lemma 6.4) the composition of n realizers $F_1, \ldots, F_n$ is a realizer whose range is included in the range of all $F_1, \ldots, F_n$. This is not true for generic functionals, but it depends on the fact that the realizer set received as input is used as a continuation, and it will be crucial in order to prove the correctness of the realization interpretation.

Lemma 6. *1.* $\mathtt{id}$ *is a realizer set. If* κ_1, κ_2 *are realizer sets, then* $\kappa_1 \circ \kappa_2$ *is a realizer set.*
2. *If* κ *is a realizer set and* F *a realizer, then* $F(\kappa)$ *is a realizer set.*
3. $\mathtt{Id}$ *is a realizer.* I_k *is a realizer map. If* F, G *are realizers, then* $F \circ G$ *is such.*
4. *If* $F_1, \ldots, F_n$ *are realizers, then for all realizer set* κ *and all state* s *if* $s' = (F_1 \circ \ldots \circ F_n)(\kappa, s)$ *then for some realizer sets* $\kappa_1, \ldots, \kappa_n$ *and states* $s_1, \ldots, s_n$ *we have* $s' = F_1(\kappa_1, s_1) = \cdots = F_n(\kappa_n, s_n)$*.*

The central notion of this paper is that of a realization relation $F \models \tau$, where F is a realizer and $\tau : \mathbb{S}_\perp \to \mathbb{B}_\perp$ is an indexed truth value. The intended meaning is that F realizes τ if for any realizer set κ, $F(\kappa)$ sends any state s to some extension $s' = F(\kappa, s) \geq s$ in which τ is true (i.e. $\tau(s') = tt$).

Definition 12. *Let $F : \mathtt{St}$ be a realizer and $\tau : \mathbb{S}_\perp \to \mathbb{B}_\perp$ a convergent map.*

1. *$F \models \tau$ if for all realizer sets κ and all $s \in \mathbb{S}$ if $s' = F(\kappa, s)$ then $\tau(s') = tt$.*
2. *If P is a closed predicate of $\mathcal{L}_1$ and F a realizer, we say that $F \models P$ if $F \models [\![P]\!]$.*
3. *If P is a predicate of $\mathcal{L}_1$ with free variables $x_1, \ldots, x_k$ all of type Nat, and $\Phi : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$ a realizer map, then $\Phi \models P$ if for all vectors $\boldsymbol{\tau} : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k$ of convergent maps $\Phi(\boldsymbol{\tau}) \models [\![P]\!]_{[\boldsymbol{\tau}/\boldsymbol{x}]}$.*

We will now prove that each rule of our system can be interpreted by an operation sending realizers of the assumptions into realizers of the conclusion. Eventually we will define, by induction over the proofs, a map $\mathcal{R}(.)$ sending a proof Π of P into a realizer $\mathcal{R}(\Pi)$ of P. The realizer will express a part of the construction hidden in Π, the part which is not expressed by the subterms of P.

Let $\Phi, \Psi : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$, then the *pointwise composition* $\Phi \bullet \Psi : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$ is defined as $(\Phi \bullet \Psi)(\boldsymbol{\tau}) = \Phi(\boldsymbol{\tau}) \circ \Psi(\boldsymbol{\tau})$. We will now check that the pointwise composition of the realizers of the assumptions of a Post rule is the realizer of the conclusion of the same rule. The proof relies on the fact that the realizer of the first assumption of the rule uses as continuation the set of all realizers of the remaining assumptions of the rule.

Lemma 7. *Let $\mathsf{P}_1, \ldots, \mathsf{P}_k$ and Q be the premises and the conclusion of an instance of the Post scheme: for any $s \in \mathbb{S}_\perp$, if $[\![\mathsf{P}_1]\!]_\rho(s) = \cdots = [\![\mathsf{P}_k]\!]_\rho(s) = tt$, then $[\![\mathsf{Q}]\!]_\rho(s) = tt$, for any ρ.*

Proposition 3. *Suppose that in system* PRA-$\exists$ *there is a derivation ending by an instance of the Post rule schema:*

$$\frac{\mathsf{P}_1 \quad \cdots \quad \mathsf{P}_k}{\mathsf{Q}}\ \mathsf{Post}$$

and let $\Phi_1 \models \mathsf{P}_1, \ldots, \Phi_k \models \mathsf{P}_k$: then $\Phi_1 \bullet \ldots \bullet \Phi_k \models \mathsf{Q}$.

Proof. It suffices to prove the statement when $\Phi_1 = F_1, \ldots, \Phi_n = F_k$ are just realizers (so that in particular $F_1 \bullet \ldots \bullet F_k = F_1 \circ \ldots \circ F_k$). By Lemma 6.4, for all realizer set κ and state s if $s' = (F_1 \circ \ldots \circ F_k)(\kappa, s)$ then for some realizer sets $\kappa_1, \ldots, \kappa_k$ and some states $s_1, \ldots, s_k$ we have $s' = F(\kappa_1, s_1) = \cdots = F(\kappa_k, s_k)$. By assumption $[\![\mathsf{P}_1]\!](F_1(\kappa_1, s_1)) = \ldots = [\![\mathsf{P}_k]\!](F_k(\kappa_k, s_k)) = tt$ for all realizer sets $\kappa_1, \ldots, \kappa_k$ and states $s_1, \ldots, s_k$. We deduce that $[\![\mathsf{P}_1]\!](s') = \ldots = [\![\mathsf{P}_k]\!](s') = tt$, and by Lemma 7 that $[\![\mathsf{Q}]\!](s') = tt$. By definition of realization we conclude $F_1 \circ \ldots \circ F_k \models \mathsf{Q}$. ∎

Note that we can replace, in the proof above, the composition $F_1 \circ \ldots \circ F_k$ by any permutation of it, and we would obtain some (in general, *different*) realizer of the *same* conclusion Q (see the end of [2]). Our interpretation of this fact is that composition is an arbitrary sequentialization of a parallel composition between "processes" $F_1 \circ \ldots \circ F_k$. Note also that if the Post Rule is unary, then any realizer of the only assumption is a realizer of the conclusion.

For each instance of the φ-axiom and of the χ-axiom we can define two realizer maps. The k-ary pointwise identity (i.e., the trivial realizer) realizes the φ-axiom with k free variables. In fact, the φ-axiom hides no construction on states, because it is true in all states by the very interpretation of χ and φ.

Proposition 4. *Fix any instance* $Q \equiv \chi_{\mathsf{P}}(\boldsymbol{x}) \implies \mathsf{P}[\boldsymbol{x}, \varphi_{\mathsf{P}}(\boldsymbol{x})]$ *of the* φ*-axiom, with* $\boldsymbol{x} = x_1, \ldots, x_n : \mathsf{Nat}$. *Then* $I_k \models \mathsf{Q}$.

The crucial step is defining some realizer $\mathtt{r}_{\mathsf{P},k+1}$ of the χ-axiom for the $k+1$-ary primitive recursive predicate $\mathsf{P}[\boldsymbol{x}, \mathsf{y}] \in \mathcal{L}_0$ instantiated in some $\boldsymbol{n}, m : \mathbb{N}$. The interpretation of such an instance might be false in some state s. Indeed, we might have $[\![\chi]\!](\boldsymbol{n}^o, m^o)(s) = \mathit{ff}$ while $[\![\mathsf{P}]\!]_{[\boldsymbol{n}^o/\boldsymbol{x}, m^o/y]} = \mathit{tt}$. In this case we extend s to some state $s' > s$ in which the given instance of the χ-axiom is true. The realizer $\mathtt{r}_{\mathsf{P},k+1}$ defines s' by first adding the triple $\langle \mathsf{P}, \mathbf{n}, \mathsf{m} \rangle$ to s, then by applying the realizer set variable κ and obtaining some $s'' = \kappa(s') \geq s'$. By definition we have $[\![\chi]\!](\boldsymbol{n}^o, m^o)(s'') = \mathit{tt}$, therefore the χ-axiom for P and $\boldsymbol{n}, m : \mathbb{N}$ is true in s'. We interpret this step as an atomic "learning" step: the realizer learns one point in the graph of the map χ (and of the map φ, since they are closely related).

Sometimes one step of learning is not enough to validate the χ-axiom. We therefore define an operator Ω, raising $\mathtt{r}_{\mathsf{P},k+1}$ to some realizer $\mathtt{R}_{\mathsf{P},k+1}$ of the χ-axiom instantiated over $\boldsymbol{\tau}, \tau : [\![\mathsf{Nat}]\!]^{k+1}$, a vector of indexed integers. Ω repeatedly applies $\mathtt{r}_{\mathsf{P},k+1}(\boldsymbol{\tau}(s), \tau(s))$ in order to validate the given instance of the χ-axiom, because the values $\boldsymbol{\tau}(s), \tau(s)$ might change with the state (see the example at the end of §5). In the rest of the paper, when we write $\mathtt{let}\ (x = u)\ \mathtt{in}\ (t)$ we mean $(\lambda x.t)(u)$.

Definition 13 (Realizer of the χ-axiom). *Assume* $\boldsymbol{n} : \mathbb{N}^k, m : \mathbb{N}$ *and* $\kappa : \mathbb{S}_\perp \to \mathbb{S}_\perp$ *and* $s : \mathbb{S}_\perp$. *Let* $\mathsf{P} \in \mathcal{L}_0$ *be a primitive recursive predicate interpreting it. Then we define* $\mathtt{r}_{\mathsf{P},k+1} : {\mathbb{N}_\perp}^{k+1} \to \mathtt{St}$ *and* $\mathtt{R}_{\mathsf{P},k+1} : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^{k+1} \to \mathtt{St}$ *by:*

1. $\mathtt{r}_{\mathsf{P},k+1}(\boldsymbol{n}, m, \kappa, s) \equiv \mathsf{if}(\neg [\![\chi]\!](\boldsymbol{n}, s) \wedge [\![\mathsf{P}]\!]_{[\boldsymbol{n}^o, m^o/\boldsymbol{x}, y]}(s), \kappa(s @ \langle \mathsf{P}, \boldsymbol{n}, m \rangle), \kappa(s)) : \mathbb{S}$.
2. *Assume* $\Phi : {\mathbb{N}_\perp}^k \to \mathtt{St}$ *is a family of realizers indexed over* ${\mathbb{N}_\perp}^k$, *and* $\boldsymbol{\tau} : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k$ *and* $\kappa : (\mathbb{S}_\perp \to \mathbb{S}_\perp), s : \mathbb{S}_\perp$. *Then the realizer map* $\Omega(\Phi) : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^k \to \mathtt{St}$ *is defined by* $\Omega(\Phi, \boldsymbol{\tau})(\kappa, s) = \mathtt{let}\ (s' = \Phi(\boldsymbol{\tau}(s))(\kappa, s))\ \mathtt{in}\ (\mathtt{if}\ \boldsymbol{\tau}(s) = \boldsymbol{\tau}(s')\ \mathtt{then}\ s'\ \mathtt{else}\ \Omega(\Phi, \boldsymbol{\tau})(\kappa, s')) : \mathbb{S}_\perp$.
3. $\mathtt{R}_{\mathsf{P},k+1} \equiv \Omega(\mathtt{r}_{\mathsf{P},k+1}) : \mathtt{St}$.

The computation of $\Omega(\Phi, \boldsymbol{\tau})(\kappa, s)$ produces a weakly increasing sequence of states $s = s_0 \leq s_1 \leq s_2 \leq \cdots$ such that $s_{n+1} = \Phi(\boldsymbol{\tau}(s_n))(\kappa, s_n)$ for all n. If $\boldsymbol{\tau}$ is convergent, then $\boldsymbol{\tau}(s_{n+1}) = \boldsymbol{\tau}(s_n)$ for some n, and the computation terminates with output s_{n+1}. Ω defines in this way a realizer map for the χ-axiom.

Proposition 5 (Realizer of the χ-axiom). *Fix any instance* $\mathsf{Q} \equiv (\mathsf{P}[\boldsymbol{x}, x] \implies \chi_{\mathsf{P}}(\boldsymbol{x}))$ *of the* χ*-axiom, for the* $k+1$*-ary primitive recursive predicate* $\mathsf{P}[\boldsymbol{x}, \mathsf{y}] \in \mathcal{L}_0$. *Let* $\mathsf{R}[\boldsymbol{y}] \in \mathcal{L}_1$.

1. $\mathtt{r}_{\mathsf{P},k+1}(\boldsymbol{n}, n) \models [\![\mathsf{Q}]\!]_{[\boldsymbol{n}^o, n^o/\boldsymbol{x}, x]}$, *for all* $\boldsymbol{n} \in \mathbb{N}^k$ *and* $n \in \mathbb{N}$.
2. *If* $\Phi(\boldsymbol{m}) \models [\![\mathsf{R}]\!]_{[\boldsymbol{m}^o]}$ *for all* $\boldsymbol{m} \in \mathbb{N}^h$, *then* $\Omega(\Phi) \models \mathsf{R}$.
3. $\mathtt{R}_{\mathsf{P},k+1} \models \mathsf{Q}$.

We can also define a realizer $\mathtt{WF}$ mapping a realizer map Φ for the assumption of an induction rule with parameters $\mathsf{t}, \boldsymbol{t}$ into a realizer map $\mathtt{WF}(\Phi, [\![\mathsf{t}]\!], [\![\boldsymbol{t}]\!])$ for the conclusion of the induction rule. Apart from a single detail we will precise in a moment, $\mathtt{WF}$ is the usual realizer of the rule of well-founded induction.

Definition 14. *Let $C : \mathbb{S}_\perp \to \mathbb{B}_\perp$. For any realizer F we define a conditional realizer, which is F when C holds, and* $\mathtt{Id}$ *otherwise:* $\mathsf{if}(C, F)(\kappa, s) = \mathsf{if}(C(s), F(\kappa, s), \kappa(s))$.

Definition 15. *Assume $w, f_1, \ldots, f_k : [\![(\mathsf{Nat} \Rightarrow \mathsf{Nat})^{\mathsf{St}}]\!]$. Let $C_i : \mathbb{S}_\perp \to \mathbb{B}_\perp$ denote the condition $w(f_i(\tau)) < w(\tau)$, for $i = 1, \ldots, k$. If $C : \mathbb{S}_\perp \to \mathbb{B}_\perp$, then $C \wedge^o C_i$ is by definition $\lambda s.C(s) \wedge C_i(s)$. Let $\Phi : (\mathbb{S}_\perp \to \mathbb{N}_\perp) \to \mathtt{St}$. Then*

1. $\mathtt{WF}_C$ *is recursively defined as follows: abbreviate* $\mathtt{WF}_C(\Phi)(\tau) \equiv \mathtt{WF}_C(\Phi, w, \boldsymbol{f})(\tau)$, *we set*

$$\mathtt{WF}_C(\Phi)(\tau) = \mathsf{if}(C, \Phi(\tau) \circ \mathtt{WF}_{C \wedge^o C_1}(\Phi)(f_1(\tau)) \ \ldots \circ \mathtt{WF}_{C \wedge^o C_k}(\Phi)(f_k(\tau)))$$

2. *Assume $\Phi : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^{k+1} \to \mathtt{St}$ and $w, f_1, \ldots, f_k : [\![\mathsf{Nat}^{k+1} \to \mathsf{Nat}]\!]$. Then we define $\mathtt{WF}(\Phi, w, \boldsymbol{f})(\boldsymbol{\tau}) = \mathtt{WF}_T(\Phi(\boldsymbol{\tau}), w(\boldsymbol{\tau}), \boldsymbol{f}(\boldsymbol{\tau}))$, where the index T is the always true condition.*

Note the "guard" $\mathsf{if}(C, \ldots)$ in front of the definition $\mathtt{WF}_C = \ldots$. By definition unfolding, this means that whenever $C(s)$ is false in a state s, the realizer $\mathtt{WF}_C$ together with all its recursive calls trivialize to the identity. The reason for having these "guards" is that the clauses $C_i = w(f_i(\tau)) < w(\tau)$ on which the recursive calls depend may change their truth value from true to false, and whenever this happens the recursive call must disappear.

We can now state that $\mathtt{WF}$ produces a realizer map for the conclusion of the induction rule.

Lemma 8. *Assume $\mathsf{P} \in \mathcal{L}_1$ is a predicate and $\mathsf{t}, \mathsf{t_1} \ldots, \mathsf{t_k} : \mathsf{Nat} \to \mathsf{Nat} \in \mathcal{L}_1$ a, with $\boldsymbol{x} = x_1, \ldots, x_n$ and $\mathtt{FV}(\mathsf{P}) = \boldsymbol{x}, x$ and $\mathtt{FV}(t, \boldsymbol{t}) = \boldsymbol{x}$. Assume $\Phi : (\mathbb{S}_\perp \to \mathbb{N}_\perp)^{k+1} \to \mathtt{St}$ be a realizer map. Abbreviate $[\![t]\!] = \lambda\boldsymbol{\tau}.[\![t]\!]_{[\boldsymbol{\tau}/\boldsymbol{x}]}$ and $[\![\boldsymbol{t}]\!] = \lambda\boldsymbol{\tau}.[\![\boldsymbol{t}]\!]_{[\boldsymbol{\tau}/\boldsymbol{x}]}$. If Φ realizes the assumption of the induction rule for P:*

$$\Phi \models \mathsf{ifz}(\mathsf{t_1}(x) \prec_{\mathsf{t}} x)\mathsf{P}[\mathsf{t_1}(x)/x] \wedge \cdots \wedge \mathsf{ifz}(\mathsf{t_k}(x) \prec_{\mathsf{t}} x)\mathsf{P}[\mathsf{t_k}(x)/x] \Longrightarrow \mathsf{P}$$

then $\mathtt{WF}(\Phi, [\![t]\!], [\![\boldsymbol{t}]\!]) \models \mathsf{P}$.

By putting all together, we define by induction on Π a map $\mathcal{R}(\Pi)$ from proofs to realizer maps: for all details we refer to [2], §5, Def. 16.

Theorem 2 (Realizability). *Let $\Pi : x_1, \ldots, x_k \vdash P$ be a proof in system* PRA-$\exists$, *then $\mathcal{R}(\Pi) \models P$, that is for all convergent $\boldsymbol{\tau} \in [\![\mathsf{Nat}^{\mathsf{St}}]\!]$, realizer set κ and state $s \in \mathbb{S}_\perp$,*

$$[\![P]\!]_{[\boldsymbol{\tau}/\boldsymbol{x}]}(\mathcal{R}(\Pi)(\boldsymbol{\tau}, \kappa, s)) = tt.$$

The realizer we compute by Theorem 2 is the pointwise identity (i.e. it is trivial) whenever the proof Π uses no χ-axiom. Indeed, a realizer returns a state with enough information about χ, ϕ in order to make the conclusion of Π true, and all rules but the χ-rule require no information whatever about χ, ϕ in order to be true.

Acknowledgments. The paper has profited of the remarks of the anonymous referees.

References

1. Akama, Y., Berardi, S., Hayashi, S., Kohlenbach, U.: An arithmetical hierarchy of the law of excluded middle and related principles. In: Proc. of LICS 2004, pp. 192–201 (2004)
2. Berardi, S., de' Liguoro, U.: A Calculus of Realizers for EM_1 Arithmetic (Full Version). Technical report, Università di Torino (2008), `http://www.di.unito.it/~stefano/RealWFA.pdf`
3. Coquand, T.: A semantics of evidence for classical arithmetic. J. Symb. Log. 60, 325–337 (1995)
4. Criscuolo, G., Minicozzi, E., Trautteur, G.: Limiting recursion and the arithmetic hierarchy. ITA 9(1), 5–12 (1975)
5. Friedman, H.: Classically and intuitionistically provably recursive functions. In: Scott, D.S., Muller, G.H. (eds.) Higher Set Theory. LNM, vol. 699, pp. 21–28. Springer, Heidelberg (1978)
6. Gold, E.M.: Limiting recursion. J. Symb. Log. 30, 28–48 (1965)
7. Griffin, T.G.: The formulae-as-types notion of control. In: Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages, POPL 1990, San Francisco, CA, USA, January 17–19, pp. 47–57. ACM Press, New York (1990)
8. Hayashi, S.: Mathematics based on incremental learning, excluded middle and inductive inference. Theor. Comp. Sci. 350, 125–139 (2006)
9. Hilbert, D., Bernays, P.: Grundlagen der Mathematik, vol. II. Springer, Heidelberg (1970)
10. Krivine, J.-L.: Dependent choice, 'quote' and the clock. Theor. Comput. Sci. 308, 259–276 (2003)
11. Mints, G.: Strong termination for the epsilong substitution method. J. Symb. Log. 61(4), 1193–1205 (1996)
12. Moser, G., Zach, R.: The epsilon calculus and herbrand complexity. Studia Logica 82(1), 133–155 (2006)
13. Murthy, C.R.: An evaluation semantics for classical proofs. In: Proc. of LICS 1991, pp. 96–107 (1991)
14. Parigot, M.: Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In: LPAR, pp. 190–201 (1992)
15. Schubert, L.K.: Iterated limiting recursion and the program minimalization problem. J. of Ass. Comp. Mach. 21(3), 436–445 (1974)
16. van Dalen, D., Troelstra, A.: Constructivism in Mathematics, vol. I. North-Holland, Amsterdam (1988)

Quantitative Game Semantics for Linear Logic

Ugo Dal Lago[1,⋆] and Olivier Laurent[2,⋆⋆]

[1] Dipartimento di Scienze dell'Informazione
Università di Bologna
dallago@cs.unibo.it
[2] Preuves Programmes Systèmes
CNRS – Université Paris 7
Olivier.Laurent@pps.jussieu.fr

Abstract. We present a game-based semantic framework in which the time complexity of any IMELL proof can be read out of its interpretation. This gives a compositional view of the geometry of interaction framework introduced by the first author. In our model the time measure is given by means of slots, as introduced by Ghica in a recent paper. The cost associated to a strategy is polynomially related to the normalization time of the interpreted proof, in the style of a complexity-theoretical full abstraction result.

1 Introduction

Implicit computational complexity (ICC) is a very active research area lying at the intersection between mathematical logic, computational complexity and programming language theory. In the last years, a myriad of systems derived from mathematical logic (often through the Curry-Howard correspondence) and characterizing complexity classes (e.g. polynomial time, polynomial space or logarithmic space) has been proposed.

The techniques used to analyze ICC systems are usually ad-hoc and cannot be easily generalized to other (even similar) systems. Moreover, checking whether extending an existing ICC system with new constructs or new rules would break the correspondence with a given complexity class is usually not easy: soundness must be (re)proved from scratch. Take, for example, the case of subsystems of Girard's linear logic capturing complexity classes: there are at least three distinct subsystems of linear logic corresponding to polynomial time, namely bounded linear logic, light linear logic and soft linear logic. All of them can be obtained by properly restricting the rules governing the exponential connectives. Even if they have *not* been introduced independently, correspondence with polynomial time had to be reproved thrice. We need to understand *why* certain restrictions on the usual comonoidal exponential discipline in linear logic lead to characterizations of certain complexity classes.

⋆ Partially supported by the Marie Curie EIF grant "ASFIX" and by FIRB grant RBIN04M8S8, "Intern. Inst. for Applicable Math.".
⋆⋆ Partially supported by the French ANR "NO-CoST" project (JC05_43380).

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 230–245, 2008.
© Springer-Verlag Berlin Heidelberg 2008

This is the typical situation where semantics can be useful. And, indeed, some proposals for semantic frameworks into which some existing ICC systems can be interpreted have already appeared in the literature. Moreover, there are some proposals for semantic models in which the interpretation "reveals" quantitative, intensional, properties of proofs and programs. One of them [5] is due to the first author and is based on context semantics. There, the complexity of a proof is obtained by a global analysis of its interpretation as a set of paths.

In this paper, we show that the above mentioned context semantics can be put into a more interactive form by defining a game model for multiplicative and exponential linear logic and showing a quantitative correspondence between the interpretation of any proof and the time needed to normalize the proof itself. This correspondence can be thought of as a complexity-theoretic full-abstraction result, where proofs with the same complexity (rather than observationally equivalent proofs) are equated in the semantics.

Context semantics is a model of Girard's geometry of interaction. As a consequence, turning it into an AJM game model should not be difficult (at least in principle), due to the well-known strong correspondence between the two frameworks (see [3], for example). But there are at least two problems: first of all, the context semantics framework described in [5] is *slightly* different from the original one and, as such, it is not a model of the geometry of interaction. This is why we introduce a lifting construction in our game model.

Moreover, the global analysis needed in [5] to extract the complexity of a proof from its interpretation cannot be easily turned into a more interactive analysis, in the spirit of game semantics. The extraction of time bounds from proof interpretations is somehow internalized here through the notion of *time analyzer* (see Section 2). One of the key technical lemmas towards the quantitative correspondence cited above is proved through a game-theoretical reducibility argument (see Section 4).

Another semantic framework which has been designed with similar goals is Ghica's slot games [8]. There, however, the idea is playing slots in correspondence with any potential redex in a program, while here we focus on exponentials. On the other hand, the idea of using slots to capture intensional properties of proofs (or programs) in an interactive way is one of the key ingredients of this paper. In Section 5 the reader can find a more detailed comparison with Ghica's work. To keep the presentation simple, we preferred to adopt Ghica's way of introducing cost into games, rather than Leperchey's time monad.

In Baillot and Pedicini's geometry of interaction model [4], the "cost" of a proof is strongly related to the length of regular paths in its interpretation. But this way, one can easily define a family of terms which normalize in linear time but have exponential cost.

2 Syntax

We here introduce multiplicative exponential linear logic as a sequent calculus. It would be more natural to deal with proof-nets instead of the sequent calculus,

$$\frac{}{A \vdash A}\ A \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\varsigma(\Gamma, \Delta) \vdash B}\ U \qquad \frac{\Gamma \vdash B}{\varsigma(\Gamma, !A) \vdash B}\ W \qquad \frac{\Gamma, !A, !A \vdash B}{\varsigma(\Gamma, !A) \vdash B}\ C$$

$$\frac{\Gamma, A \vdash B}{\varsigma(\Gamma) \vdash A \multimap B}\ R_{\multimap} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\varsigma(\Gamma, \Delta, A \multimap B) \vdash C}\ L_{\multimap} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\varsigma(\Gamma, \Delta) \vdash A \otimes B}\ R_{\otimes} \qquad \frac{\Gamma, A, B \vdash C}{\varsigma(\Gamma, A \otimes B) \vdash C}\ L_{\otimes}$$

$$\frac{A_1, \ldots, A_n \vdash B}{\varsigma(!A_1, \ldots, !A_n) \vdash !B}\ P_! \qquad \frac{\Gamma, A \vdash B}{\varsigma(\Gamma, !A) \vdash B}\ D_! \qquad \frac{\Gamma, !!A \vdash B}{\varsigma(\Gamma, !A) \vdash B}\ N_!$$

Fig. 1. A sequent calculus for IMELL

but our semantic constructions will rely on a precise sequentiality in proof constructions that we would have to rebuild in a proof-net setting.

The language of *formulas* is defined by the following productions:

$$A ::= \alpha \mid A \multimap A \mid A \otimes A \mid !A$$

where α ranges over a countable set of *atoms*. A context is a sequence $\Gamma = A_1, \ldots, A_n$ of formulas. If $\Gamma = A_1, \ldots, A_n$ is a context and $\varsigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$ is a permutation, then $\varsigma(\Gamma)$ stands for the context $A_{\varsigma(1)}, \ldots, A_{\varsigma(n)}$.

The rules in Figure 1 define a sequent calculus for (intuitionistic) multiplicative and exponential linear logic, IMELL, with an exchange rule integrated in the other ones. Our presentation uses an explicit *digging* rule $N_!$ as often done in the geometry of interaction setting (see [9] for some comments).

Given any proof $\pi : \Gamma \vdash A$, we can build another (cut-free) proof $[\pi] : \Phi, \Gamma \vdash A$, where Φ is a sequence in the form $!^{k_1}(A_1 \multimap A_1), \ldots, !^{k_n}(A_n \multimap A_n)$. We say that "cuts are exposed" in $[\pi]$. It is defined as follows, by induction on the structure of π:

- If the last rule in π is not U and the immediate subproofs of π are $\rho_1, \ldots, \rho_n$, then $[\pi]$ is obtained from $[\rho_1], \ldots, [\rho_n]$ in the natural way. For a promotion rule, as an example, π and $[\pi]$ are given by:

$$\frac{\rho : A_1, \ldots, A_n \vdash B}{\varsigma(!A_1, \ldots, !A_n) \vdash !B}\ P_! \qquad\qquad \frac{[\rho] : \Phi, A_1, \ldots, A_n \vdash B}{!\Phi, \varsigma(!A_1, \ldots, !A_n) \vdash !B}\ P_!$$

- For a cut rule, π and $[\pi]$ are given by:

$$\frac{\rho : \Gamma \vdash B \quad \sigma : \Delta, B \vdash A}{\varsigma(\Gamma, \Delta) \vdash A}\ U \qquad\qquad \frac{[\rho] : \Phi, \Gamma \vdash B \quad [\sigma] : \Psi, \Delta, B \vdash A}{\Phi, B \multimap B, \Psi, \varsigma(\Gamma, \Delta) \vdash A}\ L_{\multimap}$$

The cut-elimination steps $\pi \rightsquigarrow \rho$ are an easy adaptation of the usual ones. We just have to take care of the exchange parts, but they can be handled without any particular problem. To avoid stupid loops, we allow a cut rule c to commute upwards with another cut rule d during reduction only if d introduces the left premise of c (and not if d introduces the right premise of c).

For our complexity analysis to make sense, we will restrict the cut elimination procedure to a particular strategy of reduction called *surface reduction*. From

a proof-net point of view it corresponds to reducing cuts at depth 0 only. In a sequent calculus setting, we only apply a reduction step to a cut rule if it is not above a promotion rule $P_!$. There are several reasons why surface reduction has been considered here:

- It corresponds to various reduction strategies for the lambda calculus. In particular, if lambda terms are encoded into IMELL via the cbn encoding $A \to B \equiv !A \multimap B$, then surface reduction simulates head reduction. On the other hand, the cbv encoding $A \to B \equiv !(A \multimap B)$ induces a simulation of (weak) call-by-value reduction by surface reduction.
- An upper bound to the time complexity of cut-elimination can be obtained by considering the so-called level-by-level strategy [5]. But the level-by-level strategy is nothing more than an iteration of surface reduction. As a consequence, our semantic interpretation could be applied itself iteratively to obtain bounds on the time complexity of ordinary cut-elimination.
- Any proof whose conclusion does not contain the modal operator ! in *positive* position can be normalized using surface reduction. In the cbn encoding, formulas for infinite datatypes *do* contain ! in positive position, but those positive occurrences can be "linearized" with appropriate coercion maps. As an example, natural numbers are encoded as $N = !(!A \multimap A) \multimap !A \multimap A$, but there is an easy coercion $N \multimap N_{lin}$, where $N_{lin} = !(A \multimap A) \multimap !A \multimap A$.

For practical reasons, we introduce a particular atomic formula $\mathbb{U}$ and we extend the IMELL system with the following "pseudo"-rules (which are not valid from a logical point of view):

$$\frac{}{\vdash X}\ a \qquad\qquad \frac{\Gamma \vdash A}{\varsigma(X,\Gamma) \vdash A}\ w$$

where X is any atomic formula: α or $\mathbb{U}$.

This allows us to define a proof TA_A of $A \vdash \mathbb{U}$ and a proof TA^0_A of $\vdash A$. TA_A is called the *time analyzer* of A. They are defined by mutual induction on A:

$$\frac{\dfrac{}{\vdash \mathbb{U}}\ a}{\mathsf{TA}_\alpha : \alpha \vdash \mathbb{U}}\ w \qquad \frac{}{\mathsf{TA}^0_\alpha : \vdash \alpha}\ a \qquad \frac{\dfrac{}{\vdash \mathbb{U}}\ a}{\mathsf{TA}_{!A} : !A \vdash \mathbb{U}}\ W \qquad \frac{\mathsf{TA}^0_A : \vdash A}{\mathsf{TA}^0_{!A} : \vdash !A}\ P_!$$

$$\frac{\dfrac{\mathsf{TA}_A : A \vdash \mathbb{U} \quad \dfrac{\mathsf{TA}_B : B \vdash \mathbb{U}}{\mathbb{U}, B \vdash \mathbb{U}}\ w}{A, B \vdash \mathbb{U}}\ U}{\mathsf{TA}_{A \otimes B} : A \otimes B \vdash \mathbb{U}}\ L_\otimes \qquad \frac{\mathsf{TA}^0_A : \vdash A \quad \mathsf{TA}^0_B : \vdash B}{\mathsf{TA}^0_{A \otimes B} : \vdash A \otimes B}\ R_\otimes$$

$$\frac{\mathsf{TA}^0_A : \vdash A \quad \mathsf{TA}_B : B \vdash \mathbb{U}}{\mathsf{TA}_{A \multimap B} : A \multimap B \vdash \mathbb{U}}\ L_\multimap \qquad \frac{\mathsf{TA}_A : A \vdash \mathbb{U} \quad \dfrac{\mathsf{TA}^0_B : \vdash B}{\mathbb{U} \vdash B}\ w}{A \vdash B}\ U}{\mathsf{TA}^0_{A \multimap B} : \vdash A \multimap B}\ R_\multimap$$

3 Game Semantics

The game model we use is based on the constructions presented in [1]. We extend it with the simplest exponential construction (by enriching moves with copy

indexes [10,2], except that we use a presentation with exponential signatures in the spirit of the geometry of interaction [6]) together with a lifting operation (adding two fresh moves at the beginning of a game).

3.1 Games

A *game* A consists in:

- A set of *moves* M_A.
- A function $\lambda_A : M_A \to \{\mathsf{P}, \mathsf{O}\}$. $\overline{\lambda_A}$ denotes the function from M_A to $\{\mathsf{P}, \mathsf{O}\}$ defined by $\overline{\lambda_A}(m) \neq \lambda_A(m)$. $M_A^{\circledast}$ denotes the subset of M_A^* containing alternating, opponent-initiated sequences only, i.e., $\lambda_A(m) = \mathsf{O}$ whenever $ms \in M_A^{\circledast}$ and, moreover, $\lambda_A(m) \neq \lambda_A(n)$ whenever $smnr \in M_A^{\circledast}$. M_A^{P} and M_A^{O} are subsets of M_A defined in the natural way.
- A set P_A of *valid plays* such that $P_A \subseteq M_A^{\circledast}$ and P_A is closed under prefixes.

The language $\mathscr{E}$ of *exponential signatures* is defined by induction from the following set of productions: $t, s, u ::= \mathsf{e} \mid \mathsf{l}(t) \mid \mathsf{r}(t) \mid \mathsf{p}(t) \mid \mathsf{n}(t, t)$.

3.2 Constructions on Games

To each connective corresponds a game construction. In the particular case of the exponential connective !, we decompose its interpretation in our model into a "sequentiality construction" given by lifting and a "copying construction" given by a traditional exponential construction with copy indexes given by exponential signatures.

- **Atomic game α:**
  - $M_\alpha = \{\alpha^{\mathsf{P}}, \alpha^{\mathsf{O}}\}$.
  - $\lambda_\alpha(\alpha^{\mathsf{P}}) = \mathsf{P}$ and $\lambda_\alpha(\alpha^{\mathsf{O}}) = \mathsf{O}$.
  - P_α is $\{\varepsilon, \alpha^{\mathsf{O}}, \alpha^{\mathsf{O}} \cdot \alpha^{\mathsf{P}}\}$.

  One particular atomic game is called $\mathbb{U}$ with moves denoted by a (instead of α^{P}) and q (instead of α^{O}).
- **Tensor game $A \otimes B$:**
  - $M_{A\otimes B} = M_A + M_B$. If $s \in M_{A\otimes B}^*$, then s_A denotes the subsequence of s consisting of moves in M_A. Similarly for s_B.
  - $\lambda_{A\otimes B} = \lambda_A + \lambda_B$.
  - The elements of $P_{A\otimes B}$ are sequences $s \in M_{A\otimes B}^{\circledast}$ such that $s_A \in P_A$, $s_B \in P_B$.
- **Arrow game $A \multimap B$:**
  - $M_{A\multimap B} = M_A + M_B$.
  - $\lambda_{A\multimap B} = \overline{\lambda_A} + \lambda_B$.
  - The elements of $P_{A\multimap B}$ are sequences $s \in M_{A\multimap B}^{\circledast}$ such that $s_A \in P_A$, $s_B \in P_B$.
- **Lifting game $\downarrow A$:**
  - $M_{\downarrow A} = M_A + \{\mathsf{open}, \mathsf{close}\}$.
  - $\lambda_{\downarrow A}(m) = \lambda_A(m)$ whenever $m \in M_A$, $\lambda_{\downarrow A}(\mathsf{open}) = \mathsf{O}$, $\lambda_{\downarrow A}(\mathsf{close}) = \mathsf{P}$.

  - $P_{\downarrow A}$ is $\{\varepsilon, \mathsf{open}\} \cup \{\mathsf{open} \cdot \mathsf{close} \cdot s \mid s \in P_A\}$.
- **Exponential game $\#A$:**
  - $M_{\#A} = \mathscr{E} \times M_A$. Given any sequence s in $M^*_{\#A}$ and any exponential signature t, s_t denotes the subsequence of s consisting in moves in the form (t, m). Given any sequence s in M^*_A and any exponential signature t, $t \times s$ denotes the sequence in $M^*_{\#A}$ obtained by pairing each move in s with t.
  - $\lambda_{\#A}(t, m) = \lambda_A(m)$.
  - The elements of $P_{\#A}$ are sequences $s \in M^{\circledast}_{\#A}$ such that for every $t \in \mathscr{E}$, $s_t = t \times r$ with $r \in P_A$.

We will often use the notation $!A$ for $\#{\downarrow}A$.

3.3 Strategies

Proofs are interpreted as particular strategies over games. However since we are not looking for full completeness results (but for complexity full abstraction instead), we are not particularly restrictive on the kind of strategies we deal with. There is no particular notion of uniformity on strategies such as history-freeness, innocence, etc. Important properties of strategies coming from proofs will be recovered through realizability (see Section 4).

A *strategy* σ over a game A is a non-empty set of even-length plays in P_A satisfying the following conditions:

- σ is *even-prefix-closed*;
- σ is *deterministic*: if $smn \in \sigma$, $sml \in \sigma$, then $n = l$.

A strategy σ over A is *total* if $s \in \sigma$ and $sm \in P_A$ implies $smn \in \sigma$ for some $n \in M_A$.

Composition of strategies can be defined in the usual way. Given a strategy σ over $A \multimap B$ and τ over $B \multimap C$, we can first define $\sigma \parallel \tau$ as follows:

$$\sigma \parallel \tau = \{s \in (M_A + M_B + M_C)^* \mid s_{A,B} \in \sigma \wedge s_{B,C} \in \tau\}.$$

where $s_{A,B}$ denotes the subsequence of s consisting of moves in $M_A + M_B$ and similarly for $s_{B,C}$ and for $s_{A,C}$.

The *composition* of σ and τ, denoted $\sigma;\tau$ is simply $\sigma;\tau = \{s_{A,C} \mid s \in \sigma \parallel \tau\}$.

Proposition 1. *If σ is a strategy over $A \multimap B$ and τ is a strategy over $B \multimap C$, then $\sigma;\tau$ is a strategy over $A \multimap C$.*

A useful restriction on strategies is given by *history-free strategies* σ satisfying: if $sm \cdot next_\sigma(m) \in P_A$ then $sm \cdot next_\sigma(m) \in \sigma$ if and only if $s \in \sigma$ where $next_\sigma$ is the generating partial function from M^{O}_A to M^{P}_A. The composition of two history-free strategies is an history-free strategy generated by the composition of generating functions. Some of the strategies we use happen to be history-free, but not all of them are.

The history-free *identity strategy* $\mathbf{id}_A$ over $A \multimap A$ is given by the generating function (assume $A \multimap A$ is $A_1 \multimap A_2$):

$$\forall m \in M_A^{\mathsf{O}}.next_{\mathbf{id}_A}(m_{A_2}) = m_{A_1}$$
$$\forall m \in M_A^{\mathsf{P}}.next_{\mathbf{id}_A}(m_{A_1}) = m_{A_2}$$

According to [1], games and strategies define a symmetric monoidal closed category (SMCC).

3.4 Constructions on Strategies

We describe elementary constructions on strategies which, once plugged together, will allow us to interpret proofs in the game model.

- **Left-lifting Strategy:** Given a strategy σ over the game $A \otimes B \multimap C$, the subset $\mathbf{ll}(\sigma)$ of $P_{\downarrow A \otimes B \multimap C}$ is defined as follows:

  $$\mathbf{ll}(\sigma) = \{\varepsilon\} \cup \{m \cdot \mathsf{open}_{\downarrow A} \mid \exists ms \in \sigma\} \cup \{m \cdot \mathsf{open}_{\downarrow A} \cdot \mathsf{close}_{\downarrow A} \cdot s \mid ms \in \sigma\}$$

  In the same spirit, we can define $\mathbf{ll}_B(\sigma)$ over $A \otimes \downarrow B \multimap C$ (so that $\mathbf{ll}_A(\sigma) = \mathbf{ll}(\sigma)$).
- **Right-lifting Strategy:** Given a strategy σ over the game A, the subset $\mathbf{rl}(\sigma)$ of $P_{\downarrow A}$ is defined as follows:

  $$\mathbf{rl}(\sigma) = \{\varepsilon\} \cup \{\mathsf{open}_{\downarrow A} \cdot \mathsf{close}_{\downarrow A} \cdot s \mid s \in \sigma\}$$

  Using the immediate bijection between $M_{\downarrow(A \multimap B)}$ and $M_{A \multimap \downarrow B}$, if σ is a strategy over $A \multimap B$, we will often use $\mathbf{rl}(\sigma)$ as a strategy over $A \multimap \downarrow B$.
- **Lifting Strategy:** Given a strategy σ over the game $A_1 \otimes \cdots \otimes A_n \multimap B$, the subset $\mathbf{l}(\sigma)$ of $P_{\downarrow A_1 \otimes \cdots \otimes \downarrow A_n \multimap \downarrow B}$ is defined by $\mathbf{l}(\sigma) = \mathbf{ll}_{A_1}(\ldots \mathbf{ll}_{A_n}(\mathbf{rl}(\sigma)))$.
- **Dereliction Strategy:** The subset $\mathbf{d}_A$ of $P_{\#A \multimap A}$ is the one induced by the following (assume $\#A \multimap A$ is $\#A_1 \multimap A_2$):

  $$\forall m \in M_A^{\mathsf{O}}.next_{\mathbf{d}_A}(m_{A_2}) = (\mathsf{e}, m)_{\#A_1}$$
  $$\forall m \in M_A^{\mathsf{P}}.next_{\mathbf{d}_A}((\mathsf{e}, m)_{\#A_1}) = m_{A_2}$$

- **Digging Strategy:** The subset $\mathbf{n}_A$ of $P_{\#\downarrow A \multimap \downarrow\#\downarrow\#\downarrow A}$ is the one induced by the following (assume $\#\downarrow A \multimap \downarrow\#\downarrow\#\downarrow A$ is $\#\downarrow A_1 \multimap \downarrow\#\downarrow\#\downarrow A_2$):

  $$next_{\mathbf{n}_A}(\mathsf{open}_{\downarrow\#\downarrow\#\downarrow A_2}) = (\mathsf{e}, \mathsf{open})_{\#\downarrow A_1}$$
  $$next_{\mathbf{n}_A}((\mathsf{e}, \mathsf{close})_{\#\downarrow A_1}) = \mathsf{close}_{\downarrow\#\downarrow\#\downarrow A_2}$$
  $$next_{\mathbf{n}_A}((t, \mathsf{open})_{\downarrow\#\downarrow\#\downarrow A_2}) = (\mathsf{p}(t), \mathsf{open})_{\#\downarrow A_1}$$
  $$next_{\mathbf{n}_A}((\mathsf{p}(t), \mathsf{close})_{\#\downarrow A_1}) = (t, \mathsf{close})_{\downarrow\#\downarrow\#\downarrow A_2}$$
  $$\forall m \in M_A^{\mathsf{O}}.next_{\mathbf{n}_A}((t, (s, m))_{\downarrow\#\downarrow\#\downarrow A_2}) = (\mathsf{n}(t, s), m)_{\#\downarrow A_1}$$
  $$\forall m \in M_A^{\mathsf{P}}.next_{\mathbf{n}_A}((\mathsf{n}(t, s), m)_{\#\downarrow A_1}) = (t, (s, m))_{\downarrow\#\downarrow\#\downarrow A_2}$$

- **Contraction Strategy:** The subset $\mathbf{c}_A$ of $P_{\#\downarrow A \multimap \downarrow\#\downarrow A \otimes \#\downarrow A}$ is the one induced by the following (assume $\#\downarrow A \multimap \downarrow\#\downarrow A \otimes \#\downarrow A$ is $\#\downarrow A_1 \multimap \downarrow\#\downarrow A_2 \otimes \#\downarrow A_3$):

$$\begin{aligned}
next_{\mathbf{c}_A}(\mathsf{open}_{\downarrow\#\downarrow A_2}) &= (\mathsf{e}, \mathsf{open})_{\#\downarrow A_1} \\
next_{\mathbf{c}_A}((\mathsf{e}, \mathsf{close})_{\#\downarrow A_1}) &= \mathsf{close}_{\downarrow\#\downarrow A_2} \\
\forall m \in M_A^{\mathsf{O}}.next_{\mathbf{c}_A}((t,m)_{\downarrow\#\downarrow A_2}) &= (\mathsf{l}(t), m)_{\#\downarrow A_1} \\
\forall m \in M_A^{\mathsf{P}}.next_{\mathbf{c}_A}((\mathsf{l}(t), m)_{\#\downarrow A_1}) &= (t,m)_{\downarrow\#\downarrow A_2} \\
\forall m \in M_A^{\mathsf{O}}.next_{\mathbf{c}_A}((t,m)_{\#\downarrow A_3}) &= (\mathsf{r}(t), m)_{\#\downarrow A_1} \\
\forall m \in M_A^{\mathsf{P}}.next_{\mathbf{c}_A}((\mathsf{r}(t), m)_{\#\downarrow A_1}) &= (t,m)_{\#\downarrow A_3}
\end{aligned}$$

- **Promotion Strategy:** Given a strategy σ over the game $A_1 \otimes \cdots \otimes A_n \multimap B$, the subset $\mathbf{p}(\sigma)$ of $P_{\#A_1 \otimes \cdots \otimes \#A_n \multimap \#B}$ is defined as follows:

$$\mathbf{p}(\sigma) = \{s \in P_{\#A_1 \otimes \cdots \otimes \#A_n \multimap \#B} \mid \forall t.\exists r \in \sigma.s_t = t \times r\}$$

We use the notation $\mathbf{pl}(\sigma)$ for $\mathbf{p}(\mathbf{l}(\sigma))$.

Proposition 2. *For any game A, $\mathbf{d}_A$, $\mathbf{n}_A$ and $\mathbf{c}_A$ are strategies. Let σ be a strategy over $A_1 \otimes \cdots \otimes A_n \multimap B$. Then $\mathbf{rl}(\sigma)$ and $\mathbf{p}(\sigma)$ are strategies and, if $n \geq 1$, $\mathbf{ll}(\sigma)$ is a strategy.*

3.5 Interpretation of Proofs

We define the strategy $[\![\pi]\!]$ interpreting a proof π.

The multiplicative rules are interpreted according to the symmetric monoidal closed structure of the category of games and strategies. The interpretation of the exponential rules is based on the constructions described above.

- **Weakening:** if σ is a strategy over $\Gamma \multimap B$, it is also a strategy over $A \otimes \Gamma \multimap B$ and we can build $(\mathbf{d}_{\downarrow A} \otimes \mathbf{id}_\Gamma); \mathbf{ll}(\sigma)$ as a strategy over $!A \otimes \Gamma \multimap B$.
- **Contraction:** if σ is a strategy over $!A \otimes !A \otimes \Gamma \multimap B$, we can build $(\mathbf{c}_A \otimes \mathbf{id}_\Gamma); \mathbf{ll}(\sigma)$ as a strategy over $!A \otimes \Gamma \multimap B$.
- **Promotion:** if σ is a strategy over $A_1 \otimes \cdots \otimes A_n \multimap B$, we can build $\mathbf{pl}(\sigma)$ as a strategy over $!A_1 \otimes \cdots \otimes !A_n \multimap !B$.
- **Dereliction:** if σ is a strategy over $A \otimes \Gamma \multimap B$, we can build $(\mathbf{d}_{\downarrow A} \otimes \mathbf{id}_\Gamma); \mathbf{ll}(\sigma)$ as a strategy over $!A \otimes \Gamma \multimap B$.
- **Digging:** if σ is a strategy over $!!A \otimes \Gamma \multimap B$, we can build $(\mathbf{n}_A \otimes \mathbf{id}_\Gamma); \mathbf{ll}(\sigma)$ as a strategy over $!A \otimes \Gamma \multimap B$.

The main difference between weakening and dereliction comes from the original strategy: over $\Gamma \multimap B$ for weakening and considered over $A \otimes \Gamma \multimap B$, while "really" over $A \otimes \Gamma \multimap B$ for dereliction.

Theorem 1 (Soundness). *If $\pi \rightsquigarrow \rho$ then $[\![\pi]\!] = [\![\rho]\!]$.*

Proof. The multiplicative steps are given by the SMCC structure. The permutations of formulas are handled by the symmetry of the SMCC structure. The key properties required for the other cases are:

- If $\sigma : A_0 \multimap A$ and $\tau : A \otimes B \multimap C$ then $(\sigma \otimes \mathbf{id}_{\downarrow B}); \mathbf{ll}_B(\tau) = \mathbf{ll}_B((\sigma \otimes \mathbf{id}_B); \tau)$.
- If $\sigma : A_1 \otimes \cdots \otimes A_n \multimap A$ and $\tau : A \otimes A_{n+1} \otimes \cdots \otimes A_m \multimap B$ then $(\mathbf{p}(\sigma) \otimes \mathbf{id}_{\#A_{n+1}} \otimes \cdots \otimes \mathbf{id}_{\#A_m}); \mathbf{p}(\tau) = \mathbf{p}((\sigma \otimes \mathbf{id}_{A_{n+1}} \otimes \cdots \otimes \mathbf{id}_{A_m}); \tau)$.
- If $\sigma : A_1 \otimes \cdots \otimes A_n \multimap B$ then $\mathbf{p}(\sigma); \mathbf{d}_B = (\mathbf{d}_{A_1} \otimes \cdots \otimes \mathbf{d}_{A_n}); \sigma$.
- If $\sigma : A_1 \otimes \cdots \otimes A_n \multimap B$ then $\mathbf{pl}(\sigma); \mathbf{c}_B = (\mathbf{c}_{A_1} \otimes \cdots \otimes \mathbf{c}_{A_n}); \mathbf{l}(\mathbf{pl}(\sigma)) \otimes \mathbf{pl}(\sigma)$ (up to some permutation in the second composition turning $\downarrow!A_1 \otimes !A_1 \otimes \cdots \otimes \downarrow!A_n \otimes !A_n$ into $\downarrow!A_1 \otimes \cdots \otimes \downarrow!A_n \otimes !A_1 \otimes \cdots \otimes !A_n$).
- If $\sigma : A_1 \otimes \cdots \otimes A_n \multimap B$ then $\mathbf{pl}(\sigma); \mathbf{n}_B = (\mathbf{n}_{A_1} \otimes \cdots \otimes \mathbf{n}_{A_n}); \mathbf{l}(\mathbf{pl}(\mathbf{pl}(\sigma)))$. □

We extend the interpretation to the formula $\mathbb{U}$ and to pseudo-rules. The pseudo-rule a is interpreted by the strategy $\{\varepsilon, X^{\mathsf{O}} \cdot X^{\mathsf{P}}\}$. If σ is the interpretation of the premise of an application of the pseudo-rule w, its conclusion is interpreted by $\{\varepsilon\} \cup \{m \cdot X^{\mathsf{O}} \mid \exists ms \in \sigma\} \cup \{m \cdot X^{\mathsf{O}} \cdot X^{\mathsf{P}} \cdot s \mid ms \in \sigma\}$. X^{O} denotes α^{O} if $X = \alpha$ and q if $X = \mathbb{U}$. X^{P} denotes α^{P} if $X = \alpha$ and a if $X = \mathbb{U}$.

If σ is the interpretation of a (pseudo)-proof, then σ is *total*.

4 Realizability

In order to prove properties of the strategies interpreting proofs, we are going to define a notion of realizability between strategies and formulas.

The relations "σ P-realizes A", $\sigma \Vdash^P A$, (with σ strategy over A) and "τ O-realizes A", $\tau \Vdash^O A$, (with τ strategy over $A \multimap \mathbb{U}$) are defined in a mutually recursive way by induction on A:

- $\sigma \Vdash^P \alpha$ if $\sigma = \{\varepsilon, \alpha^{\mathsf{O}} \cdot \alpha^{\mathsf{P}}\}$
- $\tau \Vdash^O \alpha$ if $\tau = \{\varepsilon, \mathsf{q} \cdot \alpha^{\mathsf{O}}, \mathsf{q} \cdot \alpha^{\mathsf{O}} \cdot \alpha^{\mathsf{P}} \cdot \mathsf{a}\}$
- $\sigma \Vdash^P \mathbb{U}$ if $\sigma = \{\varepsilon, \mathsf{q} \cdot \mathsf{a}\}$
- $\tau \Vdash^O \mathbb{U}$ if $\tau = \mathbf{id}_{\mathbb{U}}$
- $\sigma \Vdash^P A \otimes B$ if $\sigma_A \Vdash^P A$ and $\sigma_B \Vdash^P B$ with $\sigma_A = \{s_A \mid s \in \sigma\}$. (We ask in particular that σ_A and σ_B are strategies over A and B, respectively.)
- $\tau \Vdash^O A \otimes B$ if for any $\sigma \Vdash^P A$, $\sigma; \tau \Vdash^O B$ and for any $\sigma \Vdash^P B$, $\sigma; \tau \Vdash^O A$. (Using that, up to the curryfication isomorphisms, τ can also be seen as a strategy over $A \multimap (B \multimap \mathbb{U})$ or over $B \multimap (A \multimap \mathbb{U})$.)
- $\sigma \Vdash^P A \multimap B$ if for any $\delta \Vdash^P A$, $\delta; \sigma \Vdash^P B$ and for any $\tau \Vdash^O B$, $\sigma; \tau \Vdash^O A$
- $\tau \Vdash^O A \multimap B$ if $\tau_A \Vdash^P A$ and $\tau_{B \multimap \mathbb{U}} \Vdash^O B$
- $\sigma \Vdash^P !A$ if for any exponential signature t, $\sigma\restriction_t \Vdash^P A$ with $\sigma\restriction_t = \{s\restriction_t \mid s \in \sigma\}$ and $s\restriction_t$ is obtained from s_t by replacing any move (t, m) by m and by then erasing the initial open and close moves if they appear (we ask in particular that $\sigma\restriction_t$ is a strategy over A for any t).
- $\tau \Vdash^O !A$ if τ contains the play $\mathsf{q} \cdot (\mathsf{e}, \mathsf{open})$.

An adequacy property relates proofs, strategies and realizability:

Proposition 3. *For every proof π, the strategy $[\![\pi]\!]$ P-realizes the conclusion of π.*

Proof. A first remark is that if $\sigma \Vdash^P A$ then σ contains a non-empty play and if $\tau \Vdash^O A$ then τ contains a play with a move in A (by induction on A). We now do the proof by induction on π. We only give a few typical cases.

- Right tensor: if $\sigma_1 \Vdash^P \Gamma \multimap A$ and $\sigma_2 \Vdash^P \Delta \multimap B$, and if $\delta \Vdash^P \Gamma \otimes \Delta$ then $\delta_\Gamma \Vdash^P \Gamma$ and $\delta_\Delta \Vdash^P \Delta$ so that $\delta_\Gamma; \sigma_1 \Vdash^P A$ and $\delta_\Delta; \sigma_2 \Vdash^P B$, and finally $\delta; (\sigma_1 \otimes \sigma_2) \Vdash^P A \otimes B$. If $\tau \Vdash^O A \otimes B$, $\delta_1 \Vdash^P \Gamma$ and $\delta_2 \Vdash^P \Delta$, we have: $(\delta_1 \otimes \mathbf{id}_\Delta); (\sigma_1 \otimes \sigma_2); \tau = (\delta_1; \sigma_1) \otimes \sigma_2; \tau = \sigma_2; ((\delta_1; \sigma_1); \tau)$, but $\delta_1; \sigma_1 \Vdash^P A$ thus $(\delta_1; \sigma_1); \tau \Vdash^O B$ and $(\delta_1 \otimes \mathbf{id}_\Delta); (\sigma_1 \otimes \sigma_2); \tau \Vdash^O \Delta$. In a similar way $(\mathbf{id}_\Gamma \otimes \delta_2); (\sigma_1 \otimes \sigma_2); \tau \Vdash^O \Gamma$.
- Promotion: if $\sigma \Vdash^P A_1 \otimes \cdots \otimes A_n \multimap B$ (with σ' obtained from σ by interpreting the promotion rule) and if $\delta_i \Vdash^P !A_i$ $(1 \leq i \leq n)$, for any exponential signature t we have $\delta_i\restriction_t \Vdash^P A_i$ thus $((\delta_1 \otimes \cdots \otimes \delta_n); \sigma')\restriction_t = (\delta_1\restriction_t \otimes \cdots \otimes \delta_n\restriction_t); \sigma \Vdash^P B$. If $\tau \Vdash^O !B$ and $\delta_i \Vdash^P !A_i$ $(1 \leq i \leq n)$, for any $1 \leq i \leq n$, $(\delta_1 \otimes \cdots \otimes \delta_{i-1} \otimes \mathbf{id}_{!A_i} \otimes \delta_{i+1} \otimes \cdots \otimes \delta_n); \sigma'; \tau$ plays $(\mathsf{e}, \mathsf{open})$ as first move in $!A_i$ since τ plays $(\mathsf{e}, \mathsf{open})$ as first move in $!B$ and each δ_i contains the play $(\mathsf{e}, \mathsf{open}) \cdot (\mathsf{e}, \mathsf{close})$. □

As a consequence, $[\![\mathsf{TA}_A]\!] \Vdash^P A \multimap \mathbb{U}$ thus $[\![\mathsf{TA}_A]\!] \Vdash^O A$ (since $\mathbf{id}_\mathbb{U} \Vdash^O \mathbb{U}$).

A *complete set of moves* for any game A is a subset of M_A defined by induction on the structure of A:

- If $A = \alpha$, the only complete set of moves for A is $\{\alpha^\mathsf{P}, \alpha^\mathsf{O}\}$.
- If $A = B \otimes C$ or $A = B \multimap C$, C_B is a complete set of moves for B and C_C is a complete set of moves for C, then $C_A = C_B + C_C$ is a complete set of moves for A.
- If $A = !B$, then any subset of M_A containing the move $(\mathsf{e}, \mathsf{close})$ is a complete set of moves for A.

Proposition 4. *If σ P-realizes A, τ O-realizes A and $\sigma; \tau$ is total, then the maximal sequence in $\sigma \parallel \tau$ (seen as a set of moves of A) is complete.*

5 Complexity

In this Section, we show how to instrument games with slots, in the same vein as in Ghica's framework [8]. The idea is simple: slots are used by the player to communicate some quantitative properties of the underlying proof to the opponent. But while in Ghica's work slots are produced in correspondence with any potential redex, here the player raises a slot in correspondence with boxes, i.e. instances of the promotion rule. In Ghica's slot games, the complexity of a program can be read out of any complete play in its interpretation, while here the process of measuring the complexity of proofs is internalized through the notion of time analyzer (see Section 2): the complexity of π (with conclusion A) is simply the number of slots produced in the interaction between $[\![\pi]\!]$ and $[\![\mathsf{TA}_A]\!]$. Notice that the definition of TA_A only depends on the formula A.

The symbol $\bullet$ is a special symbol called a *slot*. In the new setting, the set of moves for A, will be the usual M_A, while the notion of a play should be slightly changed. Given a game A and a sequence s in $(M_A + \{\bullet\})^*$, we denote by s° the sequence in M_A^* obtained by deleting any occurrence of $\bullet$ in s. Analogously, given any subset σ of $(M_A + \{\bullet\})^*$, σ° will denote $\{s^\circ \mid s \in \sigma\} \subseteq M_A^*$.

A *play-with-costs* for A is a sequence s in $(M_A + \{\bullet\})^*$ such that $s^\circ \in P_A$, whenever $s = r \bullet mq$ it holds that $\lambda_A(m) = \mathsf{P}$ and the last symbol in s (if any) is a move in M_A. A *strategy-with-costs* for the game A is a set σ of plays-with-costs for A such that σ° is a strategy (in the usual sense) for A and, moreover, σ is *slot-deterministic*: if $sm \bullet^k n \in \sigma$ and $sm \bullet^h n \in \sigma$, then $k = h$.

Composition of strategies-with-costs needs to be defined in a slightly different way than the one of usual strategies. In particular, we need two different notions of projections: first of all, if $s \in (M_A + M_B + \{\bullet\})^*$, we can construct s_{A^X} (where $X \subseteq \{\mathsf{P}, \mathsf{O}\}$) by extracting from s any move $m \in M_A$ together with the slots immediately before any such m provided $\lambda_A(m) \in X$. But we can even construct $s_{A^\bullet}$, by only considering the slots which precede moves in M_A *but not the moves themselves*. Given strategies-with-costs σ over $A \multimap B$ and τ over $B \multimap C$, we can first define $\sigma \parallel \tau$ as follows:

$$\sigma \parallel \tau = \{s \in (M_A + M_B + M_C + \{\bullet\})^* \mid s_{A^{\mathsf{P},\mathsf{O}}, B^{\mathsf{P}}} \in \sigma \wedge s_{B^{\mathsf{O}}, C^{\mathsf{P},\mathsf{O}}} \in \tau\}.$$

The *composition* of σ and τ, denoted $\sigma; \tau$ is now simply

$$\sigma; \tau = \{s_{A^{\mathsf{P},\mathsf{O}}, B^\bullet, C^{\mathsf{P},\mathsf{O}}} \mid s \in \sigma \parallel \tau\}.$$

In other words, we forget the moves in M_B, but we keep all the slots produced by them.

Proposition 5. *If σ is a strategy-with-costs over $A \multimap B$ and τ is a strategy-with-costs over $B \multimap C$, then $\sigma; \tau$ is a strategy-with-costs over $A \multimap C$.*

The strategy constructions we have seen so far can be turned into strategy-with-costs constructions. In the basic strategies, slots come into play only in $\mathbf{rl}(\sigma)$: in particular, $\mathbf{rl}^i(\sigma) = \{\varepsilon\} \cup \{\mathsf{open}_{\downarrow A} \cdot \bullet^i \cdot \mathsf{close}_{\downarrow A} \cdot s \mid s \in \sigma\}$. This way, the interpretation $[\![\pi]\!]^i$ of any proof π is parametrized on a natural number i.

In the particular case of a cut-free proof π with axiom rules only introducing !-free formulas, $[\![\pi]\!]^i$ can be easily deduced from $[\![\pi]\!]$ by adding $\bullet^i$ before each P-move of the shape (t, close) in each play of $[\![\pi]\!]$.

We are in a position to define the complexity $\mathcal{C}(\pi)$ of any proof π. First, consider the shape of any non-trivial play-with-costs s in a strategy-with-costs σ for $\mathbb{U}$: it must have the following shape $\mathsf{q} \bullet^i \mathsf{a}$. But observe that this play is the *only* non-trivial play-with-costs in σ, due to (slot) determinacy. The integer i is called the *complexity* of σ, denoted $\mathcal{C}(\sigma)$. This way we can define the complexity $\mathcal{C}(\pi)$ of any proof π with conclusion A as simply the complexity of π when composed with the time analyzer: $\mathcal{C}([\![\pi]\!]^1; [\![\mathsf{TA}_A]\!]^0)$. The complexity of π is defined for every π because $[\![\pi]\!]; [\![\mathsf{TA}_A]\!]$ P-realizes $\mathbb{U}$ (by Proposition 3) and, as a consequence, contains a non-empty play. Given any play-with-costs s, $\mathcal{C}(s)$ is simply the number of occurrences of $\bullet$ in s.

5.1 Dynamics under Exposed Cuts

In this Section, we will prove some lemmas about the preservation of semantics when cuts are exposed as in the $[\cdot]$ construction (see Section 2). With

$[\![\pi]\!]_{eca}$ we denote the (unique) maximal (wrt the prefix order) sequence in $\iota \parallel ([\![\pi]\!]^1_{ec}; [\![\mathsf{TA}_B]\!]^0)$, where $[\pi] :!^{k_1}(A_1 \multimap A_1), \ldots, !^{k_n}(A_n \multimap A_n), \Gamma \vdash C$, $[\![\pi]\!]^i_{ec} = [\![[\pi]]\!]^i$, $\iota = [\![!^{k_1} id_{A_1} \otimes \cdots \otimes !^{k_n} id_{A_n}]\!]^0$ and $B = \bigotimes \Gamma \multimap C$. $!^k id_A$ is the proof for $!^k(A \multimap A)$ obtained by applying k times the promotion rule (with an empty context) to the trivial proof id_A of $A \multimap A$. We are interested in studying how $[\![\pi]\!]_{eca}$ evolves during cut elimination for any proof $\pi : \Gamma \vdash C$. This will lead us to full abstraction. Indeed:

Remark 1. Please notice that the strategy from which we obtain the complexity of π is:

$$\tau = [\![\pi]\!]^1; [\![\mathsf{TA}_B]\!]^0 = (\iota; [\![\pi]\!]^1_{ec}); [\![\mathsf{TA}_B]\!]^0 = \iota; ([\![\pi]\!]^1_{ec}; [\![\mathsf{TA}_B]\!]^0).$$

This implies that $[\![\pi]\!]_{eca}$ contains exactly $\mathcal{C}(\pi)$ slots and, moreover, it contains a complete set of moves for $D = !^{k_1}(A_1 \multimap A_1) \otimes \cdots \otimes !^{k_n}(A_n \multimap A_n)$. This, in particular, is a consequence of Proposition 4, since $\iota \Vdash^P D$, $([\![\pi]\!]^1_{ec}; [\![\mathsf{TA}_B]\!]^0) \Vdash^O D$ and their composition is total.

The cut-elimination relation $\rightsquigarrow$ can be thought of as the union of nine reduction relations $\overset{x}{\rightsquigarrow}$ where x ranges over the set $\mathcal{R} = \{\mathcal{T}, \mathcal{X}, \otimes, \multimap, \mathcal{C}, \mathcal{D}, \mathcal{N}, \mathcal{W}, !-!\}$. They correspond to commuting, axiom, tensor, linear arrow, contraction, dereliction, digging, weakening and promotion-promotion cut-elimination steps. If $X \subseteq \mathcal{R}$ or $x \in \mathcal{R}$, then $\overset{X}{\rightsquigarrow}$ and $\overset{x}{\rightsquigarrow}$ have the obvious meaning. We can consider a reduction relation that postpones !−!-cuts to the very end of the computation. The resulting reduction relation is denoted with $\hookrightarrow$. Again, $\overset{X}{\hookrightarrow}$ and $\overset{x}{\hookrightarrow}$ (where $x \in \mathcal{R}$ and $X \subseteq \mathcal{R}$) have their natural meaning.

We need to analyze how $[\![\rho]\!]_{eca}$ differs from $[\![\pi]\!]_{eca}$ if $\pi \overset{x}{\rightsquigarrow} \rho$. Clearly, this crucially depends on $x \in \mathcal{R}$, since cuts are exposed in $[\pi]$ and $[\rho]$. Due to lack of space, we report just one particular case here, namely $x = \mathcal{D}$:

Lemma 1 (Dereliction). *If $\pi \overset{\mathcal{D}}{\rightsquigarrow} \rho$, then $\mathcal{C}([\![\pi]\!]_{eca}) = \mathcal{C}([\![\rho]\!]_{eca}) + 1$.*

Proof. We only consider the case where the cut reduced in π is the last rule of π. The other cases can be reduced to this one by an easy induction. With this hypothesis, $[\pi]$ is

$$\dfrac{\dfrac{[\sigma] : A_1, \ldots, A_n, D_1, \ldots, D_m \vdash B}{!A_1, \ldots, !A_n, \varsigma(!D_1, \ldots, !D_m) \vdash !B}\, P_! \quad \dfrac{[\theta] : \Phi, \Gamma, B \vdash C}{\Phi, \vartheta(\Gamma), !B \vdash C}\, D_!}{!A_1, \ldots, !A_n, !B \multimap !B, \Phi, \varpi(!D_1, \ldots, !D_m, \Gamma) \vdash C}\, L_\multimap$$

and $[\rho]$ is

$$\dfrac{\dfrac{\dfrac{\dfrac{[\sigma] : A_1, \ldots, A_n, D_1, \ldots, D_m \vdash B \quad [\theta] : \Phi, \Gamma, B \vdash C}{A_1, \ldots, A_n, B \multimap B, \Phi, \Gamma, D_1, \ldots, D_m \vdash C}\, L_\multimap}{A_1, \ldots, A_n, B \multimap B, \Phi, !D_m, \Gamma, D_1, \ldots, D_{m-1} \vdash C}\, D_!}{D_!}}{\vdots}}{\dfrac{A_1, \ldots, A_n, B \multimap B, \Phi, !D_2, \ldots, !D_m, \Gamma, D_1 \vdash C}{A_1, \ldots, A_n, B \multimap B, \Phi, \varpi(!D_1, \ldots, !D_m, \Gamma) \vdash C}\, D_!}\, D_!$$

Observe that: $[\pi] :!A_1, \ldots, !A_n, !B_1 \multimap !B_2, \Phi_1, \Gamma_1 \vdash C$ and $[\rho] : A_1, \ldots, A_n, B_1 \multimap B_2, \Phi_1, \Gamma_1 \vdash C$. Now, consider $\iota \parallel ([\![\rho]\!]^1_{ec}; [\![\mathsf{TA}_E]\!])$ and $\iota' \parallel ([\![\pi]\!]^1_{ec}; [\![\mathsf{TA}_E]\!])$, where

E is the conclusion of ρ and π. It is easy to realize that $[\![\rho]\!]_{eca}$ can be simulated by the $[\![\pi]\!]_{eca}$, in such a way that

$$[\![\rho]\!]_{eca} = [\![\pi]\!]_{eca}\{m_{B_i}/(\mathsf{e}, m_{B_i}), \cdot/\bullet^a\,(\mathsf{e},\mu)_{!B_i}, m_{A_i}/(\mathsf{e}, m_{A_i}), \cdot/\bullet^b\,(\mathsf{e},\mu)_{!A_i}\}$$

where μ is a metavariable for either open or close. Observe that $a = 1$ when $\mu = \mathsf{close}$ and $i = 1$ (a promotion in π raises a slot), $a = 0$ otherwise and $b = 0$ (ι does not raise any slot). But there is exactly one $(\mathsf{e}, \mathsf{close})_{!B_1}$ in $[\![\pi]\!]_{eca}$: at most one (the same move is not played twice); at least one from Proposition 4 (since strategies interpreting (pseudo)proofs are total). The thesis easily follows. □

5.2 Full Abstraction

We now have all the required material to give our key result: full abstraction of the game model with respect to the reduction length (Theorems 2 and 3).

Given a proof π and any reduction relation $\rightarrow$, $[\pi]_{\rightarrow}$ and $||\pi||_{\rightarrow}$ denote the maximum length of a reduction sequence starting in π (under $\rightarrow$) and the maximum size of any reduct of π (under $\rightarrow$), respectively. We note $|\pi|$ the size of a proof π.

Lemma 2. *For every proof π, $[\pi]_{\rightsquigarrow} = [\pi]_{\hookrightarrow}$ and $||\pi||_{\rightsquigarrow} = ||\pi||_{\hookrightarrow}$.*

Proof. Whenever $\pi \overset{!-!}{\rightsquigarrow} \rho \overset{x}{\rightsquigarrow} \sigma$ and $x \neq$!−!, there are $\theta_1, \ldots, \theta_n$ (where $n \geq 1$) such that $\pi \overset{x_1}{\rightsquigarrow} \theta_1 \overset{x_2}{\rightsquigarrow} \cdots \overset{x_n}{\rightsquigarrow} \theta_n \overset{x_{n+1}}{\rightsquigarrow} \sigma$, and $x_{i+1} =$!−! whenever $x_i =$!−!. For example, if $\pi \overset{!-!}{\rightsquigarrow} \rho \overset{W}{\rightsquigarrow} \sigma$ and the box erased in the second step is exactly the one created by the first step, then clearly $\pi \overset{W}{\rightsquigarrow} \theta \overset{W}{\rightsquigarrow} \sigma$. As a consequence, for any sequence $\pi_1 \rightsquigarrow \cdots \rightsquigarrow \pi_n$ there is another sequence $\rho_1 \hookrightarrow \cdots \hookrightarrow \rho_m$ such that $\pi_1 = \rho_1$, $\pi_n = \rho_m$ and $m \geq n$. This proves the first claim. Now, observe that for any $1 \leq i \leq n$ there is j such that $|\rho_j| \geq |\pi_i|$: a simple case analysis suffices. □

Proposition 6. *If $\pi \overset{\{\mathcal{C},\mathcal{D},\mathcal{N},\mathcal{W},!-!\}}{\hookrightarrow} \rho$ then $\mathcal{C}(\pi) = \mathcal{C}(\rho) + 1$.*

Proof. From Remark 1, we know that $\mathcal{C}(\pi) = \mathcal{C}([\![\pi]\!]_{eca})$. We apply Lemma 1 (and similar statements for contraction, digging and weakening). □

Proposition 7. *If $\pi \overset{\{\mathcal{T},\mathcal{X},\otimes,\multimap\}}{\hookrightarrow} \rho$ then $\mathcal{C}(\pi) = \mathcal{C}(\rho)$.*

The mismatch between the statements of Proposition 6 and Proposition 7 can be informally explained as follows. After any "exponential" reduction step (see Proposition 6) one slot is missing, namely the one raised by the promotion rule involved in the reduction step when faced with the $(\mathsf{e}, \mathsf{open})$ move raised by the left rule interacting with the promotion rule itself. Clearly, this does not happen when performing "linear" reduction steps (see Proposition 7).

Lemma 3. *If π is cut-free, then $\mathcal{C}(\pi) \leq |\pi|$.*

Proposition 8. *If π rewrites to ρ in n steps by the $\mathcal{T}$ rule, then $|\pi| = |\rho|$ and $n \leq 2|\pi|^2$.*

Proof. The equality $|\pi| = |\rho|$ can be trivially verified whenever $\pi \overset{\mathcal{T}}{\hookrightarrow} \rho$. Now, for any proof π, define $|\pi|_{comm}$ as the sum, over all instances of the U rule inside π, of $|\sigma|_{cut} + |\sigma| + |\theta|$, where σ (respectively, θ) is the left (respectively, right) premise of the cut and $|\sigma|_{cut}$ is simply the number of instances of the cut rule in σ. For obvious reasons, $0 \leq |\pi|_{comm} \leq 2|\pi|^2$. Moreover, if $\pi \overset{\mathcal{T}}{\hookrightarrow} \rho$, then $|\pi|_{comm} > |\rho|_{comm}$. For example, consider the following commutative reduction step:

$$\dfrac{\dfrac{\pi : \Gamma \vdash A \quad \rho : \Delta, A \vdash B}{\varsigma(\Gamma, \Delta) \vdash B}\, U \qquad \sigma : \Omega, B \vdash C}{\theta : \vartheta(\Gamma, \Delta, \Omega) \vdash C}\, U \quad \rightsquigarrow \quad \dfrac{\pi : \Gamma \vdash A \qquad \dfrac{\rho : \Delta, A \vdash B \quad \sigma : \Omega, B \vdash C}{\varpi(\Delta, \Omega), A \vdash C}\, U}{\xi : \vartheta(\Gamma, \Delta, \Omega) \vdash C}\, U$$

Clearly, $|\theta| = |\xi|$ but $|\theta|_{comm} > |\xi|_{comm}$. Other cases are similar. □

Theorem 2. *For every proof π, $\mathcal{C}(\pi) \leq [\pi]_{\rightsquigarrow} + ||\pi||_{\rightsquigarrow}$*

Theorem 3. *There is a polynomial $p : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ such that for every proof π, $[\pi]_{\rightsquigarrow} \leq p(\mathcal{C}(\pi), |\pi|)$ and $||\pi||_{\rightsquigarrow} \leq p(\mathcal{C}(\pi), |\pi|)$.*

Proof. By Lemma 2, the thesis easily follows from $[\pi]_{\hookrightarrow}, ||\pi||_{\hookrightarrow} \leq p(\mathcal{C}(\pi), |\pi|)$. Our first task will be to analyze the shape of any box you can find during the normalization of π by $\hookrightarrow$ up to the point where you begin to fire !−! cuts. But it is easy to prove that any such box is just a subproof of π, possibly endowed with n promotions rules (where n is less than the total number of $N_!$ cuts fired during normalization). As a consequence, any such box has at most size $|\pi| + \mathcal{C}(\pi)$. Now, we can easily bound $||\pi||_{\hookrightarrow}$: at any C or N normalization step, the size of the underlying proof increases by at most $|\pi| + \mathcal{C}(\pi)$ (but the complexity strictly decreases), while in any other case the size decreases. As a consequence, $||\pi||_{\hookrightarrow} \leq \mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi))$. Now, the total number of non-commuting reduction steps is at most $\mathcal{C}(\pi) + \mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi))$. Between any of them, there are at most $2||\pi||^2_{\hookrightarrow}$ commuting steps. As a consequence:

$$\begin{aligned}[\pi]_{\hookrightarrow} &\leq \mathcal{C}(\pi) + \mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi)) + (\mathcal{C}(\pi) + \mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi)))\, 2||\pi||^2_{\hookrightarrow} \\ &\leq (\mathcal{C}(\pi) + \mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi)))\, (1 + 2(\mathcal{C}(\pi)(|\pi| + \mathcal{C}(\pi)))^2).\end{aligned}$$

□

6 Further Work

The main defect of our approach is the strong use of sequentiality information from sequent calculus proofs in the game interpretation. The two main approaches to get rid of this sequentiality are the use of non-deterministic strategies or of clusters of moves (when interpreting the promotion rule). This way we would be able to directly interpret proof-nets. In a similar spirit, we have used an exponential construction for games based on a grammar of exponential signatures, as usually done with context semantics. This is known to lead to not-very-satisfactory properties for !: for example, weakening is not neutral with respect to contraction, contraction is not commutative, etc. However, an answer to this problem should easily come from the solution proposed in the

AJM setting with the notion of equivalence of strategies [2]. All these ingredients would probably allow us to turn our game model into a true categorical model of intuitionistic linear logic.

Another weakness is the restriction to surface reduction. We think adaptations to head reduction or to reduction strategies leading to normal forms should be possible by modifying the time analyzer in order to interactively access to "deeper" parts of proofs.

The notion of realizability we have introduced is tuned to reach the result we need, namely Proposition 4. However, it seems possible to modify it in various ways and to use it for very different applications in the more general context of game semantics.

Very recently, another proposal leading to similar observations but being based on relational semantics has appeared [7]. The authors give an exact measure of the number of steps required for surface reduction (and then level-by-level reduction). This should be also possible in our setting by adding lifting constructions to all the connectives (not only to the exponential ones). However an important difference comes from the notion of cut elimination under consideration: while they use β-reduction style exponential steps (coming from contractions of unbounded arity in particular), we consider standard exponential steps (based on binary contractions), and this may lead to an exponential blowup. Possible correspondences between our game-theoretical analysis and the analysis done in [7] could come from works about projecting strategies into relations.

Acknowledgements

The authors would like to thank the anonymous referees for their useful comments.

References

1. Abramsky, S.: Semantics of interaction. In: Dybjer, P., Pitts, A. (eds.) Semantics and Logics of Computation. Publications of the Newton Institute, pp. 1–32. Cambridge University Press, Cambridge (1997)
2. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation 163(2), 409–470 (2000)
3. Baillot, P.: Approches dynamiques en sémantique de la logique linéaire : jeux et géométrie de l'interaction. Thèse de doctorat, Université Aix-Marseille II (1999)
4. Baillot, P., Pedicini, M.: Elementary complexity and geometry of interaction. Fundamenta Informaticae 45(1-2), 1–31 (2001)
5. Lago, U.D.: Context semantics, linear logic and computational complexity. In: Proc. 21st Symposium on Logic in Computer Science, pp. 169–178. IEEE, Los Alamitos (2006)
6. Danos, V., Herbelin, H., Regnier, L.: Games semantics and abstract machines. In: Proc. 11th Symposium on Logic In Computer Science, pp. 394–405. IEEE, Los Alamitos (1996)
7. de Carvalho, D., Pagani, M., de Falco, L.T.: A semantic measure of the execution time in linear logic. Technical Report 6441, INRIA (2007)

8. Ghica, D.: Slot games: A quantitative model of computation. In: Proc. 32nd ACM Symposium on Principles of Programming Languages, pp. 85–97 (2005)
9. Girard, J.-Y.: Geometry of interaction III: accommodating the additives. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advances in Linear Logic. London Mathematical Society Lecture Note Series, vol. 222, pp. 329–389. Cambridge University Press, Cambridge (1995)
10. Hyland, M.: Game semantics. In: Dybjer, P., Pitts, A. (eds.) Semantics and Logics of Computation. Publications of the Newton Institute, pp. 131–184. Cambridge University Press, Cambridge (1997)

A Characterization of Hypercoherent Semantic Correctness in Multiplicative Additive Linear Logic

Paolo Tranquilli*

Dipartimento di Matematica – Università Roma Tre
Largo S. Leonardo Murialdo, 1 – 00146 Roma – Italy
Laboratoire PPS – Université Paris Diderot - Paris 7
Case 7014 – 75205 Paris – France
tranquil@mat.uniroma3.it

Abstract. We give a graph theoretical criterion on multiplicative additive linear logic (MALL) cut-free proof structures that exactly characterizes those whose interpretation is a hyperclique in Ehrhard's hypercoherent spaces. This criterion is strictly weaker than the one given by Hughes and van Glabbeek characterizing proof nets (i.e. desequentialized sequent calculus proofs). We thus also give the first proof of semantical soundness of hypercoherent spaces with respect to proof nets entirely based on graph theoretical trips, in the style of Girard's proof of semantical soundness of coherent spaces for proof nets of the multiplicative fragment of linear logic.

1 Introduction

Proof nets (PN) are the syntax of choice for unit-free multiplicative linear logic (MLL, [6]). The robustness of such a syntax consists in its ability to quotient proofs of MLL modulo inessential rule commutation in a canonical way. Each proof net represents in fact an equivalence class of sequential proofs, and such equivalence is validated by numerous semantic models. This is obtained by building proofs in a more general syntax, *proof structures* (PS), among which one may characterize the ones that come from sequent calculus proofs via a host of well established *correctness* criterions, where *correctness* here means *sequentializability*. The most famous ones are the long trip one due to Girard [6], and the Danos-Regnier one [4] of switching acyclicity and connectedness.

Since the beginning there was a tight pairing between linear logic and the semantic model that brought the intuitions necessary for its discovery: coherent spaces. The link is the interpretation of PNs in coherent spaces via the notion of *experiment*. As PNs live inside a more general world, also the interpretation is in fact defined on PSs in general, yielding simply sets[1].

* This work was partly supported by Università Italo-Francese (Programma Vinci 2007).

[1] In fact one may regard this interpretation as living in the category **Rel** of sets and relations, though this becomes less clear in the presence of the exponential modality !.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 246–261, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Clearly the first thing to check is the semantic soundness of such an interpretation: are PNs interpreted as objects of coherent spaces, i.e. cliques? If $[\![\ ]\!]$ stands for such an interpretation, chosen by assigning a coherent space to each type literal, the following theorem addresses such a question.

Theorem 1 (Girard, [6]). *For π an MLL-PS on a sequent Γ, if π is switching acyclic then for any interpretation $[\![\ ]\!]$ we have that $[\![\pi]\!]$ is a clique in $[\![\Gamma]\!]$.*

As the sole role of switching connectedness is to invalidate the mix rule, which is accepted by coherent spaces, one drops it from the requirements.

There is now another question one can ask. As it makes sense to interpret a PS, it also makes sense to ask when such an interpretation is a clique. Such *semantic correctness*, in the case of MLL, turns out to be equivalent to the sequentializability one, as one has the following, reverse theorem.

Theorem 2 (Retoré, [16]). *For π an MLL-PS on Γ, if $[\![\pi]\!]$ is a clique in $[\![\Gamma]\!]$ for any interpretation $[\![\ ]\!]$, then π is switching acyclic.*

This strong pairing begins to break when one extends the system with units, or exponentials, or additives, which are the main concern of this work. On one side, the problem of providing unit-free multiplicative additive linear logic (MALL) a canonical syntax extending the good properties of the MLL one proved to be a longstanding question. A partial answer was given by Girard in [7] and a more satisfactory one was developed by Hughes and van Glabbeek in [8], a work which is one of our starting points. PSs are in this framework represented as sets of purely multiplicative structures, usually referred to as *slices* (see for example [9]), identified by *linkings* (i.e. sets of axioms, see Section 2 for more details). Again [8] provides a geometrical criterion, which we call the HvG one (page 259) characterizing sequentializable structures, which we call HvG-correct.

On the other hand, one would also like to extend the good semantic pairing of MLL to MALL. Coherent spaces are known to not provide the same results for MALL PSs as for MLL. In fact there is a PS, the Gustave one, which is the proof theoretical counterpart of the Gustave function G in the stable model of PCF. In the same way as G is an unsequentializable stable function, the Gustave PS which we will show in Figure 1 at page 252 is an incorrect structure which is interpreted by a clique, so that no analog of Theorem 2 is possible for MALL and coherent spaces.

The Gustave function G is however rejected by Bucciarelli and Ehrhard's strongly stable model [3], and starting from it Ehrhard developed in [5] a new model of LL extending the coherent one: the *hypercoherent spaces* (Section 2.2). One may then turn to such a model hoping for a better account of MALL. Semantic soundness clearly holds if one passes through the sequentialization theorem of [8], though a more direct proof might be desirable (we will in fact give it, by combining Proposition 17 and Theorem 11). As for the analog of Theorem 2, the Gustave PS is indeed rejected, but one stumbles anyway upon another counterexample [12], which we show in Figure 2 on page 252. It has been conjectured [12, Conjecture 70] that such fracture between MALL syntax

and hypercoherent semantics is due to the intrinsic unconnectedness of the counterexample.

Conjecture 3 (Pagani). If θ is a proof structure, and $\forall \lambda \in \theta\colon \lambda$ is switching acyclic *and connected*, and $[\![\theta]\!]$ is a hyperclique for any interpretation $[\![\ ]\!]$, then θ is HvG-correct.

We decided to "factorize" the conjecture by first finding the criterion for semantic correctness, which we call *hypercorrectness* (Definition 5). This criterion exactly characterizes the cut-free structures which have a hyperclique as interpretation. This approach has much similarity to the work of Pagani in the framework of exponential LL, where a criterion (*visible acyclicity*) is shown to characterize nets interpreted by non-uniform cliques [11] or finitary relations [13], along with interesting computational properties. More from a distance, a similarity can be established with what happened in the study of models of PCF: once it was clear that Scott-continuous functions, or even stable ones, were not fully abstract for PCF, two directions were taken. One was to refine the models (from continuity to stability and from stability to strong stability), while the other, similar to what we do here, was to find which languages were fully abstract for these same models (parallel PCF for the continuous one [15] and stable PCF for the stable one [14]). One difference is that in our work and that of [11,13] one really finds a discerning *geometrical* criterion (something that has sense because of the presence of generally "incorrect" objects, PSs) corresponding to an *algebraic* one, apparently distant (hypercliques here, finitary relations in [13]). In MALL the approach of semantic refinement is the direction taken in [2], where a proof of full completeness is given by applying an operation of double glueing on hypercoherent spaces.

Returning to the conjecture, we set out to prove

1. for θ cut-free proof structure, θ is hypercorrect iff $[\![\theta]\!]$ is a hyperclique for any interpretation;
2. for θ proof structure with $\forall \lambda \in \theta\colon \lambda$ switching connected, θ is hypercorrect iff θ is HvG-correct.

We address here point 1, proving both sides of the equivalence in Theorems 11 and 15, and leave point 2 as a further conjecture. The computational content of the criterion, along with its extension to PS with cuts, is left for future work.

Hypercorrectness uses a notion of *&-oriented cycles*: contrary to what happens in sequentalizability criterions the orientation of paths counts. There are already many hints of such behaviour relating to semantics. The visible acyclic paths employed in [11] have such feature. The works in [2] and [1] show full completeness results by employing cycles where the orientation is decided by *jumps*, though the framework of the two is Girard's non canonical proof nets. More recently, investigation on games semantics in [10] has as well brought to the fore an oriented interpretation of the acyclicity criterion in MLL PNs.

Outline. In Section 2 we define the standard notions appearing in this work. Next, in Section 3, we define hypercorrectness and prove the characterization. Finally in Section 4 we present some contour information and results.

2 The Framework

We will here introduce the main actors involved in this work: *MALL proof structures*, *hypercoherent spaces* and *experiments*.

Given a denumerable set of type variables $\mathcal{V}$, unit-free MALL formulas are generated by the grammar

$$\mathcal{F} ::= \mathcal{V} \mid \mathcal{V}^{\perp} \mid \mathcal{F} \otimes \mathcal{F} \mid \mathcal{F} \parr \mathcal{F} \mid \mathcal{F} \oplus \mathcal{F} \mid \mathcal{F} \,\&\, \mathcal{F},$$

with linear negation $(\ )^{\perp}$ defined by De Morgan dualities $(A \otimes B)^{\perp} := A^{\perp} \parr B^{\perp}$ and $(A \oplus B)^{\perp} := A^{\perp} \,\&\, B^{\perp}$ as usual[2]. Variables and their negations are **atomic**, connectives $\otimes/\parr$ are called **multiplicative**, while $\oplus/\&$ are **additive**. A sequent Γ is a multiset of formulas $A_1, \ldots, A_n$.

We will identify a formula with its graph-theoretical representation as a syntactical tree, which has a distinguished root node (the **conclusion** of the formula), logical connectives as intermediate nodes (called **links**), and atomic formulas as leaves. The term "node" will therefore indicate any of these parts, while among edges we will call the one above the root **terminal** and the ones above a given link **premises** to that link. Every edge has a subformula corresponding to it, and it is called its **type**. Different occurrences of nodes or edges will be noted by lowercase Latin letters. Two leaves are dual if their atomic formulas are dual. Sequents are likewise identified with their representation as syntactical forests. The tree structure naturally induces an (arborescent) order on links and edges, which we will denote by $\preceq$, with conclusions being minimal. For nodes a, b connected by an edge e in Γ we will write $a \rightarrow_e b$ (resp. $a \leftarrow_e b$) if e is a premise of b (resp. a). We will omit any of a, b, e if it is of no importance, so that for example $\rightarrow_e b$ means "e is a premise of b".

2.1 MALL Proof Structures

We will now define cut-free MALL PSs, mostly following [8], though some notions are here equivalently reformulated.

In the following let us fix a sequent Γ. An **axiom** is an unordered pair of dual leaves of Γ. Any set of axioms λ naturally defines a subforest of Γ which we denote by $\Gamma\restriction\lambda$, by taking $\left(\bigcup \lambda\right)\downarrow$, the set of leaves in axioms of λ down-closed with respect to $\preceq$, i.e. the subforest of Γ obtained by taking edges and links which have an axiom in λ above them. In $\Gamma\restriction\lambda$ connectives are either binary or unary. We call λ a **linking** (on Γ) if axioms in λ are pairwise disjoint and $\Gamma\restriction\lambda$ contains all conclusions of Γ, no unary multiplicative connectives $\otimes/\parr$ and no binary additive connectives $\oplus/\&$[3]. The **slice** $\mathcal{G}_\lambda$ associated to a linking λ is the graph obtained from $\Gamma\restriction\lambda$ by adding a new node for every axiom $\{a, b\}$ of λ with edges to the leaves a and b. By extending the notation, also these new nodes in $\mathcal{G}_\lambda$ are called axioms, and the new edges are premises to the leaves. The order

[2] Here and in the rest of the paper, := means "is defined as".

[3] In [8] linkings are defined as a partition over the leaves of an *additive resolution*, a notion not appearing here. The definition is clearly equivalent.

$\preceq$ is extended to $\mathcal{G}_\lambda$ by setting the axiom nodes and edges as greater than the leaves they connect (axioms are maximal, and the order is no longer aborescent).

Given Λ a set of linkings, we define $\Gamma\restriction\Lambda := \bigcup_{\lambda\in\Lambda}\Gamma\restriction\lambda$, where superposition is trivially defined as all lives inside Γ. We define the set $\&2(\Lambda)$ as the set of binary & connectives in $\Gamma\restriction\Lambda$. For two linkings $\lambda,\mu\in\Lambda$ we use the notation $\lambda \overset{w}{\nsim} \mu$ (λ **and** μ **toggle** w **uniquely**) if $\&2(\{\lambda,\mu\}) = \{w\}$ (which implies $\lambda\neq\mu$), and the notation $\lambda \overset{w}{-} \mu$ if $\lambda \overset{w}{\nsim} \mu$ or $\lambda = \mu$[4].

A **&-resolution** G of Γ is a subforest of Γ obtained by erasing from it one whole branching (whether left or right) from each & in Γ, i.e. choosing one of its premises e and erasing all edges and nodes $x \succeq e$. A linking λ **is on** a &-resolution G if $\Gamma\restriction\lambda \subseteq G$, i.e. all axioms in λ are on leaves of G.

Definition 4 (Proof structures). *A (cut-free) PS on a sequent Γ is a set θ of linkings such that for every &-resolution G of Γ there exist a unique $\lambda\in\theta$ on G (**resolution condition**).*

2.2 Hypercoherent Spaces

The first denotational semantics of linear logic were coherent spaces [6], which in fact were the mathematical notion that gave the first intuitions for linear logic. Much later, Ehrhard introduces in [5] a refinement, the *hypercoherent spaces*, which we briefly present here.

A **hypercoherent space** $\mathcal{X}$ is given by a pair $(|\mathcal{X}|, \frown_{\mathcal{X}})$ where

- $|\mathcal{X}|$ is a set called the **web** of $\mathcal{X}$.
- $\frown_{\mathcal{X}}$, called the **hypercoherence** of $\mathcal{X}$, is a *predicate* $\frown_{\mathcal{X}} \subseteq \mathcal{P}^*_{<\omega}(|\mathcal{X}|)$, the finite non-empty subsets of the web of $\mathcal{X}$, which is *reflexive* in the sense that it contains the set of singletons $\mathcal{P}_{=1}(|\mathcal{X}|)$.

The hypercoherent space as subscript of the relation is omitted if no confusion is possible. Apart from $\frown$, one defines the following relations, from which $\frown$ can be in turn recovered: strict hypercoherence $\smile := \frown\setminus\mathcal{P}_{=1}(|\mathcal{X}|)$, hyperincoherence $\asymp := \mathcal{P}^*_{<\omega}(|\mathcal{X}|)\setminus\smile$ and strict hyperincoherence $\frowny := \mathcal{P}^*_{<\omega}(|\mathcal{X}|)\setminus\frown$. The **hypercliques** of $\mathcal{X}$ are

$$\mathcal{H}(\mathcal{X}) := \{\, h\subseteq|\mathcal{X}| \mid \forall s\subseteq^*_{<\omega} h\colon \frown s \,\},$$

where $s\subseteq^*_{<\omega} h$ means that s is a finite non-empty subset of h.

All connectives of linear logic have a corresponding operation on hypercoherent spaces. We define here all of them but the exponential one which is of no interest here.

Dual: $|\mathcal{X}^\perp| := |\mathcal{X}|$, and $\frown_{\mathcal{X}^\perp} := \asymp_{\mathcal{X}}$.
Multiplicatives: $|\mathcal{X}\otimes\mathcal{Y}| = |\mathcal{X}\parr\mathcal{Y}| := |\mathcal{X}|\times|\mathcal{Y}|$, and given $s\subseteq^*_{<\omega}|\mathcal{X}|\times|\mathcal{Y}|$ we set

$$\begin{aligned}\frown_{\mathcal{X}\otimes\mathcal{Y}} s &\iff \frown_{\mathcal{X}}\pi_0(s) \text{ and } \frown_{\mathcal{Y}}\pi_1(s),\\ \frown_{\mathcal{X}\parr\mathcal{Y}} s &\iff \frown_{\mathcal{X}}\pi_0(s) \text{ or } \frown_{\mathcal{Y}}\pi_1(s),\end{aligned}$$

with π_0 and π_1 the usual left and right projections.

[4] $\lambda \overset{w}{-} \mu$ is denoted $\lambda \overset{w}{=} \mu$ in [8].

Additives: $|\mathcal{X}_0 \oplus \mathcal{X}_1| = |\mathcal{X}_0 \,\&\, \mathcal{X}_1| := |\mathcal{X}_0| + |\mathcal{X}_1|$, the disjoint sum. We denote an element of such a disjoint sum as $x.i$, with $i = 0$ or $i = 1$ and $x \in |\mathcal{X}_i|$. Given $s \subseteq^*_{<\omega} |\mathcal{X}_0| + |\mathcal{X}_1|$, let $s_i := \{\, x \in |\mathcal{X}_i| \mid x.i \in s \,\}$. Then we set

$$\supset_{\mathcal{X}_0 \oplus \mathcal{X}_1} s \iff s_i = \emptyset \text{ and } \supset_{\mathcal{X}_{1-i}} s_{1-i} \text{ for } i = 0 \text{ or } 1,$$
$$\supset_{\mathcal{X}_0 \& \mathcal{X}_1} s \iff \text{either } s_0 \neq \emptyset \text{ and } s_1 \neq \emptyset, \text{ or } s_i = \emptyset \text{ and } \supset_{\mathcal{X}_{1-i}} s_{1-i} \text{for } i = 0 \text{ or } 1.$$

Note therefore that if s_0 and s_1 are both non-empty, one automatically has $\frown_{\mathcal{X}_0 \& \mathcal{X}_1} s$ and $\smile_{\mathcal{X}_0 \oplus \mathcal{X}_1} s$ regardless of the elements of s, as it cannot be a singleton.

The operations defined above respect De Morgan's duality.

2.3 Experiments

The notion of experiments was developed by Girard in [6] to give a way to directly interpret multiplicative proof nets in coherent semantics, without passing through sequent calculus. The remainder of this section will be devoted to defining experiments on (cut-free) linkings and PSs.

Suppose given an interpretation $[\![\ ]\!]$ on type variables, i.e. a mapping from type variables to hypercoherent spaces. It can be easily extended to all formulas A by induction, chasing down all connectives and applying the corresponding operation on hypercoherent spaces. Then the interpretation of a sequent $\Gamma = A_1, \ldots A_n$ is $[\![\Gamma]\!] := \bigparr_{i=1}^{n} [\![A_i]\!]$. We disregard any problem of bracketing, and consider the web of $[\![\Gamma]\!]$ as made up of n-uples.

Given a (cut-free) linking λ on Γ, an **experiment** e on λ (notation $e : \lambda$) is a function assigning to each axiom $\ell \in \lambda$ of type $\alpha/\alpha^\perp$ a point $e(\ell) \in \big|[\![\alpha]\!]\big|$. This function is then extended by induction to every edge f of type A in $\mathcal{G}_\lambda$, so that $e(f) \in \big|[\![A]\!]\big|$:

- if A is atomic, f has an axiom $\ell \in \lambda$ above it, and one sets $e(f) := e(\ell)$;
- if A is multiplicative, f is under a $\otimes/\parr$ link with both of its premises f_0 and f_1, and one sets $e(f) := \big(e(f_0), e(f_1)\big)$;
- if A is additive, f is under a $\oplus/\&$ with only one of its premises f_i ($i = 0$ for left, 1 for right), and one sets $e(f) := e(f_i).i$.

If $f_1, \ldots, f_n$ are the terminal edges of Γ, then the **result** of the experiment e on λ is defined as $e(\lambda) := \big(e(f_1), \ldots, e(f_n)\big) \in \big|[\![\Gamma]\!]\big|$. An experiment e on a PS θ is an experiment on any of its linkings λ, with $e(\theta) := e(\lambda)$. The interpretation of a PS is then given as

$$[\![\theta]\!] := \{\, e(\theta) \mid e \text{ experiment on } \theta \,\} \subseteq \big|[\![\Gamma]\!]\big|.$$

Given experiments $e_1, \ldots, e_k$ on θ, if an edge d is in all $\mathcal{G}_{\lambda_i}$ where $e_i : \lambda_i$, then it makes sense to ask whether $\supset\{e_i(d)\}$ holds, obviously by taking as space the interpretation of the type of d.

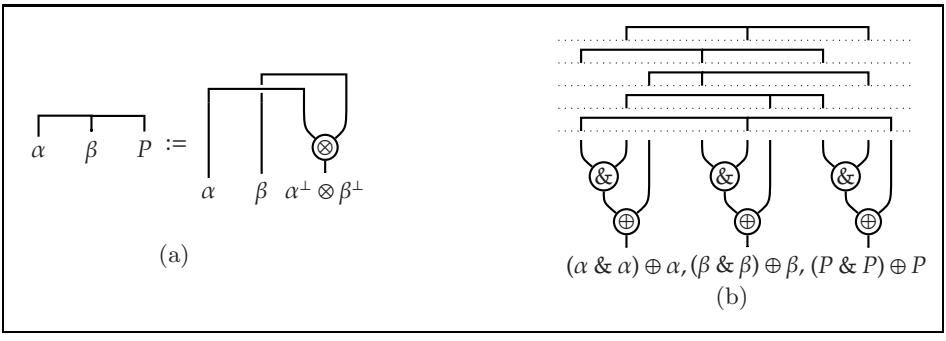


Fig. 1. The Gustave PS γ is shown in (b). P is short for $\alpha^\perp \otimes \beta^\perp$, and the three-leaves axioms shown are a short graphical representation for the trivial linking on $\alpha, \beta, \alpha^\perp \otimes \beta^\perp$, as shown in (a).

2.4 Examples

The Gustave PS γ is presented in Figure 1(b), its five linkings shown one above the other. This example is described in [8, Section 4.6.1] in the framework of Hughes and van Glabbeek PSs. It is an unsequentializable structure, as all terminal $\oplus$s are binary, so no final $\oplus$ rule may be applied in sequent calculus. In fact the HvG criterion (page 259) rejects such structure. While the interpretation of γ in coherent spaces is a clique, as coherence is checked on at most two slices at a time, $[\![\gamma]\!]$ in hypercoherent spaces is not a hyperclique.

Figure 2 shows the counterexample to hypercoherent semantic correctness being equivalent to sequentializability [12, Proposition 69]. The PS δ, whose linkings are shown in Figure 2(a), is not sequentializable as the final rule must be $\otimes$, however it cannot split the $\epsilon \oplus \epsilon, \epsilon^\perp$ part of the context as it depends on both &s. Such a dependency is registered by *jumps*, which give an illegal cycle

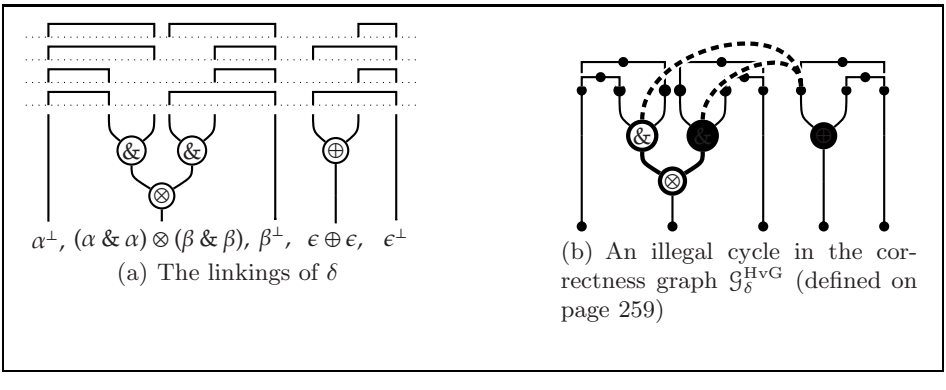


Fig. 2. The proof structure δ: an unsequentializable structure such that $[\![\delta]\!]$ is a hyperclique

in such a structure, as shown in Figure 2(b). Notice that the cycle traverses the &s in opposite directions. The interpretation $[\![\delta]\!]$ is a hyperclique because of the way binary &s entail strict coherence whatever comes above them. The slices, though switching acyclic, are not switching connected – this should always be the case for unsequentializable semantically correct structures, if the conjecture stated in point 2 of Section 1 is indeed true.

3 The Criterion

In this section we will define the criterion and then show the main results.

3.1 Hypercorrectness

We will define *correctness graphs* in the style of [8], with a substantial difference though. While jumps in [8] are drawn from the axioms, here we will draw them from the places where slices begin to differ from bottom to top. Section 4 will give equivalent forms of this criterion and a more precise comparison with the HvG criterion.

Given a set of linkings Λ, the **pre-correctness graph** $\mathcal{G}'_\Lambda$, is obtained by superposing all slices of Λ, i.e. $\mathcal{G}'_\Lambda := \bigcup_{\lambda\in\Lambda} \mathcal{G}_\lambda$. The $\Gamma \restriction \lambda$ part of each slice is inside $\Gamma \restriction \Lambda$, so in fact $\mathcal{G}'_\Lambda$ is obtained by adding axioms to it. Superposition (i.e. identification) of axiom nodes and edges happens if and only the related axiom connects the same leaves. An edge or a node in $\mathcal{G}'_\Lambda$ is said to be **total** (for Λ) if it is in all slices, i.e. in $\bigcap_{\lambda\in\Lambda} \mathcal{G}_\lambda$, **partial** otherwise. An **additive contraction**, or simply contraction, is a total non-& node with partial premises, and their set is noted as $\mathrm{contr}(\Lambda)$. Contractions are in fact binary $\oplus$s and total leaves under partial axioms.

The **correctness graph** $\mathcal{G}_\Lambda$ is obtained from $\mathcal{G}'_\Lambda$ by adding new edges, called **jumps**, from a node $c \in \mathrm{contr}(\Lambda)$ to $w \in \&2(\Lambda)$ whenever

$$\exists \lambda_1, \lambda_2 \in \Lambda \mid \lambda_1 \overset{w}{\nrightarrow} \lambda_2 \text{ and } c \in \mathrm{contr}(\{\lambda_1, \lambda_2\}).$$

A jump j from c to w is denoted $c \rightsquigarrow_j w$. Jumps are considered partial, and premises to the & they jump to. Let $\mathrm{tot}(\Lambda)$ (resp. $\mathrm{part}(\Lambda)$) denote the set of total (resp. partial) edges in $\mathcal{G}_\Lambda$.

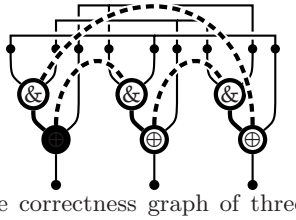
A **path** ϕ in $\mathcal{G}_\Lambda$ is a finite non-repeating sequence e_i of edges such that e_i and e_{i+1} are *adjacent*, i.e. share a node, and such that also every shared node is not repeated. As sequences, paths are *oriented*, so we can define the *source* (resp. *target*) of ϕ as the unshared node of the first (resp. last) edge in ϕ. A **cycle** is a non-empty path whose source and target coincide. We identify ϕ with the set of its edges and the nodes it traverses, so that we may write $w \in \phi$ for a node w. Paths may also be denoted with the concatenated notations for premises and jumps, as for example in $\rightarrow_e \rightarrow x \leftarrow\!\!\rightsquigarrow_j w$. Note how some node or edge names may be omitted, and recall that jumps are considered also as premises, so that in the example e may be a jump. Also arrowheads will be omitted (as in $x —_e y$) if

we do not want to specify whether the path is going upwards or downwards. For $e \in \phi$, write $\downarrow e \in \phi$ (resp. $\uparrow e \in \phi$) if e is traversed going down (resp. up), i.e. if d is traversed towards (resp. from) the node it is premise of. A path **bounces** on a node x if it contains a segment of shape $\to x \leftarrow$ or $\leftarrow x \to$. Cycles are to be considered bouncing on their source/target if their first and last edges are both immediately above or below it. A path or cycle is **switching** if it never bounces on a $\parr$ or &.

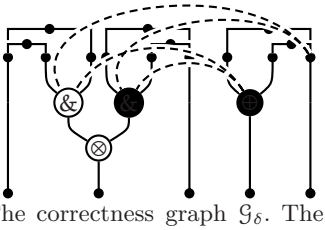
Finally, a switching path ϕ is said to be &**-oriented** if it changes from being partial to total on &s only and does viceversa on contractions only, i.e. for every $-_e\, x\, -_f$ in ϕ, if $e \in \mathrm{part}(\Lambda)$ and $f \in \mathrm{tot}(\Lambda)$ (resp. viceversa) then $x \in \&2(\Lambda)$ (resp. $x \in \mathrm{contr}(\Lambda)$). Furtherly, two paths ϕ and ψ are said to be **bounce-compatible** if whenever ϕ and ψ both bounce on the same total tensor or axiom x, traversing its adjacent edges a, b, then a, b appear in the same order in ϕ and ψ. A union of paths is said to be bounce-compatible if its paths are pairwise bounce-compatible.

Definition 5 (Hypercorrectness). *A proof structure θ is hypercorrect if for every $\Lambda \subseteq \theta$ and every bounce-compatible non-empty union S of &-oriented cycles in $\mathcal{G}_\Lambda$, there is $w \in \&2(\Lambda)$ such that $w \notin S$.*

Note that for any λ, as the whole $\mathcal{G}_{\{\lambda\}} = \mathcal{G}_\lambda$ is total and lacks binary &s, this criterion entails the absence of switching cycles, i.e. multiplicative correctness (without connectedness) of every linking. Notice also that dropping bounce-compatibility and &-orientedness of S amounts to reverting to the HvG criterion (see page 259). Revisiting the examples shown in Figures 1 and 2, we show in Figures 3(a) and 3(b) respectively one of their correctness graphs.



(a) The correctness graph of three linkings of the Gustave PS. Only three out of six jumps are shown, and axiom nodes are omitted. The cycle shown is strictly &-oriented (page 255).



(b) The correctness graph $\mathcal{G}_\delta$. The only way to form a cycle would be to bounce on the tensor, but that would not be a &-oriented one.

Fig. 3. Two examples of correctness graphs. The first one shows the rejection of the Gustave PS by the criterion, while the second structure is hypercorrect. Leaf nodes and axiom nodes are marked by •s.

3.2 Hypercorrectness Implies Hypercoherence

We will devote this section to the proof of Theorem 11, the analog of Theorem 1.

Let us fix in the following θ a cut-free PS on a sequent Γ. A set of linkings $\Lambda \subseteq \theta$ is said to be **saturated** if for every $\lambda \in \theta \setminus \Lambda$, $\Lambda \cup \{\lambda\}$ has more binary &s than Λ. A &-oriented path or cycle ϕ is **strictly &-oriented** if it always descends on partial edges, i.e. if $e \in \phi$, $e \in \mathrm{part}(\Lambda)$, then $\downarrow e \in \phi$. Note that this implies not passing any partial axioms. The following are two basic lemmas needed for our proofs later.

Lemma 6. *For Λ saturated, every $c \in \mathrm{contr}(\Lambda)$ has a jump $c \rightsquigarrow$ in $\mathcal{G}_\Lambda$.*

Lemma 7. *If θ is hypercorrect and $\Lambda \subseteq \theta$ is saturated, then every non-empty bounce-compatible union S of strictly &-oriented cycles has a jump out of it, i.e. $\exists w \in \&2(\Lambda) \setminus S$ and $c \in \mathrm{contr}(\Lambda) \cap S$ such that $c \rightsquigarrow w \in \mathcal{G}_\Lambda$.*

The following is the main lemma opening us the way for Theorem 11.

Lemma 8. *Let θ be a hypercorrect PS on a sequent Γ, $e_1, \ldots, e_n$ experiments on θ, such that $\smile\{e_i(f)\}$ on a terminal edge f. Then there exist $\Lambda \subseteq \theta$ and a strictly &-oriented path ϕ in $\mathcal{G}_\Lambda$ starting with f and ending with a terminal wire f' such that $\frown\{e_i(f')\}$.*

Proof. Consider Λ the minimal saturated set of linkings containing those on which experiments e_i are taken. By minimality binary &s are the same. We will give a precise algorithm which will build the path ϕ. The base step of such an algorithm is the non-deterministic function NEXT, taking as inputs a direction ϵ which can be $\uparrow$, $\downarrow$ and an edge $d \in \mathcal{G}_\Lambda$ such that

1. if $d \in \mathrm{part}(\Lambda)$ then $\epsilon = \downarrow$;
2. if $d \in \mathrm{tot}(\Lambda)$ and $\epsilon = \uparrow$, then $\smile\{e_i(d)\}$;
3. if $d \in \mathrm{tot}(\Lambda)$ and $\epsilon = \downarrow$, then $\frown\{e_i(d)\}$.

The output will be a direction ϵ' and an edge d' with the same properties and such that dd' is a path with $\epsilon d, \epsilon' d' \in dd'$. Let us define NEXT by the three cases described above.

1. Let $\rightarrow_d x$. If $x \in \mathrm{part}(\Lambda)$, then $x \rightarrow_{d'}$ with $d' \in \mathrm{part}(\Lambda)$, and let $\mathrm{NEXT}(\downarrow d) := \downarrow d'$. If $x \in \mathrm{tot}(\Lambda)$, then either $x \in \&2(\Lambda)$, in which case $x \rightarrow_{d'}$ and $\mathrm{NEXT}(\downarrow d) := \downarrow d'$ (note $\smile\{e_i(d')\}$ as &s binary in Λ are also binary in the linkings on which the experiments are taken), or $x \in \mathrm{contr}(\Lambda)$. By Lemma 6, there is $x \rightsquigarrow_{d'}$, and we set $\mathrm{NEXT}(\downarrow d) := \downarrow d'$.
2. Let $\leftarrow_d x$. If $x \in \mathrm{contr}(\Lambda)$, then proceed as the above case, setting $\mathrm{NEXT}(\uparrow d) := \downarrow d'$ with $x \rightsquigarrow_{d'}$. Otherwise let us define NEXT by cases on the nature of x:
   - **axiom:** x is total, and $\leftarrow_d x \rightarrow_{d'}$. Set $\mathrm{NEXT}(\uparrow d) := \downarrow d'$. The property is preserved as the value of the experiments on the two edges is the same and their types are dual;
   - **leaf or unary additive:** there is a unique $x \leftarrow_{d'}$, $d' \in \mathrm{tot}(\Lambda)$ with the same incoherence of d, so we set $\mathrm{NEXT}(\uparrow d) := \uparrow d'$;
   - **binary with:** this case is impossible, because $\leftarrow_d x$, as noted above, implies $\frown\{e_i(d)\}$;

par: we have $\to_{d_0} x \leftarrow_{d_1}$ the two premises of x, and as $\smile \{e_i(d)\}$ and $\{e_i(d_j)\} = \pi_j\{e_i(d)\}$, we have $\smile\{e_i(d_j)\}$ on both, and may set $\textsc{next}(\uparrow d) := \uparrow d_j$ for any of the two js;

tensor: we have $\to_{d_0} x \leftarrow_{d_1}$, and as $\frown\{e_i(d)\}$, one of the two projections $\{e_i(d_j)\}$ must be strictly hyperincoherent, and we may set $\textsc{next}(\uparrow d) := \uparrow d_j$ with such a j.

3. Let $\to_d x$. We have that x and all its adjacent edges are total, so x cannot be an axiom, a contraction or a binary &. Again, let us proceed by cases.

   leaf or unary additive: $x \to_{d'}$, and trivially we can set $\textsc{next}(\downarrow d) := \downarrow d'$;

   par: $\to_d x \to_{d'}$, and as $\frown\{e_i(d)\}$, then $\frown\{e_i(d')\}$, and we set $\textsc{next}(\downarrow d) = \downarrow d'$;

   tensor: let $\to_d x \leftarrow_{d'}$, d' the other premise of x, and $x \to_{d''}$; if $\frown\{e_i(d'')\}$, then set $\textsc{next}(\downarrow d) := \downarrow d''$; otherwise, necessarily $\smile\{e_i(d')\}$, and we may set $\textsc{next}(\downarrow d) := \uparrow d'$.

We say that a path $f_0 f_1 \cdots f_k$ is **admissible** if it is built by an iteration of NEXT, i.e. $f_{j+1} = \textsc{next}(f_j)$, with its first edge f_0 either a terminal one or also an output of NEXT, i.e. such that $\exists f_{-1} \mid f_0 = \textsc{next}(f_{-1})$.

Fact 9

- *The composition $\phi :: \psi$ of two admissible paths ϕ and ψ is admissible;*
- *all admissible paths are strictly &-oriented and bounce-compatible between them;*
- *in particular, an admissible path ending on one of its own nodes forms a strictly &-oriented cycle.*

Another non-deterministic function we will use is JUMP, which takes as input a non-empty union S of admissible cycles (therefore a bounce-compatible union of &-oriented cycles) and gives $\downarrow j$, where j is a jump out of S as described by Lemma 7. Notice that all jumps can always be outputted by NEXT: they are therefore admissible, and may be appended to an admissible path preserving such property.

Finally, let W and S be variables for sequences of binary &s and unions of admissible cycles. W_j (resp. S_j) will denote the j-th element of W (resp. S), with W starting from 1 and S from 0, and both ending in k (we will always use k for the size of W). The algorithm will build an admissible ϕ so that at all times W are the &s in ϕ which are not in any cycle of S. In a way W_i will be "in between" S_{i-1} and S_i (W_i will be generated by $\textsc{jump}(S_{i-1})$). Also, the algorithm will make it so that all &s touched at some time by ϕ are partitioned by W and &s in $\bigcup S_j$.

The following is a schematic example of how the algorithm works. The aim is that starting from the terminal edge f given by hypothesis the path ϕ eventually ends on another one, the f' of the thesis. Suppose that following NEXT we end up in a cycle χ_1. Applying JUMP to it, we can backtrack and jump to a & w_1 outside it and keep going (at this point, we set $W = \langle w_1 \rangle$ and $S = \langle \chi_1, \emptyset \rangle$). Now suppose the path cycles again, intersecting itself *after* w_1, forming χ_2. If

we applied JUMP to $\chi_1 \cup \chi_2$, it could answer the same jump to w_1 it told before, and it would be useless. In such a case we obtain $w_2 = \text{JUMP}(\chi_2)$, and if w_2 is "fresh" we set $W = \langle w_1, w_2 \rangle$ and $S = \rangle \chi_1, \chi_2, \emptyset \rangle$. If then at a certain point we end up again on ϕ before w_1 (and w_2) forming χ_3 then we may safely collapse the three cycles and apply JUMP to $\chi_1 \cup \chi_2 \cup \chi_3$ without risking a useless answer. W becomes $\langle w_3 \rangle$, $S = \langle \chi_1 \cup \chi_2 \cup \chi_3, \emptyset \rangle$ (note w_1, w_2 are in it, so that we may say that they are somehow "burnt" in this process).

Going back to the preliminary description of the algorithm, every time ϕ arrives to a node $x \notin \phi$, we store in x the path ϕ as it is at that moment, calling it the **history** of x. We are now ready to present the whole algorithm. Recall that by hypotheses there is a terminal edge f such that $\smile\{e_i(f)\}$, so we can apply NEXT to $\uparrow f$. The target of ϕ is denoted by $\mathrm{t}(\phi)$.

1. Start by setting $\phi := f$, $\epsilon d := \uparrow f$, $W := \langle\rangle$, $S := \langle \emptyset \rangle$ ($k := 0$).
2. Repeat...
   (a) If $\mathrm{t}(\phi) \in \bigcup S_j$ then $\mathrm{t}(\phi) \in \chi$ with χ a cycle. Let ψ be the smallest portion of χ that starting from x crosses ϕ again. $\psi = \langle\rangle$ if $\mathrm{t}(\phi) \in \phi$, and $\psi = \chi$ if χ does not intersect ϕ elsewhere. Set $\phi := \phi :: \psi$ (note that the following condition will be automatically satisfied).
   (b) If $\mathrm{t}(\phi) \in \phi$ then let χ be the cycle thus formed, and do the following steps...
      i. Let i be such that W_i is the last W_j strictly before $\mathrm{t}(\phi)$ in ϕ if one exists, $i := 0$ otherwise (note χ contains all W_j with $j > i$).
      ii. $S_i := \bigcup_{j=i}^{k} S_j \cup \chi$, and erase from W and S all following elements (in fact, set $k := i$).
      iii. $\epsilon d := \text{JUMP}(S_i) = \text{JUMP}(S_k)$, and let $c \rightsquigarrow_d w$ (note that $w \notin S_k$). Set ϕ to the history of c, and then append d to it.
   (c) ...else, do the following.
      i. If $\mathrm{t}(\phi) \in \&2(\Lambda)$, then set $W := W :: \mathrm{t}(\phi)$ and $S := S :: \emptyset$ (and in fact $k := k + 1$).
      ii. $\epsilon d := \text{NEXT}(\epsilon d)$ and $\phi := \phi :: d$.
3. ...until $t(\phi)$ is a conclusion.

Fact 10. *The algorithm shown above always terminates.*

Proof (sketch). One shows that the following measure strictly decreases for lexicographic ordering:

$$\mu := (\#\,\&2(\Lambda) - \#\,\&2(\textstyle\bigcup S_j) - k, \#\,\&2(\Lambda) - \#\,\&2(S_k \cup \{\mathrm{t}(\phi)\}), |\mathcal{G}_\Lambda| - |\phi|)$$

where $\&2(T) := \&2(\Lambda) \cap T$ and the size $|\ |$ counts the edges. The component μ_1 decreases strictly in step 2(c)i, else μ_2 does it in step 2(b)iii, else μ_3 does it in step 2(c)ii.

Therefore the lemma is proved: if f' is the terminal edge with which ϕ ends, then $\downarrow f' \in \phi$, and by the properties of NEXT we have $\smile\{e_i(f')\}$. □

Theorem 11. *If θ is a hypercorrect PS on a sequent Γ, then $[\![\theta]\!]$ is a hyperclique in $[\![\Gamma]\!]$ for every interpretation $[\![\ ]\!]$.*

Proof. Let $[\![\ ]\!]$ be any interpretation, and let $c \subseteq^*_{<\omega} [\![\theta]\!]$. By definition $c = \{\, e_1(\theta), \dots, e_n(\theta) \,\}$. Suppose $\neq c$, i.e. c is not a singleton. Then there is a conclusion c of Γ such that $\neq\{e_i(c)\}$. Either $\frown\{e_i(c)\}$ which implies $\frown\{e_i(\theta)\}$, or else $\smile\{e_i(\theta)\}$, which by above Lemma 8 entails the existence of another conclusion c' with $\frown\{e_i(c')\}$ which also implies $\frown\{e_i(\theta)\}$. In any case, coherence of c is proved, and therefore $[\![\theta]\!]$ is a hyperclique. □

3.3 Hyperincorrectness Implies Hyperincoherence

This section will prove Theorem 15, the analog of Retoré's theorem. This will be done using the following lemma, a sort of dual to Lemma 8.

Lemma 12. *Let θ be a set of linkings over Γ, f_1 and f_2 two terminal edges, and $\phi_1, \dots, \phi_k$ pairwise bounce-compatible and &-oriented paths in $\mathcal{G}_\theta$ such that every ϕ_i is either a cycle or a path starting with f_1 and ending with f_2. Suppose at least one of the ϕ_js is of the second kind, and $\&2(\theta) \subseteq \bigcup_j \phi_j$. Then there exist an interpretation $[\![\ ]\!]$ and experiments $e_1, \dots, e_n$ such that $\smile\{e_i(f_1)\}$, and $\asymp\{e_i(c)\}$ for every terminal edge $f \neq f_1, f_2$.*

Proof. The interpretation we define is $[\![\ ]\!]_{\mathcal{X}}$, which maps all literals to a space $\mathcal{X}$. We give a sketch on how to define such a space and the experiments e_i.

Fact 13. *There is a hypercoherent space $\mathcal{X}$ and experiments $e_1, \dots, e_n$ relative to $[\![\ ]\!]_{\mathcal{X}}$ with $n = \max(\#\,\Lambda, 2)$ such that*

(E1) for each total axiom ℓ such that there is ϕ_j traversing it, let a be the axiom edge under ℓ with $\uparrow a \in \phi_j$[5]: then $\smile\{e_i(a)\}$;
(E2) for each other total axiom we have $=\{e_i(\ell)\}$;
(E3) for each contraction leaf x, if f is the edge under it then $\smile\{e_i(f)\}$.

Proof (sketch). The aim is to define an experiment e_i on each λ_i (one sets $\lambda_1 = \lambda_2$ in the degenerate case $\#\,\Lambda = 1$). E1 can be easily achieved if $\mathcal{X}$ contains at least a strict coherent pair and a strict incoherent one, by making the experiments give one or the other depending on the direction of the paths traversing such an ℓ wrt duality. The problems come from E3, as there may be partial axioms linking two contractions. These are solved by building an ad-hoc space $\mathcal{X}$ having as web such partial axioms plus three distinguished points $\mathtt{c}, \mathtt{i}, \mathtt{n}$ (for coherent, incoherent and neutral).

Fact 14. *From properties E1–3 listed in Fact 13 we can deduce the following ones:*

(P1) for every $d \in \mathrm{tot}(\Lambda)$, if $\exists d' \succeq d$ and j such that $d' \in \phi_j$, then $\neq \{e_i(d)\}$, i.e. it is not a singleton;
(P2) for every $d \in \mathrm{tot}(\Lambda)$, if $\forall j\colon \downarrow d \notin \phi_j$, i.e. d is not traversed downward by any ϕ_j, then $\asymp\{e_i(d)\}$.

[5] Notice that this identifies a regardless of ϕ_j: if two of the paths traverse the axiom ℓ, they cannot do it in opposite direction because of bounce-compatibility.

Proof (sketch). The proof of P2 is done by an easy induction on the type of the edge, by regarding what happens above it. In the tensor case bounce compatibility plays a central role in order to apply i.h. Binary additive cases are trivial: for & the hypothesis never applies, for $\oplus$ the thesis always applies.

These two properties immediately entail the result, as by hypotheses $\forall j\colon \downarrow f_1 \notin \phi_j$ and $\exists j \mid f_1 \in \phi_j$, so by P1 and P2 combined we have $\smile\{e_i(f_1)\}$. Again by hypotheses for every $f \neq f_1, f_2$ we have $\downarrow f \notin \phi_j$ for any j, so that P2 gives the rest of the result. □

With the above lemma at hand, we can easily prove the second main theorem of this work. Note how we weaken the hypothesis without asking the resolution condition (Definition 4).

Theorem 15. *If θ is a set of linkings, and for every $[\![\ ]\!]$ we have that $[\![\theta]\!]$ is a hyperclique, then θ is hypercorrect.*

Proof (sketch). One shows that if θ is invalidated by a union S of cycles in $\mathcal{G}_\Lambda$ then one can build hyperincoherent experiments on Λ, by an induction on the number of links in $\mathcal{G}_\Lambda$. One disassembles $\mathcal{G}_\Lambda$ one terminal link at a time, until one arrives to break S by taking out a $\otimes$. This makes the structure fall into the hypotheses of Lemma 12, and the result easily follows by the law of hypercoherence on $\otimes$.

4 Compendium

Equivalent criterions. We define the **partial contractions** as the set $\mathrm{pcontr}(\Lambda) := \bigcup_{\lambda,\mu\in\Lambda} \mathrm{contr}(\{\lambda,\mu\})$, and the graph $\mathcal{G}^{\mathrm{p}}_\Lambda$ with jumps from $\mathrm{pcontr}(\Lambda)$ with the same rule. In fact $\mathrm{contr}(\Lambda) \subseteq \mathrm{pcontr}(\Lambda)$ and $\mathcal{G}^{\mathrm{p}}_\Lambda = \bigcup_{\lambda,\mu\in\Lambda} \mathcal{G}_{\{\lambda,\mu\}}$, with jumps identified iff they have same target and same source.

Proposition 16. *Hypercorrectness (Definition 5) is equivalent to having any number of its parts substituted in the following ways.*

1. *bounce-compatibility can be strengthened with plain compatibility, i.e. ϕ and ψ are compatible iff whenever $e \in \phi \cap \psi$ then ϕ and ψ traverse e in the same direction;*
2. *&-orientedness can be strengthened with strict &-orientedness (defined on page 255);*
3. *the condition asking the presence of $w \in \&2(\Lambda)$ outside S can be strengthened to be the presence of $w \in \&2(\Lambda) \cap \mathrm{tot}(\Lambda)$ outside S,*
4. *the graph $\mathcal{G}^{\mathrm{p}}_\Lambda$ can replace $\mathcal{G}_\Lambda$.*

Comparison with sequentializability. For $\Lambda \subseteq \theta$, let $\mathcal{G}^{\mathrm{HvG}}_\Lambda$ be the correctness graph of Λ as defined in [8]. The only difference is where jumps are drawn from. In $\mathcal{G}^{\mathrm{HvG}}_\Lambda$ one adds to $\mathcal{G}'_\Lambda$ a jump $a \rightsquigarrow w$ for every a leaf such that there are $\lambda \overset{w}{\nsim} \mu$ with an axiom $\ell \in \lambda \setminus \mu$ above a. Then the MIX-sequentializability criterion [8] is

$$\forall \Lambda \subseteq \theta\colon \exists w \in \&2(\Lambda) \mid \ w \text{ is in no switching cycle of } \mathcal{G}^{\mathrm{HvG}}_\Lambda. \qquad \text{(HvG)}$$

This form is clearly equivalent to asking that each union of switching cycles has a & outside it. We can directly infer hypercorrectness from the HvG criterion.

Proposition 17. *Every sequentializable PS is hypercorrect.*

Proof (sketch). A direct proof can be given by translating each strictly &-oriented cycle in $\mathcal{G}_A$ into one in $\mathcal{G}_A^{\mathrm{HvG}}$ containing the same &s. Each jump in $\mathcal{G}_A$ can be substituted with at least a path not intersecting the cycle and going from the contraction to a leaf jumping to the same & in $\mathcal{G}_A^{\mathrm{HvG}}$.

There is also a restating of the HvG criterion using jumps of $\mathcal{G}_A^{\mathrm{P}}$, though the not so trivial proof of equivalence is beyond our scope here, and will be detailed in future work. This may lead to employ "cleaner" correctness graphs, having in general fewer jumps, and could possibly open the way for a richer syntax (non-η-expanded proof nets, second order and/or exponential boxes).

Cut reduction. The study of the computational significance of hypercorrectness is left for future work. The main point is to give a good definition of jumps in the presence of cuts and prove stability under cut reduction. Already in the context of the HvG criterion, the latter is very delicate. One will probably have to tweak the criterion via its equivalent versions. Clearly this issue is now the most important one for this criterion. One expects, as it happens for visible acyclicity in [13] or for the PCF variants of [15,14], that such a semantic correctness is not just a static characterization but also has a dynamic content, possibly shedding light on new computational aspects of both syntax and semantics.

Acknowledgements. The author would like to especially thank Michele Pagani for the exhausting but exhaustive conversations on the topic. Thanks also to Paul-André Melliès for insightful conversations, and to Lorenzo Tortora de Falco and Antonio Bucciarelli for their helpful hints.

References

1. Abramsky, S., Melliès, P.-A.: Concurrent games and full completeness. In: LICS, pp. 431–442. IEEE Computer Society Press, Los Alamitos (1999)
2. Blute, R., Hamano, M., Scott, P.J.: Softness of hypercoherences and MALL full completeness. Ann. Pure Appl. Logic 131(1-3), 1–63 (2005)
3. Bucciarelli, A., Ehrhard, T.: Sequentiality and strong stability. In: LICS, pp. 138–145. IEEE Computer Society Press, Los Alamitos (July 1991)
4. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Mathematical Logic 28, 181–203 (1989)
5. Ehrhard, T.: Hypercoherence: A strongly stable model of linear logic. In: Advances in Linear Logic, pp. 83–108. Cambridge University Press, Cambridge (1995)
6. Girard, J.-Y.: Linear logic. Th. Comp. Sc. 50, 1–102 (1987)
7. Girard, J.-Y.: Proof-nets: the parallel syntax for proof-theory. In: Logic and Algebra. Lecture Notes in Pure and Appl. Math, vol. 180, pp. 97–124 (1996)
8. Hughes, D., van Glabbeek, R.: Proof nets for unit-free multiplicative-additive linear logic. In: LICS, pp. 1–10. IEEE Computer Society Press, Los Alamitos (2003)

9. Laurent, O., de Falco, L.T.: Slicing polarized additive normalization. In: Linear Logic in Computer Science, pp. 247–282 (2004)
10. Melliès, P.-A., Mimram, S.: Asynchronous games without alternation (submitted, 2008)
11. Pagani, M.: Acyclicity and coherence in multiplicative and exponential linear logic. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 531–545. Springer, Heidelberg (2006)
12. Pagani, M.: Proof nets and cliques: towards the understanding of analytical proofs. PhD thesis, Università Roma Tre / Université Aix-Marseille II (April 2006)
13. Pagani, M.: Visible acyclic nets: between interaction and semantics (in preparation, 2008)
14. Paolini, L.: A stable programming language. Inf. Comput. 204(3), 339–375 (2006)
15. Plotkin, G.D.: Lcf considered as a programming language. Theor. Comput. Sci. 5(3), 225–255 (1977)
16. Retoré, C.: A semantic characterisation of the correctness of a proof net. Mathematical Structures in Computer Science 7(5), 445–452 (1997)

An Indexed System for Multiplicative Additive Polarized Linear Logic

Masahiro Hamano[1,⋆] and Ryo Takemura[2]

[1] National Institute of Informatics, Tokyo 101-8430, Japan
and SFC Research Institute, Fujisawa 252-8520, Japan
hamano@jaist.ac.jp
[2] Department of Philosophy, Keio University, Tokyo 108-8345, Japan
takemura@abelard.flet.keio.ac.jp

Abstract. We present an indexed logical system $\mathsf{MALLP}(I)$ for Laurent's multiplicative additive polarized linear logic (MALLP) [14]. The system is a polarized variant of Bucciarelli-Ehrhard's indexed system for multiplicative additive linear logic [4]. Our system is derived from a web-based instance of Hamano-Scott's denotational semantics [12] for MALLP. The instance is given by an adjoint pair of right and left multi-pointed relations. In the polarized indexed system, subsets of indexes for I work as syntactical counterparts of families of points in webs. The rules of $\mathsf{MALLP}(I)$ describe (in a proof-theoretical manner) the denotational construction of the corresponding rules of MALLP. We show that $\mathsf{MALLP}(I)$ faithfully describes a denotational model of MALLP by establishing a correspondence between the provability of indexed formulas and relations that can be extended to (non-indexed) proof-denotations.

1 Introduction

In their study of logical relations and the denotational completeness of linear logic (LL), Bucciarelli and Ehrhard [4] introduced an indexed system $\mathsf{MALL}(I)$ for multiplicative additive linear logic (MALL). In their sequel [5], this system was extended into full fragment $\mathsf{LL}(I)$. The status of this indexed syntactical system is noteworthy as it stems from relational semantics **Rel**, which is one of the simplest denotational semantics for LL. Bucciarelli-Ehrhard's indexed system is designed so that each formula corresponds to a relation and each logical rule corresponds to a denotational interpretation of the corresponding rule in LL. The crucial ingredient for this correspondence is the *domains* of formulas: Each formula A of the indexed system is equipped with a domain $d(A)$ which enumerates the *locations* of points in the corresponding relation on $|A|$. Their indexed system enjoys *basic property*, which establishes a relationship between the provability of indexed formulas and the sub-definability of the corresponding relations in the denotational semantics of LL. Later A. Bruasse-Bac [3] extended the indexed

⋆ The first author is supported by Grant-in-Aid for Scientific Reserach (C) (19540145) of JSPS.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 262–277, 2008.
© Springer-Verlag Berlin Heidelberg 2008

system to the second order by adapting relational semantics to Girard's objects of variable type.

Another logical framework in which locations play a key role is that of Girard's ludics [11]. In ludics one abstracts locations, where syntactical formulas give its occurrences through construction of proofs. Several similarities (though not rigorous) have been noticed with the indexed system discussed above. Among them, one observes a common idea in the underlying *hypersequentialized calculus* [10] on which ludics is founded. In the hypersequenatialized calculus, which is a variant of MALL, each formula is equipped with a coherent space (rather than the more primitive notion of relation), and each inference rule is defined in terms of the construction of cliques for the equipped spaces. While sharing such similar syntactical construction with reflecting semantics, the key ingredient, peculiar to ludics, is polarity (see [11]). Polarity was introduced by Girard [8]. Through Laurent's formalization of polarized linear logic LLP [14], polarity turns out to be an important parameter controlling linear proof-theory. Most fundamentally, polarity enables categorization of Andreoli's [1] dual properties of *focalization* and *reversibility* of connectives for proof-search in LL. In polarized linear logic, reversible and focusing connectives are characterized simply as *negative* and *positive*. As is the case with ludics, polarity is also a crucial tool for handling locations game-theoretically. Laurent [15] establishes how polarity dominates game-theoretical computational models arising from LL. Polarity and locations are becoming a crucial tandem for understanding the computational meaning of LL.

Given the above, a natural question arises: Is there any polarized variant of Bucciarelli-Ehrhard's indexed system that naturally accommodates polarity in its indexes? The existence of such a system would guarantee that polarity is a stable core controlling both syntax and semantics uniformly, whose combination is at the heart of indexed systems. In this paper, we answer this question affirmatively by presenting an indexed system MALLP(I) for the multiplicative additive fragment MALLP [14] of Laurent's LLP. Our indexed system is designed by means of multi-pointed relational semantics, a web-based instance of Hamano-Scott's denotational semantics [12] for MALLP. The cornerstone of our multi-pointed relational semantics is a pair of contravariant categories $\mathbf{PRel}_l$ and $\mathbf{PRel}_r$. Left (resp. right) multi-pointed relational semantics $\mathbf{PRel}_l$ (resp. $\mathbf{PRel}_r$) consist of multi-pointed sets (i.e., sets with distinguished multi-points) and of relations preserving the distinguished elements from left (resp. from right). Polarity shifting operators are then interpreted as a pair of adjoint functors between the contravariant pair. In addition to the adjunction, the usual relations provide *bimodule* $\widehat{\mathbf{Rel}}$ so that it is closed under left (resp. right) compositions from $\mathbf{PRel}_r$ (resp. $\mathbf{PRel}_l$). Being a polarized variant of **Rel**, our framework $(\langle \mathbf{PRel}_l, \mathbf{PRel}_r \rangle, \widehat{\mathbf{Rel}})$ provides one of the simplest denotational semantics for MALLP.

Our MALLP(I), designed from multi-pointed relational semantics, is a polarized variant of Bucciarelli-Ehrhard's MALL(I): the usual multiplicative additive rules for the former coincide with those for the latter under the polarity constraint. It is remarkable that in our MALLP(I) there arise, corresponding to $\downarrow$,

parameterized $\downarrow_K$-rules with subsets K's of I. Each $\downarrow_K$-rule comes equipped with a side condition on domains by reflecting the corresponding categorical adjunction. In $\mathsf{MALLP}(I)$ polarity behaves compatibly with indexes since focusing/reversible properties are captured by indexed positive/negative connectives. $\mathsf{MALLP}(I)$ formulas correspond bijectively to relations arising in our relational denotational semantics. The main goal of this paper is to establish a basic property (Theorem 1), that is a polarized version of Bucciarelli-Ehrhard's property established in [4]. This basic property states that a family of points is contained in a (denotation of) MALLP proof if and only if the corresponding $\mathsf{MALLP}(I)$ formula is provable in the indexed system.

2 Multi-pointed Relational Semantics for MALLP

In this section, we introduce *multi-pointed relational* semantics, which is a variant of relational semantics. Multi-pointed relations are shown to provide a simple denotational semantics for polarized multiplicative additive linear logic (MALLP). (See [14] for the syntax of MALLP.) This is a polarized analogy of the category **Rel** of relations, which, as is well-known, provides one of the simplest denotational semantics for usual multiplicative additive linear logic (MALL) (see [5,2] for **Rel**). Let us begin by defining a pair of categories $\mathbf{PRel}_r$ and $\mathbf{PRel}_l$ of right and left multi-pointed relations. The right/left pair corresponds to negative/positive polarity of MALLP.

Notation: When X and Y are sets, we denote by $X \times Y$ the cartesian product of them; and by $X + Y$ the disjoint union of them, i.e., $\{1\} \times X \cup \{2\} \times Y$.

Definition 1 ($\mathbf{PRel}_r$ and $\mathbf{PRel}_l$). The categories $\mathbf{PRel}_r$ of right-multi-pointed relations and $\mathbf{PRel}_l$ of left-multi-pointed relations are defined as follows:

An object A is a pair $(|A|, \mathsf{mp}(A))$, where $|A|$ is a set called the web of A, and $\mathsf{mp}(A)$ is a finite subset of $|A|$. Moreover if $|A| \neq \emptyset$, then $\mathsf{mp}(A) \neq \emptyset$. Each element of $\mathsf{mp}(A)$ is called a distinguished element of A.
A morphism from A to B is a relation $R \subseteq |A| \times |B|$ which satisfies

$$\mathsf{mp}(A) = R[\mathsf{mp}(B)] \text{ for } \mathbf{PRel}_r$$
$$[\mathsf{mp}(A)]R = \mathsf{mp}(B) \text{ for } \mathbf{PRel}_l$$

where, for sets $X \subseteq |A|$ and $Y \subseteq |B|$, and for a relation $R \subseteq |A| \times |B|$,

$$R[Y] = \{a \mid \exists b \in Y, (a,b) \in R\} \quad \text{and} \quad [X]R = \{b \mid \exists a \in X, (a,b) \in R\}.$$

Compositions for each category are *relational* so that given $R : A \to B$ and $S : B \to C$, $S \circ R = \{(a,c) \mid \exists b \in B,\ (a,b) \in R \text{ and } (b,c) \in S\} : A \to C$. This composes in each categories because it holds that $R[S[\mathsf{mp}(C)]] = (S \circ R)[\mathsf{mp}(C)]$ and $[[\mathsf{mp}(A)]R]S = [\mathsf{mp}(A)](S \circ R)$.

There are obviously forgetful functors $|\ |$ both from $\mathbf{PRel}_r$ and $\mathbf{PRel}_l$ to the category **Rel** of relations.

The two categories $\mathbf{PRel}_l$ and $\mathbf{PRel}_r$ are contravariantly equivalent. This is given by $(\ )^\perp$, which leaves the objects invariant but reverses the relations and compositions:

$$(\ )^\perp : (\ \mathbf{PRel}_r)^{op} \simeq \ \mathbf{PRel}_l$$

Starting from the contravariantly equivalent pair, we give a denotational semantics for polarized MALL by the following Definition 2. See Section 3 and Definition A.1 in Hamano-Scott [12] for the definition of categorical semantics for MALLP. Note that, in the following, their general framework based on Definition A.1 is adapted so that *bimodules* play a role between the two categorical pair. See also Cockett-Seely [6] for bimodules in polarized category. Our bimodule may be seen as a concrete instance of Example 3.0.2 of [6].

Definition 2 (Polarity-changing functors and bimodule)

– *The functors* $\uparrow$ *and* $\downarrow$ are defined as follows:

$$\downarrow : \ \mathbf{PRel}_l \longrightarrow \ \mathbf{PRel}_r \tag{1}$$

$$\uparrow : \ \mathbf{PRel}_r \longrightarrow \ \mathbf{PRel}_l \tag{2}$$

(On objects) $|{\downarrow}A| = |{\uparrow}A| = \{*\} + |A|$ with $\mathsf{mp}({\downarrow}A) = \mathsf{mp}({\uparrow}A) = \{*\}$,
(On morphisms) Given a morphism $R \subseteq |A| \times |B|$,

$$\downarrow R := \uparrow R := R + \{(*, *)\},$$

which are morphisms of ${\downarrow}A \to {\downarrow}B$ and of ${\uparrow}A \to {\uparrow}B$.

The unique element of $\mathsf{mp}({\downarrow}A)$ (resp. $\mathsf{mp}({\uparrow}A)$) is often denoted by $*_\downarrow$ (resp. by $*_\uparrow$) to stress that the distinguished point arises to interpret the $\downarrow$ (resp. the $\uparrow$). The above definition yields the strict form $({\downarrow}N)^\perp = {\uparrow}N^\perp$ and $({\uparrow}P)^\perp = {\downarrow}P^\perp$ of De Morgan duality between $\downarrow$ and $\uparrow$.

Note that the functors (1) and (2) factor through $|\ \ |$ to **Rel** by inducing the functors from **Rel** respectively to $\mathbf{PRel}_r$ and to $\mathbf{PRel}_l$. By abuse of notation, the induced functors are also denoted by $\downarrow$ and $\uparrow$, respectively. See the diagram depicting Lemma 1 below, where the clockwise and the anticlockwise triangles show the factorizations.

– A *bimodule* $\widehat{\mathbf{Rel}}(P, N)$ consists of maps of the form $P \to N$ for object $P \in \mathbf{PRel}_r$ and $N \in \mathbf{PRel}_l$ so that they are closed under left (respectively, right) composition of morphisms from $\mathbf{PRel}_l$ (respectively from $\mathbf{PRel}_r$). A bimodule is thus characterized by a profunctor:

$$\widehat{\mathbf{Rel}}(-,-) : (\ \mathbf{PRel}_r)^{op} \times \ \mathbf{PRel}_l \to \mathbf{Set}$$

so that each instantiation determines a set of these maps. We define

$$\widehat{\mathbf{Rel}}(P, N) = \mathbf{Rel}(|P|, |N|) \tag{3}$$

That is, the maps $P \to N$ consist of *usual relations* of $P \to N$ (i.e., of morphism of **Rel**).

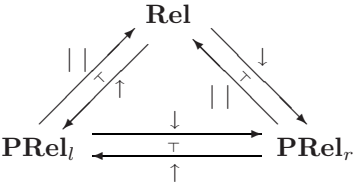
Then the bimodule obviously satisfies:

$$\widehat{\mathbf{Rel}}(\mathbf{1}, P^{\perp} \times N) \cong \widehat{\mathbf{Rel}}(P, N)$$

where $\mathbf{1}$ is an object of $\mathbf{PRel}_r$ such that $|\mathbf{1}| = \{*\}$ and $\times$ is the cartesian product of $\mathbf{Rel}$ for objects of $\mathbf{PRel}_l$.

Finally, the following series of adjunctions are crucial in order to obtain a polarized category:

Lemma 1 (Adjunctions). *The following adjunctions hold:*



That is, for every object $P \in \mathbf{PRel}_r$ and $N \in \mathbf{PRel}_l$, there are natural isomorphisms

$$\mathbf{PRel}_l(\uparrow P, N) \quad \cong \quad \widehat{\mathbf{Rel}}(P, N) \quad \cong \quad \mathbf{PRel}_r(P, \downarrow N) \tag{4}$$

Proof. We show the right-isomorphism (dually for the left). Let $R \in \mathbf{PRel}_r(P, \downarrow N)$. Since R is right-multi-pointed, we have $\mathsf{mp}(P) = R[\mathsf{mp}(\downarrow N)]$, which implies that if $(a, *_{\downarrow}) \in R$ then $a \in \mathsf{mp}(P)$ and that $\forall p \in \mathsf{mp}(P),\ (p, *_{\downarrow}) \in R$. Hence, R is written as $\{(p, *_{\downarrow}) \mid p \in \mathsf{mp}(P)\} \cup R'$ for a unique $R' \in \widehat{\mathbf{Rel}}(P, N)$. The correspondence from R to R' is bijective and gives the natural isomorphism. ■

Proposition 1 (Multi-pointed relational model for MALLP). *The pair $\langle\, \mathbf{PRel}_l,\ \mathbf{PRel}_r \rangle$ together with the module $\widehat{\mathbf{Rel}}$ forms a polarized category, and hence a denotational semantics for* MALLP. *This polarized category is denoted by $(\langle\, \mathbf{PRel}_l,\ \mathbf{PRel}_r \rangle, \widehat{\mathbf{Rel}})$.*

The categorical framework presented so far yields the following interpretation of formulas and of proofs of MALLP.

Definition 3 (Interpretation of MALLP formulas). Positive (resp. negative) formulas A of MALLP are interpreted by objects $(|A|, \mathsf{mp}(A))$ in $\mathbf{PRel}_r$ (resp. in $\mathbf{PRel}_l$):

- $|\mathbf{1}| = \{*\}$, $|\perp| = \{*\}$ and $\mathsf{mp}(\mathbf{1}) = \mathsf{mp}(\perp) = \{*\}$.
- $|\top| = |\mathbf{0}| = \emptyset$ and $\mathsf{mp}(\top) = \mathsf{mp}(\mathbf{0}) = \emptyset$.
- $|P \otimes Q| = |P| \times |Q|$, $|M \parr N| = |M| \times |N|$ and $\mathsf{mp}(X \otimes Y) = \mathsf{mp}(X \parr Y) = \mathsf{mp}(X) \times \mathsf{mp}(Y)$.
- $|P \oplus Q| = |P| + |Q|$, $|M \,\&\, N| = |M| + |N|$ and $\mathsf{mp}(X \oplus Y) = \mathsf{mp}(X \,\&\, Y) = \mathsf{mp}(X) + \mathsf{mp}(Y)$.
- $|\uparrow P| = \{*\} + |P|$, $|\downarrow N| = \{*\} + |N|$ and $\mathsf{mp}(\downarrow N) = \mathsf{mp}(\uparrow P) = \{*\}$.

Definition 4 (Interpretation of proofs). Every MALLP-proof π is interpreted in $(\langle \mathbf{PRel}_l, \mathbf{PRel}_r\rangle, \widehat{\mathbf{Rel}})$ by π^*, which is a map either in $\mathbf{PRel}_r$ or in $\widehat{\mathbf{Rel}}$ depending on whether the end sequent contains a positive formula or not, respectively.

Rules for the usual linear logic are the same as [4]. In the following, for a sequence $\mathcal{M}$ of negative formulas $N_1, \ldots, N_n$, the sequence $\mathcal{M}$ (resp. $\mathcal{M}^\perp$) is identified with the object $N_1 \parr \cdots \parr N_n$ of $\mathbf{PRel}_l$ (resp. $N_1^\perp \otimes \cdots \otimes N_n^\perp$ of $\mathbf{PRel}_r$).

- When π is $\vdash P^\perp, P$, we define $\pi^* = \{(a,a) \mid a \in |P|\} \quad \in \mathbf{PRel}_r(P,P)$
- When π is $\dfrac{\overset{\pi_1}{\vdash \Delta, N} \quad \overset{\pi_2}{\vdash \Lambda, N^\perp}}{\vdash \Delta, \Lambda}\ cut$, we define

$$\pi^* = \{(\delta, \lambda) \mid \exists a,\ (\delta, a) \in \pi_1^* \text{ and } (\lambda, a) \in \pi_2^*\}$$

  Since π_2^* is always a map in $\mathbf{PRel}_r$, π^* is a map either in $\mathbf{PRel}_r$ or in $\widehat{\mathbf{Rel}}$, depending on whether Δ contains a positive formula or not, respectively.
- When π is $\dfrac{\overset{\pi_1}{\vdash \mathcal{M}, N}}{\vdash \mathcal{M}, \downarrow N}\ \downarrow$, we define

$$\pi^* = \pi_1^* \cup \{(p, *_\downarrow) \mid p \in \mathsf{mp}(\mathcal{M})\} \quad \in \mathbf{PRel}_r(\mathcal{M}^\perp, \downarrow N)$$

  π^* is obtained from $\pi_1^* \in \widehat{\mathbf{Rel}}(\mathcal{M}^\perp, N)$ by the right adjunction of (4).
- When π is $\dfrac{\overset{\pi_1}{\vdash \mathcal{M}, P}}{\vdash \mathcal{M}, \uparrow P}\ \uparrow$, we define $\pi^* = \pi_1^* \quad \in \widehat{\mathbf{Rel}}(\mathcal{M}^\perp, \uparrow P)$.

  π^* is obtained from $\pi_1^* \in \mathbf{PRel}_r(\mathcal{M}^\perp, P)$ by composing $\eta : P \to \uparrow P$, which is the unit of the left adjunction of (4); i.e., $\eta = \{(p,p) \mid p \in |P|\}$. Note that the composition of η acts identically on morphisms.

Our simple interpretation above provides a nice framework for discriminating the two proofs of Example 1 below.

Example 1 (Denotations of proofs in $(\langle \mathbf{PRel}_l, \mathbf{PRel}_r\rangle, \widehat{\mathbf{Rel}})$*).* Let us consider a MALLP-sequent $\vdash \uparrow\downarrow\uparrow\mathbf{1}, \uparrow\downarrow\bot$ and two different proofs π_1 and π_2 for this:

$$\pi_1 = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbf{1}, \bot}{\vdash \uparrow\mathbf{1}, \bot}\uparrow}{\vdash \downarrow\uparrow\mathbf{1}, \bot}\downarrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \bot}\uparrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \downarrow\bot}\downarrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \uparrow\downarrow\bot}\uparrow \qquad \begin{array}{c}\{(1,B)\}\\ \{(1,B)\}\\ \{(\downarrow_a, B), (1,B)\}\\ \{(\downarrow_a, B), (1,B)\}\\ \{(\uparrow_a, \downarrow_c), (\downarrow_a, B), (1,B)\}\\ \{(\uparrow_a, \downarrow_c), (\downarrow_a, B), (1,B)\}\end{array}$$

$$\pi_2 = \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathbf{1}, \bot}{\vdash \uparrow\mathbf{1}, \bot}\uparrow}{\vdash \downarrow\uparrow\mathbf{1}, \downarrow\bot}\downarrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \downarrow\bot}\uparrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \uparrow\downarrow\bot}\downarrow}{\vdash \uparrow\downarrow\uparrow\mathbf{1}, \uparrow\downarrow\bot}\uparrow \qquad \begin{array}{c}\{(1,B)\}\\ \{(1,B)\}\\ \{(\uparrow_b, \downarrow_c), (1,B)\}\\ \{(\uparrow_b, \downarrow_c), (1,B)\}\\ \{(\downarrow_a, \uparrow_c), (\uparrow_b, \downarrow_c), (1,B)\}\\ \{(\downarrow_a, \uparrow_c), (\uparrow_b, \downarrow_c), (1,B)\}\end{array}$$

The right-hand side of each subproof designates its interpretation, where we take $|\uparrow\downarrow\uparrow\mathbf{1}| = \{\uparrow_a, \downarrow_a, \uparrow_b, 1\}$ and $|\uparrow\downarrow\bot| = \{\uparrow_c, \downarrow_c, B\}$ so that $\downarrow_a = \uparrow_c$, $\uparrow_b = \downarrow_c$ and $1 = B$. As is seen above, the two proofs π_1 and π_2 are interpreted by different relations.

3 Indexed Multiplicative Additive Polarized Linear Logic MALLP(I)

In this section, we present an indexed logical system MALLP(I), which is a conservative extension of MALLP. The syntactical system MALLP(I) arises from our multi-pointed relational semantics for MALLP, presented in Section 2. Each rule of MALLP(I) is designed so that it describes the denotational construction of the corresponding rule of MALLP in $(\langle \mathbf{PRel}_l, \mathbf{PRel}_r \rangle, \widehat{\mathbf{Rel}})$. Our design is inspired by Bucciarelli-Ehrhard's system [4] of MALL(I), just as their system stems from the denotational semantics **Rel** for MALL. By reflecting the adjunctions of the polarized category of Section 2, our polarity shifting rule $\downarrow_K$ for each $K \subseteq I$ is accompanied by a certain side condition. Let us begin by defining formulas of MALLP(I).

Let I be an index set which is fixed, once and for all. Each formula A of MALLP(I) is associated with a set $d(A) \subseteq I$, called the *domain* of A.

Definition 5 (Formulas and domains). *Positive* and *negative* formulas of domain J (denoted simply as P_J and N_J, respectively) are defined by the following grammar: For any sets $J, K, L \subseteq I$ such that $K \cap L = \emptyset$,

$$P_J ::= \mathbf{1}_J \mid \mathbf{0}_\emptyset \mid P_J \otimes P_J \mid P_K \oplus P_L \mid \downarrow_K N_L \text{ (positive formula)}$$
$$N_J ::= \bot_J \mid \top_\emptyset \mid N_J \parr N_J \mid N_K \,\&\, N_L \mid \uparrow_K P_L \text{ (negative formula)}$$

Note that, in contrast to the MALL connectives $\otimes, \parr, \oplus, \&$, the polarity shifting connectives $\downarrow_K$ and $\uparrow_K$ are provided for each K to have their own domains. To be precise, $\downarrow_K N$ and $\uparrow_K P$ are defined as follows (the other connectives are the same as those in [4]):

- For each $K \subseteq I$, we introduce two new connectives $\downarrow_K$ and $\uparrow_K$, both of which have K as their domains, i.e., $d(\downarrow_K) = d(\uparrow_K) = K$.
- For $L \subseteq I$ disjoint with K, if N is a negative formula with $d(N) = L$, then $\downarrow_K N$ is a positive formula with $d(\downarrow_K N) = K + L$.
- For $L \subseteq I$ disjoint with K, if P is a positive formula with $d(P) = L$, then $\uparrow_K P$ is a negative formula with $d(\uparrow_K P) = K + L$.

For any MALLP(I)-formula A with $d(A) = J$, we define its negation $A^\perp$ with $d(A^\perp) = J$ in the usual way, using the De Morgan duality for MALLP-formulas. A J-sequent is an expression of the shape $\vdash_J \Delta$ where Δ is a (possibly empty) sequence of MALLP(I)-formulas of domains J (denoted as $d(\Delta) = J$).

Definition 6 (Restriction). For a MALLP(I)-formula A with $d(A) = J$, and for $K \subseteq I$, we define the *restriction* of A by K, denoted by $A\restriction_K$, which is a MALLP(I)-formula of domain $J \cap K$ as follows:

- $\top_\emptyset\restriction_K = \top_\emptyset$ and $\mathbf{0}_\emptyset\restriction_K = \mathbf{0}_\emptyset$;
- $\bot_J\restriction_K = \bot_{J\cap K}$ and $\mathbf{1}_J\restriction_K = \mathbf{1}_{J\cap K}$;
- $(P \otimes Q)\restriction_K = P\restriction_K \otimes Q\restriction_K$, $(N \parr M)\restriction_K = N\restriction_K \parr M\restriction_K$, $(P \oplus Q)\restriction_K = P\restriction_K \oplus Q\restriction_K$, $(N \,\&\, M)\restriction_K = N\restriction_K \,\& M\restriction_K$, $(\uparrow_J P)\restriction_K = \uparrow_{J\cap K} P\restriction_K$, and $(\downarrow_J N)\restriction_K = \downarrow_{J\cap K} N\restriction_K$.

Trivially $A^{\perp}\restriction_K = (A\restriction_K)^{\perp}$. If Δ is a sequence of $\mathsf{MALLP}(I)$-formulas $A_1, \ldots, A_n$ of domains J, we define $\Delta\restriction_K = A_1\restriction_K, \ldots, A_n\restriction_K$ so that $d(\Delta\restriction_K) = d(\Delta) \cap K$.

In order to introduce polarity shifting rule $\downarrow_K$ for $\mathsf{MALLP}(I)$, we give the following definition:

Definition 7 ($d(\partial\mathcal{M})$). For a sequence $\mathcal{M}$ of negative formulas, the domain $d(\partial\mathcal{M})$, which is a subset of $d(\mathcal{M})$, is defined as follows:
First let $\dagger_1, \ldots, \dagger_n$ denote all the outermost $\uparrow$'s of $\mathcal{M}$ and all the $\perp$'s outside any scope of polarity shifting operators. Then $\mathcal{M}$ is written as $\underline{\mathcal{M}}[\dagger_1 P_1, \ldots, \dagger_n P_n]$, where P_m is a positive formula (empty if $\dagger_m$ is $\perp$) and $\underline{\mathcal{M}}[*_1, \ldots, *_n]$ is an expression made from $*_1, \ldots, *_n$ by applying only negative connectives $\parr$ and $\&$. Note that each comma of $\mathcal{M}$ is identified with $\parr$. We define

$$d(\partial\mathcal{M}) := |\underline{\mathcal{M}}|[d(\dagger_1), \ldots, d(\dagger_n)],$$

where $|\underline{\mathcal{M}}|$ is the (set-theoretical) expression resulting from the expression $\underline{\mathcal{M}}[*_1, \ldots, *_n]$ by replacing $\parr$ and $\&$ respectively with $\cap$ and $\cup$. Obviously $d(\partial\mathcal{M}) \subseteq d(\mathcal{M})$ since $d(\mathcal{M}) = |\underline{\mathcal{M}}|[d(\dagger_1 P_1), \ldots, d(\dagger_n P_n)]$.

Remark 1 ($d(\partial\mathcal{M})$ for distributed $\mathcal{M}$). By distributing $\parr$ over $\&$, every $\mathcal{M}$ is rewritten as $M_1 \,\&\, \cdots \,\&\, M_n$ so that each M_i is $\parr(\dagger_{ij} P_{ij})$. Then we can more directly define $d(\partial\mathcal{M}) = \bigcap_j d(\dagger_{1j}) + \cdots + \bigcap_j d(\dagger_{nj})$.

We introduce the inference rules of $\mathsf{MALLP}(I)$, which consist of the polarity shifting rules on top of Bucciarelli-Ehrhard's rules [4] of $\mathsf{MALL}(I)$ with the polarity constraint.

Definition 8 (Inference rules of $\mathsf{MALLP}(I)$). Inference rules of $\mathsf{MALLP}(I)$ are defined as follows (the exchange rule is left implicit):

Axioms[1] **and cut:**

$$\vdash_J \mathbf{1}_J \qquad \vdash_\emptyset \Delta, \top_\emptyset \qquad \frac{\vdash_J \Delta, N \quad \vdash_J \Lambda, N^{\perp}}{\vdash_J \Delta, \Lambda}\ cut$$

For $\top$-axiom $\vdash_\emptyset \Delta, \top_\emptyset$, its context Δ contains at most one positive formula.

Multiplicative rules:

$$\frac{\vdash_J \Delta}{\vdash_J \Delta, \perp_J}\ \perp_J \qquad \frac{\vdash_J \Delta, P \quad \vdash_J \Lambda, Q}{\vdash_J \Delta, \Lambda, P \otimes Q}\ \otimes \qquad \frac{\vdash_J \Delta, N, M}{\vdash_J \Delta, N \parr M}\ \parr$$

Additive rules:

$$\frac{\vdash_J \Delta, P}{\vdash_J \Delta, P \oplus Q}\ \oplus_1 \qquad \frac{\vdash_J \Delta, P}{\vdash_J \Delta, Q \oplus P}\ \oplus_2 \qquad \frac{\vdash_J \Delta\restriction_J, N \quad \vdash_K \Delta\restriction_K, M}{\vdash_{J+K} \Delta, N \,\&\, M}\ \&$$

For $\oplus_1$-,$\oplus_2$-rules, observe that Q has to have the empty domain.
For &-rule, it is assumed that $d(N) = J$ and $d(M) = K$ with $J \cap K = \emptyset$, and that $d(\Delta) = J + K$.

[1] $\mathsf{MALLP}(I)$ has no propositional variables, and every formula consists of constants. Hence, the usual identity axiom is derivable in Lemma 2.

Polarity shifting rules:

($\uparrow$-rule)

$$\frac{\vdash_J \Delta, P}{\vdash_J \Delta, \uparrow_\emptyset P} \uparrow$$

($\downarrow_K$-rule) For every $K \subseteq I$ such that $J \cap K = \emptyset$,

$$\frac{\vdash_J \mathcal{M}\restriction_J, N}{\vdash_{K+J} \mathcal{M}, \downarrow_K N} \downarrow_K \text{ with } \ K \subseteq d(\partial\mathcal{M}) \tag{5}$$

$\downarrow_K$-rule is applicable only when the side condition is satisfied. This condition is a syntactical description of the adjunctions (4) of Section 2. (See Proposition 2 bellow.)

Remark 2 ($\mathsf{MALLP} \prec \mathsf{MALLP}(I)$*).* For each inference rule of $\mathsf{MALLP}(I)$, if the conclusion sequent is of the domain $\emptyset$, then so is the premise sequent(s). Thus, the rules for sequents of the empty domain are identified with the standard rules of MALLP. Moreover, every $\mathsf{MALLP}(I)$-proof σ for $\vdash_\emptyset \Delta$ contains only sequents of the empty domain. Hence σ is considered as a MALLP-proof for $\vdash \Delta$. Thus $\mathsf{MALLP}(I)$ is a conservative extension of MALLP.

The following lemmas hold in the same way as in [4].

Lemma 2 (Identity). $\vdash_J A, A^\perp$ *is provable for any* $\mathsf{MALLP}(I)$*-formula* A *of domain* J*.*

Lemma 3 (Restriction). *If* $\vdash_J \Delta$ *is provable, then so is* $\vdash_{J\cap K} \Delta\restriction_K$ *for any* $K \subseteq I$*.*

In Laurent's original polarized linear logic MALLP [14], positive/negative polarities classify the dual proof-theoretical properties for connectives: *reversible* connectives $⅋, \&, \downarrow$ for negative formulas and *focusing* connectives $\otimes, \oplus, \uparrow$ for positive formulas. Our $\mathsf{MALLP}(I)$ retains the dual properties, as expected.

Lemma 4 (Focalized sequent property). *If* $\vdash_J \Delta$ *is provable in* $\mathsf{MALLP}(I)$*, then* Δ *contains at most one positive formula.*

Lemma 5 (Reversibility). $\{⅋, \&, \downarrow_K\}$*-rule is reversible. That is, if the conclusion sequent of* $\{⅋, \&, \downarrow_K\}$*-rule is provable, then so is the premise sequent.*

Proof. We prove $\downarrow_K$-rule. (The other rules are immediate.) Suppose $\vdash_{K+J} \mathcal{M}, \downarrow_K N$ is provable. Then Lemma 3, by restricting the domain of the sequent to $(K+J)\cap J$, implies that $\vdash_J \mathcal{M}\restriction_J, \downarrow_\emptyset N$ is provable. On the other hand, $\vdash_J N^\perp, N$ is provable by Lemma 2. Thus we have the following proof of $\vdash_J \mathcal{M}\restriction_J, N$:

$$\dfrac{\vdash_J \mathcal{M}\restriction_J, \downarrow_\emptyset N \quad \dfrac{\vdash_J N^\perp, N}{\vdash_J \uparrow_\emptyset N^\perp, N} \uparrow}{\vdash_J \mathcal{M}\restriction_J, N} \mathit{cut}$$

■

Let us see, by the following example, how the side condition of $\downarrow_K$-rule is applied. For its denotational characterization, see Proposition 2.

Example 2 ($\downarrow_K$-rule and the side condition). Let us consider the following provable sequent of domain $\{1,2,3,4\}$:

$$\vdash \uparrow_{\{1\}}\mathbf{1}_{\{2\}} \,\&\, \uparrow_{\{3\}}\mathbf{1}_{\{4\}} \ , \ \downarrow_{\{1,3\}}\bot_{\{2,4\}}$$

Let us search for a proof of the sequent. It is possible to apply $\downarrow_{\{1,3\}}$-rule because the side condition is satisfied: $d(\downarrow_{\{1,3\}}) \subseteq d(\partial(\uparrow_{\{1\}}\mathbf{1}_{\{2\}} \,\&\, \uparrow_{\{3\}}\mathbf{1}_{\{4\}})) = \{1\} \cup \{3\}$. Then we obtain $\vdash (\uparrow_{\{1\}}\mathbf{1}_{\{2\}} \,\&\, \uparrow_{\{3\}}\mathbf{1}_{\{4\}})\restriction_{\{2,4\}} \ , \ \bot_{\{2,4\}}$, which coincides with $\vdash \uparrow_{\emptyset}\mathbf{1}_{\{2\}} \,\&\, \uparrow_{\emptyset}\mathbf{1}_{\{4\}} \ , \ \bot_{\{2,4\}}$. Then by applying a &-rule and then $\uparrow$-rules, we have the following proof σ (the braces $\{,\}$ of domains are omitted for simplicity):

$$\sigma = \dfrac{\dfrac{\dfrac{\vdash \mathbf{1}_2 \ , \ \bot_2}{\vdash \uparrow_{\emptyset}\mathbf{1}_2 \ , \ \bot_2}\uparrow \quad \dfrac{\vdash \mathbf{1}_4 \ , \ \bot_4}{\vdash \uparrow_{\emptyset}\mathbf{1}_4 \ , \ \bot_4}\uparrow}{\vdash \uparrow_{\emptyset}\mathbf{1}_2 \,\&\, \uparrow_{\emptyset}\mathbf{1}_4 \ , \ \bot_{2,4}}\,\&}{\vdash \uparrow_1\mathbf{1}_2 \,\&\, \uparrow_3\mathbf{1}_4 \ , \ \downarrow_{1,3}\bot_{2,4}}\downarrow_{1,3}$$

Let us consider another example by modifying the domains of the above example: $\vdash \uparrow_{\{1\}}\mathbf{1}_{\emptyset} \,\&\, \uparrow_{\emptyset}\mathbf{1}_{\{2\}} \ , \ \downarrow_{\{1,2\}}\bot_{\emptyset}$. This sequent is shown to be unprovable by the cut-elimination (Corollary 1): first, $\downarrow_{\{1,2\}}$-rule is not applicable since the sequent fails to satisfy the side condition: $d(\downarrow_{\{1,2\}}) \not\subseteq d(\partial(\uparrow_{\{1\}}\mathbf{1}_{\emptyset} \,\&\, \uparrow_{\emptyset}\mathbf{1}_{\{2\}})) = \{1\} \cup \emptyset$. Therefore, the last rule must be:

$$\dfrac{\vdash \uparrow_1\mathbf{1}_{\emptyset} \ , \ \downarrow_1\bot_{\emptyset} \quad \vdash \uparrow_{\emptyset}\mathbf{1}_2 \ , \ \downarrow_2\bot_{\emptyset}}{\vdash \uparrow_1\mathbf{1}_{\emptyset} \,\&\, \uparrow_{\emptyset}\mathbf{1}_2 \ , \ \downarrow_{1,2}\bot_{\emptyset}}\,\&$$

Although the left premise sequent is provable, the right premise sequent is not: By Lemma 4 the last rule must be $\downarrow_{\{2\}}$-rule, which is not applicable because of the violation of the side condition: $d(\downarrow_{\{2\}}) \not\subseteq d(\partial(\uparrow_{\emptyset}\mathbf{1}_{\{2\}})) = \emptyset$. ■

The formulas of indexed system $\mathsf{MALLP}(I)$ are designed so that the domain of each formula indicates (syntactically) a family of points, thus a relation, in the multi-pointed relational semantics of Section 2. Hence, there is a bijective correspondence between $\mathsf{MALLP}(I)$-formulas and families of points in the webs for the corresponding MALLP-formulas. Let us describe this correspondence precisely.

Notation: For $a \in X^J$ and $j \in J$, a_j denotes the j-th element of a. For $a \in X^J$ and $b \in Y^J$, we denote by $a \times b$ the element of $(X \times Y)^J$ given by $(a \times b)_j = (a_j, b_j)$ for each $j \in J$. If $K, L \subseteq I$ are disjoint and if $a \in X^K$ and $b \in Y^L$, we denote by $a + b$ the element of $(X + Y)^{K+L}$ defined by case; $(a + b)_k = a_k$ if $k \in K$; and $(a + b)_l = b_l$ if $l \in L$. For $J \subseteq I$, we denote by $(c)_J$ the J-indexed family of the constant c, i.e., the unique element of $\{c\}^J$.

Definition 9 (Translation of MALLP-formulas). To any MALLP-formula A, and any family $a \in |A|^J$, we associate a $\mathsf{MALLP}(I)$-formula $A\langle a \rangle$ of domain J inductively as follows:

We here treat only $\downarrow N$ and $\uparrow P$. (The other connectives are the same as in [4].)

– If $A \equiv \downarrow N$, then $a = (*_\downarrow)_K + b$ with the family $(*_\downarrow)_K \in \{*_\downarrow\}^K$ and $b \in |N|^L$ such that $K+L = J$. Then we set $A\langle a\rangle = \downarrow_K N\langle b\rangle$, which is a $\mathsf{MALLP}(I)$-formula of domain J.
– If $A \equiv \uparrow P$, then $a = (*_\uparrow)_K + b$ with the family $(*_\uparrow)_K \in \{*_\uparrow\}^K$ and $b \in |P|^L$ such that $K+L = J$. Then we set $A\langle a\rangle = \uparrow_K P\langle b\rangle$, which is a $\mathsf{MALLP}(I)$-formula of domain J.

If $\Delta = A_1, \ldots, A_n$ is a sequence of MALLP-formulas, we define $|\Delta| = |A_1| \times \cdots \times |A_n|$. If $\gamma \in |\Delta|^J$, then, using our usual notational conventions, we can write $\gamma = \gamma^1 \times \cdots \times \gamma^n$ with $\gamma^m \in |A_m|^J$, and we set $\Delta\langle\gamma\rangle = A_1\langle\gamma^1\rangle, \ldots, A_n\langle\gamma^n\rangle$.

We have the following lemma, which will be used to prove Proposition 3.

Lemma 6. *Let $\vdash \Delta$ be a sequent in* MALLP *and let $\gamma \in |\Delta|^J$. Let $K \subset I$. Let $\gamma\restriction_{J\cap K}$ be the restriction of γ to $J \cap K$. Then $\Delta\langle\gamma\restriction_{J\cap K}\rangle = \Delta\langle\gamma\rangle\restriction_K$.*

The following Lemma 7 ensures that the correspondence given in Definition 9 is bijective to the $\mathsf{MALLP}(I)$-formulas:

Lemma 7. *If A is a $\mathsf{MALLP}(I)$-formula of domain J and $A\restriction_\emptyset$ is the corresponding* MALLP*-formula, there is a unique family $a \in |A\restriction_\emptyset|^J$ such that $A = A\restriction_\emptyset\langle a\rangle$.*

For a typographical convenience, a MALLP-formula $A\restriction_\emptyset$ and a sequent $\mathcal{M}\restriction_\emptyset$ are sometimes denoted by A and M, respectively.

We see the above bijective correspondence by the following example.

Example 3 ($\mathsf{MALLP}(I)$*-formula as a relation).* Let us consider the $\mathsf{MALLP}(I)$-sequent $\vdash \uparrow_{\{1\}}\mathbf{1}_{\{2\}} \,\&\, \uparrow_{\{3\}}\mathbf{1}_{\{4\}}\ ,\ \downarrow_{\{1,3\}}\bot_{\{2,4\}}$ of Example 2. We determine the corresponding family $\gamma \in |\uparrow\mathbf{1} \,\&\, \uparrow\mathbf{1}, \downarrow\bot|^{\{1,2,3,4\}}$ as follows: Let us represent the webs $|\uparrow\mathbf{1} \,\&\, \uparrow\mathbf{1}| = \{\uparrow_a, 1_a, \uparrow_b, 1_b\}$ and $|\downarrow\bot| = \{\downarrow_c, B_c\}$. The representation designates the correspondence between components of formulas and points. We determine γ_1 as follows: Since it is the domains of $\uparrow_{\{1\}}$ and $\downarrow_{\{1,3\}}$ that contain the index 1, γ_1 is a pair $(\uparrow_a, \downarrow_c)$ of the corresponding points to $\uparrow_{\{1\}}$ and $\downarrow_{\{1,3\}}$. Similar calculations for $\gamma_2, \gamma_3, \gamma_4$ yield $\gamma_1 = (\uparrow_a, \downarrow_c),\ \gamma_2 = (1_a, B_c), \gamma_3 = (\uparrow_b, \downarrow_c),\ \gamma_4 = (1_b, B_c)$. In fact, γ happens to be the denotation of the MALLP-proof, which is obtained from $\mathsf{MALLP}(I)$-proof σ of Example 2 by forgetting all the domain symbols. ■

By means of the above bijective correspondence, the side condition of $\downarrow_K$-rule turns out to be a syntactic counterpart of multi-pointedness of relations:

Proposition 2 (Semantical characterization of the side condition of $\downarrow_K$-rule). *Let J and K be disjoint subsets of I. Let $\mathcal{M}$ be a sequence of negative formulas of domain $K + J$, and N be a formula of domain J in* $\mathsf{MALLP}(I)$*. Let $\gamma \times a \in |(\mathsf{M}, \downarrow\mathsf{N})|^{K+J}$ be the family of points associated with $(\mathcal{M}, \downarrow_K N)$. Then the following two conditions are equivalent:*

1. *$K \subseteq d(\partial\mathcal{M})$*
2. *$\gamma \times a$ is right-multi-pointed, that is, for any index $i \in K$, $\gamma_i \in \mathsf{mp}(\mathsf{M})$.*

Proof. By induction on the number of & in $\mathcal{M}[*_1, \ldots, *_n]$. ■

Example 4 (A non-↑-soft sequent of MALLP(I)*).* Let us consider the sequent $\vdash \uparrow_{\{1\}}\mathbf{1}_{\{2\}}\ ,\ \uparrow_{\{1\}}\downarrow_{\emptyset}\bot_{\{2\}}$, which is not provable in MALLP(I). By the cut- elimination theorem of MALLP(I) (Corollary 1 below), the last rule should be a ↑-rule. However, it is impossible to apply the rule because both the outermost ↑'s have the non-empty domain $\{1\}$. The unprovability corresponds, by virtue of Theorem 1 bellow, to the non-↑-softness of the corresponding relation $\gamma \subseteq |\uparrow\mathbf{1}, \uparrow\downarrow\bot|$ such that $\gamma_1 = (*_\uparrow, *_\uparrow)$. Note that if a relation is not ↑-soft (i.e., does not factor through any outermost ↑), it cannot be contained in any denotations of MALLP-proofs since MALLP syntax is ↑-soft. See Section 7.1.1 of [12] for the ↑-softness.

4 A Correspondence between MALLP(I)-Provability and Denotations of MALLP-Proofs

This section is devoted to proving the *basic property* of MALLP(I), which is the main theorem of this paper. The property, named after Bucciarelli-Ehrhard [4], characterizes a relationship between provability of formulas of MALLP(I) and denotations of proofs of MALLP. The characterization is a polarized version of Bucciarelli-Ehrhard's Proposition 20 of [4]. As a corollary of the basic property, the cut-elimination theorem of MALLP(I) is obtained.

Theorem 1 (Basic property of MALLP(I)**).** *Let Δ be a sequence of formulas of* MALLP*, and let $\gamma \in |\Delta|^J$. The following two statements are equivalent.*

i) There exists a proof π of Δ in MALLP *such that $\gamma \in (\pi^*)^J$.*
ii) The sequent $\vdash_J \Delta\langle\gamma\rangle$ is provable in MALLP(I)*.*

The theorem is proved by the following Proposition 3 and Proposition 4, which are converse to each other: Proposition 3 shows an implication from (i) to (ii) and conversely for Proposition 4. In the following proofs, we sometimes denote by A_J a MALLP(I)-formula A with domain J. We first show the following:

Proposition 3. *Let Δ be a sequent in* MALLP *and let π be a proof (resp. cut-free proof) of $\vdash \Delta$ in* MALLP*. Let $\gamma \in (\pi^*)^J$ (for some $J \subseteq I$). Then the sequent $\vdash_J \Delta\langle\gamma\rangle$ has a proof (resp. cut-free proof) σ in* MALLP(I) *such that $\sigma\restriction_\emptyset = \pi$.*

Proof. By induction on the MALLP-proof π. We consider only the polarity shifting rules (the other rules are the same as Lemma 18 of Bucciarelli-Ehrhard [4]).

– When π is $\dfrac{\overset{\pi_1}{\vdash \mathcal{M}, P}}{\vdash \mathcal{M}, \uparrow P}\ \uparrow$

since $\gamma \in (\pi^*)^J = (\pi_1^*)^J \subseteq |\mathcal{M}, P|^J$ by the interpretation of ↑-rule, γ is of the form $\delta \times a$ with $\delta \in |\mathcal{M}|^J$ and $a \in |P|^J$. Thus by the induction hypothesis, the MALLP(I)-sequent $\vdash_J \mathcal{M}\langle\delta\rangle, P\langle a\rangle$ has a proof σ_1 such that $\sigma_1\restriction_\emptyset = \pi_1$. By applying ↑-rule to σ_1, we obtain the following proof σ of $\vdash_J (\mathcal{M}, \uparrow P)\langle\gamma\rangle$ so that $\sigma\restriction_\emptyset = \pi$:

$$\frac{\overset{\sigma_1}{\vdash_J \mathcal{M}\langle\delta\rangle, P\langle a\rangle}}{\vdash_J \mathcal{M}\langle\delta\rangle, \uparrow_\emptyset P\langle a\rangle}\ \uparrow$$

– When π is $\dfrac{\overset{\pi_1}{\vdash \mathcal{N}, M}}{\vdash \mathcal{N}, \downarrow M}\ \downarrow$

since $\gamma \in (\pi^*)^J = \left(\{(p, *_\downarrow) \mid p \in \mathsf{mp}(\mathcal{N})\} \cup \pi_1^*\right)^J \subseteq |\mathcal{N}, \downarrow M|^J$ by the interpretation of $\downarrow$-rule, γ is of the form $\delta \times a$ with $\delta \in |\mathcal{N}|^J$ and $a \in |\downarrow M|^J$. Since $|\downarrow M|^J = (\{*_\downarrow\} + |M|)^J$, there are two uniquely defined disjoint sets K and L such that $K + L = J$, and a is of the form $(*_\downarrow)_K + b$ with $(*_\downarrow)_K \in \{*_\downarrow\}^K$ and $b \in |M|^L$. According to this decomposition of J, δ is also written as $\delta\restriction_K + \delta\restriction_L$ with $\delta\restriction_K \in |\mathcal{N}|^K$ and $\delta\restriction_L \in |\mathcal{N}|^L$. Note that we have $\delta\restriction_K \in \mathsf{mp}(\mathcal{N})$ by the interpretation of $\downarrow$-rule in the multi-pointed relational semantics. Thus γ is written as the disjoint union $\gamma = (\delta\restriction_K \times (*_\downarrow)_K) + (\delta\restriction_L \times b)$, where $\delta\restriction_K \times (*_\downarrow)_K \in \{(p, *_\downarrow) \mid p \in \mathsf{mp}(\mathcal{N})\}^K$ and $\delta\restriction_L \times b \in (\pi_1^*)^L$. By the induction hypothesis for π_1, the $\mathsf{MALLP}(I)$-sequent $\vdash_L \mathcal{N}\langle\delta\restriction_L\rangle, M\langle b\rangle$ has a proof σ_1 such that $\sigma_1\restriction_\emptyset = \pi_1$. Since $\mathcal{N}\langle\delta\restriction_L\rangle = \mathcal{N}\langle\delta\rangle\restriction_L$ by Lemma 6, we apply $\downarrow_K$-rule to σ_1 in order to obtain the following proof σ of $\vdash_J (\mathcal{N}, \downarrow M)\langle\gamma\rangle$ such that $\sigma\restriction_\emptyset = \pi$:

$$\frac{\overset{\sigma_1}{\vdash_L \mathcal{N}\langle\delta\rangle\restriction_L\ ,\ M\langle b\rangle}}{\vdash_{K+L} \mathcal{N}\langle\delta\rangle\ ,\ (\downarrow M)\langle(*_\downarrow)_K + b\rangle}\ \downarrow_K$$

The side condition for $\downarrow_K$-rule is satisfied since γ is of the form $(\delta\restriction_K \times (*_\downarrow)_K) + (\delta\restriction_L \times b)$ with $\delta\restriction_K \in \mathsf{mp}(\mathcal{N})^K$ (see Proposition 2). ■

Next, we show the converse of Proposition 3:

Proposition 4. *Let Δ be a sequent in* MALLP. *Let $\gamma \in |\Delta|^J$ (for some $J \subseteq I$), and let σ be a proof of $\vdash_J \Delta\langle\gamma\rangle$ in* $\mathsf{MALLP}(I)$. *Then $\gamma \in (\sigma\restriction_\emptyset\ ^*)^J$.*

Proof. By induction on the $\mathsf{MALLP}(I)$-proof σ. We consider the polarity shifting rules since the other rules are the same as those in Lemma 19 of [4]. In the following proof, a MALLP-formula $A\restriction_\emptyset$ and a sequent $\Delta\restriction_\emptyset$ are denoted by A and $\mathbf{\Delta}$, respectively.

– When σ is $\dfrac{\overset{\sigma_1}{\vdash_J \Delta, P}}{\vdash_J \Delta, \uparrow_\emptyset P}\ \uparrow$

there is, by Lemma 7, $\gamma = \delta \times a \in |\mathbf{\Delta}, \uparrow\mathsf{P}|^J$ such that $\Delta_J, \uparrow_\emptyset P_J = (\mathbf{\Delta}, \uparrow\mathsf{P})\langle\gamma\rangle$. Note first that $a \in |\mathsf{P}|^J$ holds since the domain of the outermost $\uparrow$ of $\uparrow_\emptyset P_J = (\uparrow\mathsf{P})\langle a\rangle$ is empty. Thus $\vdash_J \Delta, P$ coincides with $\vdash_J \mathbf{\Delta}\langle\delta\rangle, \mathsf{P}\langle a\rangle$, and hence we have $\delta \times a \in (\sigma_1\restriction_\emptyset\ ^*)^J$ by the induction hypothesis. Then, by the denotational interpretation of $\uparrow$-rule of MALLP, we conclude:

$$\gamma = \delta \times a \in (\sigma_1\restriction_\emptyset\ ^*)^J = (\sigma\restriction_\emptyset\ ^*)^J.$$

– When σ is $\dfrac{\overset{\sigma_1}{\vdash_L \mathcal{M}\restriction_L, N}}{\vdash_{K+L} \mathcal{M}, \downarrow_K N}\ \downarrow_K$ with $K \subseteq d(\partial\mathcal{M})$, there is, by Lemma 7, $\gamma \in |\mathsf{M}, \downarrow\mathsf{N}|^{K+L}$ such that $\mathcal{M}_{K+L}, \downarrow_K N_L = (\mathsf{M}, \downarrow\mathsf{N})\langle\gamma\rangle$. Because K and L are disjoint, γ is written as the following disjoint union:

$$\gamma = (\delta\restriction_K \times (*_\downarrow)_K) + (\delta\restriction_L \times b) \in |\mathsf{M}, \downarrow\mathsf{N}|^{K+L},$$

where $\delta\restriction_K \times(*_\downarrow)_K \in |\mathsf{M}|^K \times \{*_\downarrow\}^K$ and $\delta\restriction_L \times b \in |\mathsf{M}|^L \times |\mathsf{N}|^L$. Since $\vdash_L \mathcal{M}\restriction_L, N$ coincides with $\vdash_L \mathsf{M}\langle\delta\restriction_L\rangle, \mathsf{N}\langle b\rangle$, we have $\delta\restriction_L \times b \in (\sigma_1\restriction_\emptyset\ ^*)^L$ by the induction hypothesis. Since the side condition of $\downarrow_K$-rule implies $\delta\restriction_K \in \mathsf{mp}(\mathsf{M})^K$ (see Proposition 2), we obtain:

$$(\delta\restriction_K \times(*_\downarrow)_K) + (\delta\restriction_L \times b) \in \big(\{(p, *_\downarrow) \mid p \in \mathsf{mp}(\mathsf{M})\} \cup \sigma_1\restriction_\emptyset\ ^*\big)^{K+L}.$$

Thus, by the interpretation of $\downarrow$-rule of MALLP, we conclude $\gamma \in (\sigma\restriction_\emptyset\ ^*)^{K+L}$. ■

As a corollary, we have the following semantical cut-elimination à la Bucciarelli-Ehrhard [5].

Corollary 1 (Cut-elimination of MALLP(I)). *The sequent calculus system* MALLP(I) *enjoys cut-elimination. That is, if a sequent is provable, then it is provable without using cut-rule.*

Proof. Assume $\vdash_J \Delta$ is provable with a MALLP(I)-proof σ. By Lemma 7, Δ is of the form $\Delta\langle\gamma\rangle$ for a sequence Δ of MALLP-formulas and $\gamma \in |\Delta|^J$. Then by Proposition 4, $\gamma \in (\sigma\restriction_\emptyset\ ^*)^J$. Since $\sigma\restriction_\emptyset$ is a MALLP-proof of $\vdash \Delta$, there exists a cut-free MALLP-proof π for the sequent by the cut-elimination of MALLP. Since $\sigma\restriction_\emptyset\ ^* = \pi^*$ by Proposition 1, we have $\gamma \in (\pi^*)^J$. Then, by Proposition 3, there exists a cut-free MALLP(I)-proof ρ of $\vdash_J \Delta\langle\gamma\rangle$, which sequent is $\vdash_J \Delta$. ■

5 Discussions and Future Work

Let us discuss several comparisons of our polarity shifting operators with Bucciarelli and Ehrhard's exponentials of [5]. First, our multi-pointed relational interpretation of $\downarrow A$ and $\uparrow A$ is seen as a restriction of their interpretation of $!A$ and $?A$ (pg.212 of [5]) to the multisets of cardinality *at most one* (i.e., $|\uparrow A| = |\downarrow A| = \{[a] \mid a \in |A|\} \cup \{[\,]\}$). Note that the empty multiset $[\,]$ corresponds to our distinguished element $*$. Due to this restriction, the contraction rule is absent in our interpretation. On the other hand, the interpretation of the promotion rule of LL (pg.240 of [5]) simulates ours of the $\downarrow$-rule for MLLP (without additives) by restricting the cardinality n for the index of the family to either 0 or 1. Second, our indexed $\downarrow_K N$ and $\uparrow_K P$ of Definition 5 coincide with Bucciarelli-Ehrhard's $!_u N$ and $?_u P$ when u is the injection from L to $L + K$. Then our translation of Definition 9 corresponds to theirs (pg.213 of [5]).

As another comparison, it is straightforward to generalize our construction of this paper into a polarized variant of LL(I) of [5] with exponentials. By weakening the bijective correspondence of Lemma 7 into surjective one, the construction yields an indexed system for Laurent's $\mathsf{LL}_{\mathrm{pol}}$ augmented with polarity shifting operators $\uparrow$ and $\downarrow$.

Regarding future works, a phase semantics for MALLP(I) should be examined. Phase semantics is a standard truth-value semantics for linear logic. Such a semantics for MALLP(I) is obtained by a generalization of our polarized phase semantics [13] for MALLP. In [13] a topological structure was given to a phase

space by interpreting $\downarrow$ and $\uparrow$ as interior and closure operators, respectively. For the generalization, an I-product phase spaces become crucial analogously to Bucciarelli-Ehrhard [4,5] and Ehrhard [7]. In our polarized setting the product topology on this phase space is important to understand parameterized $\downarrow_K$ connectives for $\mathsf{MALLP}(I)$. A phase semantics for $\mathsf{MALLP}(I)$ yields, by virtue of Theorem 1, a new denotational semantics for MALLP. Moreover a truth valued completeness of such a phase semantics leads naturally to a weak denotational completeness in the sense of Girard [9] and Bucciarelli-Ehrhard. In particular, such a denotational completeness explicates an I-indexed topological logical relations for polarized linear logic.

Acknowledgements

The authors would like to express their sincere thanks especially to one of the anonymous referees for careful reading and for valuable comments and suggestions.

References

1. Andreoli, J.-M.: Logic Programming with Focusing Proofs in Linear Logic. Journal of Logic and Computation 2(3), 297–347 (1992)
2. Blute, R., Scott, P.J.: Category theory for linear logicians. In: Ruet, P., Ehrhard, T., Girard, J.-Y., Scott, P. (eds.) Proceedings Linear Logic Summer School, pp. 1–52. Cambridge University Press, Cambridge (2004)
3. Bruasse-Bac, A.: Logique linéaire indexée du second ordre, Thèse de doctorat, Institut de Mathématiques de Luminy, Université Aix-Marseille II (2001)
4. Bucciarelli, A., Ehrhard, T.: On phase semantics and denotational semantics in multiplicative-additive linear logic. Annals of Pure and Applied Logic 102(3), 247–282 (2000)
5. Bucciarelli, A., Ehrhard, T.: On phase semantics and denotational semantics: the exponentials. Annals of Pure and Applied Logic 109(3), 205–241 (2001)
6. Cockett, R., Seely, R.: Polarized category theory, modules, and game semantics. Theory and Applications of Categories 18(2), 4–101 (2007)
7. Ehrhard, T.: A completeness theorem for symmetric product phase spaces. Journal of Symbolic Logic 69(2), 340–370 (2004)
8. Girard, J.-Y.: A New Constructive Logic: Classical Logic. Mathematical Structures in Computer Science 1(3), 255–296 (1991)
9. Girard, J.-Y.: On denotational completeness. Theoretical Computer Science 227(1), 249–273 (1999) (special issue)
10. Girard, J.-Y.: On the meaning of logical rules II: multiplicative/additive case. In: Foundation of Secure Computation. NATO Series F, vol. 175, pp. 183–212. IOS Press, Amsterdam (2000)
11. Girard, J.-Y.: Locus Solumn from the rules of logic to the logic of rules. Mathematical Structures in Computer Science 11(3), 301–506 (2001)
12. Hamano, M., Scott, P.: A Categorical Semantics for Polarized MALL. Annals of Pure and Applied logic 145, 276–313 (2007)

13. Hamano, M., Takemura, R.: A Phase Semantics for Polarized Linear Logic and Second Order Conservativity (submitted, 2007)
14. Laurent, O.: Étude de la polarisation en logique, Thèse de Doctorat, Institut de Mathématiques de Luminy, Université Aix-Marseille II (2002)
15. Laurent, O.: Polarized games. Annals of Pure and Applied Logic 130(1-3), 79–123 (2004)

A Characterisation of Lambda Definability with Sums Via $\top\top$-Closure Operators

Shin-ya Katsumata

Research Institute for Mathematical Sciences, Kyoto University
Kyoto, 606-8502, Japan
sinya@kurims.kyoto-u.ac.jp

Abstract. We give a new characterisation of morphisms that are definable by the interpretation of the simply typed lambda calculus with sums in *any* bi-Cartesian closed category. The $\top\top$-closure operator will be used to *construct* the category in which the collection of definable morphisms at sum types can be characterised as the coproducts of such collections at lower types.

1 Introduction

The *λ-definability problem* is to characterise the semantic elements that are definable by denotational / categorical semantics of the simply typed λ-calculus. A characterisation of the λ-definable elements in full type hierarchies was first given by Plotkin using *Kripke logical relations*. This result was later generalized by Jung and Tiuryn to any Henkin model using *Kripke logical predicates with varying arity* [15]; its categorical formulation was also given by Alimohamed [1].

These precursors considered the definability problem in the simply typed lambda calculus with only arrow types (and possibly product types). The problem becomes more subtle when sum types are added. There is a natural definition of coproducts for Kripke predicates with varying arity, but these coproducts are not sufficient to characterise the definable elements at sum types. In [11] Fiore and Simpson overcome this difficulty by introducing a new concept called *Grothendieck logical predicates*. They used Grothendieck topology to improve the definition of coproducts of Kripke predicates. By constructing a suitable category of worlds and topology on it, Fiore and Simpson succeeded in giving a characterisation of definable morphisms in any bi-CCC with *stable* coproducts.

In this paper we approach the definability problem in the simply typed lambda calculus with sums using a different technique called *$\top\top$-closure operator*. The main contribution of this paper is the following:

1. We characterise the definability predicate (=the collection of definable morphisms) at sum types by means of the standard coproducts for Kripke predicates and the *semantic $\top\top$-closure operator*:

$$\mathrm{Def}\, 0 = \dot{0}^{\top\top}, \qquad \mathrm{Def}(\tau + \tau') = (\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau')^{\top\top}.$$

   This characterisation holds with respect to the interpretation of the lambda calculus with sums in *any* bi-CCC. We also give a characterisation of morphisms definable by the simply typed lambda calculus with sums by means of *weak logical predicates*.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 278–292, 2008.
© Springer-Verlag Berlin Heidelberg 2008

2. We analyse the underlying categorical essence of the above arguments, and present it as the *restriction theorem*. The statement of the theorem is the following: let P be a logical predicate in a sufficiently rich fibration $p : \mathbb{P} \to \mathbb{C}$. If P respects product and arrow types, then we can restrict P to a full reflective subcategory $\mathbb{P}^{\top\top}$ of $\mathbb{P}$ so that P respects sum types as well.

The characterisation stated in item 1 implies that in the category $\mathbb{K}^{\top\top}$ of $\top\top$-closed objects the definability predicates at sum types are given by the coproducts of the definability predicates at lower types. By employing $\mathbb{K}^{\top\top}$ as a gluing category, we also show that the inclusion from the free distributive category $L_0(B)$ over the set B of base types to the free bi-CCC $L_1(B)$ over B is full. We note that this result is proved in [10], but there a different gluing category is employed.

Preliminary. We define categories and functors by the following table:

A category / functor is ...	when it has / preserves ...
Cartesian	finite products
co-Cartesian	finite coproducts
bi-Cartesian	finite products and finite coproducts
Cartesian closed (CC)	finite products and exponentials
bi-Cartesian closed (bi-CC)	finite products, finite coproducts and exponentials

2 Definability of Calculi with Sums

We deal with the definability problem in the context of functorial semantics, where syntatic theories are treated as freely generated categories, and interpretations are represented by structure-preserving functors. We fix a small Cartesian category L that plays the role of a syntactic theory, a bi-CCC $\mathbb{C}$ that plays the role of a semantic domain, and a strict Cartesian functor F:

$$L \xrightarrow{F} \mathbb{C}$$

that gives an interpretation of the syntactic theory. We first review some basic properties of the definability predicate for F in this setting, then extend L and F with additional structures (coproducts / exponentials) toward the main theorems of this paper. At this moment we do not require that L is a freely generated category, as the freeness does not play any role in the following discussion.

2.1 Kripke Predicates with Varying Arity

We first introduce the poset $\mathbf{Ctx}_L$ that plays the role of Kripke structure for Kripke predicates with varying arity. The carrier of the poset is $(\mathbf{Obj}(L))^*$, the set of finite sequences of L-objects, and these sequences are ordered by the prefix ordering (that is, $\tau \leq \sigma$ if τ is a prefix of σ). Below we treat $\mathbf{Ctx}_L$ as a category.

When L is identified as a syntactic (type) theory, the poset $\mathbf{Ctx}_L$ expresses inclusions of typing contexts. We associate these inclusions with projections in L by the following functor $|-| : \mathbf{Ctx}_L \to L^{op}$:

$$|\tau_1 \cdots \tau_n| = (\cdots((1 \times \tau_1) \times \tau_2) \cdots) \times \tau_n, \qquad |\tau_1 \cdots \tau_n \leq \tau_1 \cdots \tau_{n+m}| = \mathrm{id} \overbrace{\circ \pi \circ \cdots \circ \pi}^{m},$$

where π is the first projection for appropriate objects.

We next define the category $\mathbb{K}_F$ of *Kripke predicates with varying arity*.

- An object of $\mathbb{K}_F$ is a pair (C, X) where C is a $\mathbb{C}$-object and X is a subpresheaf of the contravariant presehaf $\mathbb{C}(F|-|, C)$ on $\mathbf{Ctx}_L$.
- A morphism from (C, X) to (D, Y) is a $\mathbb{C}$-morphism $f : C \to D$ such that for any $\mathbf{Ctx}_L$-object Γ and $\mathbb{C}$-morphism $g \in X\Gamma$, we have $f \circ g \in Y\Gamma$.

The category $\mathbb{K}_F$ is constructed as follows. Let $H_F : \mathbb{C} \to [\mathbf{Ctx}_L, \mathbf{Set}]$ be a functor defined by $H_F(C) = \mathbb{C}(F|-|, C)$. Then $\mathbb{K}_F$ is the vertex of the change-of-base (pullback) of the subobject fibration $p : \mathbf{Sub}([\mathbf{Ctx}_L, \mathbf{Set}]) \to [\mathbf{Ctx}_L, \mathbf{Set}]$ along H_F (Figure 1). This construction is an instance of *subscone* [22] or *categorical gluing* [1].

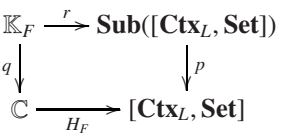


Fig. 1. Derivation of q

Proposition 1. *The leg $q : \mathbb{K}_F \to \mathbb{C}$ of the change-of-base (Figure 1) is a strict bi-CC functor and a partial order fibration with fibred small products.*

Proof. Since p is a partial order fibration with fibred small products, q inherits these structures via the change-of-base along the Cartesian functor H_F. The bi-CC structure of $\mathbb{K}_F$, which is strictly preserved by q, is given as follows (see also Section 1.5, [1]):

$$\begin{aligned}
\dot{1} &= (1, \{!_{F|\Gamma|}\}_\Gamma) \\
(C, X) \dot{\times} (D, Y) &= (C \times D, \{h \mid \pi_1 \circ h \in X\Gamma, \pi_2 \circ h \in Y\Gamma\}_\Gamma) \\
\dot{0} &= (0, \emptyset) \\
(C, X) \dot{+} (D, Y) &= (C + D, (\{inl \circ f \mid f \in X\Gamma\} \cup \{inr \circ f \mid f \in Y\Gamma\})_\Gamma) \\
(C, X) \dot{\Rightarrow} (D, Y) &= (C \Rightarrow D, \{f \mid \forall \Gamma' \geq \Gamma \,.\, \forall x \in X\Gamma' \,.\, ev \circ \langle f \circ F|\Gamma \leq \Gamma'|, x\rangle \in Y\Gamma'\}_\Gamma).
\end{aligned}$$

(here $\{\cdots\}_\Gamma$ denotes a presheaf described by an auxiliary parameter $\Gamma \in \mathbf{Ctx}_L$).

We define the target of our study, the *definability functor* $\mathrm{Def} : L \to \mathbb{K}_F$, by

$$\begin{aligned}
\mathrm{Def}\, \tau &= (\tau, \{Fg \in \mathbb{C}(F|\Gamma|, F\tau) \mid g \in L(|\Gamma|, \tau)\}_\Gamma) \\
\mathrm{Def}\, f &= Ff.
\end{aligned}$$

We call $\mathrm{Def}\, \tau$ the *definability predicate* (of F) at τ. We refer to the presheaf part of $\mathrm{Def}\, \tau$ by $D\tau$; in other words, $D = r \circ \mathrm{Def}$ (r is the other leg of the change-of-base).

Proposition 2. $\mathrm{Def} : L \to \mathbb{K}_F$ *is a full strict Cartesian functor, and* $q \circ \mathrm{Def} = F$.

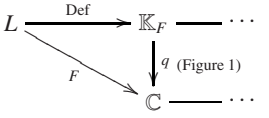
Figure 2 summarises categories and functors we have introduced so far.

One important property of the functor Def, which is implicit in the characterisation of λ-definability by Jung and Tiuryn [15], is the following:

Lemma 1. *For any L-objects τ, τ' and $\mathbf{Ctx}_L$-object Γ, we have*

$$f \in r(\mathrm{Def}\, \tau \dot{\Rightarrow} \mathrm{Def}\, \tau')(\Gamma) \iff \lambda^{-1}(f) \in D\tau'(\Gamma\tau),$$

(here λ^{-1} denotes the uncurrying operator).



$\mathbb{K}_F$ bi-CCC $(\dot{1}, \dot{\times}, \dot{0}, \dot{+}, \dot{\Rightarrow})$
$\mathbb{C}$ bi-CCC $(1, \times, 0, +, \Rightarrow)$
q strict bi-CC functor and partial order fibration with fibered small products
L small Cartesian category
F strict Cartesian functor
Def full strict Cartesian functor

Fig. 2. Categories and Functors for the Argument of Definability

2.2 A Characterisation of Definability with Sums

We next assume that the category L in Figure 2 is a distributive category (in the sense of Walters [26,7]) and F is a strict bi-Cartesian functor. Recall that a distributive category $\mathbb{C}$ is a bi-Cartesian category such that the canonical morphism

$$[A \times inl, A \times inr] : (A \times B) + (A \times C) \to A \times (B + C)$$

has the inverse (called *distributive law*) [1] :

$$m^{\mathbb{C}}_{A,B,C} : A \times (B + C) \to (A \times B) + (A \times C).$$

We note that F strictly preserves distributive laws, that is, $F(m^{L}_{\tau,\tau',\rho}) = m^{\mathbb{C}}_{F\tau,F\tau',F\rho}$.

The functor Def in Figure 2 is still full strict Cartesian from Proposition 2, but not co-Cartesian. We merely have the following inequations:

$$\mathrm{Def}\, 0 \geq \dot{0}, \qquad \mathrm{Def}(\tau + \tau') \geq \mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau'. \tag{1}$$

Interestingly, these inequations are equated when applied to the contravariant functor $(- \dot{\Rightarrow} \mathrm{Def}\, \rho)$.

Lemma 2. *For any L-objects τ, τ', ρ, we have*

$$\begin{aligned} \mathrm{Def}\, 0 \dot{\Rightarrow} \mathrm{Def}\, \rho &= \dot{0} \dot{\Rightarrow} \mathrm{Def}\, \rho \\ \mathrm{Def}(\tau + \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho &= (\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho. \end{aligned}$$

Proof. We leave the proof of the first equation to the reader. We show the second equation. Let τ, τ', ρ be L-objects. The inequation (1) implies half of the equation to be proved. We therefore show the other half displayed below:

$$\mathrm{Def}(\tau + \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho \geq (\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho.$$

Let Γ be a $\mathbf{Ctx}_L$-object and $f \in r((\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho)(\Gamma)$. The isomorphism

$$(\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau') \dot{\Rightarrow} \mathrm{Def}\, \rho \cong (\mathrm{Def}\, \tau \dot{\Rightarrow} \mathrm{Def}\, \rho) \dot{\times} (\mathrm{Def}\, \tau' \dot{\Rightarrow} \mathrm{Def}\, \rho)$$

implies that $\lambda(ev \circ (f \times inl)) \in r(\mathrm{Def}\, \tau \dot{\Rightarrow} \mathrm{Def}\, \rho)(\Gamma)$ and $\lambda(ev \circ f \times inr) \in r(\mathrm{Def}\, \tau' \dot{\Rightarrow} \mathrm{Def}\, \rho)(\Gamma)$. From Lemma 1, we obtain

$$(g_1 =)\ ev \circ (f \times inl) \in D\rho(\Gamma\tau), \qquad (g_2 =)\ ev \circ (f \times inr) \in D\rho(\Gamma\tau').$$

[1] The distributive law implies that the unique map $0 \to A \times 0$ is the isomorphism; see [7].

We thus take L-morphisms $h_1 : |\Gamma\tau| \to \rho$ and $h_2 : |\Gamma\tau'| \to \rho$ such that $g_1 = Fh_1$ and $g_2 = Fh_2$. Since F is a strict bi-Cartesian functor, we have

$$[g_1, g_2] \circ m^{\mathbb{C}}_{F|\Gamma|, F\tau, F\tau'} = F([h_1, h_2] \circ m^{L}_{|\Gamma|, \tau, \tau'}).$$

The left hand side is equal to $ev \circ (f \times (F\tau + F\tau'))$:

$$\begin{aligned}[g_1, g_2] \circ m &= ev \circ (f \times (F\tau + F\tau')) \circ [F|\Gamma| \times inl, F|\Gamma| \times inr] \circ m \\ &= ev \circ (f \times (F\tau + F\tau')).\end{aligned}$$

Hence $ev \circ (f \times (F\tau + F\tau')) \in D\rho(\Gamma(\tau + \tau'))$. From Lemma 1, we obtain $f \in r(\mathrm{Def}(\tau + \tau') \Rightarrow \mathrm{Def}\,\rho)(\Gamma)$.

We combine this lemma and the *semantic* $\top\top$*-closure operator* [16] to extract $\mathbb{K}_F$'s full reflective sub bi-CCC whose coproducts can characterise the definability predicates at sum types. The semantic $\top\top$-closure operator (we may drop the word "semantic" thereafter) is a semantic analogue of Pitt's $\top\top$-closure technique in [25], and is an instance of the author's semantic $\top\top$-lifting [16]. The $\top\top$-closure operator in this section is specialised to the argument of definability. In Section 3 it will be re-introduced in more general form, together with the proofs of propositions and theorems in this section.

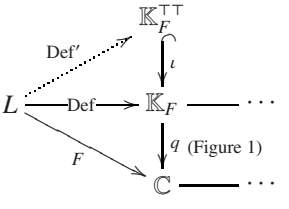


Fig. 3. Restriction of Definability Functor

The $\top\top$-closure operator is defined as follows. Let X be a $\mathbb{K}_F$-object above a $\mathbb{C}$-object I. For each L-object ρ, we define $X^{\top\top(\rho)}$ to be the vertex of the following inverse image in the fibration $q : \mathbb{K}_F \to \mathbb{C}$:

$$\begin{array}{ccc} X^{\top\top(\rho)} & \dashrightarrow & (X \Rightarrow \mathrm{Def}\,\rho) \Rightarrow \mathrm{Def}\,\rho \\ \downarrow & & \downarrow \\ I & \xrightarrow[\eta_I^{F\rho} = \lambda(ev \circ \langle \pi', \pi \rangle)]{} & (I \Rightarrow F\rho) \Rightarrow F\rho \end{array} \qquad \begin{array}{c} \mathbb{K}_F \\ \downarrow q \\ \mathbb{C} \end{array}$$

where $\eta_I^{F\rho}$ is the unit of the *continuation monad*. We then define $X^{\top\top}$, the $\top\top$-closure of X by

$$X^{\top\top} = \bigwedge_{\rho \in \mathbf{Obj}(L)} X^{\top\top(\rho)}.$$

Proposition 3. *The assignment $X \mapsto X^{\top\top}$ extends to a monad over $\mathbb{K}_F$ whose unit and multiplication are vertical (c.f. Proposition 7).*

Below we call the assignment *(semantic)* $\top\top$*-closure operator*. It indeed gives an idempotent closure operator at every fibre, as unit and multiplication are vertical.

Corollary 1. *We have $X \leq X^{\top\top}$ and $(X^{\top\top})^{\top\top} = X^{\top\top}$, and the monad is idempotent (c.f. Corollary 2).*

We then consider $\mathbb{K}_F$'s full reflective subcategory $\mathbb{K}_F^{\top\top}$ consisting of $\top\top$-closed objects (that is, objects X such that $X^{\top\top} = X$); see Figure 3. Some calculation shows that $\top\top$-closed objects form an *exponential ideal*. Therefore we obtain the following:

Proposition 4. *The category $\mathbb{K}_F^{\top\top}$ is a bi-CCC and $q \circ \iota$ is a strict bi-CC functor (c.f. Theorem 4).*

The CC structure in $\mathbb{K}_F^{\top\top}$ is inherited from $\mathbb{K}_F$, while the co-Cartesian structure is given by $\dot{0}^{\top\top}$ and $(X \dot{+} Y)^{\top\top}$. That the $\top\top$-closure operator is defined in terms of the definability predicates themselves implies the following important property:

Proposition 5. *For every L-object ρ,* Def ρ *is $\top\top$-closed (c.f. Proposition 8).*

Thus functor Def can be restricted to the full Cartesian functor (Def′ in Figure 3) to $\mathbb{K}_F^{\top\top}$. Furthermore, from Lemma 2 we obtain a *characterisation of the definability predicates at sum types* (c.f. Theorem 5-2):

$$\mathrm{Def}\, 0 = (\mathrm{Def}\, 0)^{\top\top} = \dot{0}^{\top\top} \tag{2}$$

$$\mathrm{Def}(\tau + \tau') = (\mathrm{Def}(\tau + \tau'))^{\top\top} = (\mathrm{Def}\, \tau \dot{+} \mathrm{Def}\, \tau')^{\top\top}. \tag{3}$$

This is equivalent to saying that Def′ is a strict co-Cartesian functor. To summarise:

Theorem 1 (Restriction Theorem for Definability Functor). *In Figure 3, assume that L is a small distributive category and F is a strict bi-Cartesian functor. Then* Def′ *is a full strict bi-Cartesian functor.*

We next let L be a small bi-CCC and $F : L \to \mathbb{C}$ be a strict bi-CC functor. Under this situation, the restriction of the definability functor to $\mathbb{K}_F^{\top\top}$ becomes a bi-CC functor. Since any bi-CCC is a distributive category, Def′ in Figure 3 is full bi-Cartesian from the restriction theorem. Moreover, as shown in [1] (c.f. [15]), the functor Def (and Def′) strictly preserves exponentials. Therefore we obtain the following theorem:

Theorem 2. *In figure 3, assume that L is a small bi-CCC and F is a strict bi-CC functor. Then* Def′ *in Figure 3 is a full strict bi-CC functor.*

2.3 Fullness of Free Distributive Categories in Free Bi-CCCs

As an application of the restriction theorem, we show that the canonical inclusion from the free distribute category to the free bi-CCC is full. We note that this result (and faithfulness) is proved in [10] using a different gluing category.

We fix the set B of base types and regard it as a discrete category. In this paper, by the free distributive category $(L_0(B), \eta_0 : B \to L_0(B))$ over B, we mean the distributive category with the following universal property: for any distributive category $\mathbb{C}$ and a functor $F : B \to \mathbb{C}$, there exists a unique strict bi-Cartesian functor $\overline{F} : L_0(B) \to \mathbb{C}$ such that $\overline{F} \circ \eta_0 = F$. We also define the free bi-CCC $(L_1(B), \eta_1 : B \to L_1(B))$ over B as the one having the similar universal property. Such free categories arise as term categories of the simply typed (lambda) calculus with sums. We omit the detail of the construction of free categories due to lack of space; see e.g. [18].

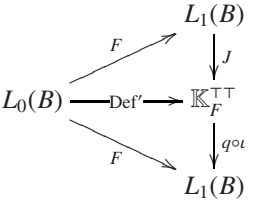
We instantiate Figure 3 with the following data:

1. $L = L_0(B)$, the free distributive category over B.
2. $\mathbb{C} = L_1(B)$, the free bi-CCC over B.
3. $F = \overline{\eta_1} : L_0(B) \to L_1(B)$, the strict bi-Cartesian functor derived from the universal property of $L_0(B)$.

Lafont applied categorical gluing to show that any small Cartesian category $\mathbb{C}$ can be fully embedded into the CCC that is relatively free with respect to $\mathbb{C}$ [17]. We apply his proof technique to the show that $\overline{\eta_1}$ is full. Here we use $\mathbb{K}_F^{\top\top}$ as a substitute for the gluing category.

Theorem 3. *The strict bi-Cartesian functor $\overline{\eta_1} : L_0(B) \to L_1(B)$ is full.*

Proof. Below we write F for $\overline{\eta_1}$. From Theorem 1 we obtain a full strict bi-Cartesian functor $\mathrm{Def}' : L_0(B) \to \mathbb{K}_F^{\top\top}$. From Proposition 4, $\mathbb{K}_F^{\top\top}$ is a bi-CCC; hence we obtain a strict bi-CC functor $J = \overline{\mathrm{Def}' \circ \eta_0} : L_1(B) \to \mathbb{K}_F^{\top\top}$. Furthermore, $q \circ \iota$ is a strict bi-CC functor, so $q \circ \iota \circ J = \mathrm{Id}$ by the universal property of $L_1(B)$. This implies that J is faithful.



In the above diagram, the upper half of the triangle commutes from the universal property of $L_0(B)$. The lower half of the triangle also commutes from Figure 3. We now show that F is full. Let $f : F\tau \to F\sigma$ be a $L_1(B)$-morphism. We seek for a $L_0(B)$-morphism g such that $f = Fg$. We first have $Jf : \mathrm{Def}'\,\tau \to \mathrm{Def}'\,\sigma$. Since Def' is full, there exists a $L_0(B)$-morphism $g : \tau \to \sigma$ such that $Jf = \mathrm{Def}'\,g = J(Fg)$. Since J is faithful, we obtain $f = Fg$.

3 $\top\top$-Closure Operators and the Restriction Theorem

In this section we focus on the general scheme that underlies in the derivation of the restriction theorem (Theorem 1), and re-establish it in more general form.

We first identify the class of fibrations in which we can consider $\top\top$-closure operators. If a functor $U : \mathbb{P} \to \mathbb{C}$ satisfies the following conditions:

$$\begin{array}{c} \mathbb{P} \\ \downarrow U \\ \mathbb{C} \end{array} \qquad \begin{array}{l} \mathbb{P} \text{ bi-CCC } (\dot{1}, \dot{\times}, \dot{0}, \dot{+}, \dot{\Rightarrow}, \dot{!}, \dot{\pi}, \dot{\pi}', \dot{\lambda}, \dot{ev}, \cdots) \\ \mathbb{C} \text{ bi-CCC } (1, \times, 0, +, \Rightarrow, !, \pi, \pi', \lambda, ev, \cdots) \\ U \text{ strict bi-CC functor and partial order} \\ \quad \text{fibration with fibered small products} \end{array}$$

we say that *U admits $\top\top$-closure operators*. Below we give a sufficient condition for ensuring that a fibration admits $\top\top$-closure operators.

Proposition 6. *Let $p : \mathbb{E} \to \mathbb{B}$ be a partial order bifibration such that $\mathbb{B}$ is a bi-CCC, p has fibred small products, fibred finite coproducts, fibred exponentials and simple products (see e.g. Jacobs [14]). Then p admits $\top\top$-closure operators.*

3.1 ⊤⊤-Closure Operators

We fix a fibration $U : \mathbb{P} \to \mathbb{C}$ which admits $\top\top$-closure operators. Each $\top\top$-closure operator takes a $\mathbb{P}$-object as a parameter called *closure parameter*. Let S be a closure parameter. For a $\mathbb{P}$-object X, we define $X^{\top\top(S)}$ to be the vertex of the following inverse image:

$$\begin{array}{ccc} X^{\top\top(S)} & \dashrightarrow & (X \mathrel{\dot{\Rightarrow}} S) \mathrel{\dot{\Rightarrow}} S \\ \downarrow & & \downarrow \\ UX & \xrightarrow[\eta^{US}_{UX} = \lambda(ev \circ \langle \pi', \pi \rangle)]{} & (UX \Rightarrow US) \Rightarrow US \end{array} \qquad \begin{array}{c} \mathbb{P} \\ \downarrow U \\ \mathbb{C} \end{array}$$

We note that the $\mathbb{C}$-morphism η^{US}_{UX} is the unit of the continuation monad $(- \Rightarrow US) \Rightarrow US$. This construction exactly coincides with the *semantic* $\top\top$*-lifting* [16] of the identity monad.

Proposition 7. *[16] Let S be a closure parameter. The assignment $X \mapsto X^{\top\top(S)}$ extends to an endofunctor $(-)^{\top\top(S)} : \mathbb{P} \to \mathbb{P}$ such that $U \circ (-)^{\top\top(S)} = U$. Furthermore, there exists vertical natural transformations $\eta^{\top\top(S)}$ and $\mu^{\top\top(S)}$ that make the triple $((-)^{\top\top(S)}, \eta^{\top\top(S)}, \mu^{\top\top(S)})$ a monad.*

Corollary 2. *Let S be a closure parameter. For any $\mathbb{P}$-object X, we have*

$$X \leq X^{\top\top(S)}, \qquad (X^{\top\top(S)})^{\top\top(S)} = X^{\top\top(S)}, \qquad S = S^{\top\top(S)}.$$

Proof. In this proof we simply write $\top\top$ for $\top\top(S)$. The first two (in)equations are immediate consequences of the previous lemma. To show $S^{\top\top} = S$, it is sufficient to show $S^{\top\top} \leq S$. We consider the following diagram:

$$\begin{array}{ccccc} S^{\top\top} & \dashrightarrow & (S \mathrel{\dot{\Rightarrow}} S) \mathrel{\dot{\Rightarrow}} S & \xrightarrow[\dot{ev} \circ \langle id, \dot{\lambda}(\pi') \circ \dot{!} \rangle]{} & S \\ \downarrow & & \downarrow & & \downarrow \\ US & \xrightarrow[\eta^{US}_{US}]{} & (US \Rightarrow US) \Rightarrow US & \xrightarrow[ev \circ \langle id, \lambda(\pi') \circ ! \rangle]{} & US \end{array} \qquad \begin{array}{c} \mathbb{P} \\ \downarrow U \\ \mathbb{C} \end{array}$$

The composite of morphisms in $\mathbb{C}$ is the identity. Hence $S^{\top\top} \leq S$ holds.

We next generalise $\top\top$-closure operators to take multiple closure parameters. Let $\mathbf{S} = \{S_i\}_{i \in I}$ be a set-indexed family of closure parameters. We define $(-)^{\top\top(\mathbf{S})}$ by

$$X^{\top\top(\mathbf{S})} = \bigwedge_{i \in I} X^{\top\top(S_i)}$$

where $\bigwedge$ denotes the fibred product. Below we only consider set-indexed family of closure parameters.

Proposition 8. *Let $\mathbf{S} = \{S_i\}_{i \in I}$ be a family of closure parameters. The mapping $X \mapsto X^{\top\top(\mathbf{S})}$ extends to a monad over $\mathbb{P}$ whose unit and multiplication are vertical. Furthermore, for any $\mathbb{P}$-object X, we have*

$$X \leq X^{\top\top(\mathbf{S})}, \qquad (X^{\top\top(\mathbf{S})})^{\top\top(\mathbf{S})} = X^{\top\top(\mathbf{S})}, \qquad S_i = S_i^{\top\top(\mathbf{S})} \quad (i \in I).$$

3.2 Full Reflective Subcategory of $\top\top$-Closed Objects

We investigate the structure of the full reflective subcategory of $\top\top$-closed objects. Let $\mathbf{S}$ be a family of closure parameters. We write $\mathbb{P}^{\top\top(\mathbf{S})}$ for $\mathbb{P}$'s full reflective subcategory consisting of $\top\top(\mathbf{S})$-closed objects (that is, objects X such that $X^{\top\top(\mathbf{S})} = X$). We write ι for the inclusion functor from $\mathbb{P}^{\top\top(\mathbf{S})}$ to $\mathbb{P}$.

Since $\mathbb{P}$ is bi-Cartesian, $\mathbb{P}^{\top\top(\mathbf{S})}$ is also bi-Cartesian (see e.g. Proposition 3.5.3 and 3.5.4, [6]). The Cartesian structure is inherited from $\mathbb{P}$, while the co-Cartesian structure is given by the following diagram:

$$X \xrightarrow{\dot{inl}} X \dot{+} Y \xrightarrow{\leq} (X \dot{+} Y)^{\top\top(\mathbf{S})} \xleftarrow{\geq} X \dot{+} Y \xleftarrow{\dot{inr}} Y \tag{4}$$

We next show that $\top\top(\mathbf{S})$-closed objects form an *exponential ideal*.

Lemma 3. *Let $\mathbf{S} = \{S_i\}_{i\in I}$ be a family of closure parameters. Then for any $\mathbb{P}$-object X and Y above I and J respectively, $Y^{\top\top(\mathbf{S})} = Y$ implies $(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} = X \dot{\Rightarrow} Y$.*

Proof. Below we only show $(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \leq X \dot{\Rightarrow} Y$; the other direction is clear as $(-)^{\top\top(\mathbf{S})}$ is a closure operator. Let $i \in I$. We define $\dot{w} : (X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \to (((X \dot{\Rightarrow} Y) \dot{\Rightarrow} S_i) \dot{\Rightarrow} S_i)$ to be the composite of $\mathbb{P}$-morphisms in the following diagram:

$$\begin{array}{ccc}
(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} & & \\
\leq \big\downarrow & & \\
(X \dot{\Rightarrow} Y)^{\top\top(S_i)} & \dashrightarrow & ((X \dot{\Rightarrow} Y) \dot{\Rightarrow} S_i) \dot{\Rightarrow} S_i \qquad \mathbb{P} \\
 & & \qquad\qquad \big\downarrow U \\
I \Rightarrow J & \xrightarrow[\eta^{US_i}_{I\Rightarrow J}]{} & ((I \Rightarrow J) \Rightarrow US_i) \Rightarrow US_i \qquad \mathbb{C}
\end{array}$$

From the diagram, $\dot{w}$ is above $\eta^{US_i}_{I\Rightarrow J}$. We also define a $\mathbb{P}$-morphism $\dot{c} : X \dot{\times} (Y \dot{\Rightarrow} S_i) \to (X \dot{\Rightarrow} Y) \dot{\Rightarrow} S_i$ by

$$\dot{c} = \dot{\lambda}(\dot{ev} \circ (id \dot{\times} \dot{ev}) \circ \langle \dot{\pi}' \circ \dot{\pi}, \langle \dot{\pi}', \dot{\pi} \circ \dot{\pi} \rangle \rangle),$$

which is above the following $\mathbb{C}$-morphism $c : I \times (J \Rightarrow US_i) \to (I \Rightarrow J) \Rightarrow US_i$:

$$c = \lambda(ev \circ (id \times ev) \circ \langle \pi' \circ \pi, \langle \pi', \pi \circ \pi \rangle \rangle).$$

By combining these, we obtain a $\mathbb{P}$-morphism

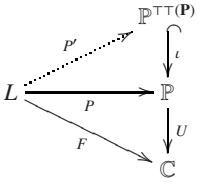
$$\dot{\lambda}(\dot{ev} \circ (\dot{w} \dot{\times} \dot{c}) \circ \dot{a}) : (X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \dot{\times} X \to (Y \dot{\Rightarrow} S_i) \dot{\Rightarrow} S_i$$

above

$$\eta^{US_i}_J \circ ev_{I,J} = \lambda(ev \circ (\eta^{US_i}_{I\Rightarrow J} \times c) \circ a) : (I \Rightarrow J) \times I \to (J \Rightarrow US_i) \Rightarrow US_i$$

(where a and $\dot{a}$ are associativity morphisms in $\mathbb{C}$ and $\mathbb{P}$ respectively). This implies that the following inequation holds for every $i \in I$ in the fibre over $(I \Rightarrow J) \times I$:

$$(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \dot{\times} X \leqslant ev^*_{I,J}(Y^{\top\top(S_i)}).$$



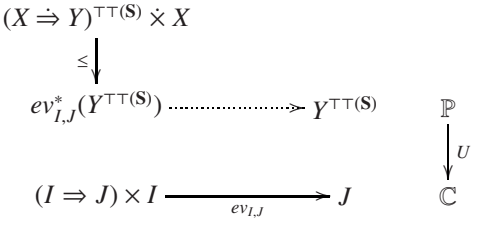
$\mathbb{P}$	bi-CCC $(\dot{1}, \dot{\times}, \dot{0}, \dot{+}, \dot{\Rightarrow})$
$\mathbb{C}$	bi-CCC $(1, \times, 0, +, \Rightarrow)$
U	strict bi-CC functor and partial order fibration with fibred small products
L	*small* category
P, F	functors
$\mathbb{P}^{\top\top(\mathbf{P})}$	bi-CCC of $\top\top(\mathbf{P})$-closed objects
ι	strict CC inclusion functor

Fig. 4. Restriction of P to $\top\top(\mathbf{P})$-Closed Objects

Therefore we have

$$(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \dot{\times} X \leqslant \bigwedge_{i \in I} ev^*_{I,J}(Y^{\top\top(S_i)}) = ev^*_{I,J}(Y^{\top\top(\mathbf{S})}).$$

Now the composite, say $\dot{v}$, of $\mathbb{P}$-morphisms in the following diagram is above $ev_{I,J}$:



so $\dot{\lambda}(\dot{v}) : (X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \to X \dot{\Rightarrow} Y^{\top\top(\mathbf{S})}$ is above $\lambda(ev_{I,J}) = \mathrm{id}$. Hence $(X \dot{\Rightarrow} Y)^{\top\top(\mathbf{S})} \leq X \dot{\Rightarrow} Y^{\top\top(\mathbf{S})} = X \dot{\Rightarrow} Y$.

Theorem 4. *For any family* $\mathbf{S}$ *of closure parameters,* $\top\top(\mathbf{S})$*-closed objects form a full reflective sub bi-CCC* $\mathbb{P}^{\top\top(\mathbf{S})}$ *of* $\mathbb{P}$*, and* $U \circ \iota : \mathbb{P}^{\top\top(\mathbf{S})} \to \mathbb{C}$ *is a faithful strict bi-CC functor.*

Proof. That $\mathbb{P}^{\top\top(\mathbf{S})}$ is a bi-CCC follows from Lemma 3 and Day's reflection theorem [8]. The CC structure on $\mathbb{P}^{\top\top(\mathbf{S})}$ is inherited from $\mathbb{P}$; so ι is a strict CC functor. In general, ι is not a co-Cartesian functor, but the coproduct diagram in (4) is strictly mapped to the coproduct diagram in $\mathbb{C}$ by $U \circ \iota$. Hence $U \circ \iota$ is a strict bi-CC functor. The faithfulness is obvious.

3.3 Restriction Theorem

We next consider a small category L and functors $F : L \to \mathbb{C}$ and $P : L \to \mathbb{P}$ such that $U \circ P = F$ (see the lower half of the commutative diagram in Figure 4). The functor P specifies a family of closure parameters $\mathbf{P} = \{P\tau\}_{\tau \in \mathbf{Obj}(L)}$.

Proposition 9. *The functor* $P : L \to \mathbb{P}$ *restricts to* $\mathbb{P}^{\top\top(\mathbf{P})}$ *(see* P' *in Figure 4).*

Proof. From Proposition 8, $(P\tau)^{\top\top(\mathbf{P})} = P\tau$ holds for any L-object τ, that is, $P\tau$ is an object in the full subcategory $\mathbb{P}^{\top\top(\mathbf{P})}$ of $\mathbb{P}$. Hence P restricts to $\mathbb{P}^{\top\top(\mathbf{P})}$.

Theorem 5 (Restriction Theorem). *In the commutative diagram in Figure 4,*

1. *If L is a Cartesian (closed) category and F and P are strict Cartesian (closed) functors, then P' is also a strict Cartesian (closed) functor.*
2. *If F is a strict co-Cartesian functor and P satisfies*

$$P0 \dot{\Rightarrow} P\rho = \dot{0} \dot{\Rightarrow} P\rho, \qquad P(\tau + \tau') \dot{\Rightarrow} P\rho = (P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho$$

   then P' is a strict co-Cartesian functor.
3. *If L is a bi-CCC, F is a strict bi-CC functor and P is a strict CC functor, then P satisfies the above equations (hence P' is a strict bi-CC functor).*

Proof. 1. The Cartesian (closed) structure in $\mathbb{P}^{\top\top(\mathbf{P})}$ is the restriction of that in $\mathbb{P}$ to $\mathbb{P}^{\top\top(\mathbf{P})}$. Since P strictly preserves Cartesian (closed) structure, so does P'.

2. Suppose $P(\tau+\tau') \dot{\Rightarrow} P\rho = (P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho$. From the definition of $\top\top(\mathbf{P})$, we have

$$\begin{aligned} P(\tau + \tau') &= P(\tau + \tau')^{\top\top(\mathbf{P})} \\ &= \bigwedge_{\rho \in \mathbf{Obj}(L)} (\eta^{F\rho}_{F\tau+F\tau'})^*((P(\tau + \tau') \dot{\Rightarrow} P\rho) \dot{\Rightarrow} P\rho) \\ &= \bigwedge_{\rho \in \mathbf{Obj}(L)} (\eta^{F\rho}_{F\tau+F\tau'})^*(((P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho) \dot{\Rightarrow} P\rho) \\ &= (P\tau \dot{+} P\tau')^{\top\top(\mathbf{P})}. \end{aligned}$$

   One can similarly show $P0 = (P0)^{\top\top(\mathbf{P})}$.

3. We show that the equations in 2 holds for each strict CC functor P such that $U \circ P = F$. In any bi-CCC $\mathbb{D}$ there is an isomorphism

$$(A + B) \Rightarrow C \underset{\beta^{A,B,C}_{\mathbb{D}}}{\overset{\alpha^{A,B,C}_{\mathbb{D}}}{\rightleftarrows}} (A \Rightarrow C) \times (B \Rightarrow C)$$

   which is preserved by strict bi-CC functors. Consider the following diagram:

$$\begin{array}{ccc} P(\tau + \tau') \dot{\Rightarrow} P\rho & & \\ \| & & \\ P((\tau + \tau') \Rightarrow \rho) & \xrightarrow{P(\alpha_L^{\tau,\tau',\rho})} & P((\tau \Rightarrow \rho) \times (\tau' \Rightarrow \rho)) \qquad \mathbb{P} \\ & & \| \qquad\qquad \downarrow U \\ (P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho & \xleftarrow[\beta^{P\tau,P\tau',P\rho}_{\mathbb{P}}]{} & (P\tau \dot{\Rightarrow} P\rho) \dot{\times} (P\tau' \dot{\Rightarrow} P\rho) \\ (F\tau + F\tau') \Rightarrow F\rho & \underset{\beta^{F\tau,F\tau',F\rho}_{\mathbb{C}}}{\overset{\alpha^{F\tau,F\tau',F\rho}_{\mathbb{C}}}{\rightleftarrows}} & (F\tau \Rightarrow F\rho) \times (F\tau' \Rightarrow F\rho) \qquad \mathbb{C} \end{array}$$

   From $U \circ P = F$, the morphism $P(\alpha_L^{\tau,\tau',\rho})$ is above $\alpha^{F\tau,F\tau',F\rho}_{\mathbb{C}}$. Therefore the composition of morphisms in $\mathbb{P}$ is above $\beta^{F\tau,F\tau',F\rho}_{\mathbb{C}} \circ \alpha^{F\tau,F\tau',F\rho}_{\mathbb{C}} = \mathrm{id}$. Thus we obtain $P(\tau + \tau') \dot{\Rightarrow} P\rho \leq (P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho$. The other direction, $P(\tau + \tau') \dot{\Rightarrow} P\rho \geq (P\tau \dot{+} P\tau') \dot{\Rightarrow} P\rho$, follows from a similar argument.
   We leave the proof of $P0 \dot{\Rightarrow} P\rho = \dot{0} \dot{\Rightarrow} P\rho$ to the reader.

Theorem 1 is an instance of this general restriction theorem. In Figure 4 we instantiate U with $q : \mathbb{K}_F \to \mathbb{C}$, L with a small distributive category, F with a bi-Cartesian functor and P with the definability functor of F. From Proposition 2, Lemma 2 and Theorem 5-2, we obtain Theorem 1.

3.4 A Characterisation of Definable Morphisms by Weak Logical Predicates

We finally give a characterisation of morphisms definable by the simply typed lambda calculus with sums by means of *weak logical predicates*. Let B be the set of base types, $F : L_1(B) \to \mathbb{C}$ be a bi-CC functor and $U : \mathbb{P} \to \mathbb{C}$ be a fibration admitting ⊤⊤-closure operators. An $\mathbf{Obj}(L_1(B))$-indexed family P of $\mathbb{P}$-objects is called *weak logical predicate* (with respect to F and U) if the following holds for any $L_1(B)$-objects τ, τ', ρ:

- $P\tau$ is above $F\tau$,
- $P(\tau \times \tau') = P\tau \mathbin{\dot\times} P\tau'$, $P1 = \dot{1}$, $P(\tau \Rightarrow \tau') = P\tau \mathbin{\dot\Rightarrow} P\tau'$, and
- $(P\tau + P\tau) \mathbin{\dot\Rightarrow} P\rho = P((\tau + \tau') \Rightarrow \rho)$, $\dot{0} \mathbin{\dot\Rightarrow} P\rho = P(0 \Rightarrow \rho)$; (c.f. Theorem 5-2).

We say that a $\mathbb{C}$-morphism $f : F\tau \to F\tau'$ is *invariant under* P if there exists a (necessarily unique) $\mathbb{P}$-morphism $g : P\tau \to P\tau'$ above f.

Lemma 4 (Basic Lemma for Weak Logical Predicates). *Let P be a weak logical predicate with respect to a bi-CC functor $F : L_1(B) \to \mathbb{C}$ and a fibration $U : \mathbb{P} \to \mathbb{C}$ admitting ⊤⊤-closure operators. Then for any $L_1(B)$-morphism $f : \tau \to \sigma$, Ff is invariant under P.*

Theorem 6. *Let $\mathbb{C}$ be a bi-CCC and $F : L_1(B) \to \mathbb{C}$ be a bi-CC functor. Then a $\mathbb{C}$-morphism f is definable by F (i.e. f is in the image of F) if and only if f is invariant under any weak logical predicate with respect to any fibration $U : \mathbb{P} \to \mathbb{C}$ admitting ⊤⊤-closure operators.*

Proof. If f is invariant under any weak logical predicate, then it should be so under Def with respect to F and q in Section 2. Since Def is full, f is definable by F. The converse is immediate from Lemma 4.

4 Related Work

4.1 Grothendieck Logical Predicates

We briefly review Fiore and Simpson's *Grothendieck logical predicates* [11]. They are a further refinement of Jung and Tiuryn's Kripke predicates with varying arity using *Grothendieck topology*. Let $\mathbf{C}$ be a small category, K be a Grothendieck topology on $\mathbf{C}$ and $a : \mathbf{C} \to \mathbb{C}$ be a functor called *arity functor*. The topology K induces an *idempotent monad* $\mathcal{K}$ over the category $\mathbf{Sub}([\mathbf{C}^{op}, \mathbf{Set}])$ of subpresheaves [4], and one obtains the full reflective subcategory $\mathbf{ClSub}_{\mathcal{K}}([\mathbf{C}, \mathbf{Set}]) \to \mathbf{Sub}([\mathbf{C}, \mathbf{Set}])$ of $\mathcal{K}$-closed subobjects. One can verify that $\mathbf{ClSub}_{\mathcal{K}}([\mathbf{C}, \mathbf{Set}])$ is a bi-CCC, and the composite of functors $\mathbf{ClSub}_{\mathcal{K}}([\mathbf{C}, \mathbf{Set}]) \to \mathbf{Sub}([\mathbf{C}, \mathbf{Set}]) \to [\mathbf{C}, \mathbf{Set}]$ strictly preserves the bi-CC structure. We then take the pullback of the composite along $H_a : \mathbb{C} \to [\mathbf{C}^{op}, \mathbf{Set}]$ defined by

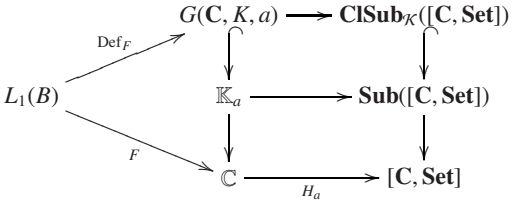


Fig. 5. Construction of the Category of Grothendieck Predicates

$H_a(C) = \mathbb{C}(a-, C)$. This yields the category $G(\mathbf{C}, K, a)$ of Grothendieck predicates, which is also a bi-CCC (see Figure 5). Every Grothendieck logical predicate is then formulated as a bi-CC functor from $L_1(B)$ (the free bi-CCC over the set B of base types) to $G(\mathbf{C}, K, a)$.

Let $\mathbb{C}$ be a bi-CCC whose coproducts are stable and $F : L \to \mathbb{C}$ be a strict bi-CC functor. For the characterisation of the morphisms definable by F, Fiore and Simpson instantiated $\mathbf{C}$ with a syntactically constructed category of constrained contexts, K with a suitable topology on $\mathbf{C}$ and a with the interpretation of contexts by F. They showed that the functor $\mathrm{Def} : L_1(B) \to G(\mathbf{C}, K, a)$ that captures the morphisms definable by F is a bi-CC functor, that is, a Grothendieck logical predicate.

We give an informal comparison of their approach and our approach.

1. In our approach the parameter category for Kripke predicates is the partial order $\mathbf{Ctx}_L$ of context inclusions, while in [11] a non-partial order category of constrained contexts and renamings is used (although it can be switched to the partial order called *Diaconescu cover* without affecting the result; see Section 5, [11]).
2. The closure operator $\mathcal{K}$ can be restricted to the one $\mathcal{K}|_{\mathbb{K}_a}$ over $\mathbb{K}_a$, and $G(\mathbf{C}, K, a)$ can be seen as the full reflective subcategory of the $\mathcal{K}|_{\mathbb{K}_a}$-closed subobjects. In our approach we derived the ($\top\top$-)closure operator over $\mathbb{K}_F$ from the definability predicates itself, and considered the full reflective subcategory $\mathbb{K}_F^{\top\top}$ of $\top\top$-closed subobjects. Both approaches perform a similar categorical construction to obtain the category for characterising the definability predicates, but with different closure operators.
3. One drawback of our characterisation is that the definability predicates at sum types are *not* inductively characterised, although they are coproducts of the definability predicates at lower types. This is because the $\top\top$-closure operator used in equations (2) and (3) refers to the definability predicates at every type. On the other hand, in Fiore and Simpson's work the definability predicates at sum types are completely determined by those at lower types.
4. One advantage of our characterisation is that it holds for any interpretation of the simply typed lambda calculus with sums in *any* bi-CCC.

4.2 Other Related Work

Pitts introduced $\top\top$-closure operator for capturing the concept of admissible relations in the syntactic study of a polymorphic functional language [24]. Operators that are

similar to the TT-closure had already appeared in various forms: the duality operator in the phase-space semantics of linear logic [13] and Parigot's technique of the strong normalisation of the second order classical natural deduction [23] are such instances. The notion of TT-closure operators also appears in other studies [21,5].

Hinted from Pitts' TT-closure operator, Lindley and Stark introduced a new technique called TT-lifting for extending the strong normalisation proof using computability predicate technique to Moggi's computational metalanguage [20,19]. Their TT-lifting was later categorically formulated as a method to lift strong monads on the base category of a fibration to the one on its total category [16] by the author. There, TT-closure operators are formulated as the TT-lifting of the identity monad.

It is widely recognised that re-establishing properties that hold in the lambda calculus with only arrow types is difficult under the presence of sum types. For instance, the design of a confluent and strongly normalising rewriting system (with β-reduction and η-expansion) for the simply typed lambda calculus with sums [12] and the proof of the completeness of the equational theory of the lambda calculus with sums in **Set** [9] exhibits the intrinsic difficulty in handling sums. In this stream of research Grothendieck logical predicates are shown to be an effective tool in reasoning about the lambda calculus with sums. They are applied to the correctness of the normalisation-by-evaluation algorithm [2] and the proof of the extensional normalisation [3] for the lambda calculus with sums.

Acknowledgement

The author is grateful to anonymous reviewers for their constructive comments. The author thank Masahito Hasegawa, Kazushige Terui, Susumu Nishimura, Jacques Garrigue, Ichiro Hasuo and the members of RIMS computer science group for valuable discussions.

References

1. Alimohamed, M.: A characterization of lambda definability in categorical models of implicit polymorphism. Theor. Comput. Sci. 146(1&2), 5–23 (1995)
2. Altenkirch, T., Dybjer, P., Hofmann, M., Scott, P.: Normalization by evaluation for typed lambda calculus with coproducts. In: LICS, pp. 303–310 (2001)
3. Balat, V., Di Cosmo, R., Fiore, M.: Extensional normalisation and type-directed partial evaluation for typed lambda calculus with sums. In: Jones, N., Leroy, X. (eds.) POPL, pp. 64–76. ACM, New York (2004)
4. Barr, M., Wells, C.: Toposes, Triples and Theories. Springer, Heidelberg (1998)
5. Benton, N.: A typed, compositional logic for a stack-based abstract machine. In: Yi, K. (ed.) APLAS 2005. LNCS, vol. 3780, pp. 364–380. Springer, Heidelberg (2005)
6. Borceux, F.: Handbook of Categorical Algebra 1. Encyclopedia of Mathematics and Its Applications, vol. 50. Cambridge University Press, Cambridge (1994)
7. Cockett, J.: Introduction to distributive categories. Mathematical Structures in Computer Science 3(3), 277–307 (1993)
8. Day, B.: A reflection theorem for closed categories. Journal of pure and applied algebra 2(1), 1–11 (1972)

9. Dougherty, D., Subrahmanyam, R.: Equality between functionals in the presence of coproducts. In: LICS, pp. 282–291. IEEE Computer Society, Los Alamitos (1995)
10. Fiore, M., Di Cosmo, R., Balat, V.: Remarks on isomorphisms in typed lambda calculi with empty and sum types. In: LICS, pp. 147–157. IEEE Computer Society, Los Alamitos (2002)
11. Fiore, M., Simpson, A.: Lambda definability with sums via grothendieck logical relations. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 147–161. Springer, Heidelberg (1999)
12. Ghani, N.: $\beta\eta$-equality for coproducts. In: Dezani-Ciancaglini, M., Plotkin, G.D. (eds.) TLCA 1995. LNCS, vol. 902, pp. 171–185. Springer, Heidelberg (1995)
13. Girard, J.-Y.: Linear logic. Theor. Comp. Sci. 50, 1–102 (1987)
14. Jacobs, B.: Categorical Logic and Type Theory. Elsevier, Amsterdam (1999)
15. Jung, A., Tiuryn, J.: A new characterization of lambda definability. In: Bezem, M., Groote, J.F. (eds.) TLCA 1993. LNCS, vol. 664, pp. 245–257. Springer, Heidelberg (1993)
16. Katsumata, S.: A semantic formulation of $\mathtt{TT}$-lifting and logical predicates for computational metalanguage. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 87–102. Springer, Heidelberg (2005)
17. Lafont, Y.: Logiques, Categóries et Machines. PhD thesis, Universit''e Paris VII (1988)
18. Lambek, J., Scott, P.J.: Introduction to Higher Order Categorical Logic. In: Cambridge studies in advanced mathematics. CUP (1986)
19. Lindley, S.: Normalisation by Evaluation in the Compilation of Typed Functional Programming Languages. PhD thesis, University of Edinburgh (2004)
20. Lindley, S., Stark, I.: Reducibility and $\mathtt{TT}$-lifting for computation types. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 262–277. Springer, Heidelberg (2005)
21. Melliès, P.-A., Vouillon, J.: Recursive polymorphic types and parametricity in an operational framework. In: Proc. LICS 2005, pp. 82–91. IEEE Computer Society, Los Alamitos (2005)
22. Mitchell, J.: Representation independence and data abstraction. In: Proc. POPL 1986, pp. 263–276 (1986)
23. Parigot, M.: Proofs of strong normalisation for second order classical natural deduction. Journal of Symbolic Logic 62(4), 1461–1479 (1997)
24. Pitts, A.: Parametric polymorphism and operational equivalence. Mathematical Structures in Computer Science 10(3), 321–359 (2000)
25. Pitts, A., Stark, I.: Operational reasoning for functions with local state. In: Gordon, A.D., Pitts, A.M. (eds.) Higher Order Operational Techniques in Semantics. Publications of the Newton Institute, pp. 227–273. Cambridge University Press, Cambridge (1998)
26. Walters, R.F.C.: Categories and Computer Science. Cambridge Computer Science Texts. Cambridge University Press, Cambridge (1992)

Superposition for Fixed Domains

Matthias Horbach and Christoph Weidenbach

Max-Planck-Institut für Informatik
Saarbrücken, Germany
{horbach,weidenb}@mpii.de

Abstract. Superposition is an established decision procedure for a variety of first-order logic theories represented by sets of clauses. A satisfiable theory, saturated by superposition, implicitly defines a perfect term-generated model for the theory. Proving universal properties with respect to a saturated theory directly leads to a modification of the perfect model's term-generated domain, as new Skolem functions are introduced. For many applications, this is not desired. Therefore, we propose the first superposition calculus that can explicitly represent existentially quantified variables and can thus compute with respect to a given domain. This calculus is sound and complete for a first-order fixed domain semantics. For some classes of formulas and theories, we can even employ the calculus to prove properties of the perfect model itself, going beyond the scope of known superposition based approaches.

1 Introduction

One of the most powerful calculi for first-order logic with equality is superposition [1,13,16]. This is in particular demonstrated by superposition instances effectively deciding almost any known decidable classical subclass of first-order logic, e.g. the monadic class with equality [2] or the guarded fragment with equality [7], as well as a number of decidable first-order classes that have been proven decidable for the first time by means of the superposition calculus [12,9,15,10]. The key to this success is an inherent redundancy notion based on the term-generated minimal model $\mathcal{I}_N$ of a clause set N. If all inferences from a clause set N are redundant (then N is called *saturated*) and N does not contain the empty clause, then $\mathcal{I}_N$ is a minimal model of N.

A formula Φ is entailed by a clause set N with respect to the standard first-order semantics, written $N \models \Phi$, if Φ holds in all models of N over all possible domains. For a number of applications, this is not the desired semantics. Instead, only Herbrand models of N over the signature $\mathcal{F}$ should be considered, written $N \models_{\mathcal{F}} \Phi$. Even stronger, the validity of Φ is considered with respect to the model $\mathcal{I}_N$, written $\mathcal{I}_N \models \Phi$ or alternatively $N \models_{Ind} \Phi$. It holds that $\mathcal{I}_N \in \{\mathcal{M} \mid \mathcal{M} \models_{\mathcal{F}} N\} \subseteq \{\mathcal{M} \mid \mathcal{M} \models N\}$ and the opposite inclusions hold for the sets of valid formulas: $\{\Phi \mid N \models_{Ind} \Phi\} \supseteq \{\Phi \mid N \models_{\mathcal{F}} \Phi\} \supseteq \{\Phi \mid N \models \Phi\}$.

Consider the following small example, demonstrating the differences of the three semantics. The clause set $N = \{ \rightarrow G(s(0), 0),\ G(x, y) \rightarrow G(s(x), s(y)) \}$

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 293–307, 2008.
© Springer-Verlag Berlin Heidelberg 2008

is finitely saturated by superposition, where the domain of $\mathcal{I}_N$ is isomorphic to the naturals and $G_{\mathcal{I}_N}$ is a subset of the greater relation. Now for the different entailment relations the following holds:

$$\begin{array}{lll} N \models G(s(s(0)), s(0)) & N \models_{\mathcal{F}} G(s(s(0)), s(0)) & N \models_{Ind} G(s(s(0)), s(0)) \\ N \not\models \forall x.G(s(x), x) & N \models_{\mathcal{F}} \forall x.G(s(x), x) & N \models_{Ind} \forall x.G(s(x), x) \\ N \not\models \forall x.\neg G(x, x) & N \not\models_{\mathcal{F}} \forall x.\neg G(x, x) & N \models_{Ind} \forall x.\neg G(x, x) \end{array}$$

Superposition is a sound and complete calculus for the standard semantics $\models$. In this paper, we develop a sound and complete calculus for $\models_{\mathcal{F}}$. Given a clause set N and a purely existentially quantified conjecture, standard superposition is also complete for $\models_{\mathcal{F}}$. The problem arises with universally quantified conjectures that become existentially quantified after negation. Then, as soon as these existentially quantified variables are Skolemized, the standard superposition calculus applied afterwards no longer computes modulo $\models_{\mathcal{F}}$, but modulo $\models_{\mathcal{F} \cup \{f_1, \ldots, f_n\}}$ where $f_1, \ldots, f_n$ are the introduced Skolem functions. The idea behind our new calculus is not to Skolemize existentially quantified variables, but to treat them explicitly by the calculus. This is represented by an extended clause notion, containing a constraint for the existentially quantified variables. For example, the above conjecture $\forall x.G(s(x), x)$ results after negation in the clause $u \approx x \,\|\, G(s(x), x) \rightarrow$ with existential variable u. In addition to standard first-order equational reasoning, the inference and reduction rules of the new calculus take also care of the constraint (see section 3).

In general, a $\models_{\mathcal{F}}$ unsatisfiability proof of a constrained clause set requires the computation of infinitely many empty clauses. This does not come as a surprise because we have to show that an existentially quantified clause cannot be satisfied by a term-generated infinite domain. In order to represent this infinite set of empty clauses finitely, a further induction rule, based on the minimal model semantics $\models_{Ind}$, can be employed. We prove the new rule sound in section 4 and show its potential. In general, our calculus can cope with (conjecture) formulas of the form $\forall^* \exists^* \Phi$ and does not impose special conditions on N (except saturation for $\models_{Ind}$), which is beyond any known result on superposition based calculi proving properties of $\models_{\mathcal{F}}$ or $\models_{Ind}$ [11,4,8,3,6,14]. This, together with potential extensions and directions of research, is discussed in the final section 5.

2 Preliminaries

We build on the notions of [1,16] and shortly recall here the most important concepts as well as the specific extensions needed for the new superposition calculus. Let $\mathcal{F}$ be a *signature*, i.e. a set of function symbols of fixed arity, and X an infinite set of variables. 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 *ground terms* over $\mathcal{F}$. A (standard universal) *clause* is a pair of multisets of equations, written $\Gamma \rightarrow \Delta$, interpreted as the conjunction of all atoms in Γ implying the disjunction of all atoms in Δ. The *empty clause* is denoted by $\square$. Any (reduction) ordering $\prec$ on terms can be lifted to clauses in the usual way as its twofold multiset extension over equations and

clauses (cf. [1]). Predicates can be encoded in this setting as equations with a special "true" constant on the right hand side as usual. Please note that in this case we consider a many-sorted framework where the predicative sort is separated from the sort of all other terms. As there are no variables of the predicative sort, we never explicitly express the sorting.

By $t|_p$ we denote the subterm of t at *position* p. The term that arises from t by replacing the subterm at position p by the term r is $t[r]_p$. A *substitution* σ is a map from a set $X' \subseteq X$ of variables to $\mathcal{T}(\mathcal{F}, X)$, where the domain $\mathrm{dom}(\sigma) = \{x \in X' \mid x\sigma \neq x\}$ is finite. The most general unifier of two terms $s, t \in \mathcal{T}(\mathcal{F}, X)$ is denoted by $\mathrm{mgu}(s, t)$. Remark that, even if we consider predicates, there are no variables of the predicative sort and hence substitutions do not introduce symbols of this sort.

A *Herbrand interpretation* over the signature $\mathcal{F}$ is a congruence on the ground terms $\mathcal{T}(\mathcal{F})$. We recall the construction of the special Herbrand interpretation $\mathcal{I}_N$ derived from a clause set N in [1]. If N is consistent and saturated with respect to a certain inference system, then $\mathcal{I}_N \models N$ and $\mathcal{I}_N$ is called a minimal model of N. Let $\prec$ be a well-founded reduction ordering that is total on ground terms. We use induction on the clause ordering $\prec$ to define sets of equations E_C, R_C and I_C for all ground clauses over $\mathcal{T}(\mathcal{F})$ by $R_C = \bigcup_{C \succ C'} E_{C'}$, and $I_C = R^*_C$, i.e. the reflexive, transitive closure of R_C. Moreover $E_C = \{s \approx t\}$ (and we say that C *produces* $s \approx t$), if $C = \Gamma \to \Delta, s \approx t$ is a ground instance of a clause from N such that (i) $s \succ t$ and $s \approx t$ is a strictly maximal occurrence of an equation in C, (ii) s is irreducible by R_C, (iii) $\Gamma \subseteq I_C$, and (iv) $\Delta \cap I_C = \emptyset$. Otherwise $E_C = \emptyset$. Finally, we define the confluent and terminating ground rewrite system $R = \bigcup_C E_C$ as the set of all produced equations and set $\mathcal{I}_N = R^*$ over the domain $\mathcal{T}(\mathcal{F})$.

We distinguish a finite set $V \subset X$ of *existential variables*. Elements of V are denoted as u, v and elements of $X \setminus V$ as x, y, z. A *constrained clause* $v_1 \approx t_1, \ldots, v_n \approx t_n \parallel C$ consists of a sequence of equations $v_1 \approx t_1, \ldots, v_n \approx t_n$ called the *constraint* and a clause C, such that $V = \{v_1, \ldots, v_n\}$, $v_i \neq v_j$ for $i \neq j$, and neither C nor $t_1, \ldots, t_n$ contain an existential variable. In particular, constraints always constitute a solved unification problem. The constrained clause is called *ground* if C and $t_1, \ldots, t_n$ are ground. A constraint α induces a substitution $V \to \mathcal{T}(\mathcal{F}, X)$, which we will denote by σ_α.

Constrained clauses are considered equal up to renaming of non-existential variables. For example, the clauses $u \approx x, v \approx y \parallel P(x)$ and $u \approx y, v \approx x \parallel P(y)$ are considered equal, but they are both different from $v \approx x, u \approx y \parallel P(x)$. If the universal variable x does not appear in C, we abbreviate $v_1 \approx t_1, \ldots, v_n \approx t_n, v \approx x \parallel C$ as $v_1 \approx t_1, \ldots, v_n \approx t_n \parallel C$. A clause $\parallel C$ is *unconstrained*. If $V = \{v_1, \ldots, v_n\}$, and $x_1, \ldots, x_m$ are the non-existential variables in a clause set N, the semantics of N is $\exists \boldsymbol{v}. \forall \boldsymbol{x}. \bigwedge_{(\alpha \parallel C) \in N} \alpha \to C$, i.e. an interpretation $\mathcal{M}$ is said to *model* N, written $\mathcal{M} \models N$ iff the formula $\exists \boldsymbol{v}. \forall \boldsymbol{x}. \bigwedge_{(\alpha \parallel C) \in N} \alpha \to C$ is valid in $\mathcal{M}$.

We extend $\prec$ to constraints by $v_1 \approx s_1, \ldots, v_n \approx s_n \prec v_1 \approx t_1, \ldots, v_n \approx t_n$ iff $s_1 \preceq t_1 \wedge \ldots \wedge s_n \preceq t_n$ and $s_1 \neq t_1 \vee \ldots \vee s_n \neq t_n$. Constrained clauses are ordered lexicographically with priority on the constraint, i.e. $\alpha \parallel C \prec \beta \parallel D$ iff $\alpha \prec \beta$,

or $\alpha = \beta$ and $C \prec D$. This ordering is not total on ground clauses, but strong enough to support our completeness result and the usual notion of redundancy.

We use the symbols $\forall$ and $\exists$ both in first-order formulas and on the meta level, where they are also used for higher-order quantification.

3 First-Order Reasoning in Fixed Domains

In this section, we will give a saturation procedure for sets of constrained clauses over a domain $\mathcal{T}(\mathcal{F})$ and show how it is possible to decide whether a saturated clause set possesses a Herbrand model over $\mathcal{F}$.

We consider the following inference rules, which are defined with respect to a reduction ordering $\prec$ on $\mathcal{T}(\mathcal{F}, X)$ that is total on ground terms. Most of the rules are quite similar to the usual superposition rules, just generalized to constrained clauses. However, they also require additional treatments of the constraints. To simplify the presentation below we do not enrich the calculus by the use of a negative literal selection function, although this is also possible. As usual, we consider the universal variables in different appearing clauses to be renamed apart. If $\alpha_1 = v_1 \approx s_1, \ldots, v_n \approx s_n$ and $\alpha_2 = v_1 \approx t_1, \ldots, v_n \approx t_n$ are two constraints, then we write $\alpha_1 \approx \alpha_2$ for the equations $s_1 \approx t_1, \ldots, s_n \approx t_n$, which do not contain any existential variables, and we write $\mathrm{mgu}(\alpha_1, \alpha_2)$ for the most general common unifier of $(s_1, t_1), \ldots, (s_n, t_n)$.

- Equality Resolution:
$$\frac{\alpha \parallel \Gamma, s \approx t \to \Delta}{(\alpha \parallel \Gamma \to \Delta)\sigma}$$
where (i) $\sigma = \mathrm{mgu}(s, t)$ and (ii) $(s \approx t)\sigma$ is maximal in $(\Gamma, s \approx t \to \Delta)\sigma$.
- Equality Factoring:
$$\frac{\alpha \parallel \Gamma \to \Delta, s \approx t, s' \approx t'}{(\alpha \parallel \Gamma, t \approx t' \to \Delta, s' \approx t')\sigma}$$
where (i) $\sigma = \mathrm{mgu}(s, s')$, (ii) $(s \approx t)\sigma$ is maximal in $(\Gamma \to \Delta, s \approx t, s' \approx t')\sigma$, and (iii) $t\sigma \not\succeq s\sigma$
- Superposition, Right:
$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \quad \alpha_2 \parallel \Gamma_2 \to \Delta_2, s[l']_p \approx t}{(\alpha_1 \parallel \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2, s[r]_p \approx t)\sigma_1\sigma_2}$$
where (i) $\sigma_1 = \mathrm{mgu}(l, l')$, $\sigma_2 = \mathrm{mgu}(\alpha_1\sigma_1, \alpha_2\sigma_1)$, (ii) $(l \approx r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l \approx r)\sigma_1\sigma_2$ and $(s \approx t)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_2 \to \Delta_2, s \approx t)\sigma_1\sigma_2$, (iii) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$ and $t\sigma_1\sigma_2 \not\succeq s\sigma_1\sigma_2$, and (iv) l' is not a variable.
- Superposition, Left:
$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \quad \alpha_2 \parallel \Gamma_2, s[l']_p \approx t \to \Delta_2}{(\alpha_1 \parallel \Gamma_1, \Gamma_2, s[r]_p \approx t \to \Delta_1, \Delta_2)\sigma_1\sigma_2}$$

where (i) $\sigma_1 = \mathrm{mgu}(l, l')$, $\sigma_2 = \mathrm{mgu}(\alpha_1\sigma_1, \alpha_2\sigma_1)$, (ii) $(l \approx r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l \approx r)\sigma_1\sigma_2$, $(s \approx t)\sigma_1\sigma_2$ is maximal in $(\Gamma_2 \to \Delta_2, s \approx t)\sigma_1\sigma_2$, (iii) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$ and $t\sigma_1\sigma_2 \not\succeq s\sigma_1\sigma_2$, and (iv) l' is not a variable.

– Constraint Superposition:

$$\frac{\alpha_1 \parallel \Gamma_1 \to \Delta_1, l \approx r \quad v \approx t[l']_p, \alpha_2 \parallel \Gamma_2 \to \Delta_2}{(v \approx t[r]_p, \alpha_2 \parallel \alpha_1 \approx (v \approx t[r]_p, \alpha_2), \Gamma_1, \Gamma_2 \to \Delta_1, \Delta_2)\sigma}$$

where (i) $\sigma = \mathrm{mgu}(l, l')$, (ii) $(l \approx r)\sigma$ is strictly maximal in $(\Gamma_1 \to \Delta_1, l \approx r)\sigma$, (iii) $r\sigma \not\succeq l\sigma$, and (iv) l' is not a variable.

– Equality Elimination:

$$\frac{\alpha_1 \parallel \Gamma \to \Delta, l \approx r \quad v \approx t[r']_p, \alpha_2 \parallel \square}{(\alpha_1 \parallel \Gamma \to \Delta)\sigma_1\sigma_2}$$

where (i) $\sigma_1 = \mathrm{mgu}(r, r')$, $\sigma_2 = \mathrm{mgu}(\alpha_1\sigma_1, (v \approx t[l]_p, \alpha_2)\sigma_1)$, (ii) $(l \approx r)\sigma_1\sigma_2$ is strictly maximal in $(\Gamma \to \Delta, l \approx r)\sigma_1\sigma_2$, (iii) $r\sigma_1\sigma_2 \not\succeq l\sigma_1\sigma_2$, and (iv) r' is not a variable.

While the other rules keep clauses with different constraints strictly separate, constraint superposition and equality elimination transfer information across these bounds, allowing to derive, e.g., $u \approx b \parallel \square$ from $u \approx b \parallel \to a \approx b$ and $u \approx a \parallel \square$ where $a \succ b$.

This inference system contains the standard universal superposition calculus as the special case when there are no existential variables at all present, i.e. $V = \emptyset$ and all constraints are empty: The rules equality resolution, equality factoring, and superposition right and left reduce to their non-constrained counterparts and the constraint superposition and equality elimination rules become obsolete.

A ground constrained clause $\alpha \parallel C$ is called *redundant* with respect to a set N of constrained clauses if there are ground instances $\alpha \parallel C_1, \ldots, \alpha \parallel C_k$ of clauses in N, such that $C_1, \ldots, C_k \models C$ and $C_i \prec C$ for all i. As an alternative, we could choose $\models_{\mathcal{F}}$ for defining redundancy. A non-ground constrained clause is redundant if all its ground instances are redundant. A ground inference with conclusion $\beta \parallel B$ is called *redundant* with respect to N if either some premise is redundant or if there are ground instances $\beta \parallel C_1, \ldots, \beta \parallel C_k$ of clauses in N, such that $C_1, \ldots, C_k \models B$ and $C_1, \ldots, C_n$ are smaller than the maximal premise of the ground inference. A non-ground inference is redundant if all its ground instances are redundant. A clause set N is *saturated* if each inference with premises in N is redundant with respect to N.

As constrained clauses are just special classes of clauses, the construction of a Herbrand model of N is almost identical to the one for universal clause sets [1]. The main difference is that we now have to account for existential variables before starting the construction. To define a Herbrand interpretation $\mathcal{I}_N$ of a set N of constrained clauses, we proceed in two steps:

1. Let $A_N = \{\alpha \mid (\alpha \parallel \square) \in N\}$. Let α_N be a minimal ground constraint with respect to $\prec$ such that α_N is not an instance of any $\alpha \in A_N$ if such a constraint exists. Otherwise we say that A_N is *covering*. In this case let α_N be an arbitrary ground constraint.

2. The Herbrand interpretation $\mathcal{I}_N$ is defined as the (classical) minimal model of the unconstrained clause set $\{C\sigma \mid (\alpha \parallel C) \in N \wedge \alpha\sigma = \alpha_N\}$.

Note that even if A_N is not covering, α_N is usually not uniquely defined, e.g. $\alpha_N = \{u{\approx}0, v{\approx}s(0)\}$ or $\alpha_N = \{u{\approx}s(0), v{\approx}0\}$ for $\mathcal{F} = \{0, s\}$ and the clause set $N = \{u{\approx}0, v{\approx}0 \parallel \Box\}$, which results in $A_N = \{(u{\approx}0, v{\approx}0)\}$.

While it is well known how the second step works, it is not that obvious that one can decide whether A_N is covering and, if it is not, effectively compute α_N. This is, however, possible for finite N: Let $\{x_1, \ldots, x_m\} \subseteq X \setminus V$ be the set of non-existential variables appearing in A_N. A_N is covering if and only if the formula $\forall x_1, \ldots, x_m. \bigwedge_{\alpha \in A_N} \neg\alpha$ is satisfiable in $\mathcal{T}(\mathcal{F})$. Such so-called disunification problems have been studied among others by Comon and Lescanne [5], who gave a terminating algorithm that eliminates the universal quantifiers from this formula and transforms the initial problem into an equivalent formula from which the set of solutions can easily be read off.

We will now show that a saturated constrained clause set N has a Herbrand model over $\mathcal{F}$ (namely $\mathcal{I}_N$) if and only if A_N is not covering, and call $\mathcal{I}_N$ a *minimal model* of N in this case. Since $\mathcal{I}_N$ is defined via a set of unconstrained clauses, it inherits all properties of models of purely universal clause sets. Above all, we will use the property that the rewrite system R constructed in parallel with this interpretation is confluent and terminating. We write $s \to_R t$ if there is a rule $l{\approx}r \in R$, also written $l{\to}r \in R$, and a position p of s such that $s = s[l]_p$ and $t = s[r]_p$. In this case, s is called *reducible* by R. The notions of positions, $\to_R$ and reducibility lift naturally to constraints.

Lemma 1. *Let N be saturated. If A_N is not covering then α_N is irreducible by R.*

Proof. Assume that there is a position p and a rule $l\sigma \to r\sigma \in R$ produced by a ground instance $(\beta \parallel \Lambda \to \Pi, l{\approx}r)\sigma$ of a clause $\beta \parallel \Lambda \to \Pi, l{\approx}r \in N$, such that $l\sigma = \alpha_N|_p$.

Because of the minimality of α_N, there must be a clause $\gamma \parallel \Box \in N$ with $\gamma\sigma'{\approx}\alpha_N[r\sigma]_p$. Since by definition α_N is not an instance of γ, the position p is a non-variable position of γ. Since furthermore $\beta\sigma = \alpha_N = \gamma\sigma'[l\sigma]_p$, there is an equality elimination inference

$$\frac{\beta \parallel \Lambda \to \Pi, l{\approx}r \quad \gamma \parallel \Box}{(\beta \parallel \Lambda \to \Pi)\sigma_1\sigma_2}\ \sigma_1 = \mathrm{mgu}(\gamma|_p, r),\ \sigma_2 = \mathrm{mgu}(\beta\sigma_1, \gamma[l]_p\sigma_1)$$

with $\mathrm{dom}(\sigma_2) \cap V = \emptyset$. The instance $(\beta \parallel \Lambda \to \Pi)\sigma$ of the result clause prevents the production of $l\sigma{\approx}r\sigma$ by the above clause, which is a contradiction.

Lemma 2. *Let N be saturated, A_N not covering and $\mathcal{I}_N \not\models N$. If $(\alpha \parallel C)\sigma$ is a minimal ground instance of a clause in N such that $\mathcal{I}_N \not\models (\alpha \parallel C)\sigma$, then $\alpha\sigma = \alpha_N$.*

Proof. Let $C = \Gamma \to \Delta$. By definition of entailment, $\mathcal{I}_N \models \alpha_N{\approx}\alpha\sigma$, which is equivalent to $\alpha_N \leftrightarrow_R^* \alpha\sigma$. We have already seen in lemma 1 that α_N is irreducible. Because of the confluence of R, either $\alpha_N = \alpha\sigma$ or $\alpha\sigma$ must be reducible.

Assume the latter, i.e. that $\alpha\sigma|_p = l\sigma'$ for a position p and a rule $l\sigma' \to r\sigma' \in R$ that is produced by a clause $\beta \parallel \Lambda \to \Pi, l \approx r \in N$. If p is not a non-variable position in α, then the rule actually reduces σ, which contradicts the minimality of $(\alpha \parallel C)\sigma$. Otherwise there is a constraint superposition inference

$$\frac{\beta \parallel \Lambda \to \Pi, l \approx r \quad \alpha \parallel \Gamma \to \Delta}{(\alpha[r]_p \parallel \beta \approx \alpha[r]_p, \Lambda, \Gamma \to \Pi, \Delta)\tau} \; \tau = \mathrm{mgu}(\alpha|_p, l) \; .$$

The ground instance $\delta \parallel D := (\alpha[r]_p \parallel \beta \approx \alpha[r]_p, \Lambda, \Gamma \to \Pi, \Delta)\sigma\sigma'$ of the conclusion is not true in $\mathcal{I}_N$. On the other hand, as the inference is redundant, so is the clause $\delta \parallel D$, i.e. it follows from ground instances of clauses of N all of which are smaller than $\delta \parallel D$. Since moreover $\delta \parallel D \prec (\alpha \parallel C)\sigma$ (remember that the ordering prioritizes constraints), all these ground instances hold in $\mathcal{I}_N$, hence $\mathcal{I}_N \models \delta \parallel D$ by minimality of $(\alpha \parallel C)\sigma$. This is a contradiction to $\mathcal{I}_N \not\models \delta \parallel D$.

Proposition 1. *Let N be a saturated set of constrained clauses such that A_N is not covering. Then $\mathcal{I}_N \models N$.*

Proof. Assume, contrary to the proposition, that N is not modeled by $\mathcal{I}_N$. Then there is a minimal ground instance $(\alpha \parallel C)\sigma$ of a clause $\alpha \parallel C \in N$ that is not modeled by $\mathcal{I}_N$. We will refute this minimality. We proceed by a case analysis of the position of the maximal literal in $C\sigma$.

- $C\sigma$ does not contain any maximal literal at all, i.e. $C = \Box$. Since $\alpha\sigma = \alpha_N$ by lemma 2, but $\mathcal{I}_N \not\models \alpha\sigma \approx \alpha_N$ by definition of α_N, this cannot happen.
- $C = \Gamma \to \Delta, s \approx t$ and $s\sigma \approx t\sigma$ is maximal in $C\sigma$ with $s\sigma = t\sigma$. This cannot happen because then $C\sigma$ would be a tautology.
- $C = \Gamma, s \approx t \to \Delta$ and $s\sigma \approx t\sigma$ is maximal in $C\sigma$ with $s\sigma \succ t\sigma$. Since $\mathcal{I}_N \not\models C\sigma$, we know that $s\sigma \approx t\sigma \in \mathcal{I}_N$, and because R only rewrites larger to smaller terms $s\sigma$ must be reducible by a rule $l\sigma' \to r\sigma' \in R$ produced by a clause $\beta \parallel \Lambda \to \Pi, l \approx r \in N$. So $s\sigma|_p = l\sigma'$ for some position p in $s\sigma$.

  Case 1: p is a non-variable position in s. Since $\beta\sigma' = \alpha_N = \alpha\sigma$ and $s\sigma|_p = l\sigma'$, there is an inference by superposition (left) as follows:

  $$\frac{\beta \parallel \Lambda \to \Pi, l \approx r \quad \alpha \parallel \Gamma, s \approx t \to \Delta}{(\alpha \parallel \Lambda, \Gamma, s[r]_p \approx t \to \Pi, \Delta)\sigma_1\sigma_2} \; \sigma_1 := \mathrm{mgu}(s|_p, l), \; \sigma_2 = \mathrm{mgu}(\beta\sigma_1, \alpha\sigma_1)$$

  The ground instance $\delta \parallel D := (\alpha \parallel \Lambda, \Gamma, s[r]_p \approx t \to \Pi, \Delta)\sigma$ of the result clause is not modeled by $\mathcal{I}_N$.

  On the other hand, as the inference is redundant, so is the clause $\delta \parallel D$, i.e. it follows from ground instances of clauses of N all of which are smaller than $\delta \parallel D$. Since moreover $\delta \parallel D \prec (\alpha \parallel C)\sigma$, all these ground instances hold in $\mathcal{I}_N$, whence $\mathcal{I}_N \models \delta \parallel D$. A contradiction.

  Case 2: $p = p'p''$, where $s|_{p'} = x$ is a variable. Then $(x\sigma)|_{p''} = l\sigma$. If τ is the substitution that coincides with σ except that $x\tau = x\sigma[r\sigma]_{p''}$, then $\mathcal{I}_N \not\models C\tau$ and $C\tau$ contradicts the minimality of $C\sigma$.
- The other cases are handled analogously.

For the construction of $\mathcal{I}_N$, we chose α_N to be minimal. For non-minimal α_N, the proposition does not hold: If, e.g., $N = \{u{\approx}a \,\|\, \rightarrow a{\approx}b,\ u{\approx}b \,\|\, a{\approx}b \rightarrow\}$ and $a \succ b$, then N is saturated, but N implies $u{\approx}a \,\|\, \square$. So the model constructed with $\alpha'_N = \{u{\approx}a\}$ is not a model of N. On the other hand, A_N is not covering whenever N has any Herbrand model over $\mathcal{F}$:

Proposition 2. *Let N be a set of clauses such that A_N is covering. Then N does not have any Herbrand model over $\mathcal{F}$.*

Proof. Let $\mathcal{M}$ be a Herbrand model of N over $\mathcal{F}$. Then

$$\begin{aligned} & \mathcal{M} \models \{(\alpha \,\|\, \square) \,|\, (\alpha \,\|\, \square) \in N\} \\ \iff\ & \exists\sigma.\forall(\alpha \,\|\, \square) \in N.\forall\tau.(\mathcal{M} \models \alpha\sigma\tau \implies \mathcal{M} \models \square) \\ \iff\ & \exists\sigma.\forall(\alpha \,\|\, \square) \in N.\mathcal{M} \models \neg\alpha\sigma \\ \implies\ & \exists\sigma.\forall(\alpha \,\|\, \square) \in N.\mathcal{T}(\mathcal{F}) \models \neg\alpha\sigma\ , \end{aligned}$$

where $\sigma\colon V \to \mathcal{T}(\mathcal{F})$ and $\tau\colon X \setminus V \to \mathcal{T}(\mathcal{F})$. But then the constraint $\bigwedge_{v \in V} v{\approx}v\sigma$ is not an instance of the constraint of any clause of the form $\alpha \,\|\, \square$, so A_N is not covering.

A saturated clause set N for which A_N is covering may nevertheless have both non-Herbrand models and Herbrand models over an extended signature: If $\mathcal{F} = \{a\}$ and $N = \{u{\approx}a \,\|\, \square\}$, then A_N is covering, but any standard first-order interpretation with a universe of at least two elements is a model of N.

Propositions 1 and 2 constitute the following theorem:

Theorem 1. *Let N be a saturated set of constrained clauses. Then N has a Herbrand model over $\mathcal{F}$ iff A_N is not covering.*

Moreover, the classical notions of theorem proving derivations and fairness from [1] carry over to our setting. A (finite or countably infinite) *theorem proving derivation* is a sequence $N_0, N_1, \ldots$ of constrained clause sets, written $N_0 \vdash N_1 \vdash \ldots$, such that either

- (Deduction) $N_{i+1} = N_i \cup \{C\}$ and $N_i \models_{\mathcal{F}} N_{i+1}$, or
- (Deletion) $N_{i+1} = N_i \setminus \{C\}$ and C is redundant with respect to N_i.

We may use our existential superposition calculus for deductions in a theorem proving derivation:

Proposition 3. *Let $\alpha \,\|\, C$ be the conclusion of an inference with premises in N. Then $N \models_{\mathcal{F}} N \cup \{\alpha \,\|\, C\}$. More precisely: if $\tau\colon V \to \mathcal{T}(\mathcal{F})$ is a substitution then $N\tau \models N\tau \cup \{\alpha\tau \rightarrow C\tau\}$.*

Proof. Let $\alpha \,\|\, C$ be the conclusion of an inference from $\alpha_1 \,\|\, C_1, \alpha_2 \,\|\, C_2 \in N$. Then $\alpha\tau \rightarrow C\tau$ is (modulo (unconstrained) equality resolution) an instance of the conclusion of a standard paramodulation inference from $\alpha_1\tau \rightarrow C_1\tau$ and $\alpha_2\tau \rightarrow C_2\tau$. Because of the soundness of the paramodulation rules, we have $N\tau \models N\tau \cup \{\alpha\tau \rightarrow C\tau\}$.

A derivation using our calculus is *fair* if every inference with premises in $\bigcup_j \bigcap_{k \geq j} N_k$ is redundant with respect to $\bigcup_j N_j$. As usual, fairness can be ensured by systematically adding conclusions of non-redundant inferences, making this inference redundant.

As it relies on redundancy and fairness rather than a concrete inference system, the proof of the next theorem is exactly as in the unconstrained case:

Theorem 2. *Let $N_0, N_1, \ldots$ be a fair theorem proving derivation. $\bigcup_j \bigcap_{k \geq j} N_k$ is saturated, and this set has a Herbrand model over $\mathcal{F}$ if and only if N_0 does.*

4 Finite Domain and Minimal Model Validity of Constrained Clauses

Given a clause set N, we are often not only interested in the (un)satisfiability of N (with or without respect to a fixed domain), but also in properties of Herbrand models of a satisfiable clause set N over $\mathcal{F}$, especially of the model $\mathcal{I}_N$.

These are not always disjoint problems: For some N, whole classes of first-order properties and properties of $\mathcal{I}_N$ coincide, so that we can explore the latter with first-order techniques:

Proposition 4. *If N is a saturated set of unconstrained Horn clauses and Γ is a conjunction of positive literals with existential closure $\exists \boldsymbol{x}.\Gamma$, then*

$$N \models_{Ind} \exists \boldsymbol{x}.\Gamma \iff N \models \exists \boldsymbol{x}.\Gamma$$

Proof. $N \models \exists \boldsymbol{x}.\Gamma$ holds iff the set $N \cup \{\forall \boldsymbol{x}.\neg\Gamma\}$ is unsatisfiable. N is Horn, so during saturation of $N \cup \{\neg\Gamma\}$ using a set-of-support strategy (which is complete as N is saturated), only purely negative, hence non-productive, clauses can appear. So $N \cup \{\forall \boldsymbol{x}.\neg\Gamma\}$ is unsatisfiable iff $N \not\models_{Ind} \forall \boldsymbol{x}.\neg\Gamma$, which is in turn equivalent to $N \models_{Ind} \exists \boldsymbol{x}.\Gamma$.

If N and Γ additionally belong to the Horn fragment of a first-order logic (clause) class decidable by (unconstrained) superposition, such as the monadic class with equality [2] or the guarded fragment with equality [7], it is thus decidable whether $N \models_{Ind} \exists \boldsymbol{x}.\Gamma$.

Our goal in this section is to extend these results further. We will first show how to use our superposition calculus to prove or refute the validity of a set H of constrained clauses with respect to a Herbrand model $\mathcal{M}$ over $\mathcal{F}$ of a saturated clause set N, i.e. to decide whether or not $\mathcal{M} \models H$ (Theorem 3). Moreover, we will demonstrate classes of clause sets N and properties H for which $N \models_{\mathcal{F}} H$ and $N \models_{Ind} H$ coincide (Proposition 5). Finally, we will look at ways to improve the termination of our approach for properties of $\mathcal{I}_N$ (Theorem 4).

Since existential variables of N and H can be renamed apart and then do not interact in our inference system, we can and will assume that N consists only of unconstrained clauses.

As in the unconstrained context, a set-of-support strategy is complete for our calculus if the support set is saturated. We write $H \vdash_N^* H'$ if $N \cup H \vdash^* N \cup H'$ using N as set of support.

Theorem 3. *Let N be saturated and let $H \vdash_N^* H'$ such that $N \cup H'$ is saturated. Then $A_{H'}$ is covering iff $\mathcal{M} \not\models H$ for each Herbrand model $\mathcal{M}$ of N over $\mathcal{F}$.*

Proof. This is a direct consequence of theorem 2 in the context of a set-of-support strategy for N.

Example 1. We consider the partial definition of the usual ordering on the naturals given by $N = \{\rightarrow G(s(0), 0),\ G(x, y) \rightarrow G(s(x), s(y))\}$, as shown in the introduction. We want to check whether or not $N \models_{\mathcal{F}} \forall x.G(s(x), x)$. The first steps of a possible derivation are as follows:

$$
\begin{array}{lll}
\text{clauses in } N\text{:} & 1: & \| \quad \rightarrow G(s(0), 0) \\
 & 2: & \| \quad G(x, y) \rightarrow G(s(x), s(y)) \\
\text{negated conjecture:} & 3: u \approx x & \| \ G(s(x), x) \rightarrow \\
\text{superposition}(1,3) = & 4: u \approx 0 & \| \quad \square \\
\text{superposition}(2,3) = & 5: u \approx s(y) & \| \ G(s(y), y) \rightarrow \\
\text{superposition}(1,5) = & 6: u \approx s(0) & \| \quad \square \\
\text{superposition}(2,5) = & 7: u \approx s(s(z)) & \| \ G(s(z), z) \rightarrow
\end{array}
$$

If the sequel, we repeatedly superpose clauses 1 and 2 into (descendents of) clause 5 and successively derive all clauses of the forms $u \approx s^n(0) \,\|\, \square$ and $u \approx s^n(x) \,\|\, G(s(x), x) \rightarrow$, where, e.g., $s^n(0)$ denotes the n-fold application $s(\ldots s(s(0))\ldots)$ of s to 0. Since the constraints of the derived $\square$ clauses are covering, we know that $N \models_{\mathcal{F}} \forall x.G(s(x), x)$.

Given our superposition calculus for fixed domains, we can show that a result similar to proposition 4 holds for positive universal clauses.

Proposition 5. *If N is a saturated set of (unconstrained) Horn clauses and Γ is a conjunction of positive literals with universal closure $\forall \boldsymbol{v}.\Gamma$, then*

$$N \models_{Ind} \forall \boldsymbol{v}.\Gamma \iff N \models_{\mathcal{F}} \forall \boldsymbol{v}.\Gamma$$

Proof. $N \models_{\mathcal{F}} \forall \boldsymbol{v}.\Gamma$ holds iff $N \cup \{\exists \boldsymbol{v}.\neg\Gamma\}$ does not have a Herbrand model over $\mathcal{F}$. If $N \cup \{\exists \boldsymbol{v}.\neg\Gamma\}$ does not have a Herbrand model over $\mathcal{F}$, then obviously $N \not\models_{Ind} \exists \boldsymbol{v}.\neg\Gamma$. Otherwise, the minimal models of N and $N \cup \{\exists \boldsymbol{v}.\neg\Gamma\}$ are identical, since during the saturation of $N \cup \{ \,\|\, \Gamma \rightarrow\}$ with our algorithm using a set-of-support strategy (which again is complete as N is saturated), only purely negative, hence non-productive, clauses can appear. This in turn just means that $N \models_{Ind} \exists \boldsymbol{v}.\neg\Gamma$.

Using proposition 5, we can decide properties of minimal models for which neither the approach of Ganzinger and Stuber [8] nor the one of Comon and Nieuwenhuis [6] works.

Example 2. Consider yet another partial definition of the usual ordering on the naturals given by the saturated set $N = \{\rightarrow G(s(x), 0),\ G(x, s(y)) \rightarrow G(x, 0)\}$ over the signature $\mathcal{F} = \{0, s\}$. We want to prove both $N \not\models_{\mathcal{F}} \forall x, y.G(x, y)$ and

$N \not\models_{Ind} \forall x, y.G(x, y)$. We start with the clause $u{\approx}x, v{\approx}y \parallel G(x, y) \rightarrow$ and do the following one step derivation:

$$\begin{array}{lllll}
\text{clauses in } N: & 1: & \parallel & \rightarrow G(s(x), 0) \\
& 2: & \parallel G(x, s(y)) & \rightarrow G(x, 0) \\
\text{negated conjecture:} & 3: u{\approx}x, v{\approx}y & \parallel \quad G(x, y) & \rightarrow \\
\text{superposition}(1,3) = & 4: u{\approx}s(x), v{\approx}0 & \parallel & \square
\end{array}$$

All further inferences are redundant, thus the counter examples to the query are exactly those for which no $\square$ clause was derived, i.e. instantiations of u and v which are not an instance of $\{u \mapsto s(x), v \mapsto 0\}$. Hence these counter examples take on exactly the form $\{u \mapsto 0, v \mapsto t_2\}$ or $\{u \mapsto t_1, v \mapsto s(t_2)\}$ for any $t_1, t_2 \in \mathcal{T}(\mathcal{F})$. Thus we know that $N \not\models_{\mathcal{F}} \forall x, y.G(x, y)$, but since the query is positive, we also know that $N \not\models_{Ind} \forall x, y.G(x, y)$.

In comparison, the base approach by Ganzinger and Stuber starts a derivation with the clause $\rightarrow G(x, y)$, derives in one step the potentially productive clause $\rightarrow G(x, 0)$ and finishes with the answer "don't know". The extended approach that uses the predicate gnd defined by $\{\rightarrow \text{gnd}(0),\ \text{gnd}(x) \rightarrow \text{gnd}(s(x))\}$ starts the derivation with the clause $\text{gnd}(x), \text{gnd}(y) \rightarrow G(x, y)$, where at least one of $\text{gnd}(x)$ and $\text{gnd}(y)$ is selected, and diverges.

The approach by Comon and Nieuwenhuis fails as well. Before starting the actual derivation, a so-called I-axiomatization of the negation of G has to be computed. This involves a quantifier elimination procedure as in [5], that fails since G is not universally reductive (because, e.g., the head of one clause does not contain all variables of the clause): G is defined in the minimal model $\mathcal{I}_N$ by $G(x, y) \iff (y = 0 \wedge \exists u.x = s(u)) \vee (y = 0 \wedge \exists v.G(x, s(v)))$, so its negation is defined by $\neg G(x, y) \iff (y \neq 0 \vee \forall u.x \neq s(u)) \wedge (y \neq 0 \vee \forall v.\neg G(x, s(v)))$. Quantifier elimination simplifies this to $\neg G(x, y) \iff (y \neq 0 \vee x = 0) \wedge (y \neq 0 \vee \forall v.\neg G(x, s(v)))$ but cannot get rid of the remaining universal quantor.

As we have seen in example 1, a proof of $\models_{\mathcal{F}}$ validity often requires the computation of infinitely many empty clauses. This is not surprising, because we have to show that an existentially quantified clause cannot be satisfied by a term-generated infinite domain. In the context of a concrete model $\mathcal{M}$ of N, we can make use of additional structure provided by this model. To do so, we introduce a further inference rule that often drastically decreases the number of possibly non-implied query instances to be considered and allows more derivations to terminate. This rule is not sound for $\models_{\mathcal{F}}$ but always glued to the currently considered model $\mathcal{M}$. While the results in this section hold for all (sets of) Herbrand models of N, they are most likely to be used for the minimal model $\mathcal{I}_N$ of a saturated clause set and hence presented in the context of $\models_{Ind}$ only.

Over any domain where the induction theorem holds, i.e. a domain on which a well-founded ordering can be defined, we can exploit this structure to concentrate on finding minimal solutions. We do this by adding a form of induction hypothesis to the clause set. If e.g. P is a unary predicate over the natural numbers and n is the minimal number such that $P(n)$ holds, then we know that at

the same time $P(n-1), P(n-2), \ldots$ do not hold. This idea will now be cast into an inference rule that can be used during a theorem proving derivation.

Let $< \subseteq (\mathcal{T}(\mathcal{F})/_{\overset{*}{\leftrightarrow}_R})^2$ be a well-founded partial ordering on the elements of $\mathcal{I}_N$. If s, t are non-ground terms with equivalence classes $[s]$ and $[t]$, then we define $[s] < [t]$ iff $[s\sigma] < [t\sigma]$ for all grounding substitutions $\sigma : X \to \mathcal{T}(\mathcal{F})$. The definition lifts pointwise to substitutions $[\sigma], [\rho]\colon X' \to \mathcal{T}(\mathcal{F})/_{\overset{*}{\leftrightarrow}_R}$, where we say that $[\rho] < [\sigma]$ iff $[x\rho] < [x\sigma]$ for all $x \in X'$.

Lemma 3. *Let $\alpha = v_1 \approx x_1, \ldots, v_k \approx x_k$ be a constraint containing only variables, $X_\alpha = \{x_1, \ldots, x_k\}$, and let $H = \{\alpha \parallel C_1, \ldots, \alpha \parallel C_n\}$ be a set of clauses where only variables of $V \cup X_\alpha$ occur. Furthermore, let $\sigma, \rho\colon X_\alpha \to \mathcal{T}(\mathcal{F}, X)$ be substitutions with $[\rho] < [\sigma]$. If $N \models_{Ind} H$, then also $N \models_{Ind} \alpha\sigma \to (\neg C_1\rho \vee \ldots \vee \neg C_n\rho)$.*

Proof. Let $[\sigma_0]\colon V \to \mathcal{T}(\mathcal{F})/_{\overset{*}{\leftrightarrow}_R}$ be minimal with respect to $<$ such that $N \models_{Ind} \{(\alpha \parallel C_1)\sigma_0, \ldots, (\alpha \parallel C_n)\sigma_0\}$. Furthermore, let $X_{\alpha\sigma}$ be the set of non-existential variables in $\alpha\sigma$ and $\tau\colon X_{\alpha\sigma} \to \mathcal{T}(\mathcal{F})$ such that $N \models_{Ind} \alpha\sigma\sigma_0\tau$. We have to show that $N \models_{Ind} \neg C_1\rho\tau \vee \ldots \vee \neg C_n\rho\tau$.

The restriction of a substitution to the set V of existential variables is denoted by $(\cdot)|_V$. In the fifth line below, we use that α is a conjunction of equations $v \approx v\sigma_\alpha$ and that τ' and $\sigma_\alpha\rho\tau|_V$ affect different sides of each such equation.

$$
\begin{aligned}
& [\rho] < [\sigma] && \overset{X_\rho \subseteq X}{\Longleftrightarrow} [\sigma_\alpha\rho] < [\sigma_\alpha\sigma] \\
\overset{\text{def.}<}{\Longrightarrow}\ & [(\sigma_\alpha\rho\tau)|_V] < [(\sigma_\alpha\sigma\tau)|_V] = [\sigma_0] \\
\overset{[\sigma_0]\ \text{minimal}}{\Longrightarrow}\ & N \not\models_{Ind} \{(\alpha \parallel C_1)(\sigma_\alpha\rho\tau)|_V, \ldots, (\alpha \parallel C_n)(\sigma_\alpha\rho\tau)|_V\} \\
\overset{\text{def.}\models}{\Longrightarrow}\ & \exists\tau'.\, N \models_{Ind} \alpha(\sigma_\alpha\rho\tau)|_V\tau' \text{ and } N \not\models_{Ind} C_1\tau' \wedge \ldots \wedge C_n\tau' \\
\Longrightarrow\ & \exists\tau'.\, \forall v \in V.\, N \models_{Ind} v\sigma_\alpha\rho\tau \approx v\sigma_\alpha\tau' \text{ and } N \not\models_{Ind} C_1\tau' \wedge \ldots \wedge C_n\tau' \\
\Longrightarrow\ & \exists\tau'.\, \forall x \in X_\alpha.\, N \models_{Ind} x\rho\tau \approx x\tau' \text{ and } N \not\models_{Ind} C_1\tau' \wedge \ldots \wedge C_n\tau' \\
\overset{C_i\tau'\ \text{ground}}{\Longrightarrow}\ & \exists\tau'.\, \forall x \in X_\alpha.\, N \models_{Ind} x\rho\tau \approx x\tau' \text{ and } N \models_{Ind} \neg C_1\tau' \vee \ldots \vee \neg C_n\tau' \\
\overset{\text{var}(C_i) \subseteq X_\alpha}{\Longrightarrow}\ & N \models_{Ind} \neg C_1\rho\tau \vee \ldots \vee \neg C_n\rho\tau
\end{aligned}
$$

for $\tau'\colon X \setminus V \to \mathcal{T}(\mathcal{F})$.

Since the preserved solution $[\sigma_0]$ is independent of the choices of σ and ρ, any clauses derived by this lemma will have a common solution with $\alpha \parallel C$.

Example 3. Let $\mathcal{F} = \{0, s\}$, $N = \{P(s(s(x)))\}$ and $H = \{u \approx x \parallel P(x)\}$. The formulas derivable by the lemma are of one of the forms $u \approx s^n(0) \to \neg P(s^{n+m}(0))$, $u \approx s^n(0) \to \neg P(s^{n+m}(x))$ or $u \approx s^n(x) \to \neg P(s^{n+m}(x))$ for natural numbers n, m with $m > 0$. All these formulas and the initial clause $u \approx x \parallel P(x)$ have the common solution $\{u \mapsto s(s(0))\}$ in $\mathcal{I}_N$.

The formula $\alpha\sigma \to (\neg C_1\rho \vee \ldots \vee \neg C_n\rho)$ can usually not be written as a single equivalent constrained clause if some C_i contains more than one literal. However,

if $D_1 \wedge \ldots \wedge D_m$ is a conjunctive normal form of $\neg C_1 \vee \ldots \vee \neg C_n$, then each D_j is a disjunction of literals and so $\alpha\sigma \parallel D_j\rho$ is a constrained clause.

We can thus, to decide the validity of $\{\alpha \parallel C_1, \ldots, \alpha \parallel C_n\}$ in $\mathcal{I}_N$, use information taken from the lemma in the theorem proving derivation:

Theorem 4. *Let N be a saturated set of clauses, let $\alpha = v_1 \approx x_1, \ldots, v_k \approx x_k$ be a constraint containing only variables, $X_\alpha = \{x_1, \ldots, x_k\}$, and let $H = \{\alpha \parallel C_1, \ldots, \alpha \parallel C_n\}$ be a set of clauses where only variables of $V \cup X_\alpha$ occur. Moreover, let $D_1 \wedge \ldots \wedge D_m$ be a conjunctive normal form of $\neg C_1 \vee \ldots \vee \neg C_n$. Consider the inference rule*

$$\frac{}{\alpha\sigma \parallel D_j\rho} \text{ if } \sigma, \rho\colon X_\alpha \to \mathcal{T}(\mathcal{F}, X) \text{ and } [\rho] < [\sigma] \text{ and } 1 \leq j \leq m$$

which is specialized for the one fixed clause set H, and the theorem proving system combining this rule and $\vdash_N$. If H' is derived from H using this combined inference system, then $N \models_{Ind} H \iff N \models_{Ind} H'$.

Proof. This follows directly from proposition 3 and lemma 3.

Example 4. Consider the following theory of the addition on the naturals: $N = \{\to 0 + y \approx y,\ \to s(x) + y \approx s(x + y)\}$. The proof of $N \models_{Ind} \forall x.x + 0 \approx x$ with the induction inference rule terminates quickly:

clauses in N:	$1:$	$\parallel$		$\to 0 + y \approx y$
	$2:$	$\parallel$		$\to s(x) + y \approx s(x + y)$
negated conjecture:	$3: u \approx x$	$\parallel$	$x + 0 \approx x \to$	
superposition(1,3)	$= 4: u \approx 0$	$\parallel$	$0 \approx 0 \to$	
equality res.(4)	$= 5: u \approx 0$	$\parallel$	$\square$	
superposition(2,3)	$= 6: u \approx s(y)$	$\parallel$	$s(y + 0) \approx s(y) \to$	
induction rule(3)	$= 7: u \approx s(z)$	$\parallel$		$\to z + 0 \approx z$
superposition(7,6)	$= 8: u \approx s(z)$	$\parallel$	$s(z) \approx s(z) \to$	
equality res.(8)	$= 9: u \approx s(z)$	$\parallel$	$\square$	

At this point, the clauses $u \approx 0 \parallel \square$ and $u \approx s(z) \parallel \square$ have been derived. Their constraints cover all of $\mathcal{T}(\mathcal{F})$, which means that $N \models_{Ind} u \approx x \parallel x + 0 \approx x \to$, i.e. $N \models_{Ind} \forall x.x + 0 \approx x$.

Example 5. Given the theory $N = \{\to E(0),\ E(x) \to E(s(s(x)))\}$ of the natural numbers together with a predicate describing the even numbers, we check whether $N \models_{Ind} \forall x.E(x)$. A possible derivation runs as follows:

clauses in N:	$1:$	$\parallel$		$\to E(0)$
	$2:$	$\parallel$	$E(x) \to$	$E(s(s(x)))$
negated conjecture:	$3: u \approx x$	$\parallel$	$E(x) \to$	
superposition(1,3)	$= 4: u \approx 0$	$\parallel$	$\square$	
superposition(2,3)	$= 5: u \approx s(s(y))$	$\parallel$	$E(y) \to$	
induction rule(3)	$= 6: u \approx s(s(z))$	$\parallel$		$\to E(z)$
superposition(6,5)	$= 7: u \approx s(s(z))$	$\parallel$	$\square$	

This set is saturated. The derived contradictions are $u \approx 0 \parallel \square$ and $u \approx s(s(z)) \parallel \square$. Their constraints are not covering, and in fact $N \models_{Ind} E(s(0)) \to$.

5 Conclusion

We have presented a sound and complete superposition calculus for a fixed domain semantics. Compared to other approaches in model building over fixed domains, our approach is applicable to a larger class of clause sets. While most works in the tradition of Caferra and Zabel [4] consider only very restricted forms of equality literals and even more recent publications by Peltier [14] pose strong restrictions on the clause sets (e.g. that they have a unique Herbrand model), we do not have such restrictions.

Moreover, we presented a way to prove the validity of minimal model properties by using a specific induction rule. We even showed that standard first-order and fixed domain superposition based reasoning, respectively, delivers minimal model results for some cases. The most general methods based on saturation so far are those by Ganzinger and Stuber [8] and Comon and Nieuwenhuis [6]. Both approaches work only on sets of purely universal and universally reductive (Horn) clauses. We gave an example of a purely universal problem that our algorithm can solve while neither of the above approaches works. Additionally, we showed how we can also prove the validity of $\forall^*\exists^*$-quantified formulas.

Another intensely studied approach is via test sets [11,3]. Test sets rely on the existence of a set of constructor symbols that are either free or specified by unconditional equations only. Again, such properties are not needed for the applicability of our calculus. Example 1 is not solvable via test sets, whereas Example 2 is.

In analogy to the work of Bachmair and Ganzinger [1], it is also possible to extend the new superposition calculus by negative literal selection, with the restriction that no constraint literals may be selected. still hold in this setting. Theorem 1 still holds in this setting. For universally reductive clause sets N, it is also possible to make the inductive theorem proving algorithm (with selection) refutationally complete, following the approach of Ganzinger and Stuber [8].

In summary, our approach does not need many of the prerequisites required by previous approaches, like solely universally reductive clauses in N, solely Horn clauses, solely purely universal clauses, solely non-equational clauses, the existence of an "A" set fixing the standard first-order interpretation to the minimal model, or the existence of explicit constructor symbols. Its success is build on a superposition based saturation concept.

Our hope is that the success of the superposition based saturation approach on identifying decidable classes with respect to the classical first-order semantics can be extended to some new classes for the fixed domain or minimal model semantics. In case we can finitely saturate a clause set, the ordering $<$ on $\mathcal{I}_N$ elements may become effective and hence the induction rule of theorem 4 can then be effectively used to finitely saturate clause sets that otherwise have an infinite saturation. Decidability results for the fixed domain semantics are hard to obtain for infinite Herbrand domains but the problem can now be attacked using the sound and complete calculus presented in this paper. They will require in addition the extension of the redundancy notion suggested in section 3 possibly

using more expressive languages of existential constraints. Here, concepts and results from tree automata could play a role.

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. Bachmair, L., Ganzinger, H., Waldmann, U.: Superposition with simplification as a decision procedure for the monadic class with equality. In: Mundici, D., Gottlob, G., Leitsch, A. (eds.) KGC 1993. LNCS, vol. 713, pp. 83–96. Springer, Heidelberg (1993)
3. Bouhoula, A.: Automated theorem proving by test set induction. J. Symb. Comp. 23(1), 47–77 (1997)
4. Caferra, R., Zabel, N.: A method for simultanous search for refutations and models by equational constraint solving. J. Symb. Comp. 13(6), 613–642 (1992)
5. Comon, H., Lescanne, P.: Equational problems and disunification. Journal of Symbolic Computation 7(3-4), 371–425 (1989)
6. Comon, H., Nieuwenhuis, R.: Induction = I-axiomatization + first-order consistency. Information and Computation 159(1/2), 151–186 (2000)
7. Ganzinger, H., Nivelle, H.D.: A superposition decision procedure for the guarded fragment with equality. In: Proc. of LICS 1999, pp. 295–305. IEEE, Los Alamitos (1999)
8. 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)
9. Jacquemard, F., Meyer, C., Weidenbach, C.: Unification in extensions of shallow equational theories. In: Nipkow, T. (ed.) RTA 1998. LNCS, vol. 1379, pp. 76–90. Springer, Heidelberg (1998)
10. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 557–571. Springer, Heidelberg (2006)
11. Kapur, D., Narendran, P., Zhang, H.: Automating inductionless induction using test sets. Journal of Symbolic Computation 11(1/2), 81–111 (1991)
12. Nieuwenhuis, R.: Basic paramodulation and decidable theories (extended abstract). In: Proc. of LICS 1996, pp. 473–482. IEEE Computer Society Press, Los Alamitos (1996)
13. Nieuwenhuis, R., Rubio, A.: Paramodulation-based theorem proving. In: Handbook of Automated Reasoning, ch. 7, vol. I, pp. 371–443. Elsevier, Amsterdam (2001)
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. 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)
16. Weidenbach, C.: Combining superposition, sorts and splitting. In: Handbook of Automated Reasoning, ch. 27, vol. 2, pp. 1965–2012. Elsevier, Amsterdam (2001)

Non-finite Axiomatizability and Undecidability of Interval Temporal Logics with C, D, and T

Ian Hodkinson[1], Angelo Montanari[2], and Guido Sciavicco[3]

[1] Department of Computing, Imperial College London,
South Kensington Campus London SW7 2AZ (UK)
imh@doc.ic.ac.uk
[2] Department of Mathematics and Computer Science,
University of Udine (Italy)
angelo.montanari@dimi.uniud.it
[3] Department of Information Engineering and Communications,
University of Murcia, Murcia (Spain)
guido@um.es

Abstract. Interval logics are an important area of computer science. Although attention has been mainly focused on unary operators, an early work by Venema (1991) introduced an expressively complete interval logic language called CDT, based on binary operators, which has many potential applications and a strong theoretical interest. Many very natural questions about CDT and its fragments, such as (non-)finite axiomatizability and (un-)decidability, are still open (as a matter of fact, only a few undecidability results, including the undecidability of CDT, are known). In this paper, we answer most of these questions, showing that almost all fragments of CDT, containing at least one binary operator, are neither finitely axiomatizable with standard rules nor decidable. A few cases remain open.

1 Introduction

Interval-based temporal logic represents an important area of computer science. The main species of propositional interval temporal logics studied so far include Mozskowski's Propositional Interval Logic (PITL) [22], Halpern and Shoham's Modal Logic of Allen's Relations (HS) [16], and Venema's CDT logic [25] (extended to branching-time frames with linear intervals in [14]). Important fragments of HS studied in more detail include, among others, the logic of *begins/ends* Allen's relations (BE) [19], the logics of temporal neighborhood [8,9,12] and the logics of subinterval structures [4,5].

(Un-)decidability. The logic PITL, which features the binary modality C (*chop*) and the modal constant π for point-intervals, has been shown to be undecidable in [22] when interpreted in the class of all finite linearly ordered sets and in classes based on $\mathbb{N}$ or $\mathbb{Z}$; in [19] the result has been extended to the dense case (as pointed out in [13], this implies undecidability in the class of all linearly ordered sets). Satisfiability for HS is also undecidable in many cases, as it has

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 308–322, 2008.
© Springer-Verlag Berlin Heidelberg 2008

been shown in [16]. HS can be thought of as a modal logic with a unary modal operator for any of Allen's relations between two intervals. Undecidability has been shown for various interesting classes of linearly ordered sets, and the proof hinges on a non-trivial reduction from the halting problem for a Turing machine. In some cases, such as Dedekind-complete linearly ordered sets with an infinite unbounded ascending sequence, validity is non recursively enumerable, which implies non-finite axiomatizability (no matters what kind of deduction rules are used). Undecidability of CDT, over the same classes of structures, immediately follows from that of HS. In [19], the fragment of HS with only two modalities, corresponding to Allen's relations *begins/ends* (BE), has been shown undecidable when interpreted over dense linear orderings. On the positive side, decidability of the fragments of HS with modalities corresponding to Allen's relations *begins/begun-by* and *ends/ended-by* has been obtained by means of a translation to Linear Temporal Logic [13]. Moreover, decidability of the satisfiability problem for the class of HS fragments featuring only two modalities, corresponding to Allen's relations *meets/met-by* ($A\overline{A}$, or PNL), has been shown in [6] (as a matter of fact, a number of natural extensions of it turn out to be undecidable ([7])). Finally, decidability of the logics of subinterval structures over dense ordering has been proved in [4,5].

(Non-)finite axiomatizability. Sound and complete axiomatic systems in interval temporal logics are scarce. PITL has been axiomatized both at the first-order and the propositional level in [3] for discrete/finite linearly ordered sets, but under the hypothesis of *locality*, that is, with the semantical assumption that each propositional letter is true over an interval if and only if it is true at the first point of the interval. The logics HS and CDT have been finitely axiomatized, respectively in [24] and [25], for various classes of linearly ordered sets, but using a Burgess/Gabbay-style non-orthodox 'irreflexivity' rule [11]. Finally, logics of neighborhood modalities have complete axiomatic systems with standard rules in various cases; some of them are infinite, as in the case in which point-intervals are allowed, but no modal constant for them is included in the language [12].

In this paper we focus on the logic CDT. Its language includes three binary operators C, D, and T, which correspond to the ternary interval relations occurring when an extra point is added in one of the three possible distinct positions with respect to the two endpoints of the current interval (*between, before,* and *after*), plus a modal constant π which holds at a given interval if and only if it is a point-interval. We prove the following results. First, we show that the undecidability of CDT can be extended to some sub-languages. For the language with C and π, undecidability was known (as recalled above). Here we show that D alone and T alone are undecidable as well, when interpreted in interval structures over any class of linearly ordered sets containing at least one linearly ordered set with an infinite sequence — ascending for T, descending for D — and that $D + \pi$ and $T + \pi$ are undecidable when interpreted in interval structures over finite strict linear orders. Second, the logic CDT is not finitely axiomatizable with standard rules over interval structures based on any class of linearly ordered sets containing $(\mathbb{Q}, <)$. The same holds for any sub-language containing at least one of

the modalities C, D, T. This result holds even if we drop the modal constant π. Notice that the undecidability and non-finite axiomatizability results are independent from each other. The question of whether there exists a finite axiomatic system for CDT with standard rules only was open since [24,25]. The results in the present paper correct a previous claim in [23]. Because of space restrictions, many proofs are only sketched in bare outline or omitted.

2 Basic Notions

Venema's interval-based temporal logic CDT [25] features a denumerable set $\mathcal{AP}$ of propositional letters, the classical operators $\wedge$ and $\neg$ (the remaining ones can be considered as abbreviations), the modal constant π, and three binary modal operators, namely, C, T, and D.[1] *Well-formed* formulas, denoted by $\phi, \psi, \ldots$, can be obtained by the following grammar:

$$\phi := p \mid \neg\phi \mid \phi \wedge \psi \mid \pi \mid \phi \sharp \psi \ (\sharp \in \mathit{Mod}),$$

where $\mathit{Mod} = \{C, D, T\}$ and $p \in \mathcal{AP}$. We will denote the sublanguage of CDT featuring only the non-empty set $S \subseteq \mathit{Mod}$ of the operators by L^{π}_{S}, and by L_S when π is not allowed (so CDT becomes L^{π}_{CDT} in our notation). In the next section, we will show that, for every S, L_S is not expressive enough to define π.

Definition 1 (abstract semantics). *Let $S \subseteq \mathit{Mod}$. An S-*frame *is a structure of the form $\mathcal{F} = (I, \Pi, R_\sharp : \sharp \in S)$, where I is a non-empty set, $\Pi \subseteq I$, and $R_\sharp \subseteq I^3$ for each $\sharp \in S$. An S-*model *is a structure of the form $\mathcal{M} = (\mathcal{F}, h)$, where $\mathcal{F} = (I, \Pi, R_\sharp : \sharp \in S)$ is an S-frame and $h : \mathcal{AP} \to \mathcal{P}(I)$. The* frame of $\mathcal{M}$ *is $\mathcal{F}$. We evaluate an L^{π}_{S}-formula ϕ in $\mathcal{M}$ at $i \in I$ as follows:*

- $\mathcal{M}, i \models p$ *iff* $i \in h(p)$, *for* $p \in \mathcal{AP}$;
- $\mathcal{M}, i \models \neg\phi$ *iff it is not the case that* $\mathcal{M}, i \models \phi$;
- $\mathcal{M}, i \models \phi \wedge \psi$ *iff* $\mathcal{M}, i \models \phi$ *and* $\mathcal{M}, i \models \psi$;
- $\mathcal{M}, i \models \pi$ *iff* $i \in \Pi$;
- $\mathcal{M}, i \models \phi \sharp \psi$ *iff there exist* $j, k \in I$ *with* $R_\sharp(i, j, k)$, $\mathcal{M}, j \models \phi$, *and* $\mathcal{M}, k \models \psi$.

Throughout, we often identify (notationally) a model or frame with its domain: so, for example, we write $i \in \mathcal{F}$ or $i \in \mathcal{M}$ to mean $i \in I$ as above. Let $S \subseteq \mathit{Mod}$, $\mathcal{M}$ be an S-model, and $\mathcal{K}$ be a class of S-frames. An L^{π}_{S}-formula ϕ is said to be satisfiable in $\mathcal{M}$ *if there is some $i \in \mathcal{M}$ such that $\mathcal{M}, i \models \phi$, and* satisfiable over $\mathcal{K}$ *if it is satisfiable in a model whose frame is in $\mathcal{K}$. Similarly, an L^{π}_{S}-formula ϕ is* valid in $\mathcal{M}$ *(resp.,* valid over $\mathcal{K}$*) if for every $i \in \mathcal{M}$, it is the case that $\mathcal{M}, i \models \phi$ (resp., if ϕ is valid in every model whose frame is in $\mathcal{K}$). These definitions naturally generalize to the case of sets of formulas.*

Notice that L_S-formulas are L^{π}_{S}-formulas, so we can evaluate them in S-models. We do not need to include Π in our frames in this case, but it can always be

[1] C stands for 'chop'. One may think of D as standing for 'Done' and T for 'To come'.

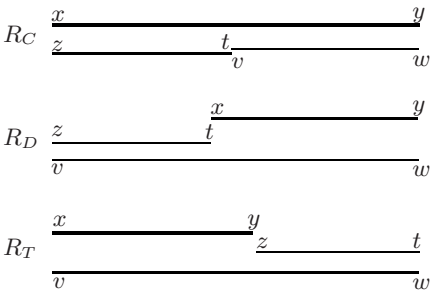


Fig. 1. The relations R_C, R_D, R_T; the thick line represents the current interval $[x, y]$

added without affecting L_S-semantics. So rather than to complicate our notation by introducing frames without Π, we give L_S-formulas semantics in L_S^π-models, which include Π.

While the above semantics can be considered *abstract*, we can formalize a *concrete* one based on pairs of points (*intervals*) over linear orders.

Definition 2 (concrete semantics). *Let $(T, <)$ be a strict linear order. We write $Int(T, <)$ for the set $\{[t, u] : t, u \in T,\ t \leq u\}$. As usual, $[t, u] = \{v \in T : t \leq v \leq u\}$, and $t \leq u$ abbreviates $t < u \vee t = u$. Let $\Pi = \{[t, t] : t \in T\}$. For any $[x, y], [z, t], [v, w] \in Int(T, <)$, we define*

- *$R_C([x, y], [z, t], [v, w])$ iff $x = z$, $t = v$, and $y = w$;*
- *$R_D([x, y], [z, t], [v, w])$ iff $x = t$, $z = v$, and $y = w$;*
- *$R_T([x, y], [z, t], [v, w])$ iff $x = v$, $y = z$, and $t = w$.*

For $S \subseteq Mod$, we write $\mathfrak{Int}_S(T, <)$ for the S-frame $(Int(T, <), \Pi, R_\sharp : \sharp \in S)$.

A graphical account of the three relations is given in Fig. 1. Intuitively, for an interval $[x, y]$, we have $\mathcal{M}, [x, y] \models \phi C \psi$ iff there is $z \in [x, y]$ with $\mathcal{M}, [x, z] \models \phi$ and $\mathcal{M}, [z, y] \models \psi$, and similarly for the other two modalities. We will use alternatively the abstract or the concrete semantics.

Definition 3. *Let $S \subseteq Mod$ and let $\mathcal{K}$ be a class of strict linear orders. We write $S(\mathcal{K})$ (resp., $S^\pi(\mathcal{K})$) for the set of all L_S-formulas (resp., L_S^π-formulas) valid over $\{\mathfrak{Int}_S(T, <) : (T, <) \in \mathcal{K}\}$. Thus, $S(\mathcal{K})$ and $S^\pi(\mathcal{K})$ are the* logics of (intervals over) *$\mathcal{K}$ in their respective languages.*

Beside the usual $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$, we introduce notation for some common classes:

- Lin = the class of all strict linear orders
- Fin = the class of all finite strict linear orders
- Dense = the class of all strict dense linear orders
- Dis = the class of all strict discrete linear orders
- Asc = the class of all strict linear orders that contain an infinite ascending sequence (i.e., they have $(\mathbb{N}, <)$ as a suborder)

– Des = the class of all strict linear orders that contain an infinite descending sequence

We usually write $S \subseteq \mathit{Mod}$ as a sequence consisting of its members: thus, we have the logics $CDT(\mathsf{Lin})$, $C^{\pi}(\mathsf{Dense})$, etc. When $\mathcal{K} = \{(T, <)\}$, we write $S(\mathcal{K})$ as $S(T, <)$ and $S^{\pi}(\mathcal{K})$ as $S^{\pi}(T, <)$.

3 Expressive Power of L_{CDT} and L^{π}_{CDT}

While in this paper we are mainly interested in binary modalities, unary modalities are most studied in interval logics. In general, one can introduce a unary interval modality for each of Allen's relations between two intervals. In [16] it was shown that only four of them are sufficient to express any of Allen's relations between two intervals, namely, $\langle B \rangle$ (*begins*), $\langle E \rangle$ (*ends*) and their inverses. In the language L^{π}_{CDT} these operators can be easily expressed, e.g.: $\langle B \rangle \phi = \phi C \neg \pi$, or $\langle \overline{E} \rangle \phi = \neg \pi D \phi$. In [24] the undecidability of $CDT(\mathsf{Asc})$ (and, by symmetry, of $CDT(\mathsf{Des})$) has been proved by exploiting the undecidability of HS over the same classes. Notice that the modal constant π plays an important role here. So, the question is whether π is definable in the language L_{CDT} or not. Here we show that it is not the case, proving that L_{CDT} is strictly less expressive than L^{π}_{CDT}, by applying a simple bisimulation argument to exhibit two models that can be distinguished by an L^{π}_{CDT}-formula, but not by any L_{CDT}-formula.

Theorem 1. *The modal constant π cannot be defined in L_{CDT}.*

Proof. Consider the pair of models $(\mathcal{M}_0, \mathcal{M}_1)$, based, respectively, on $\mathit{Int}(T_0, <)$ and $\mathit{Int}(T_1, <)$, where $T_0 = \{x_0, z_0, y_0\}$ with $x_0 < z_0 < y_0$, and $T_1 = \{x_1, y_1\}$ with $x_1 < y_1$. Clearly, $\mathit{Int}(T_0, <) = \{[x_0, x_0], [y_0, y_0], [z_0, z_0], [x_0, y_0], [x_0, z_0], [z_0, y_0]\}$, and $\mathit{Int}(T_1, <) = \{[x_1, x_1], [y_1, y_1], [x_1, y_1]\}$. The valuation functions h_0 and h_1 are void. It is easily seen that the intervals $[x_0, z_0]$ and $[x_1, x_1]$ are bisimilar. By the bisimulation-invariance of modal formulas (see, e.g., [1, Theorem 2.20]), they satisfy the same L_{CDT}-formulas in their respective models. But $\mathcal{M}_0, [x_0, z_0] \models \neg \pi$ and $\mathcal{M}_1, [x_1, x_1] \models \pi$. □

Because π is equivalent to $[B]\bot$ (and to $[E]\bot$), this means that L_{CDT} cannot define all of the modalities corresponding to Allen's relations, and the original argument for the undecidability cannot be applied anymore. As for fragments of L^{π}_{CDT}, the logic C^{π} has been shown to be undecidable for discrete linear orders in [22]; later, in [19], the undecidability was shown also for dense orders, and thus, since density and the universal operator can be defined in the language (see [13]), for all linear orders. The other fragments, with or without π, have received no attention so far.

4 Undecidability

In this section, we establish undecidability of any logic $D(\mathcal{K})$ where $\mathcal{K} \subseteq \mathsf{Lin}$ and $\mathcal{K} \cap \mathsf{Des} \neq \emptyset$, and $T(\mathcal{K})$, where $\mathcal{K} \subseteq \mathsf{Lin}$ and $\mathcal{K} \cap \mathsf{Asc} \neq \emptyset$. Recall that Asc (resp., Des)

is the class of all linearly ordered sets with an infinite ascending (resp., descending) sequence. We consider the case of T, the other one is symmetric. We show how to encode in L_T a variant of the $\mathbb{N} \times \mathbb{N}$-tiling problem called the *Octant Tiling Problem*. Given a set of tiles $\mathcal{T} = \{t_1, \ldots, t_k\}$, the octant tiling problem consists in establishing whether $\mathcal{T}$ can tile one octant of the Cartesian plane over $\mathbb{N}$; in our case, it will be the second octant $\mathcal{O} = \{(p, q) \mid p, q \in \mathbb{N}, p \leq q\}$. Each tile t_i has four colors, namely, $right(t_i)$, $left(t_i)$, $up(t_i)$, and $down(t_i)$, and neighboring tiles must have matching colors. Formally, we say that a set $\mathcal{T}$ can tile $\mathcal{O}$ if there exists a function $f : \mathcal{O} \mapsto \mathcal{T}$ such that $right(f(p, q)) = left(f(p + 1, q))$ and $up(f(p, q)) = down(f(p, q + 1))$, where $f(p, q)$ represents the tile to be placed in the position (p, q), whenever all relevant coordinates ($(p, q), (p + 1, q)$ etc.) lie in $\mathcal{O}$. The undecidability of the $\mathbb{N} \times \mathbb{N}$-tiling problem is well-known (e.g., see [2]); one can easily prove that the octant tiling problem is undecidable as well.

4.1 Language, Shortcuts, and *u*-Intervals

Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be an instance of the octant tiling problem. We will assume that $\mathcal{AP}$ contains at least the propositional letters u, t_1,..., and t_k.

As a preliminary step, we introduce a sort of *universal operator* (denoted by G) that looks only at the future of the current interval, and, then, we set our framework by forcing the existence of *unit-intervals* (or *u-intervals*) working like atomic elements. Such intervals will be denoted by the propositional letter u. We will impose that u-intervals are disposed in an unbounded unique (uninterrupted) sequence:

$$G\phi ::= \neg(\top T(\neg\phi T\top)), \tag{1}$$

$$uT\top \wedge G(u \rightarrow uT\neg u). \tag{2}$$

It is not difficult to see that $\mathcal{M}, [x, y] \models G\phi$ iff $\forall z, t(y \leq z \leq t \rightarrow \mathcal{M}, [z, t] \models \phi)$.

Lemma 1. *Suppose that $\mathcal{M}$ is a model with frame $\mathfrak{Int}_T(T, <)$ for any strict linear order $(T, <)$, that $[x, y] \in \mathcal{M}$, and that $\mathcal{M}, [x, y] \models (2)$. Then, there exists an infinite sequence of points $y_0 < y_1 < \cdots$ such that*

1. *$y = y_0$;*
2. *For every $l \in \mathbb{N}$, $\mathcal{M}, [y_l, y_{l+1}] \models u$.*

4.2 The Encoding of the Tiling Problem and Undecidability

Since the set of tiles $\mathcal{T}$ is finite, the set of colors is finite as well. Let us define an arbitrary order over it and denote the i-th color by $col(i)$. The propositional letters $t_1, \ldots, t_k$ are used to encode the colors of each side of the corresponding tiles $t_1, \ldots, t_k$, that is, to state that $right(t_i) = color(k_r)$, $left(t_i) = color(k_l)$, $up(t_i) = color(k_u)$, and $down(t_i) = color(k_d)$, for suitable k_r, k_l, k_u, and k_d.

The next formulas state that all u-intervals are tiles and that, whenever a tile is placed, it is unique. Moreover, they guarantee that tiles are placed in such a way that they respect conditions on colors.

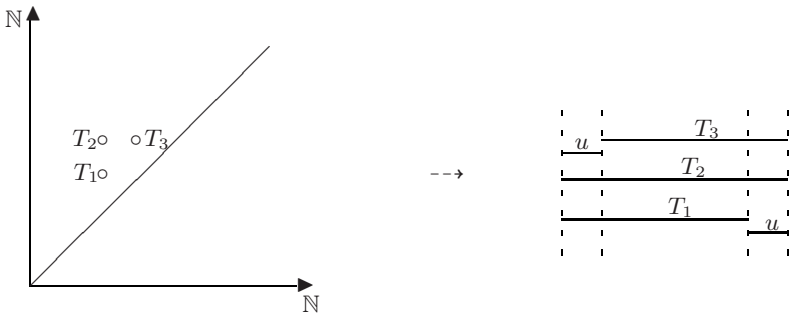


Fig. 2. A pictorial representation of the encoding

$$G(u \rightarrow \bigvee_{i=1}^{|\mathcal{T}|} t_i), \tag{3}$$

$$G \bigwedge_{i \neq j} \neg(t_i \wedge t_j), \tag{4}$$

$$G \bigwedge_{i=1}^{|\mathcal{T}|} (t_i \rightarrow \neg(uT\neg \bigvee_{j=1, up(t_i)=down(t_j)}^{|\mathcal{T}|} t_j)), \tag{5}$$

$$G\Big(u \rightarrow \bigwedge_{i=1, j=1, right(t_j) \neq left(t_i)}^{|\mathcal{T}|} \neg(t_i T t_j)\Big). \tag{6}$$

Now we define

$$\phi_{\mathcal{T}} = (2) \wedge (3) \wedge (4) \wedge (5) \wedge (6). \tag{7}$$

Lemma 2 (soundness). *Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be a set of tiles. If $\phi_{\mathcal{T}}$ is satisfiable, then $\mathcal{T}$ tiles the second octant $\mathcal{O}$.*

Proof. Let $\mathcal{M}, [x, y] \models \phi_{\mathcal{T}}$. We show that there exists a tiling function $f : \mathcal{O} \mapsto \mathcal{T}$. By Lemma 1, we know that there exists a unbounded sequence of points $y_0 < y_1 < \ldots$ such that $y = y_0$, and, for every $l \in \mathbb{N}$, $\mathcal{M}, [y_l, y_{l+1}] \models u$. Now, for each $l, m \in \mathbb{N}, l \leq m$, we put:

$$f(l, m) = t, \text{ where } \mathcal{M}, [y_l, y_{m+1}] \models t.$$

First, we have to show that f is well-defined, that is, that each $f(l, m)$ is a tile. We proceed by induction on $(m-l)$. If $(m-l) = 0$, then we are on a u-interval, and, by (3), it must be a tile. If more than one tile is placed over, it suffices to choose one of them randomly, and thus (4), f is a function. Suppose now that $f(l, m)$ is a tile whenever $m-l \leq p$, and consider $m-l = p+1$. Since $((m-1)-l) \leq p$, by inductive hypothesis $f(l, m-1)$ is a tile, say t_i. This means that $\mathcal{M}, [y_l, y_m] \models t_i$, and, by (5), $\mathcal{M}, [y_l, y_m] \models \neg(uT\neg \bigvee_{up(t_i)=down(t_j)} t_j)$. So, for every $y \geq y_m$, if $\mathcal{M}, [y_m, y] \models u$, it must be the case that $\mathcal{M}, [y_l, y] \models \bigvee_{up(t_i)=down(t_j)} t_j$.

This applies to the particular case $y = y_{m+1}$, since $\mathcal{M}, [y_m, y_{m+1}] \models u$. Hence, we have that $\mathcal{M}, [y_l, y_{m+1}] \models t_j$, that is, $f(l, m) = t_j$, for some j such that $down(t_j) = up(t_i)$. This guarantees us that f respects the 'vertical' condition of a tiling function. To conclude the proof, we need to show that that the 'horizontal' condition is respected as well. To this end, consider $f(l, m)$ and $f(l + 1, m)$. By definition, the corresponding tiles are $[y_l, y_{m+1}]$ and $[y_{l+1}, y_{m+1}]$. Since, by definition, the interval $[y_l, y_{l+1}]$ is a u-interval, by (6) it cannot be the case that $left(f(l + 1, m)) \neq right(f(l, m))$,which implies that $left(f(l + 1, m)) = right(f(l, m))$. □

Lemma 3 (completeness). *Let $\mathcal{T} = \{t_1, \ldots, t_k\}$ be a set of tiles and $f : \mathcal{O} \mapsto \mathcal{T}$ be a tiling function. Then $\phi_{\mathcal{T}}$ is satisfiable over (the intervals of) any linearly ordered set with an infinite ascending sequence.*

Proof. Assuming for simplicity that the order is $(\mathbb{N}, <)$, one can make $\phi_{\mathcal{T}}$ true at $[0, 0]$ by letting u be true at all intervals of length 1, and each t_i be true at all intervals of the form $[x, y + 1]$, where $f(x, y) = t_i$. □

We can prove similar results for $S \subseteq \{C, D, T\}$ containing C, without using the modal constant π, over any $\mathcal{K} \subseteq \mathsf{Lin}$ containing some $(T, <)$ that has an *infinite interval* $[x, y]$. The construction is analogous to the previous one. Assuming that $[x, y]$ contains an infinite increasing sequence, we substitute T by C in (1), and modify (2), (5), and (6), as follows:

$$uC\top \wedge G(u \rightarrow \top C \neg u) \wedge \neg(\top C(uC\neg(uC\top))), \tag{8}$$

$$G \bigwedge_{i=1}^{|\mathcal{T}|} \Big(t_i C u \rightarrow \bigvee_{j=1, up(t_i)=down(t_j)}^{|\mathcal{T}|} t_j \Big), \tag{9}$$

$$G \bigwedge_{i=1}^{|\mathcal{T}|} \Big(uC t_i \rightarrow \bigvee_{j=1,\, right(t_j)=left(t_i)}^{|\mathcal{T}|} t_j \Big). \tag{10}$$

If the resulting conjunction $\phi_{\mathcal{T}}$ is satisfiable over Lin, then $\mathcal{T}$ tiles the second octant; this implies that $\phi_{\mathcal{T}}$ is satisfiable on $[x, y]$. A mirror image formula works when $[x, y]$ contains an infinite descending sequence.

Theorem 2. *For any $S \subseteq \{C, D, T\}$ containing T, the logics $S(\mathcal{K})$ and $S^{\pi}(\mathcal{K})$ are not decidable for any $\mathcal{K} \subseteq \mathsf{Lin}$ with $\mathcal{K} \cap \mathsf{Asc} \neq \emptyset$, in particular when $\mathcal{K}$ is any of the following:* Lin, Dense, Dis, Asc, *the class of all infinite linearly ordered sets, any of the usual linear orders based on $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, etc. The same applies for any $S \subseteq \{C, D, T\}$ containing D, substituting* Asc *with* Des, *and $\mathbb{N}$ with $\mathbb{Z}$ or $\mathbb{Z}^{\leq 0}$. Finally, for any $S \subseteq \{C, D, T\}$ containing C, $S(\mathcal{K})$ and $S^{\pi}(\mathcal{K})$ are undecidable whenever $\mathcal{K} \subseteq \mathsf{Lin}$ contains some $(T, <)$ such that some interval in $Int(T, <)$ is infinite — e.g., when $\mathcal{K}$ is* Lin, Asc, Des, Dense, Dis, $\{(\mathbb{Q}, <)\}$, *or* $\{(\mathbb{R}, <)\}$.

Proof. If $\mathcal{K}$ is any class with $\mathcal{K} \subseteq \mathsf{Lin}$ and $\mathcal{K} \cap \mathsf{Asc} \neq \emptyset$, then $\phi_{\mathcal{T}}$ is satisfiable over $\mathcal{K}$ (and hence $\neg\phi_{\mathcal{T}} \notin T(\mathcal{K}), T^{\pi}(\mathcal{K})$) iff $\mathcal{T}$ tiles $\mathcal{O}$. Since $\phi_{\mathcal{T}}$ can be constructed from $\mathcal{T}$ by an algorithm, the result for T follows from the undecidability of the octant tiling problem. The other cases are similar. □

4.3 Undecidability over Finite Models

The above argument cannot be applied when we interpret the language over finite models. On the other hand, in [22] the language L_C^π has been shown to be undecidable when interpreted in the class of all finite linearly ordered sets, or, equivalently (since this logic can only 'look' inside the initial interval), over $\mathbb{N}$ or $\mathbb{Z}$. Here we show how to adapt the same argument to show that the logics $D^\pi(\mathsf{Fin})$ and $T^\pi(\mathsf{Fin})$ are undecidable too; again, we focus on the latter only. We closely follow [22]. In more detail, let $\mathcal{G}$ be a context-free grammar in Greibach normal form with terminals 0, 1, say, and set of non-terminals $\mathcal{N}(\mathcal{G}) = \{A_0, \dots, A_N\}$, say, where A_0 is initial. We assume that the language generated by $\mathcal{G}$ does not contain the empty string ε, and hence that $\mathcal{G}$ has no productions of the form $A_i \longrightarrow \varepsilon$. So all productions are of the form $A_i \longrightarrow V_1 V_2 \dots V_m$, where $m \geq 1$, V_1 is a terminal, and $V_2, \dots, V_m$ are terminals or non-terminals.

Whenever a word $w_0 w_1 \dots w_{n-1}$ is generated by $\mathcal{G}$, there is a finite *derivation tree* D for it (and conversely). Each node d of D has a label $\lambda(d)$: it is 0 or 1 if d is a leaf, and some non-terminal A_i otherwise, in which case the children of d taken in left-right order are labeled $V_1, V_2, \dots, V_m$, respectively, for some production rule $A_i \longrightarrow V_1 V_2 \dots V_m$ of $\mathcal{G}$ (V_1 is always a terminal and so the leftmost child of any node is a leaf). The root of D is labeled A_0, and the leaves taken in left-right order are labeled $w_0, w_1, \dots, w_{n-1}$, respectively.

We regard each non-terminal as a propositional letter, and we also use two more propositional letters, p and u. Thus, our set of propositional letters is $\mathcal{L}(\mathcal{G}) := \mathcal{N}(\mathcal{G}) \cup \{p, u\}$. We let $\phi_\mathcal{G}$ be the conjunction of the following L_T^π-formulas, where $G\phi$ is defined as in (1), and A_0 is the starting symbol for $\mathcal{G}$:

$$G(u \to \neg\pi) \wedge G\neg(\neg\pi \wedge (\neg\pi T u)), \tag{11}$$

$$\big(A_0 \wedge \neg(\neg\pi T\top)\big)T\top, \tag{12}$$

$$G\neg(A_i \wedge (\neg\pi T A_i)), \forall\ i = 0, \dots, N, \tag{13}$$

$$G\Big(\pi \wedge A_i T\top \to \bigvee_{j=1}^{n(i)} \widehat{V_1^j} T(\widehat{V_2^j} T(\cdots T(\widehat{V_{m(i,j)}^j} T A_i)) \cdots)\Big), \forall\ i = 0, \dots, N, \tag{14}$$

where

$$A_i \longrightarrow V_1^1 V_2^1 \dots V_{m(i,1)}^1 \mid V_1^2 V_2^2 \dots V_{m(i,2)}^2 \mid \cdots \mid V_1^{n(i)} V_2^{n(i)} \cdots V_{m(i,n(i))}^{n(i)}$$

is the 'consolidated' production rule of $\mathcal{G}$ for A_i, and for any terminal or non-terminal V:

$$\widehat{V} = \begin{cases} u \wedge p, & \text{if } V = 1, \\ u \wedge \neg p, & \text{if } V = 0, \\ V, & \text{otherwise.} \end{cases} \tag{15}$$

Lemma 4 (soundness). *Let $\mathcal{G}_1, \mathcal{G}_2$ be grammars as above, with (wlog.) $\mathcal{N}(\mathcal{G}_1) \cap \mathcal{N}(\mathcal{G}_2) = \emptyset$. Note that $\mathcal{L}(\mathcal{G}_1) \cap \mathcal{L}(\mathcal{G}_2) = \{p, u\}$. If the L_T^π-formula $\psi = \phi_{\mathcal{G}_1} \wedge \phi_{\mathcal{G}_2}$ is satisfiable in a model with frame of the form $\mathfrak{Int}_T(T, <)$ for some finite strict linear order $(T, <)$, then the languages generated by $\mathcal{G}_1, \mathcal{G}_2$ have a word in common.*

Proof. Suppose that ψ is satisfiable in a model $\mathcal{M}$ with frame $\mathfrak{Int}_T(T,<)$, where $(T,<) \in \mathsf{Fin}$. We can suppose wlog. that $\mathcal{M},[0,0] \models \psi$ for some point $0 \in T$, where $\{x \in T : x \geq 0\} = \{0,1,\ldots,n\}$ for some natural number $n \geq 0$, and $<$ is the usual ordering on this set. We will see below that $n > 0$. For $x < n$ put

$$w_x = \begin{cases} 1, & \text{if } \mathcal{M},[x,x+1] \models p, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

We show that the word $w = w_0 \ldots w_{n-1}$ is generable by $\mathcal{G}_1$ by constructing a derivation tree D for it. Each node d of D will be associated with an interval $[x_d, y_d]$ of $(T,<)$, where $0 \leq x_d \leq y_d$. We will add nodes d to D step by step, in such a way that

$$\mathcal{M},[x_d,y_d] \models \widehat{\lambda(d)}. \tag{17}$$

Given this, if $d \in D$ is labeled by a terminal $\lambda(d) = \alpha \in \{0,1\}$, then $\mathcal{M},[x_d,y_d] \models \widehat{\alpha}$; by (15), $\widehat{\alpha} \vdash u$, so $\mathcal{M},[x_d,y_d] \models u$. As $\mathcal{M},[0,0] \models (11)$, we have $y_d = x_d + 1$.

First we add the root r to D, and put $[x_r,y_r] = [0,n]$ and $\lambda(r) = A_0$. As $\mathcal{M},[0,0] \models (12)$, we have $\mathcal{M},[0,n] \models A_0$, so (17) holds. Then we repeat the following while D has leaves labeled by non-terminals. Pick such a leaf d, where $\lambda(d)$ is a non-terminal A_i, say. By (17), $\mathcal{M},[x_d,y_d] \models A_i$, so $\mathcal{M},[x_d,x_d] \models \pi \wedge A_i T \top$. Since $\mathcal{M},[0,0] \models (14)$, there exist a production rule $A_i \longrightarrow V_1 V_2 \ldots V_m$ of $\mathcal{G}_1$ and points $x_d = x_0 \leq x_1 \leq \ldots \leq x_m \leq n$ with $\mathcal{M},[x_{j-1},x_j] \models \widehat{V_j}$ for each $j = 1,\ldots,m$, and $\mathcal{M},[x_d,x_m] \models A_i$. But $\mathcal{M},[0,0] \models (13)$, so $x_m = y_d$. We now add new nodes $d_1,\ldots,d_m$ to D as the left-to-right children of d, and we put $[x_{d_j},y_{d_j}] = [x_{j-1},x_j]$ and $\lambda(d_j) = V_j$ for each $j = 1,\ldots,m$. By choice of the x_j, (17) is preserved. Note that since V_1 is a terminal, $x_{d_1} < y_{d_1}$. It follows that $x_d < y_d$ for each $d \in D$; $n > 0$; and the lengths of intervals associated with non-terminals strictly decrease as we move from the root. So as T is finite, the process terminates. Let $l_0,\ldots,l_{m-1}$ be the leaves of the final D in left-to-right order. Each l_i is labeled by a terminal, and so is associated with a two-point interval. Now the construction preserves the property that for any $x < n$, there is always a leaf of D whose associated interval contains $[x,x+1]$. So at the end, this is still true, and it follows that $[x,x+1]$ is associated with some l_i. Therefore, $m = n$ and $l_0,\ldots,l_{n-1}$ are associated with $[0,1],[1,2],\ldots,[n-1,n]$, respectively. For each $x < n$ we have $w_x = 1$ iff $\mathcal{M},[x,x+1] \models p$ (by (16)), iff $\lambda(l_x) = 1$ (by (15) and (17)). So $\lambda(l_x) = w_x$. Hence, D is a derivation tree for w in $\mathcal{G}_1$. Since we can prove by the same argument that w is generable also by $\mathcal{G}_2$, the languages generated by $\mathcal{G}_1$, and $\mathcal{G}_2$ both contain w, and thus the thesis. □

Lemma 5 (completeness). *Let $\mathcal{G}_1,\mathcal{G}_2$ be as in the previous lemma. If the languages generated by $\mathcal{G}_1,\mathcal{G}_2$ have a word in common, then the L^{π}_T-formula $\psi = \phi_{\mathcal{G}_1} \wedge \phi_{\mathcal{G}_2}$ is satisfiable in a model with frame of the form $\mathfrak{Int}_T(T,<)$ for some finite strict linear order $(T,<)$.*

Proof. Run Lemma 4's proof backwards: use derivation trees in $\mathcal{G}_1,\mathcal{G}_2$ for the common word to read off models of $\phi_{\mathcal{G}_1},\phi_{\mathcal{G}_2}$ on the same base, on any $(T,<) \in \mathsf{Fin}$ with long enough intervals. □

Theorem 3. *The logics* $T^{\pi}(\mathsf{Fin})$ *and* $D^{\pi}(\mathsf{Fin})$ *are undecidable.*

Proof. By [18, theorem 8.10], it is undecidable whether the languages generated by two context-free grammars have a common word. It is not hard to see that this is also true for context-free grammars in the form we have used. By Lemma 4 and 5, this problem reduces to satisfiability of L_T^{π}-formulas over the intervals of finite linear orders, which is therefore also undecidable. The argument for $D^{\pi}(\mathsf{Fin})$ is symmetrical. □

The same argument shows that $D^{\pi}(\mathbb{N}, <)$ and $T^{\pi}(\mathbb{N}, >)$ are undecidable.

5 Non-finite Axiomatizability

In this section we will show that the logics $S(\mathcal{K})$ and $S^{\pi}(\mathcal{K})$ are not finitely axiomatizable using orthodox inference rules, for any non-empty $S \subseteq \{C, D, T\}$ and any class $\mathcal{K}$ with $(\mathbb{Q}, <) \in \mathcal{K} \subseteq \mathsf{Lin}$. For example, $CDT^{\pi}(\mathsf{Lin})$ and $CDT^{\pi}(\mathsf{Dense})$ are not finitely axiomatizable. The argument is based on the so-called 'Monk algebras' from the theory of relation algebra (see, e.g., [21,20,17]), but we do not assume any knowledge of that topic here.

5.1 Basic Definitions

Let $\eta, \kappa > 0$ be cardinals, and define an equivalence relation $\sim$ on $\eta \times \kappa = \{(i, j) : i < \eta,\ j < \kappa\}$ by $(i, j) \sim (i', j')$ if and only if $j = j'$. We think of j as the *color* of (i, j), so $\sim$ is the 'same color' relation. Then, we form a relational structure $\alpha(\eta, \kappa)$ as follows.

Definition 4. *We define* $\alpha(\eta, \kappa)$ *as a tuple of the type* (I, Π, ρ)*, where:*

- $I = (\eta \times \kappa) \cup \{1'\}$*, where* 1'*, called* identity*, is assumed not to be in* $\eta \times \kappa$*;*
- $\Pi = \{1'\}$*;*
- ρ *is the set of all triples* $(a, b, c) \in I^3$ *such that 1) one of* a, b, c *is* 1' *and the other two are equal, or 2)* $1' \notin \{a, b, c\}$ *and* a, b, c *are not all* $\sim$*-equivalent.*

The relations $R_C, R_D, R_T \subseteq I^3$ *(for* I *as above) are defined as follows:*

- $R_C = \rho \setminus \{(1', a, a) : a \in I, a \neq 1'\}$,
- $R_T = \rho \setminus \{(a, 1', a) : a \in I, a \neq 1'\}$,
- $R_D = \rho \setminus \{(a, a, 1') : a \in I, a \neq 1'\}$.

We define the S*-frame* $\mathcal{F}_S(\eta, \kappa) = (I, \Pi, R_\sharp : \sharp \in S)$*, for each* $S \subseteq \{C, D, T\}$.

The $\mathcal{F}_S(\eta, \kappa)$ are 'non-standard' frames. They are not based on intervals, and indeed, this is detectable in L_S: if η, κ are finite and $\eta \gg \kappa$ then $\mathcal{F}_S(\eta, \kappa)$ does not even validate the logic $S(\mathsf{Lin})$. But $\mathcal{F}_S(\omega, \omega)$ is a bounded morphic image of $\mathfrak{Int}(\mathbb{Q}, <)$ (cf., e.g., [1, def. 2.10]), and so does validate $S^{\pi}(\mathcal{K})$. Since we can obtain $\mathcal{F}_S(\omega, \omega)$ as a 'limit' (via ultraproducts or compactness) of the finite frames $\mathcal{F}_S(\eta, \kappa)$, this will show that $S(\mathcal{K})$ and $S^{\pi}(\mathcal{K})$ are not finitely axiomatizable.

5.2 Frames $\mathcal{F}_S(\eta,\kappa)$ Not Validating $S(\mathsf{Lin})$

Define $f:\omega\to\omega$ by $f(0)=1$, and $f(k+1)=1+(k+1)\cdot f(k)$ for every $k<\omega$. For any set X, and $1\leq k<\omega$, a *k-coloring of X* is a coloring of the edges of the complete undirected loop-free graph whose set of nodes is X, using at most k colors, such that no 'monochromatic triangle' appears — that is, there is no 3-element subset of X such that all three edges lying within it are given the same color. The following Ramsey-type theorem can be found in [15].

Proposition 1. *Let $k\geq 1$. If a set X has more than $f(k)$ elements then X has no k-coloring.*

Theorem 4. *Suppose that $S\subseteq\{C,D,T\}$ is non-empty, $\eta>0$, $\kappa>1$, and $f(\kappa)<|\mathcal{F}_S(\eta,\kappa)|<\omega$. Then $\mathcal{F}_S(\eta,\kappa)$ does not validate $S(\mathsf{Lin})$.*

Proof. One can write down a 'Jankov–Fine' formula ϕ (see, e.g., [1, §3.4]) describing $\mathcal{F}_S(\eta,\kappa)$ and satisfiable only in bounded morphic pre-images of $\mathcal{F}_S(\eta,\kappa)$. If $\mathcal{F}_S(\eta,\kappa)$ validated $S(\mathsf{Lin})$, ϕ would be consistent with $S(\mathsf{Lin})$, and so there would be a bounded morphism h from some concrete interval frame $\mathfrak{Int}_S(T,<)$ onto $\mathcal{F}_S(\eta,\kappa)$. This would induce a partial κ-coloring of T: the color of the edge xy for $x<y$ would be the color of $h([x,y])$ when $h([x,y])\neq 1$'. As $\mathcal{F}_S(\eta,\kappa)$ is large, Lemma 1 shows that (assuming $C\in S$) some interval of T would chop into two subintervals, all three having the same color. This contradicts the definition of R_C. The cases $D\in S$, $T\in S$ are similar. □

5.3 Frames $\mathcal{F}_S(\eta,\kappa)$ That Do Validate $S^\pi(\mathbb{Q},<)$

We show here that $\mathcal{F}_S(\omega,\omega)$ does validate $S^\pi(\mathbb{Q},<)$, for any $S\subseteq\{C,D,T\}$. To do this, we will construct a bounded morphism from $\mathfrak{Int}_S(\mathbb{Q},<)$ onto $\mathcal{F}_S(\omega,\omega)$ step by step, using the concepts of 'networks' and 'representations'.

Definition 5. *A* network *is a map $N:B\times B\to\alpha(\omega,\omega)$, for some $B\subseteq\mathbb{Q}$, such that for all $x,y,z\in B$ we have:*

1. $N(x,y)=1'\iff x=y$,
2. $N(x,y)=N(y,x)$,
3. $\big(N(x,z),N(x,y),N(y,z)\big)\in\rho$.

We write $B(N)$ (the 'base' of N) for the set $B=\{x\in\mathbb{Q}:(x,x)\in\mathrm{dom}(N)\}$. A network can be thought of as a complete directed graph with edges labeled by elements of $\alpha(\omega,\omega)$, satisfying the conditions above. As networks N,M are functions, it follows from the definitions that $N\subseteq M$ iff $B(N)\subseteq B(M)$ and $N=M\restriction(B(N)\times B(N))$.

Definition 6. *A network N is said to be a* representation[2] *of $\alpha(\omega,\omega)$ if $B(N)=\mathbb{Q}$ and the following hold for all $x,y\in\mathbb{Q}$, where the quantifiers range over $\mathbb{Q}$:*

[2] The term 'complete representation' would be more in accordance with usual algebraic terminology.

1. $\exists z(z \leq x \wedge N(x,z) = a)$ *and* $\exists z(z \geq x \wedge N(x,z) = a)$ *for every* $a \in \alpha(\omega,\omega)$,
2. $x < y \rightarrow \exists z(x \leq z \leq y \wedge N(x,z) = a \wedge N(z,y) = b)$ *whenever* $(N(x,y),a,b) \in \rho$,
3. $x < y \rightarrow \exists z(z \geq y \wedge N(x,z) = a \wedge N(z,y) = b)$ *whenever* $(N(x,y),a,b) \in \rho$ *and* $a \neq 1$',
4. $x < y \rightarrow \exists z(z \leq x \wedge N(x,z) = a \wedge N(z,y) = b)$ *whenever* $(N(x,y),a,b) \in \rho$ *and* $b \neq 1$'.

Our aim is to show that $\alpha(\omega,\omega)$ has a representation. We can derive validity of $S^\pi(\mathbb{Q},<)$ in $\mathcal{F}_S(\omega,\omega)$ from this quite easily.

Lemma 6. $\alpha(\omega,\omega)$ *has a representation.*

Proof. We can build a representation by a 'step by step' game played on finite networks. In each round, one player challenges some failure of a property of Definition 6 for the current network. The other player replaces the network by an extension lacking this defect. This is always possible since we are using $\alpha(\eta,\kappa)$ with infinite κ, so that brand new colors can be used to label network edges whenever necessary. This avoids monochromatic triangles of the kind we met in Theorem 4. At the end of the game, a representation results. □

Theorem 5. $\mathcal{F}_S(\omega,\omega)$ *validates* $S^\pi(\mathbb{Q},<)$ *for any* $S \subseteq \{C,D,T\}$.

Proof. A representation N of $\alpha(\omega,\omega)$ induces a bounded morphism h from $\mathfrak{Int}_S(\mathbb{Q},<)$ onto $\mathcal{F}_S(\omega,\omega)$, via $h([x,y]) = N(x,y)$. Bounded morphisms preserve validity of formulas. □

5.4 Non Finite Axiomatizability

Theorem 6. *For any non-empty* $S \subseteq \{C,D,T\}$ *and any class* $\mathcal{K} \subseteq \mathsf{Lin}$ *that contains* $(\mathbb{Q},<)$*, the logics* $S(\mathcal{K})$ *and* $S^\pi(\mathcal{K})$ *are not finitely axiomatizable using standard inference rules (modus ponens, universal generalization, substitution).*

Proof. Let Λ be either one of $S(\mathcal{K}), S^\pi(\mathcal{K})$. Note that

$$S(\mathsf{Lin}) \subseteq \Lambda \subseteq S^\pi(\mathbb{Q},<). \tag{18}$$

Assume for contradiction that Φ is a finite set of axioms for Λ in its own signature (L_S or L_S^π). Then $(\bigwedge \Phi) = \phi(p_1,\ldots,p_k)$, say, axiomatises Λ alone. For each finite $n > 1$, choose finite η_n so large that $|\alpha(\eta_n,n)| > f(n)$. By Theorem 4, $\mathcal{F}_S(\eta_n,n)$ does not validate even $S(\mathsf{Lin})$, so by (18), it certainly does not validate Λ. Because the standard inference rules preserve validity over any frame, there is a model $\mathcal{M}_n = (\mathcal{F}_S(\eta_n,n),h_n)$ in which ϕ is not valid. We regard $\mathcal{M}_n$ as a first-order structure in an appropriate signature. Let $\mathcal{M}$ be a countable elementary substructure of a non-principal ultraproduct of the $\mathcal{M}_n$ (cf. [10]). By Łoś's Theorem, it follows that the frame of $\mathcal{M}$ is isomorphic to $\mathcal{F}_S(\omega,\omega)$, and that ϕ is not valid in $\mathcal{M}$. This contradicts Theorem 5 that $\mathcal{F}_S(\omega,\omega)$ validates $S^\pi(\mathbb{Q},<)$, and so (by (18)) certainly validates Λ. □

Table 1. A resume of the results of this paper

$\mathcal{K}$	$C(\mathcal{K})$	$C^{\pi}(\mathcal{K})$	$D(\mathcal{K})$	$D^{\pi}(\mathcal{K})$	$T(\mathcal{K})$	$T^{\pi}(\mathcal{K})$
Lin	Und., NFA	Und.[13],NFA	Und., NFA	Und., NFA	Und., NFA	Und., NFA
Asc	Und., NFA	Und., NFA	NFA	NFA	Und., NFA	Und., NFA
Des	Und., NFA	Und., NFA	Und.,NFA	Und., NFA	NFA	NFA
Fin	?	Und.[22]	?	Und.	?	Und.
Dense	Und., NFA	Und.[19],NFA	Und., NFA	Und., NFA	Und., NFA	Und., NFA
Dis	Und.	Und.	Und.	Und.	Und.	Und.
$\mathbb{N}$	?	Und.[22],	?	Und.	Und.	Und.
$\mathbb{Z}$	?	Und.[22],	Und.	Und.	Und.	Und.
$\mathbb{Q}$	Und., NFA	Und.[19],NFA	Und., NFA	Und, NFA	Und., NFA	Und., NFA

The logics $CDT^{\pi}(\mathsf{Lin}), CDT^{\pi}(\mathsf{Dis}), CDT^{\pi}(\mathsf{Dense}), CDT^{\pi}(\mathbb{Q})$ were finitely axiomatized by Venema [25], using an unorthodox Burgess/Gabbay-style inference rule. There is no conflict between this result and Theorem 6. A formula derived from a formula valid in $\mathcal{F}_S(\eta, \kappa)$ using Venema's rule need not be valid in $\mathcal{F}_S(\eta, \kappa)$. So the argument of Theorem 6 fails when this rule is added.

6 Conclusions

In this paper we have considered the temporal logic for intervals CDT introduced by Venema in [25]. Finite axiomatizability with standard rules and decidability/undecidability of CDT itself and of its possible single-modality fragment is a natural question that was open since the introduction of the logic, and to which we have answered here. Results are summarized in Table 1, where 'Und.' means undecidable and 'NFA' indicates non-finite axiomatizability.

Acknowledgments. Angelo Montanari was co-financed by the Italian national project on "Constraint and preferences as unifying formalism for system analysis and solutions of real problems", and Guido Sciavicco was co-financed by the Spanish MEC - FEDER project "IDEATIO", Ref. No: TIN2006-15460-C04-01.

References

1. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2002)
2. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives of Mathematical Logic. Springer, Heidelberg (1997)
3. Bowman, H., Thompson, S.: A decision procedure and complete axiomatization of finite interval temporal logic with projection. Journal of Logic and Computation 13(2), 195–239 (2003)
4. Bresolin, D., Goranko, V., Montanari, A., Sala, P.: Tableau-based Decision Procedure for the Logic of Proper Subinterval Structures over Dense Orderings. In: Areces, C., Demri, S. (eds.) Proceedings of M4M-5: 5th International Workshop on Methods for Modalities, pp. 335–351 (2007)

5. Bresolin, D., Goranko, V., Montanari, A., Sala, P.: Tableau Systems for Logics of Subinterval Structures over Dense Orderings. In: Olivetti, N. (ed.) TABLEAUX 2007. LNCS (LNAI), vol. 4548, pp. 73–89. Springer, Heidelberg (2007)
6. Bresolin, D., Goranko, V., Montanari, A., Sciavicco, G.: On Decidability and Expressiveness of Propositional Interval Neighborhood Logics. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 84–99. Springer, Heidelberg (2007)
7. Bresolin, D., Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood logics: Expressiveness, decidability, and undecidable extensions. Technical Report 05, Department of Mathematics and Computer Science, University of Udine, Italy (2008)
8. Bresolin, D., Montanari, A., Sala, P.: An optimal tableau-based decision algorithm for Propositional Neighborhood Logic. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 549–560. Springer, Heidelberg (2007)
9. Bresolin, D., Montanari, A., Sciavicco, G.: An optimal decision procedure for Right Propositional Neighborhood Logic. Journal of Automated Reasoning 38(1-3), 173–199 (2007)
10. Chang, C.C., Keisler, H.J.: Model theory. North-Holland, Amsterdam (1990)
11. Gabbay, D.M.: An irreflexivity lemma with applications to axiomatizations of conditions on tense frames. In: Monnich, U. (ed.) Aspects of Philosophical Logic, pp. 67–89. Reidel, Dordrecht (1981)
12. Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood temporal logics. Journal of Universal Computer Science 9(9), 1137–1167 (2003)
13. Goranko, V., Montanari, A., Sciavicco, G.: A road map of interval temporal logics and duration calculi. J. of Applied Non-Classical Logics 14(1–2), 9–54 (2004)
14. Goranko, V., Montanari, A., Sciavicco, G., Sala, P.: A general tableau method for propositional interval temporal logics: Theory and implementation. Journal of Applied Logic 4(3), 305–330 (2006)
15. Greenwood, R.E., Gleason, A.M.: Combinatorial relations and chromatic graphs. Canadian Journal of Mathematics 7, 1–7 (1955)
16. Halpern, J., Shoham, Y.: A propositional modal logic of time intervals. Journal of the ACM 38(4), 935–962 (1991)
17. Hirsch, R., Hodkinson, I.: Relation algebras by games. Studies in Logic and the Foundations of Mathematics, vol. 147. North-Holland, Amsterdam (2002)
18. Hopcroft, J.E., Ullman, J.D.: Introduction to automata theory, languages, and computation. Addison-Wesley, Reading (1979)
19. 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)
20. Maddux, R.: Relation algebras. Studies in Logic and the Foundations of Mathematics, vol. 150. Elsevier, Amsterdam (2006)
21. Monk, J.D.: Nonfinitizability of classes of representable cylindric algebras. Journal of Symbolic Logic 34, 331–343 (1969)
22. Moszkowski, B.: Reasoning about digital circuits. Tech. rep. stan-cs-83-970, Dept. of Computer Science, Stanford University, Stanford, CA (1983)
23. Roy, S., Sciavicco, G.: Completeness of chop. In: Guesguen, H.W., Ligozat, G., Rodriguez, R.V. (eds.) Proc. of the IJCAI 2007 Workshop on Spatial and Temporal Reasoning, pp. 90–95 (2007)
24. Venema, Y.: Expressiveness and completeness of an interval tense logic. Notre Dame Journal of Formal Logic 31(4), 529–547 (1990)
25. Venema, Y.: A modal logic for chopping intervals. Journal of Logic and Computation 1(4), 453–476 (1991)

On the Almighty Wand*

Rémi Brochenin, Stéphane Demri, and Etienne Lozes

LSV, ENS Cachan, CNRS, INRIA Saclay, France

Abstract. We investigate decidability, complexity and expressive power issues for (first-order) separation logic with one record field (herein called SL) and its fragments. SL can specify properties about the memory heap of programs with singly-linked lists. Separation logic with *two* record fields is known to be undecidable by reduction of finite satisfiability for classical predicate logic with one binary relation. Surprisingly, we show that second-order logic is as expressive as SL and as a by-product we get undecidability of SL. This is refined by showing that SL without the separating conjunction is as expressive as SL, whence undecidable too. As a consequence of this deep result, in SL the magic wand can simulate the separating conjunction. By contrast, we establish that SL without the magic wand is decidable with non-elementary complexity by reduction from satisfiability for the first-order theory over finite words. Equivalence between second-order logic and separation logic extends to the case with more than one selector.

1 Introduction

Separation logic. Programming languages with pointer variables have seldom mechanisms to detect errors. An inappropriate management of memory is the source of numerous security holes. Prominent logics for analysing such pointer programs include separation logic [Rey02], pointer assertion logic PAL [JJKS97], TVLA [LAS00], alias logic [BIL04], BI (Bunched Implication) [IO01] and LRP (logic of reachable patterns) [YRS+05] to quote a few examples. Separation logic is an assertion language used in Hoare-like proof systems [Rey02] that are dedicated to verify programs manipulating heaps. Any procedure mechanizing the proof search requires subroutines that check satisfiability of formulae from the assertion language. The main concern of the paper is to analyze the expressive power of the assertion language and its decidability status. Recall that separation logic contains a structural *separation* connective and its adjoint (the separating implication $-\!\!*$, also known as the *magic wand*). Concise and modular proofs can be derived using these connectives, since they can express properties such as non-aliasing and disjoint concurrency. In this perspective, the models of separation logic are pairs made of a store (variable valuation) and a memory heap (partial function with finite domain) that are understood as memory states.

The decidability of the satisfiability problem for separation logic has been intensively studied so far: first-order separation logic with at least two selectors (record fields) is known to be undecidable [CYO01b] by reduction of finite satisfiability for classical

* Supported by RNTL project AVERILES – R. Brochenin is supported by a DGA/CNRS fellowship.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 323–338, 2008.
© Springer-Verlag Berlin Heidelberg 2008

predicate logic with one binary relation [Tra50] (even with no separating connectives). Decidable fragments have been introduced and investigated, see e.g. [BCO04]; such fragments involve some specialized predicates for lists or trees, and some restrictions on first-order quantification. The complexity of various quantifier-free fragments has also been characterized in the past, see e.g. [CYO01b, CYO01a, Rey02, BDL07]. As far as the expressive power is concerned, propositional separation logic can be naturally reduced to propositional calculus or to a fragment to Presburger arithmetic, see [Loz04, CGH05]. On several spatial logics, the adjunct elimination property holds in the absence of first-order quantification, but does not extend to the first-order case, see e.g. [DGG04, Loz05]. To summarize, all the proof techniques used in these works do not adapt to first-order separation logic over the class of models with only one selector, for which both decidability status and the characterization of expressiveness are open.

Our motivations. A long-standing question about separation logic is how it compares with second-order logic. This is a very natural question since separating conjunction and its adjoint are essentially second-order connectives (see also a similar concern on graphs with spatial logics [DGG07]). Moreover, many properties on heaps require second-order logic, for instance to express recursive predicates, or lists and trees properties. In a sense, being able to distinguish the expressive power of these two logical formalisms would justify the use of separation logic. An attempt to make such a comparison can be found in [KR04] but the models are quite different from those considered herein (relational structures with no constraint on the finiteness of the domain of relations). In this paper, our aim is to investigate decidability, complexity and expressive power issues for first-order separation logic with one selector (record field) and its fragments when the models are the standard memory states.

Our contributions. We show that first-order separation logic with one selector (called herein SL) is as expressive as second-order logic over the class of memory states. As a by product, we get that even in presence of a unique selector, 1) first-order separation logic is undecidable (solving an open problem stated in [GM08]), 2) it may express heap properties involving several selectors (passing via second-order logic). This is refined by showing that SL without the separating conjunction is as expressive as SL, whence undecidable too. Our proof also shows that the two formalisms have the same conciseness modulo logspace translations. As a consequence of this deep result, in SL the magic wand can simulate the separating conjunction. So, the magic wand is very powerful, which is interesting because very often this connective, that is not very natural we admit, is excluded from studied fragments of separation logic. Equivalence between second-order logic and separation logic extends to the case with more than one selector by simple adaptation of the one selector case. By contrast, in [KR04], the separating conjunction is sufficient to capture second-order logic but on a different class of models. We also establish that SL without the magic wand is decidable with non-elementary complexity by reduction from satisfiability for the first-order theory over finite words [Sto74] (this result holds already with three variables). Decidability is shown by reduction into weak monadic second-order theory of one unary function that is shown decidable in [Rab69]. It is worth noting that even though the first-order theory of one unary function is known to be not elementary recursive [BGG97], we cannot

take advantage of this result since in our models the domain of the unary function is necessarily finite and finiteness cannot be expressed in most first-order dialects. As a by-product, we obtain that the entailment problem considered in [BCO04] for a fragment of separation logic with one selector is decidable.

Related work. As seen previously, heap properties are formalized in various logical languages [JJKS97, LAS00, Rey02, BIL04, YRS+05] and separation logic is just one prominent of these logics. Verification methods and logics for verifying programs with singly-linked lists can be found for instance in [BCO04, BHMV05, RZ06]. From another perspective, the relationships between logics on graphs with separating features and second-order logic can be found in [DGG07]. Finally, we would like to mention that sabotage modal logics (SML), see e.g. in [LR03], have also the ability to modify the model under evaluation. So far, we are not aware of any work relating separation logic and SML.

Omitted proofs can be found in [BDL08].

2 Preliminaries

In this section, we recall the definition of first-order separation logic with one selector (called herein SL) and second-order logic (SO) over the same class of structures. We introduce a formal notion of expressiveness, provide examples of properties that can be expressed in SL and present a straightforward encoding of SL into a fragment of SO.

2.1 Separation Logic and Second-Order Logic

Memory states. Memory states are models for all the logical formalisms we consider herein. They represent the states of the memory for programs manipulating lists. Let Loc be a countably infinite set of *locations* ranged over with $l, l', \ldots$ that represents the set of addresses. A memory state is composed of a pair made of a *store* and a *heap*. Let Var be a countably infinite set of (first-order) *variables* $\mathtt{x}, \mathtt{y}, \mathtt{z}, \ldots$. A *memory state* (also called a *model* in the rest of the document) is a pair (s, h) such that

- s is a variable valuation of the form $s : \mathtt{Var} \to \mathtt{Loc}$ (store),
- h is a partial function $h : \mathtt{Loc} \rightharpoonup \mathtt{Loc}$ with finite domain (heap). We write $\mathtt{dom}(h)$ to denote its domain and $\mathtt{ran}(h)$ to denote its range.

Given a finite set X of variables (for instance occurring in a given formula), we can assume that a model is finite by restricting the domain of the store to X. The variables in Var can be viewed as programming variables, the domain of h as the set of addresses of allocated cells, and $h(l)$ as the value held by the cell at the address l. Two heaps h_1, h_2 are said to be *disjoint*, noted $h_1 \perp h_2$, if their domains are disjoint; when this holds, we write $h_1 * h_2$ to denote the disjoint union $h_1 \uplus h_2$. Given a memory state (s, h) and a location l we write $\widetilde{\sharp} l$ to denote the cardinal of the set $\{l' \in \mathtt{Loc} : h(l') = l\}$ (number of *predecessors* of the location l in (s, h)). A location l' is a *descendant* of l if there is $n \geq 0$ such that $h^n(l) = l'$ ($h^n(l)$ is not always defined).

Formulae in SL *and* SO. Formulae of first-order separation logic with one selector SL are defined by the grammar $\phi := \neg\phi \mid \phi \wedge \phi \mid \exists \mathtt{x}.\phi \mid \mathtt{x} \hookrightarrow \mathtt{y} \mid \mathtt{x} = \mathtt{y} \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi$.

The connective $*$ is called *separating conjunction* whereas the adjoint operator $-\!\!*$ is usually called the *magic wand*. We will make use of standard notations for the derived connectives $\forall, \vee, \Rightarrow, \Leftrightarrow$... We also introduce a slight variant of the dual connective for the magic wand, also called the *septraction*: $\phi \mathbin{-\!\!\!\not*} \psi$ is defined as the formula $\neg((\phi) \mathbin{-\!\!*} (\neg(\psi)))$. We write $\mathtt{FV}(\phi)$ to denote the set of free variables occurring in ϕ and $\mathtt{SL}(*)$ [resp. $\mathtt{SL}(-\!\!*)$] to denote the restriction of $\mathtt{SL}$ without the magic wand [resp. the separating conjunction].

In order to define formulae in $\mathtt{SO}$, we consider a family $\mathtt{VAR} = (\mathtt{VAR}^i)_{i\geq 0}$ of second-order variables, denoted by $\mathtt{P}, \mathtt{Q}, \mathtt{R}, \ldots$ that will be interpreted as finite relations over $\mathtt{Loc}$ ($= \mathtt{Val} = \mathbb{N}$). Each variable in $\mathtt{VAR}^i$ is interpreted as an i-ary relation. An *environment* $\mathcal{E}$ is an interpretation of the second-order variables such that for every $\mathtt{P} \in \mathtt{VAR}^i$, $\mathcal{E}(\mathtt{P})$ is a finite subset of $\mathtt{Loc}^i$. Since we require finiteness of models, the version of second-order logics we shall consider is usually called *weak*.

Formulae of (weak) second-order logic $\mathtt{SO}$ are defined by the grammar $\phi := \neg\phi \mid \phi \wedge \phi \mid \exists \mathtt{x}.\phi \mid \mathtt{x} \hookrightarrow \mathtt{y} \mid \mathtt{x} = \mathtt{y} \mid \exists \mathtt{P}.\phi \mid \mathtt{Q}(\mathtt{x}_1, \ldots, \mathtt{x}_n)$, where $\mathtt{P}, \mathtt{Q}$ are second-order variables and $\mathtt{Q} \in \mathtt{VAR}^n$. We write $\mathtt{MSO}$ [resp. $\mathtt{DSO}$] to denote the restriction of $\mathtt{SO}$ to second-order variables in $\mathtt{VAR}^1$ [resp. $\mathtt{VAR}^2$]. As usual, a *sentence* is defined as a formula with no free occurrence of second-order variables.

Satisfaction relations for $\mathtt{SL}$ *and* $\mathtt{SO}$. The logics $\mathtt{SL}$ and $\mathtt{SO}$ share the same class of models, namely the set of memory states. The satisfaction relation for $\mathtt{SO}$ is defined below with argument an environment $\mathcal{E}$ (below $\mathtt{P} \in \mathtt{VAR}^n$).

$$
\begin{array}{lll}
(s,h),\mathcal{E} \models \exists \mathtt{P}.\ \phi & \text{iff} & \text{there is a finite subset } \mathcal{R} \text{ of } \mathtt{Loc}^n, \\
 & & \text{such that } (s,h),\mathcal{E}[\mathtt{P} \mapsto \mathcal{R}] \models \phi \\
(s,h),\mathcal{E} \models \mathtt{P}(\mathtt{x}_1, \cdots, \mathtt{x}_n) & & \\
 & \text{iff} & (s(\mathtt{x}_1), \ldots, s(\mathtt{x}_n)) \in \mathcal{E}(\mathtt{P}) \\
(s,h),\mathcal{E} \models \neg\phi & \text{iff} & \text{not } (s,h),\mathcal{E} \models \phi \\
(s,h),\mathcal{E} \models \phi \wedge \psi & \text{iff} & (s,h),\mathcal{E} \models \phi \text{ and } (s,h),\mathcal{E} \models \psi \\
(s,h),\mathcal{E} \models \exists \mathtt{x}.\ \phi & \text{iff} & \text{there is } l \in \mathtt{Loc} \text{ such that } (s[\mathtt{x} \mapsto l],h),\mathcal{E} \models \phi \\
(s,h),\mathcal{E} \models \mathtt{x} \hookrightarrow \mathtt{y} & \text{iff} & h(s(\mathtt{x})) = s(\mathtt{y}) \\
(s,h),\mathcal{E} \models \mathtt{x} = \mathtt{y} & \text{iff} & s(\mathtt{x}) = s(\mathtt{y})
\end{array}
$$

As usual, when ϕ is a sentence, we write $(s,h) \models \phi$ to denote $(s,h),\mathcal{E} \models \phi$ for any environment $\mathcal{E}$ since $\mathcal{E}$ has no influence on the satisfaction of ϕ. The satisfaction relation for $\mathtt{SL}$ is defined without any environment (or equivalently with no influence of the environment) whereas the clauses that are specific to $\mathtt{SL}$ are the following ones:

$$
\begin{array}{lll}
(s,h) \models \phi_1 * \phi_2 & \text{iff} & \text{there are two heaps } h_1, h_2 \\
 & & \text{such that } h = h_1 * h_2 \text{ and } (s,h_i) \models \phi_i \ (i = 1,2) \\
(s,h) \models \phi_1 -\!\!* \phi_2 & \text{iff} & \text{for all heaps } h' \bot h, \\
 & & \text{if } (s,h') \models \phi_1 \text{ then } (s, h' * h) \models \phi_2.
\end{array}
$$

So, $(s,h) \models \phi_1 \mathbin{-\!\!\!\not*} \phi_2$ iff there is $h' \perp h$ such that $(s,h') \models \phi_1$ and $(s, h * h') \models \phi_2$. Validity and satisfiability problems are defined in the usual way. The connective $-\!\!*$ is the *adjunct* of $*$, meaning that $(\phi * \psi) \Rightarrow \varphi$ is valid iff $\phi \Rightarrow (\psi -\!\!* \varphi)$ is valid. Observe that $*$ and $-\!\!*$ are not interdefinable since typically the formula $((\phi * \psi) \Rightarrow \varphi) \Leftrightarrow (\phi \Rightarrow (\psi -\!\!* \varphi))$ is not valid. This shall be strengthened in the sequel by establishing that $\mathtt{SL}(*)$ is decidable whereas $\mathtt{SL}(-\!\!*)$ is not.

Let F and F′ be two fragments of SL or SO. We say that F′ is at least as expressive as F (written $\mathrm{F} \sqsubseteq \mathrm{F}'$) whenever for every sentence $\phi \in \mathrm{F}$, there is $\phi' \in \mathrm{F}'$ such that for every model (s,h), we have $(s,h) \models \phi$ iff $(s,h) \models \phi'$. We write $\mathrm{F} \equiv \mathrm{F}'$ if $\mathrm{F} \sqsubseteq \mathrm{F}'$ and $\mathrm{F}' \sqsubseteq \mathrm{F}$. A translation from F to F′ is a computable function $t : \mathrm{F} \to \mathrm{F}'$ such that for every sentence $\phi \in \mathrm{F}$, for every model (s,h), we have $(s,h) \models \phi$ iff $(s,h) \models t(\phi)$.
Arithmetical constraints. Observe that SL does not contain explicitly arithmetical constraints as in [KR03, MBCC07, BIP08]. However, in Section 4 we show how to compare number of predecessors. Similar developments can be performed to compare lengths of lists but this will come as a corollary of the equivalence between SL and SO.

Another model with data. A more realistic approach to model lists consists in considering two selectors. However, SL behaves as separation logic with two selectors for which one selector is never used. Indeed, we already know that an unrestricted use of the two selectors leads to undecidability. In the paper, we show that even SL satisfiability/validity is already undecidable. It is open how to refer to data values while preserving the decidable results for SL fragments. Possible directions consist either in imposing syntactic restrictions (like the guarded fragment for classical predicate logic) or in forbidding a direct access to data values but allowing predicates of the form "there is a list from x to y with increasing data values" for instance.

More than one selector. It is easy to extend the above definitions to the case with $k \geq 1$ selectors that is also used in the literature. A heap h becomes then a partial function $h : \mathtt{Loc} \rightharpoonup \mathtt{Loc}^k$ with finite domain and atomic formulae of the form $\mathtt{x} \hookrightarrow \mathtt{y}$ are replaced by $\mathtt{x} \hookrightarrow \mathtt{y}_1, \ldots, \mathtt{y}_k$. We write $k\mathtt{SL}$ [resp. $k\mathtt{SO}$] to denote the variant of SL [resp. SO] with k selectors. Obviously $1\mathtt{SL}$ [resp. $1\mathtt{SO}$] corresponds to SL [resp. SO]. We write $k\mathtt{SO}^{k'}$ to denote the restriction of $k\mathtt{SO}$ to second-order variables in $\mathtt{VAR}^{k'}$.

2.2 A Selection of Properties

We present below a series of properties that can be expressed in $\mathtt{SL}(*)$.

- The value of x is in the domain of the heap: $\mathtt{alloc}\,(\mathtt{x}) \triangleq \exists \mathtt{y}.\mathtt{x} \hookrightarrow \mathtt{y}$.
- The domain of the heap is restricted to the value of x, and maps it to that of y: $\mathtt{x} \mapsto \mathtt{y} \triangleq \mathtt{x} \hookrightarrow \mathtt{y} \wedge \neg\exists \mathtt{y}.\mathtt{y} \neq \mathtt{x} \wedge \mathtt{alloc}\,(\mathtt{y})$.
- The domain of the heap is empty: $\mathsf{emp} \triangleq \neg\exists \mathtt{x}.\mathtt{alloc}\,(\mathtt{x})$.

Predecessors and special nodes. A *predecessor* of the variable x in the model (s,h) is a location l such that $h(l) = s(\mathtt{x})$. There are formulae in $\mathtt{SL}(*)$, namely $\sharp\mathtt{x} \geq n$ and $\sharp\mathtt{x} = n$, such that $\sharp\mathtt{x} \geq n$ [resp. $\sharp\mathtt{x} = n$] holds true exactly in models such that x has more than n predecessors [resp. exactly n predecessors]. For instance, $\sharp\mathtt{x} \geq n$ can defined in the following ways:

$$\overbrace{(\exists \mathtt{y}.\ \mathtt{y} \hookrightarrow \mathtt{x}) * \cdots * (\exists \mathtt{y}.\ \mathtt{y} \hookrightarrow \mathtt{x})}^{n \text{ times}} * \top \quad \text{or} \quad \exists \mathtt{x}_1, \ldots, \mathtt{x}_n.\ \bigwedge_{i \neq j} \mathtt{x}_i \neq \mathtt{x}_j \wedge \bigwedge_{i=1}^{n} \mathtt{x}_i \hookrightarrow \mathtt{x}$$

We define an *extremity* as a location l in a model such that l has at least one predecessor and no predecessor of l has a precedessor. The following formula states that $s(\mathtt{x})$ is an extremity: $\mathtt{extr}(\mathtt{x}) \triangleq (\neg\exists \mathtt{y}.\ \mathtt{y} \hookrightarrow \mathtt{x} \wedge \exists \mathtt{z}.\mathtt{z} \hookrightarrow \mathtt{y}) \wedge \exists \mathtt{y}.\ \mathtt{y} \hookrightarrow \mathtt{x}$.

Reachability and list predicates. Reachability in a graph is a standard property that can be expressed in monadic second-order logic. In separation logic, very often a built-in predicate for lists is added, sometimes noted $\mathsf{ls}(\mathtt{x},\mathtt{y})$. Adapting some technique used in the graph logics [DGG07], we show below how this very predicate can be expressed in $\mathtt{SL}(*)$ as well as the reachability predicate $\mathtt{x}{\rightarrow}^{*}\mathtt{y}$.

A *cyclic list* in a model (s,h) is a non-empty finite sequence $l_1,\ldots,l_n$ $(n \geq 1)$ of locations such that $h(l_n) = l_1$ and for every $i \in \{1,\ldots,n-1\}$, $h(l_i) = l_{i+1}$. A model (s,h) is a *list segment* between $\mathtt{x}$ and $\mathtt{y}$ if there are locations $l_1,\ldots,l_n$ $(n \geq 2)$ such that $s(\mathtt{x}) = l_1$, $s(\mathtt{y}) = l_n$, $l_1 \neq l_n$, $\mathtt{dom}(h) = \{l_1,\ldots,l_{n-1}\}$, and for every $i \in \{1,\ldots,n-1\}$, $h(l_i) = l_{i+1}$. Consider the formula below

$$\mathtt{x} \xrightarrow{\circlearrowleft}{}^{+} \mathtt{y} \triangleq \sharp\mathtt{x} = 0 \wedge \mathtt{alloc}\,(\mathtt{x}) \wedge \sharp\mathtt{y} = 1 \wedge \neg\mathtt{alloc}\,(\mathtt{y}) \\ \wedge \forall \mathtt{z}.\ \mathtt{z} \neq \mathtt{y} \Rightarrow (\sharp\mathtt{z} = 1 \Rightarrow \mathtt{alloc}\,(\mathtt{z})) \wedge \forall \mathtt{z}.\ \sharp\mathtt{z} \leq 1$$

Lemma 1. *Let (s,h) be a model. $(s,h) \models \mathtt{x} \xrightarrow{\circlearrowleft}{}^{+} \mathtt{y}$ iff h is undefined for $s(\mathtt{y})$ and there are unique heaps h_1, h_2 such that $h_1 * h_2 = h$, (s,h_1) is a list segment between $\mathtt{x}$ and $\mathtt{y}$ and (s,h_2) can be decomposed uniquely as a set of cyclic lists.*

We introduce additional formulae: $\mathsf{ls}(\mathtt{x},\mathtt{y}) \triangleq \mathtt{x} \xrightarrow{\circlearrowleft}{}^{+} \mathtt{y} \wedge \neg(\mathtt{x} \xrightarrow{\circlearrowleft}{}^{+} \mathtt{y} * \neg\mathsf{emp})$, $\mathtt{x} \rightarrow^{+} \mathtt{y} \triangleq \top * \mathsf{ls}(\mathtt{x},\mathtt{y})$ and $\mathtt{x}{\rightarrow}^{*}\mathtt{y} \triangleq \mathtt{x} = \mathtt{y} \vee \mathtt{x} \rightarrow^{+} \mathtt{y}$. They express the properties below.

Lemma 2. *Let (s,h) be a model. (I) $(s,h) \models \mathsf{ls}(\mathtt{x},\mathtt{y})$ iff (s,h) is a list segment between $\mathtt{x}$ and $\mathtt{y}$. (II) $(s,h) \models \mathtt{x}{\rightarrow}^{*}\mathtt{y}$ [resp. $(s,h) \models \mathtt{x}{\rightarrow}^{+}\mathtt{y}$] iff $\mathtt{y}$ is a descendant [resp. strict descendant] of $\mathtt{x}$.*

2.3 Preliminary Translations

Before showing advanced results in the forthcoming sections, we show below how $\mathtt{SL}$ can be encoded into $\mathtt{SO}$ by simply mimicking the original semantics and how $\mathtt{SO}$ can be encoded in its fragment $\mathtt{DSO}$ by representing multiedges by finite sets of edges.

Proposition 3. $\mathtt{SL} \sqsubseteq \mathtt{SO} \equiv \mathtt{DSO}$ *via logspace translations.*

Sections 4 and 5 are devoted to prove that $\mathtt{DSO} \sqsubseteq \mathtt{SL}(\mathrel{-\!\!*})$. We will obtain that $\mathtt{SL}(\mathrel{-\!\!*})$, $\mathtt{SL}$, $\mathtt{DSO}$ and $\mathtt{SO}$ have the same expressive power (via logspace translations). Consequently, this implies undecidability of the validity problem for any of these logics by the undecidability of classical predicate logic with one binary relation [Tra50]. By contrast, we prove below that $\mathtt{SL}(*)$ is decidable.

3 On the Complexity of $\mathtt{SL}(*)$

In this section, we show that $\mathtt{SL}(*)$ satisfiability is decidable but with non-elementary recursive complexity (by reduction from first-order theory over finite words).

Lemma 4. $\mathtt{MSO}$ *satisfiability is decidable.*

The proof is based on [Rab69, BGG97]. Using a technique similar to the proof of Lemma 4, we can translate $\mathtt{SL}(*)$ into $\mathtt{MSO}$.

Proposition 5. $\mathtt{SL}(*) \sqsubseteq \mathtt{MSO}$ *via a logspace translation.*

We conjecture that $\mathtt{MSO}$ is strictly more expressive than $\mathtt{SL}(*)$, see the related paper [Mar06].

Corollary 6. $\mathtt{SL}(*)$ *satisfiability is decidable.*

In order to show that satisfiability in $\mathtt{SL}(*)$ is not elementary recursive, we explain below how to encode finite words as memory states. Let $\Sigma = \{a_1, \ldots, a_n\}$ be a finite alphabet. A finite word $w = a_{i_1} \cdot a_{i_2} \cdots a_{i_m}$ is usually represented as the first-order structure $(\{1, \ldots, m\}, <, (\mathtt{P}_a)_{a \in \Sigma})$ where $\mathtt{P}_a$ is the set of positions labelled by the letter a. Similarly, the word w can be represented as a memory state (s_w, h_w) in which

- $\mathtt{x}_{beg} \rightarrow^{+} \mathtt{x}_{end}$ for which $\mathtt{x}_{beg}$ and $\mathtt{x}_{end}$ are distinguished variables marking respectively, the beginning and the end of the encoding of w,
- the list segment induced from the satisfaction of $\mathtt{x}_{beg} \rightarrow^{+} \mathtt{x}_{end}$ has exactly $m+2$ locations, and any location l of position $j \in \{2, \ldots, m+1\}$ in the list segment (hence excluding $s_w(\mathtt{x}_{beg})$ and $s_w(\mathtt{x}_{end})$) has exactly i_j predecessors. Since $s_w(\mathtt{x}_{beg})$ and $s_w(\mathtt{x}_{end})$ do not encode any position in w, there is no constraint on them.

Similarly, any memory state (s, h) containing a list segment between $\mathtt{x}_{beg}$ and $\mathtt{x}_{end}$ and such that any location on the list segment that is different from $s(\mathtt{x}_{beg})$ and $s(\mathtt{x}_{end})$ has at most $|\Sigma|$ predecessors corresponds to a unique finite word with the above encoding. In this direction, the memory state may contain other dummy locations but they are irrelevant for the representation of the finite word. Moreover, a memory state can encode only one word since $\mathtt{x}_{beg}$ and $\mathtt{x}_{end}$ are end-markers.

Proposition 7. $\mathtt{SL}(*)$ *is not elementary recursive (even its restriction with 5 variables).*

The proof uses the predicate $\mathtt{x} \rightarrow^{+} \mathtt{y}$ from Section 2.2. As a corollary, we obtain an alternative decidability proof of the entailment problem for the fragment of $\mathtt{SL}$ considered in [BCO04]. We have established decidability for a fragment of $\mathtt{SL}$ larger than the one considered in [BCO04] (for which the entailment problem is shown in coNP) but of higher complexity.

It is probable that the number of variables can be reduced further while preserving non-elementarity, but it is not very essential at this point, for instance by identifying the limits of the words by patterns (see e.g. Section 4) instead of distinguished variables.

4 Advanced Arithmetic Constraints

In this section, we show how $\mathtt{SL}(-\!\!*)$ can express properties of the form $\widetilde{\sharp \mathtt{x}} \bowtie \widetilde{\sharp \mathtt{y}} + c$ for any $c \in \mathbb{N}$ and for any relation $\bowtie \in \{=, \geq, \leq\}$ where $\widetilde{\sharp \mathtt{x}}$ denotes the number of predecessors of $s(\mathtt{x})$ in a model. Note that $\widetilde{\sharp \mathtt{x}} \bowtie c$ can be easily expressed in $\mathtt{SL}(-\!\!*)$, even without magic wand. By contrast, expressing a constraint $\widetilde{\sharp \mathtt{x}} \bowtie \widetilde{\sharp \mathtt{y}} + c$ is natural in

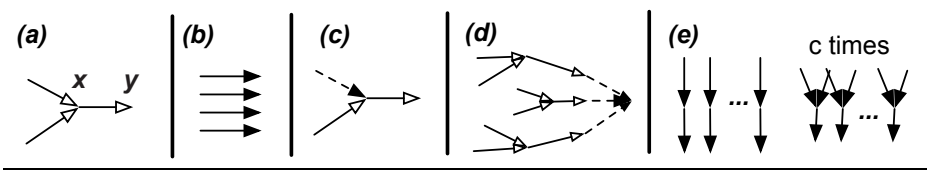


Fig. 1. Some examples of markers, marked locations, and possible aliasing

second-order logic. We show below that this can be done also in $\mathtt{SL}(-\!\!*)$. In the sequel, we assume that the current model is denoted by (s, h).

A *marker* in a model (s, h) is a specific pattern in the memory heap that we will intensively use. Formally, a *[resp. strict] marker* in (s, h) is a sequence of distinct locations $l, l_0, \ldots, l_n$ for some $n \geq 0$ such that (a) $h(l_0) = l$ [resp. and $\mathtt{dom}(h) = \{l_0, \ldots, l_n\}$], (b) for $i \in \{1, \ldots, n\}$, $h(l_i) = l_0$ and $\widetilde{\sharp l_i} = 0$ and (c) $\widetilde{\sharp l_0} = n$. We say that this marker is of *degree* n with *endpoint* l. Markers should be thought graphically. Figure 1(a) represents a marker of degree 2 with endpoint $s(\mathtt{y})$.

A model (s, h) is said to be *k-marked* whenever there is no location in $\mathtt{dom}(h)$ that does not belong to a marker of degree k. Moreover it is *strictly* k-marked when no distinct markers share the same endpoint. A model (s, h) is *segmented* whenever $\mathtt{dom}(h) \cap \mathtt{ran}(h) = \emptyset$ and no location has strictly more than one predecessor. Finally, (s, h) is *drown* when no location has one or two predecessors. The lemma below states that many properties involving these notions can be expressed in $\mathtt{SL}(-\!\!*)$.

Lemma 8. *There are formulae* drown, $\mathtt{seg}$, $\mathtt{preM}^2(\mathtt{x})$ *and* $(\mathrm{one} \mid \mathrm{two}[c])$ *($c \geq 0$) in* $\mathtt{SL}$ *with no separating connectives such that for every model* (s, h)

(I) $(s, h) \models \mathsf{drown}$ *iff* (s, h) *is drown,* **(II)** $(s, h) \models \mathtt{seg}$ *iff* (s, h) *is segmented,*
(III) $(s, h) \models \mathtt{preM}^2(\mathtt{x})$ *iff the predecessors of* $s(\mathtt{x})$ *are endpoints of markers of degree 2,*
(IV) $(s, h) \models (\mathrm{one} \mid \mathrm{two}[c])$ *iff there are* h_1, h_2 *such that* $h = h_1 * h_2$, (s, h_1) *is 1-marked and* (s, h_2) *is strictly 2-marked with exactly* c *distinct 2-markers.*

Figure 1(e) contains a model satisfying $(\mathrm{one} \mid \mathrm{two}[c])$ whereas Figure 1(d) presents a location $s(\mathtt{x})$ satisfying $\mathtt{preM}^2(\mathtt{x})$. The above formulae allow to insert in a drown model markers of degree strictly less than 3 and still to safely identify them as markers in the new model. Actually, any location with less than 2 predecessors will necessarily be part of a newly introduced marker. We will assume for now that we are working with a drown model, and later reduce the general case to this one. Observe also if h_1 is segmented and h_2 contains 1-markers, we can obtain 2-markers in $h_1 * h_2$.

We say that two heaps h_1, h_2 are *completely disjoint* if $(\mathtt{dom}(h_1) \cup \mathtt{ran}(h_1)) \cap (\mathtt{dom}(h_2) \cup \mathtt{ran}(h_2)) = \emptyset$. Now assume that h_1 is segmented with $\mid \mathtt{dom}(h_1) \mid = n$, h_2 is drown and, h_1 and h_2 are completely disjoint. Then, there is a 1-marked heap h_1' such that all the predecessors of $s(\mathtt{x})$ are endpoints of 2-markers iff $\widetilde{\sharp \mathtt{x}} \leq n$ (in h_2). If we have the possibility to add to h_1' a strict 2-marked heap with exactly c distinct 2-markers then all the predecessors of $s(\mathtt{x})$ are endpoints of 2-markers iff $\sharp \mathtt{x} \leq c + n$. This is formalized below.

Lemma 9. *Let s be a store and h_1, h_2 be two completely disjoint heaps such that $(s, h_1) \vDash \mathsf{drown} \wedge \sharp\mathtt{x} = i$ and $(s, h_2) \vDash \mathtt{seg} \wedge \sharp\mathtt{x} = 0$. Then, (i) $(s, h_1 * h_2) \models (\mathrm{one} \mid \mathrm{two}[c]) \mathrel{-\!\!\!\!\!\!\exists\!\!*} \mathtt{preM}^2(\mathtt{x})$ iff (ii) $\mid \mathtt{dom}(h_2) \mid \geq (i - c)$.*

So we may test that the number of extra arrows that satisfy $\mathtt{seg}$ is less than $\widetilde{\sharp\mathtt{x}}$. Call n the number of these extra arrows, then this scenario says $\widetilde{\sharp\mathtt{x}} - c \leq n$. Now, in order to express properties of the form $\widetilde{\sharp\mathtt{x}} \bowtie \widetilde{\sharp\mathtt{y}} + c$, it is sufficient to express properties of the form $\widetilde{\sharp\mathtt{x}} + c \leq \widetilde{\sharp\mathtt{y}} + c'$ $(c, c' \in \mathbb{N})$ thanks to Boolean connectives. Note moreover that this constraint is equivalent to: for all $n \in \mathbb{N}$, $\widetilde{\sharp\mathtt{y}} - c \leq n$ implies $\widetilde{\sharp\mathtt{x}} - c' \leq n$. This suggests a contest between two players: Spoiler that aims at disproving that the constraint holds, and Duplicator tries to prove it. The contest is the following:

1. Spoiler reduces to the case of a drown model - this will be formalized later.
2. Spoiler picks a segmented heap h' with $\mid \mathtt{dom}(h') \mid = n$.
3. He proves that $\widetilde{\sharp\mathtt{y}} - c \leq n$ using the previous scenario.
4. Then Duplicator must prove $\widetilde{\sharp\mathtt{x}} - c' \leq n$ playing with the same scenario.

Figure 2 summarizes a contest with a successful outcome for Duplicator.

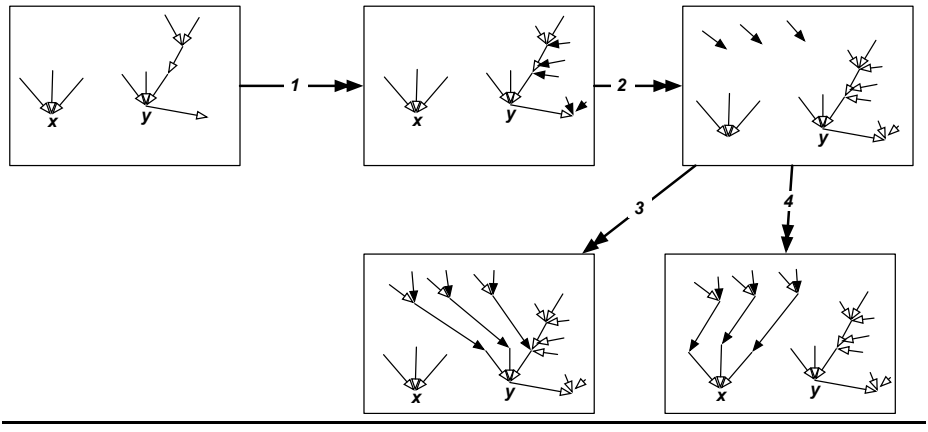


Fig. 2. A contest won by Duplicator. $n = 3$, $c = c' = 0$

Let φ_{TD} be the formula that specifies that if a model is not drown then it is only due to the fact that it contains a segmented subheap that is not drown:

$$(\forall \mathtt{x}, \mathtt{y}.\ (\mathtt{x} \hookrightarrow \mathtt{y} \wedge \sharp\mathtt{y} = 1) \Rightarrow (\sharp\mathtt{x} = 0 \wedge \neg\mathtt{alloc}\ (\mathtt{y}))) \wedge (\neg(\exists \mathtt{x}.\ \sharp\mathtt{x} = 2))$$

The formula $\mathsf{contest}(\mathtt{x}, \mathtt{y}, c, c')$ defines a contest (essential to establish Theorem 10).

$$\mathsf{drown} \wedge ((\mathtt{seg} \wedge \sharp\mathtt{x} = 0 \wedge \sharp\mathtt{y} = 0) \mathrel{-\!\!*} \varphi_{TD} \Rightarrow (((\mathrm{one} \mid \mathrm{two}[c]) \mathrel{-\!\!\!\!\!\!\exists\!\!*} \mathtt{preM}^2(\mathtt{y})) \Rightarrow ((\mathrm{one} \mid \mathrm{two}[c]) \mathrel{-\!\!\!\!\!\!\exists\!\!*} \mathtt{preM}^2(\mathtt{x}))).$$

Theorem 10. *For $c, c' \geq 0$, there is a formula ϕ in $\mathtt{SL}(-\!\!*)$ of quadratic size in $c + c'$ such that for every model (s, h), we have $(s, h) \models \phi$ iff $\widetilde{\sharp \mathtt{x}} + c \leq \widetilde{\sharp \mathtt{y}} + c'$.*

The proof contains a simple case analysis depending whether the model can be transformed into a drown one (in order to use $\mathtt{contest}(\mathtt{x}, \mathtt{y}, c, c')$) without altering the number of precedessors for $s(\mathtt{x})$ and $s(\mathtt{y})$. In Section 5, constraints $\widetilde{\sharp \mathtt{x}} + c \leq \widetilde{\sharp \mathtt{y}} + c'$ with $c, c' \leq 3$ shall be used.

5 $\mathtt{SL}(-\!\!*)$ Is Equivalent to $\mathtt{SO}$

By Proposition 3, we know that $\mathtt{SL} \sqsubseteq \mathtt{DSO}$ and there is a logspace translation from $\mathtt{SL}$ into $\mathtt{DSO}$ (logspace reductions are closed under compositions). Now, we show the converse. In the sequel, without any loss of generality we require that the sentences in $\mathtt{DSO}$ satisfy the Barendregt convention as far as the second-order variables are concerned. Assuming that a sentence contains the second-order variables $\mathtt{P}_1, \ldots, \mathtt{P}_n$, quantifications over $\mathtt{P}_i$ can only occur in the scope of quantifications over $\mathtt{P}_1$, ..., $\mathtt{P}_{i-1}$ (we call this restriction the *extended Barendregt convention*). Typically, we exclude sentences of the form $\exists \mathtt{P}_2\ \exists \mathtt{P}_1\ \phi$. Observe that any sentence in $\mathtt{DSO}$ can be transformed in logspace into an equivalent sentence verifying this convention. The *quantifier depth* of the occurrence of a subformula ψ in ϕ is therefore the maximal i such that this occurrence is in the scope of $\exists \mathtt{P}_i$ (by convention it is zero if it is not in the scope of any quantification).

Before defining the translation of a $\mathtt{DSO}$ sentence ϕ (with second-order variables $\mathtt{P}_1, \ldots, \mathtt{P}_n$), let us explain how environments can be encoded in $\mathtt{SL}$ which is the key point to simulate second-order quantification. An environment is encoded as part of the memory heap with a specific shape. The interpretation of each variable $\mathtt{P}_i$ is performed as follows. A pair (l, l') is in the interpretation of $\mathtt{P}_i$ iff there are markers with respective endpoint l and l' whose degrees are *consecutive* values strictly between some fixed values $\widetilde{\sharp \mathtt{z}_i^m}$ and $\widetilde{\sharp \mathtt{z}_i^M}$. Here, the distinguished variables $\mathtt{z}_i^m$ and $\mathtt{z}_i^M$ are interpreted as locations that are not in the domain of the original memory heap. In order to avoid confusions between the original memory heap and the part that is dedicated to the encoding of the environment, $\widetilde{\sharp \mathtt{z}_i^m}$ is strictly greater than the degree of any location in the original memory heap. In other words, instead of making the original model drown and using small markers, the model is unchanged and large markers are used.

The translation of the formula ϕ, written $T(\phi)$, is defined with the help of the translation t_j where j records the quantifier depth.

$$T(\phi) \triangleq \exists \mathtt{z}_0^m\ \mathtt{z}_0^M .\ \mathtt{isol}(\mathtt{z}_0^M) \wedge \mathtt{isol}(\mathtt{z}_0^m) \wedge$$

$$[((\forall \mathtt{x}.\ \mathtt{alloc}\,(\mathtt{x}) \Rightarrow (\mathtt{x} \hookrightarrow \mathtt{z}_0^M \vee \mathtt{x} \hookrightarrow \mathtt{z}_0^m \vee \mathtt{x} = \mathtt{z}_0^M \vee \mathtt{x} = \mathtt{z}_0^m)) \wedge \mathtt{alloc}\,(\mathtt{z}_0^M) \wedge \mathtt{alloc}\,(\mathtt{z}_0^m)) \mathrel{\overrightarrow{-\!\!*}}$$

$$(\forall \mathtt{x}.\mathtt{x} \neq \mathtt{z}_0^M \wedge \mathtt{x} \neq \mathtt{z}_0^m \Rightarrow (\sharp \mathtt{z}_0^m > 2 + \sharp \mathtt{x})) \wedge (\sharp \mathtt{z}_0^M = 2 + \sharp \mathtt{z}_0^m) \wedge \mathtt{extr}(\mathtt{z}_0^m) \wedge \mathtt{extr}(\mathtt{z}_0^M) \wedge t_0(\phi))]$$

The formula $\mathtt{isol}(\mathtt{x})$ is an abbreviation for $\neg \exists \mathtt{y}.\ (\mathtt{x} \hookrightarrow \mathtt{y}) \vee (\mathtt{y} \hookrightarrow \mathtt{x})$. Any location with number of predecessors greater than $\widetilde{\sharp \mathtt{z}_0^M}$ is useful to encode a pair of locations for the interpretation of some second-order variable (except the interpretation of

distinguished variables of the form either $\mathtt{z}_i^m$ or $\mathtt{z}_i^M$). The translation of the atomic formula $\mathtt{P}_j(\mathtt{x},\mathtt{y})$ in the scope of the quantifications over $\mathtt{P}_1,\ldots,\mathtt{P}_i$ ($j \leq i$) is defined below:

$$t_i(\mathtt{P}_j(\mathtt{x},\mathtt{y})) \triangleq \exists \mathtt{z},\mathtt{z}'\,(\mathtt{z} \hookrightarrow \mathtt{x}) \wedge (\mathtt{z}' \hookrightarrow \mathtt{y}) \wedge (\sharp\mathtt{z} > \sharp\mathtt{z}_j^m) \wedge (\sharp\mathtt{z} < \sharp\mathtt{z}_j^M) \wedge (\sharp\mathtt{z}' = 1{+}\sharp\mathtt{z}) \wedge \mathtt{extr}(\mathtt{z}) \wedge \mathtt{extr}(\mathtt{z}')$$

So $(s(\mathtt{x}), s(\mathtt{y}))$ belongs to the interpretation of $\mathtt{P}_j$ when $s(\mathtt{x})$ and $s(\mathtt{y})$ are endpoints of markers with consecutive degrees between $\widetilde{\sharp\mathtt{z}_j^m}$ and $\widetilde{\sharp\mathtt{z}_j^M}$. However, in order to avoid interferences between the encoding of the environment and the original memory heap, we require that the new memory heap satisfies structural properties described below.

Proposition 11. *There is a formula* $\mathtt{envir}(\mathtt{z},\mathtt{z}')$ *such that* $(s,h) \models \mathtt{envir}$ *iff* $\widetilde{\sharp\mathtt{z}} < \widetilde{\sharp\mathtt{z}'}$, $\widetilde{\sharp\mathtt{z}} \equiv \widetilde{\sharp\mathtt{z}'} + 2\ [3]$ *and for all* i *in* $[\widetilde{\sharp\mathtt{z}},\ldots,\widetilde{\sharp\mathtt{z}'}]$,

- *if* $i \equiv \widetilde{\sharp\mathtt{z}} + 1\ [3]$ *then there is no extremity* l *in* (s,h) *such that* $\widetilde{\sharp l} = i$,
- *if* $i \not\equiv \widetilde{\sharp\mathtt{z}} + 1\ [3]$ *then there is exactly one location* l *such that* l *is an extremity and* $\widetilde{\sharp l} = i$. *This unique location* l *belongs to* $\mathtt{dom}(h)$.

An i-well-formed model, defined below, can be divided into two disjoint parts such that one part encodes the interpretation of the second-order variables $\mathtt{P}_1,\ldots,\mathtt{P}_i$.

Definition 12. *A memory state* (s,h) *is* i-well-formed *for some* $i \geq 0$ *iff there are heaps* h_1, h_2 *with* $h = h_1 * h_2$ *satisfying the properties below:*

(I) *for every variable* $\mathtt{x}$ *in* $\{\mathtt{z}_1^m,\ldots,\mathtt{z}_i^m\} \cup \{\mathtt{z}_0^M,\ldots,\mathtt{z}_{i-1}^M\}$, $s(\mathtt{x})$ *is an extremity in* (s,h) *and* $\widetilde{\sharp\mathtt{z}_0^m} < \widetilde{\sharp\mathtt{x}} < \widetilde{\sharp\mathtt{z}_i^M}$,
(II) *when* $i \geq j > 0$, $\widetilde{\sharp\mathtt{z}_{j-1}^M} + 1 = \widetilde{\sharp\mathtt{z}_j^m}$,
(III) *there is no location* l *such that* $\widetilde{\sharp l}$ *in* (s,h_1) *is strictly greater than* $\widetilde{\sharp\mathtt{z}_0^m} - 2$ *in* (s,h),
(IV) *if* $l \in \mathtt{dom}(h_2)$ *then there is* $l' \in \{l, h_2(l)\}$ *such that* l' *is an extremity in* h_2,
(V) *any extremity* l *in the model* (s,h_2) *satisfies (1)* $l \notin \mathtt{ran}(h_1)$, *(2)* l *is an extremity in* (s,h), *(3)* $\widetilde{\sharp\mathtt{z}_0^m} \leq \widetilde{\sharp l} \leq \widetilde{\sharp\mathtt{z}_i^M}$ *and (4)* $h(l) \notin \mathtt{dom}(h_2)$.
(VI) $(s,h_2) \models \mathtt{envir}(\mathtt{z}_0^m, \mathtt{z}_i^M)$.

(s,h_1) *is called the* base part *and* (s,h_2) *the* environment part.

Note that in any such decomposition, we have $\mathtt{dom}(h_2) \cap \mathtt{ran}(h_1) = \emptyset$. Moreover, any extremity in h with more than $\widetilde{\sharp\mathtt{z}_0^m}$ predecessors has all predecessors in $\mathtt{dom}(h_2)$. The above decomposition is indeed unique.

Lemma 13. *Whenever* (s,h) *is i-well-formed with base part* h_1 *and environment part* h_2, *there is no* $(h_1', h_2') \neq (h_1, h_2)$ *such that* (s,h) *is i-well-formed with base part* h_1' *and environment part* h_2'.

The *spectrum* of an i-well-formed memory state is defined as the set of numbers of predecessors in the environment part that are greater than 3. Hence such a spectrum has the following shape ● ○ ● ● ○ ● ● ⋯ ○ ● ● ○ ● ● ○ ● ● ⋯ ○ ● ● ○● corresponding

to a finite sequence of successive integers where $\bullet$ indicates the presence of the integer and $\circ$ its absence. The smallest value is $\widetilde{\sharp \mathtt{z}_0^m}$ and the greatest value is $\widetilde{\sharp \mathtt{z}_i^M}$. Two consecutive values in the sequence encode one pair from the interpretation of a second-order variable. Observe that the concatenation of two spectra is still a spectrum. The formula $\mathtt{relation}_{i,X}$ defined below states that part of the memory is 0-well-formed and can serve to encode the interpretation of the variable $\mathtt{P}_i$.

Proposition 14. *Given $i \geq 0$ and X a finite set of variables disjoint from the set of auxiliary variables $\{\mathtt{z}_0^m, \mathtt{z}_0^M, \ldots, \mathtt{z}_i^m, \mathtt{z}_i^M\}$, there is a formula $\mathtt{relation}_{i,X}$ such that for every model (s,h), we have $(s,h) \models \mathtt{relation}_{i,X}$ iff $(s[\mathtt{z}_0^m \mapsto s(\mathtt{z}_i^m), \mathtt{z}_0^M \mapsto s(\mathtt{z}_i^M)], h)$ is 0-well-formed, its base part is empty and for every $\mathtt{x} \in X$, $s(\mathtt{x}) \notin \mathtt{dom}(h)$.*

In order to translate the subformula $\exists \mathtt{P}_i.\ \psi$, we introduce two locations whose numbers of predecessors determine the bounds for the degrees for any marker used to encode a pair for the interpretation of $\mathtt{P}_i$. There is a way to add markers (expressed thanks to the connective $\mathrel{-\!\!*}$) that guarantees that the new part of the heap encodes the interpretation of the variable $\mathtt{P}_i$ by using the above formula $\mathtt{relation}_{i,X}$. The translation of $\exists \mathtt{P}_i.\ \psi$ at the $(i-1)$ quantification depth, noted $t_{i-1}(\exists \mathtt{P}_i, \psi)$, is defined by $\exists \mathtt{z}_i^m, \mathtt{z}_i^M.\ \mathtt{isol}(\mathtt{z}_i^m) \wedge \mathtt{isol}(\mathtt{z}_i^M) \wedge (\mathtt{relation}_{i,\mathtt{FV}(\psi)} \mathrel{-\!\!*} (\mathtt{envir}(\mathtt{z}_0^m, \mathtt{z}_i^M) \wedge \sharp \mathtt{z}_{i-1}^M + 1 = \sharp \mathtt{z}_i^m \wedge t_i(\psi)))$.

Definition 15. *Let (s,h) be an i-well-formed model with environment part (s,h_2). An environment $\mathcal{E}$* extracted from *(s,h) satisfies for $j \in \{1,\ldots,i\}$, $\mathcal{E}(\mathtt{P}_j) = \mathcal{R}_j$ with $\mathcal{R}_j = \{(h_2(l), h_2(l')) :\ \widetilde{\sharp \mathtt{z}_j^m} < \widetilde{\sharp l},\ \widetilde{\sharp l} + 1 = \widetilde{\sharp l'},\ \widetilde{\sharp l'} < \widetilde{\sharp \mathtt{z}_j^M}$ in $h_2\}$.*

The map t_i is homomorphic for Boolean connectives, and is the identity for atomic formulae of the form either $\mathtt{x} = \mathtt{y}$ or $\mathtt{x} \hookrightarrow \mathtt{y}$. It remains to treat the case for first-order quantification. The main difficulty is to guarantee that first-order variables are not interpreted as locations used in markers encoding second-order quantification. Typically, the number of predecessors of $s(\mathtt{x})$ and $h(s(\mathtt{x}))$ (if its exists) should be less than $\widetilde{\sharp \mathtt{z}_0^M}$ and none of these locations is an extremity. The formula $\mathtt{notonenv}$ is introduced for this purpose: $\mathtt{notonenv}(\mathtt{x}) \triangleq \neg(\exists \mathtt{y}.\ (\mathtt{y} = \mathtt{x} \vee \mathtt{x} \hookrightarrow \mathtt{y}) \wedge (\sharp \mathtt{y} \geq \sharp \mathtt{z}_0^m) \wedge \mathtt{extr}(\mathtt{y}))$. The main reason for introducing $\mathtt{notonenv}(\mathtt{x})$ is to be able to identify locations from the environment part of i-well-formed models, as stated below.

Lemma 16. *Let (s,h) be an i-well-formed with environment part h_2. Then $(s,h) \models \mathtt{notonenv}(\mathtt{x})$ iff $s(\mathtt{x}) \notin \mathtt{dom}(h_2)$.*

The translation $t_i(\exists \mathtt{x}.\ \psi)$ is defined as $\exists \mathtt{x}.\ \mathtt{notonenv}(\mathtt{x}) \wedge t_i(\psi)$. Observe that $T(\phi)$ and ϕ have the same first-order free variables. Correctness of $T(\cdot)$ is based on Proposition 17.

Proposition 17. *Let ϕ be a $\mathtt{DSO}$ formula with second-order variables $\{\mathtt{P}_1, \ldots, \mathtt{P}_n\}$ using the extended Barendregt convention. Let ψ be a subformula of ϕ which occurs under the scope of $\mathtt{P}_1, \ldots, \mathtt{P}_j$ ($0 \leq j \leq n$) with quantified second-order variables in $\{\mathtt{P}_{j+1}, \ldots, \mathtt{P}_n\}$. Let (s,h) be a j-well-formed model with base part h_1 and environment part h_2 such that for each $\mathtt{x} \in \mathtt{FV}(\psi)$, $s(\mathtt{x}) \notin \mathtt{dom}(h_2)$. Let $\mathcal{E}_j$ be an environment extracted from (s,h) ($\{\mathtt{P}_1 \mapsto \mathcal{R}_1, \ldots, \mathtt{P}_j \mapsto \mathcal{R}_j\}$). Then, $(s,h) \models t_j(\psi)$ iff $(s,h_1), \mathcal{E}_j \models \psi$.*

Full proof of Proposition 17 can be found in [BDL08]. We present below simple cases in the analysis. This will entail our main result (Theorem 18).

Proof (sketch with simple cases)**.** Let us start by a preliminary definition. We say that a location l *occurs* in a binary relation $\mathcal{R}$ when there is a location l' such that either $(l,l') \in \mathcal{R}$ or $(l',l) \in \mathcal{R}$. Let ϕ be a $\mathtt{DSO}$ sentence satisfying the extended Barendregt convention. We want to show by induction on ψ that given:

- ψ is a subformula of ϕ which occurs under the scope of $\mathtt{P}_1, \ldots, \mathtt{P}_j$, and with second-order quantifications over elements from $\{\mathtt{P}_{j+1}, \ldots, \mathtt{P}_n\}$,
- (s,h) is j-well-formed with base part h_1 and environment part h_2 such that for every variable $\mathtt{x} \in \mathtt{FV}(\psi)$, we have $s(\mathtt{x}) \notin \mathtt{dom}(h_2)$,
- $\mathcal{E}_j$ is the environment $\{\mathtt{P}_1 \mapsto \mathcal{R}_1, \ldots, \mathtt{P}_j \mapsto \mathcal{R}_j\}$ extracted from h_2,
- no location occurring in $\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_j$ belongs to $\mathtt{dom}(h_2)$,

we have $(s,h) \models t_j(\psi)$ iff $(s,h_1), \mathcal{E}_j \models \psi$.

Base case 1: $\psi = \mathtt{P}_k(\mathtt{x},\mathtt{y})$ with $k \leq j$.
($\rightarrow$) Suppose that $(s,h) \models t_j(\mathtt{P}_k(\mathtt{x},\mathtt{y}))$. Then, $s(\mathtt{x})$ and $s(\mathtt{y})$ have predecessors in h that are extremities, let us call them respectively $l_{\mathtt{x}}$ and $l_{\mathtt{y}}$. In the heap h, we have $\widetilde{\sharp \mathtt{z}_k^m} < \widetilde{\sharp l_{\mathtt{x}}} = \widetilde{\sharp l_{\mathtt{y}}} - 1 < \widetilde{\sharp \mathtt{z}_k^M} - 1$. By Definition 12, both $l_{\mathtt{x}}$ and $l_{\mathtt{y}}$ have predecessors in $\mathtt{dom}(h_2)$ and all of their predecessors are also in $\mathtt{dom}(h_2)$. Since $\mathtt{z}_k^m$ and $\mathtt{z}_k^M$ have also all of their predecessors in $\mathtt{dom}(h_2)$, we have $\widetilde{\sharp \mathtt{z}_k^m} < \widetilde{\sharp l_{\mathtt{x}}}$, $\widetilde{\sharp l_{\mathtt{x}}} + 1 = \widetilde{\sharp l_{\mathtt{y}}}$ and $\widetilde{\sharp l_{\mathtt{y}}} < \widetilde{\sharp \mathtt{z}_k^M}$ in h_2. By Definition 15, we get $(h(l_{\mathtt{x}}), h(l_{\mathtt{y}})) \in \mathcal{R}_k$, that is $(s(\mathtt{x}), s(\mathtt{y})) \in \mathcal{R}_k$. Consequently, $(s,h_1), \mathcal{E}_j \models \mathtt{P}_k(\mathtt{x},\mathtt{y})$.

($\leftarrow$) Suppose that $(s,h_1), \mathcal{E}_j \models \mathtt{P}_k(\mathtt{x},\mathtt{y})$. By definition of $\models$ and $\mathcal{E}_j$, $(s(\mathtt{x}), s(\mathtt{y})) \in \mathcal{R}_k$. So $s(\mathtt{x})$ and $s(\mathtt{y})$ have respectively predecessors $l_{\mathtt{x}}$ and $l_{\mathtt{y}}$ in $\mathtt{dom}(h_2)$. In the heap h_2, $l_{\mathtt{x}}$ and $l_{\mathtt{y}}$ are extremities and $\widetilde{\sharp \mathtt{z}_k^m} < \widetilde{\sharp l_{\mathtt{x}}} = \widetilde{\sharp l_{\mathtt{y}}} - 1 < \widetilde{\sharp \mathtt{z}_k^M} - 1$. By Definition 12, the predecessors of any locations among $s(\mathtt{z}_k^m)$, $l_{\mathtt{x}}$, $l_{\mathtt{y}}$ and $s(\mathtt{z}_k^M)$ belong to $\mathtt{dom}(h_2)$. So the above inequalities and equality are also true in h. By Definition 12, the locations $s(\mathtt{z}_k^m)$, $l_{\mathtt{x}}$, $l_{\mathtt{y}}$ and $s(\mathtt{z}_k^M)$ are extremities in h. So $(s,h) \models t_j(\mathtt{P}_k(\mathtt{x},\mathtt{y}))$.
The other base cases are by an easy verification.

Induction step – Case 1: $\psi = \exists \mathtt{x}.\ \psi'$. The statements below are equivalent:

(0) $(s,h) \models t_j(\exists \mathtt{x}\ \psi')$,
(1) there is $l \in \mathtt{Loc}$ such that $(s',h) \models t_j(\psi')$ and $(s',h) \models \mathtt{notonenv}(\mathtt{x})$ with $s' = s[\mathtt{x} \mapsto l]$ (by definition of t_j),
(2) there is $l \in \mathtt{Loc}$ such that $(s',h) \models t_j(\psi')$ and $l \notin \mathtt{dom}(h_2)$ with $s' = s[\mathtt{x} \mapsto l]$ (by Lemma 16),
(3) there is $l \in \mathtt{Loc}$ such that $(s',h_1), \mathcal{E}_j \models \psi'$ and $l \notin \mathtt{dom}(h_2)$ with $s' = s[\mathtt{x} \mapsto l]$ (by induction hypothesis since $\mathtt{FV}(\psi') \subseteq \mathtt{FV}(\exists \mathtt{x}.\ \psi') \cup \{\mathtt{x}\}$),
(4) there is $l \in \mathtt{Loc}$ such that $(s',h_1), \mathcal{E}_j \models \psi'$ with $s' = s[\mathtt{x} \mapsto l]$,
(5) $(s,h_1), \mathcal{E}_j \models \psi$ (by definition of $\models$).

Let us justify below why (4) implies (3). Suppose (4) and $l \in \mathtt{dom}(h_2)$. Since (s,h) is i-well-formed, $l \notin (\mathtt{dom}(h_1) \cup \mathtt{ran}(h_1))$. Since $\mathtt{Loc}$ is an infinite set, there is a location $l' \in (\mathtt{Loc} \setminus (\mathtt{dom}(h_1) \cup \mathtt{ran}(h_1) \cup \mathtt{dom}(h_2))$ such that l' does not occur in $(\mathcal{R}_1 \cup \cdots \cup \mathcal{R}_j)$. By equivariance shown in [BDL08], we get $(s[\mathtt{x} \mapsto l'], h_1), \mathcal{E}_j[l \leftarrow l'] \models \psi'$. Suppose

ad absurdum that l occurs in $\mathcal{R}_k$ for some $1 \leq k \leq j$. So, l has a predecessor that is an extremity in $\mathtt{dom}(h_2)$ and by Definition 12(V(4)), $l \notin \mathtt{dom}(h_2)$, which leads to a contradiction. Hence, $\mathcal{E}_j[l \leftarrow l'] = \mathcal{E}_j$. We have established that $(s[\mathtt{x} \mapsto l'], h_1), \mathcal{E}_j \models \psi'$ and $l' \notin \mathtt{dom}(h_2)$. □

Theorem 18. $\mathtt{SL}(-\!\!*) \equiv \mathtt{SL} \equiv \mathtt{SO} \equiv \mathtt{DSO}$.

$\mathtt{DSO} \sqsubseteq \mathtt{SL}$ stems from Proposition 17. Observe that all the equivalences are obtained with logspace translations. Consequently,

Corollary 19. $\mathtt{SL}(-\!\!*)$ *validity problem is undecidable.*

Undecidability of $\mathtt{SL}(-\!\!*)$ can be obtained more easily by encoding the halting problem for Minsky machines by using the fact that $\sharp\mathtt{x} = \sharp\mathtt{y}$ and $\sharp\mathtt{x} = \sharp\mathtt{y} + 1$ can be expressed in $\mathtt{SL}(-\!\!*)$ (Section 4).

6 Concluding Remarks

We have shown that first-order separation logic with one selector is as expressive as second-order logic over the class of memory states, whence undecidable too. Moreover, the restriction to the magic wand preserves the expressive power. This solves two central open problems: the decidability status of SL and the characterization of its expressive power. Additionnally, we have proved that SL without the magic wand is decidable with non-elementary complexity whereas SL restricted to the magic wand is also undecidable. By adapting the above developments, a generalization in presence of $k > 1$ selectors (see Section 2.1) is possible.

Theorem 20. *For every* $k \geq 1$, $k\mathtt{SL} \equiv k\mathtt{SL}(-\!\!*) \equiv k\mathtt{SO}$.

The case $k = 1$ requires a lot of care but a simpler direct proof is possible for $k \neq 1$. Indeed, for $k = 1$ the identification of auxiliary memory cells is performed thanks to structural properties whereas for $k \neq 1$, this can be done by simply checking the presence of distinguished values.

References

[BCO04] Berdine, J., Calcagno, C., O'Hearn, P.: A decidable fragment of separation logic. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 97–109. Springer, Heidelberg (2004)

[BDL07] Brochenin, R., Demri, S., Lozes, E.: Reasoning about sequences of memory states. In: Artemov, S.N., Nerode, A. (eds.) LFCS 2007. LNCS, vol. 4514, pp. 100–114. Springer, Heidelberg (2007)

[BDL08] Brochenin, R., Demri, S., Lozes, E.: On the almighty wand. Technical report, LSV, ENS de Cachan (2008)

[BGG97] Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic. Springer, Heidelberg (1997)

[BHMV05] Bouajjani, A., Habermehl, P., Moro, P., Vojnar, T.: Verifying programs with dynamic 1-selector-linked structured in regular model-checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 13–29. Springer, Heidelberg (2005)

[BIL04] Bozga, M., Iosif, R., Lakhnech, Y.: On logics of aliasing. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 344–360. Springer, Heidelberg (2004)

[BIP08] Bozga, M., Iosif, R., Perarnau, S.: Quantitative separation logic and programs with lists. In: IJCAR 2008 (to appear, 2008)

[CGH05] Calcagno, C., Gardner, P., Hague, M.: From separation logic to first-order logic. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 395–409. Springer, Heidelberg (2005)

[CYO01a] Calcagno, C., Yang, H., O'Hearn, P.: Computability and complexity results for a spatial assertion language. In: APLAS 2001, pp. 289–300 (2001)

[CYO01b] Calcagno, C., Yang, H., O'Hearn, P.: 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)

[DGG04] Dawar, A., Gardner, P., Ghelli, G.: Adjunct elimination through games in static ambient logic. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 211–223. Springer, Heidelberg (2004)

[DGG07] Dawar, A., Gardner, P., Ghelli, G.: Expressiveness and complexity of graph logic. I & C 205(3), 263–310 (2007)

[GM08] Galmiche, D., Méry, D.: Tableaux and resource graphs for separation logic (submitted, 2008)

[IO01] Ishtiaq, S., O'Hearn, P.: BI as an assertion language for mutable data structures. In: POPL 2001, pp. 14–26 (2001)

[JJKS97] Jensen, J., Jorgensen, M., Klarlund, N., Schwartzbach, M.: Automatic verification of pointer programs using monadic second-order logic. In: PLDI 1997, pp. 226–236. ACM, New York (1997)

[KR03] Klaedtke, F., Rueb, H.: Monadic second-order logics with cardinalities. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 681–696. Springer, Heidelberg (2003)

[KR04] Kuncak, V., Rinard, M.: On spatial conjunction as second-order logic. Technical report, MIT CSAIL (October 2004)

[LAS00] Lev-Ami, T., Sagiv, M.: TVLA: A system for implementing static analyses. In: Palsberg, J. (ed.) SAS 2000. LNCS, vol. 1824, pp. 280–302. Springer, Heidelberg (2000)

[Loz04] Lozes, E.: Separation logic preserves the expressive power of classical logic. In: 2nd Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE 2004) (2004)

[Loz05] Lozes, E.: Elimination of spatial connectives in static spatial logics. TCS 330(3), 475–499 (2005)

[LR03] Löding, C., Rohde, P.: Model checking and satisfiability for sabotage modal logic. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 302–313. Springer, Heidelberg (2003)

[Mar06] Marcinkowski, J.: On the expressive power of graph logic. In: Ésik, Z. (ed.) CSL 2006. LNCS, vol. 4207, pp. 486–500. Springer, Heidelberg (2006)

[MBCC07] Magill, S., Berdine, J., Clarke, E., Cook, B.: Arithmetic strengthening for shape analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 419–436. Springer, Heidelberg (2007)

[Rab69] Rabin, M.: Decidability of second-order theories and automata on infinite trees. Transactions of the American Mathematical Society 41, 1–35 (1969)

[Rey02] Reynolds, J.C.: Separation logic: a logic for shared mutable data structures. In: LICS 2002, pp. 55–74. IEEE, Los Alamitos (2002)

[RZ06] Ranise, S., Zarba, C.: A theory of singly-linked lists and its extensible decision procedure. In: SEFM 2006, pp. 206–215. IEEE, Los Alamitos (2006)

[Sto74] Stockmeyer, L.: The complexity of decision problems in automata theory and logic. PhD thesis, Department of Electrical Engineering. MIT (1974)

[Tra50] Trakhtenbrot, B.A.: The impossibility of an algorithm for the decision problem for finite models. Dokl. Akad. Nauk SSSR 70, 566–572 (1950); English translation in: AMS Transl. Ser. 2, 23(1063), 1–6

[YRS+05] Yorsh, G., Rabinovich, A.M., Sagiv, M., Meyer, A., Bouajjani, A.: A logic of reachable patterns in linked data structures. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 94–110. Springer, Heidelberg (2005)

On Counting Generalized Colorings

T. Kotek[1], J.A. Makowsky[1,⋆], and B. Zilber[2,⋆⋆]

[1] Department of Computer Science
Technion–Israel Institute of Technology, Haifa, Israel
{tkotek,janos}@cs.technion.ac.il
[2] Mathematical Institute,
University of Oxford
zilber@maths.ox.ac.uk

Abstract. It is well known that the number of proper k-colorings of a graph is a polynomial in k. We investigate in this talk under what conditions a numeric graph invariant which is parametrized with parameters $k_1, \ldots, k_m$ is a polynomial in these parameters. We give a sufficient conditions for this to happen which is general enough to encompass all the graph polynomials which are definable in Second Order Logic. This not only covers the various generalizations of the Tutte polynomials, Interlace polynomials, Matching polynomials, but allows us to identify new graph polynomials related to combinatorial problems discussed in the literature.

1 Introduction

Graph invariants and graph polynomials. A *graph invariant* is a function from the class of (finite) graphs $\mathcal{G}$ into some domain $\mathcal{D}$ such that isomorphic graphs have the same picture. Usually such invariants are meant to be uniformly defined in some formalism. If $\mathcal{D}$ is the two-element boolean algebra we speak of *graph properties*. Examples are the properties of being connected, planar, Eulerian, Hamiltonian, etc. If $\mathcal{D}$ consists of the natural numbers, we speak of *numeric graph invariants*. Examples are the number of connected components, the size of the largest clique or independent set, the diameter, the chromatic number, etc. But $\mathcal{D}$ could also be a polynomial ring $\mathbb{Z}[\bar{X}]$ over $\mathbb{Z}$ with a set of indeterminates $\bar{X}$. Here examples are the characteristic polynomial, the chromatic polynomial, the Tutte polynomial, etc.

There are many graph invariants discussed in the literature, which are polynomials in $\mathbb{Z}[\bar{X}]$, but there are hardly any papers discussing classes of graph polynomials as an object of study in its generality. An outline of such a study was presented in [Mak06]. In [Mak04] the second author has introduced the **MSOL**-definable and the **SOL**-definable graph polynomials, the class of graph

⋆ Partially supported by the Israel Science Foundation for the project "Model Theoretic Interpretations of Counting Functions" (2007-2010).
⋆⋆ Partially supported by MODNET: Marie Curie Research Training Network MRTN-CT-2004-512234.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 339–353, 2008.
© Springer-Verlag Berlin Heidelberg 2008

polynomials where the range of summation is definable in (monadic) second order logic. He has verified that all the examples of graph polynomials discussed in the literature, with the exception of the weighted graph polynomial of [NW99], are actually **SOL**-polynomials over some expansions (by adding order relations) of the graph, cf. also [Mak06]. In some cases this is straight forward, but in some cases it follows from intricate theorems.

A proper k-vertex-colorings of a graph $G = (V, E)$ with colors from a set $\{0, \ldots, k-1\} = [k]$ is a function from $f : V \to [k]$ such that no two distinct vertices connected by an edge have the same value. A simple case of generalized colorings are the ϕ-colorings, k-vertex-colorings definable by a first order formula $\phi(F)$ over graphs with an additional function symbol F, and we allow all functions f which are interpretations of F satisfying $\phi(F)$. It will become clearer later that to define a ϕ-coloring, the formula has to be subject to certain semantic restrictions such as invariance under permutation of the colors, the existence of a bound on the colors used, and independence of the colors not used. The general case arises by expanding the graph, allowing several color sets, and replacing functions by relations. The associated counting function $\chi_\phi(G, k)$ counts the number of generalized colorings satisfying ϕ as a function of k.

Our first result is

Proposition A. *Let $\bar{k} = (k_1, \ldots k_\alpha)$ be the cardinalities of the various color sets. For ϕ subject to the conditions above, the counting function $\chi_\phi(G, \bar{k})$ is a polynomial in $\bar{k}$.*

The purpose of this paper is to present a framework, which we call *counting functions of generalized colorings*, for defining graph invariants.

In particular we shall compare the counting functions of generalized colorings with the **SOL**-definable polynomials. A special case of these, the **MSOL**-definable polynomials were first introduced in [Mak04]. **SOL**-definable polynomials define invariants of finite first order τ-structures for arbitrary vocabularies (similarity types). In contrast to counting functions of generalized colorings, they are polynomials by their very definition. The definability condition refers to summation over definable sets of relations and products over definable sets of elements of the underlying structure.

In order to relate the counting functions of generalized colorings to the **SOL**-definable polynomials, we allow additional combinatorial functions as monomials. We call the corresponding polynomials *extended* **SOL***-polynomials.*

Main result. Our main results here are:

Theorem B. *Every extended* **SOL***-definable polynomial is a counting function of a generalized coloring of graphs definable in* **SOL***.*

Theorem C. *Every counting function of a generalized coloring of ordered graphs definable in* **SOL** *is an extended* **SOL***-definable polynomial.*

Outline of the paper. We assume the reader is familiar with the basics of graph theory as, say, presented in [Die96, Bol99]. We also assume the reader is

familiar the basics of finite model theory as, say, presented in [EFT94, EF95, Lib04]

Section 2 is a prelude to our general discussion, in which we discuss the chromatic polynomial and explain how it fits into the various frameworks. In Section 3 we introduce our notion of counting functions of generalized colorings definable in **SOL**. We prove they are polynomials in the number of colors and show examples of graph polynomials from the literature which fall under this class of graph polynomials. In Section 4 we give precise definition of extended **SOL**-definable polynomials.

An earlier version of some of the material of this paper was posted as [MZ06].

2 Prelude: The Chromatic Polynomial

Before we introduce our general definitions, we discuss the oldest graph polynomial studied in the literature, the classical *chromatic polynomial* $\chi_G(k)$. It has a very rich literature. For an excellent and exhaustive monograph, cf. [DKT05].

We denote by $\mathcal{G}$ the set of graphs of the form $G = ([n], E)$. A k-vertex-coloring of G is a function $f : [n] \to [k]$ such that whenever $(u, v) \in E$ then $f(u) \neq f(v)$. $\chi_G(k)$ denotes the number of k-vertex-colorings of G. $\chi_G(k)$ defines, for each graph, a function $\chi_G(\lambda) : \mathcal{G} \to \mathbb{N}$ which turns out to be a polynomial in λ.

We note that $\chi_G(\lambda)$ really denotes a family of polynomials indexed by graphs from $\mathcal{G}$. This family is furthermore uniformly defined based on some of the properties of the graph G. We are interested in various formalisms in which such uniform definitions can be given. We isolate the following themes:

(i) A *recursive* definition of $\chi_G(k)$ (using an order on the vertices or edges).
(ii) A uniform *explicit* definition of $\chi_G(k)$ over the graph using a second order logic formalism. In [Cou] it is called a *static* definition of the polynomial.
(iii) We associate with each $k \in \mathbb{N}$ a two-sorted structure $\mathcal{G}_k = \langle G, [k] \rangle$ and interpret $\chi_G(k)$ as counting the number of expansions $\langle \mathcal{G}_k, F \rangle$ satisfying some first order formula $\phi(F)$.

In [CGM07] the relationship between recursive and explicit definitions is studied. There, a framework is provided which allows to show that every recursive definition of a graph polynomial also allows an explicit definition. The converse is open but seems not to be true. Here we are interested in the relationship between explicit definition, and counting expansions.

Recursive definition. The first proof that $\chi_G(\lambda)$ is a polynomial used the observation that $\chi_G(\lambda)$ has a recursive definition using the order of the edges, which can be taken as the order induced by the lexical ordering on $[n]^2$. However, the object defined does not depend on the particular order of the edges. For details, cf. [Big93, Bol99]. The essence of the proof is as follows:

For $e = (v_1, v_2)$, we put $G - e = (V, E')$ with $E' = E - \{e\}$, and $G/e = (V^*, E^*)$ $V^* = V - \{v_2\}$ and $E^* = (E \cap (V^*)^2) \cup \{(v_1, v); (v_2, v) \in E\}$. The operation passing from G to $G - e$ is called *edge removal*, and the operation passing from G to G/e is called *edge contraction*.

Lemma 1. *Let e, f be two edges of G. Then we have $(G-e)-f = (G-f)-e$, $(G/e)-f = (G-f)/e$, $(G-e)/f = (G/f)-e$ and $(G/e)/f = (G/f)/e$.*

Let $E_n = ([n], \emptyset)$. We have $\chi_{E_n}(\lambda) = \lambda^n$. Furthermore, for any edge $e \in E$ we have $\chi_G(\lambda) = \chi_{G-e}(\lambda) - \chi_{G/e}(\lambda)$.

Explicit descriptions. There are other proofs that $\chi_G(\lambda)$ is a polynomial.

Proof. We first observe that any coloring uses at most n of the λ colors. For any $m \leq n$, let $c(m)$ be the number of colorings, with a fixed set of m colors, which are vertex colorings and use all m of the colors. Then, given λ colors, the number of vertex colorings that use exactly m of the λ colors is the product of $c(m)$ and the binomial coefficient $\binom{\lambda}{m}$. So

$$\chi_G(\lambda) = \sum_{m \leq n} \binom{\lambda}{m} c(m)$$

The right side here is a polynomial in λ, because each of the binomial coefficients is. We also use that for $\lambda \leq m$ we have $\binom{\lambda}{m} = 0$.

If both the set of colors and the set of vertices are initial segments of the natural numbers with their order, we can also rewrite this in the following way:

$$\chi_G(\lambda) = \sum_{A:init(A,V)} \sum_{f:ontocol(f,A)} \binom{\lambda}{card(A)} \qquad \text{(chrom-1)}$$

where $init(A, V)$ says that A is an initial segment of V, and $ontocol(f, A)$ says that f is a vertex coloring using all the colors of A.

Equation chrom-1 is an example of a explicit definition of the chromatic polynomial.

In [DKT05, Theorem 1.4.1] another explicit description of $\chi_G(\lambda)$ is given: Let $a(G, m)$ be the number of partitions of V into m independent sets, and let

$$(\lambda)_m = \lambda \cdot (\lambda - 1) \cdot \ldots \cdot (\lambda - m + 1)$$

be the *falling factorial*. Then $\chi_G(\lambda) = \sum_m a(G, m) \cdot (\lambda)_m$. This again be written

$$\chi_G(\lambda) = \sum_{P:indpart(P,A_P,V)} (\lambda_{card(A_P)}) \qquad \text{(chrom-2)}$$

where $indpart(P, A_P, V)$ says that P is an equivalence relation on V and A_P consists of the first elements (with respect to the order on $V = [n]$) of each equivalence class.

A third explicit description for $\chi_G(\lambda)$ is given in [DKT05, Theorem 2.2.1]. It can be obtained from a two-variable polynomial $Z_G(\lambda, V)$ defined by

$$Z_G(\lambda, V) = \sum_{S:S \subseteq E} \left(\prod_{v:fcomp(v,S)} \lambda \cdot \prod_{e:e \in S} V \right) = \sum_{S:S \subseteq E} \left(\lambda^{k(S)} \cdot \prod_{e:e \in S} V \right)$$

where $fcomp(v, S)$ is the property "v is the first vertex in the order of V of some connected component of the spanning subgraph $< S : V >$ on V induced by S", and $k(S)$ is the number of connected components of $< S : V >$. Now we have

$$\chi_G(\lambda) = Z_G(\lambda, -1) \qquad \text{(chrom-3)}$$

The three explicit descriptions of the chromatic polynomial chrom-1, chrom-2, chrom-3 have several properties in common:

(i) They satisfy the same recursive definition, because they define the same polynomial.
(ii) They are of the form $\sum_k A_k(G)P_k(\lambda)$ where $P_k(\lambda)$ is a polynomial in λ with integer coefficients of degree k.
(iii) The coefficients $A_k(G)$ are positive and have a combinatorial interpretation.
(iv) The coefficients can be alternatively obtained by collecting the terms $P_k(\lambda)$ of a summation over certain relations definable in second order logic over the graph with an order on the vertices and interpreting k as the cardinality of such a relation.
(v) Although the order on the vertices is used in the explicit description of the polynomial, the polynomial is *invariant under permutations of the ordering.*

There are also significant differences.

(i) In chrom-1 it is important that the set of colors and the set of vertices are initial segments of the natural numbers with their natural order. The summation involves *one unary relation* and *one unary function.*
(ii) In chrom-2 the summation involves a *binary relation* on the vertices which is not a subset of the edge relation, but of its complement. The order relation is only needed to identify equivalence classes.
(iii) In chrom-3 we actually use a two-variable polynomial and then substitute for one variable -1. The summation involves a *binary relation* on vertices which is a subset of the edge relation. It can be also viewed as a *unary relation* on the set of edges. The order relation is only needed to identify connected components.

Counting expansions. Let $\mathbf{F}$ be a unary function symbol and let $\phi(\mathbf{F}, \mathbf{E})$ be the formula which says that F is a proper k-vertex coloring for the edge relation E. Then $\chi_G(k) = \chi_\phi(G, k)$ is the number of interpretations F of $\mathbf{F}$ in $\langle [n], [k], E, F\rangle$ which satisfy $\phi(\mathbf{F}, \mathbf{E})$. We note that

(i) a coloring is invariant under permutations of the colors,
(ii) the number of colors is bounded by the size of V, and
(iii) the property of being a coloring is independent of the colors not used.

This is readily generalized to other formulas $\psi(F, E)$ satisfying similar properties, and will be the starting point for our notion of generalized coloring.

3 Generalized Chromatic Polynomials

Generalized colorings. Let $\mathcal{M}$ be a τ-structure with universe M. We say a two-sorted structure $\langle \mathcal{M}, [k], R\rangle$ for the vocabulary τ_R is a generalized coloring of $\mathcal{M}$ with k colors. The set $[k]$ will be referred to as the color set. We denote relation symbols by bold-face letters, and their interpretation by the corresponding roman-face letter.

Definition 1 (Coloring Property). *A class $\mathcal{P}$ of generalized colorings*

$$\mathcal{P} = \{\langle \mathcal{M}, [k], R\rangle \mid R \subseteq M^m \times [k]\}$$

of τ_R structures is a coloring property if it satisfies the following conditions:

Isomorphism property. *$\mathcal{P}$ is closed under τ_R-isomorphisms.*
In particular, $\mathcal{P}$ is closed under permutations of the color set $[k]$.

Extension property. *For every $\mathcal{M}$, k, k' and R, if $k' \geq k$ and $\langle \mathcal{M}, [k], R\rangle \in \mathcal{P}$ then $\langle \mathcal{M}, [k'], R\rangle \in \mathcal{P}$.*
Namely, the extension property requires that increasing the number of colors of a generalized coloring does not affect whether it belongs to the property.

We refer to the relation symbol $\mathbf{R}$ *and its interpretations R as* coloring predicates*. For fixed k, a specific interpretation is called a k-$\mathcal{P}$-coloring.*

Definition 2 (Bounded coloring properties)

(i) A coloring property $\mathcal{P}$ is bounded*, if for every $\langle \mathcal{M}, [k], R\rangle \in \mathcal{P}$ there is a number N_M such that the set of colors $\{x \in [k] : \exists \bar{y} \in M^m R(\bar{y}, x)\}$ has size at most N_M, and N_M does not depend on k.*

(ii) A coloring property $\mathcal{P}$ is range bounded*, if its range is bounded in the following sense: There is a number $d \in \mathbb{N}$ such that for every $\langle \mathcal{M}, [k], R\rangle \in \mathcal{P}$ and $\bar{y} \in M^m$ the set $\{x \in [k] : R(\bar{y}, x)\}$ has at most d elements.*

Clearly, if a coloring property is range bounded, it is also bounded.

Proposition 1 (Special case of Proposition A). *Let $\mathcal{P}$ be a bounded coloring property. For every $\mathcal{M}$ the number $\chi_{\mathcal{P}}(\mathcal{M}, k)$ is a polynomial in k of the form*

$$\sum_{j=0}^{N_M} c_{\mathcal{P}}(\mathcal{M}, j)\binom{k}{j}$$

where $c_{\mathcal{P}}(\mathcal{M}, j)$ is the number of generalized k-$\mathcal{P}$-colorings R with a fixed set of j colors. If $\mathcal{P}$ is range-bounded then $N_M \leq d \cdot |M|^m$.

Proof. We first observe that any generalized coloring R uses at most N_M of the k colors, if it is bounded. If R is range-bounded then $N_M \leq d \cdot |M|^m$. For any $j \leq N_M$, let $c_{\mathcal{P}}(\mathcal{M}, j)$ be the number of colorings, with a fixed set of j colors, which are generalized vertex colorings and use all j of the colors. So any permutation of the set of colors used is also a coloring. Therefore, given k colors,

the number of vertex colorings that use exactly j of the k colors is the product of $c_{\mathcal{P}}(\mathcal{M}, j)$ and the binomial coefficient $\binom{k}{j}$. So

$$\chi_{\mathcal{P}}(\mathcal{M}, k) = \sum_{j \leq N_M} c_{\mathcal{P}}(\mathcal{M}, j) \binom{k}{j}$$

The right side here is a polynomial in k, because each of the binomial coefficients is. We also use that for $k < j$ we have $\binom{k}{j} = 0$.

We note bounded properties which are not range-bounded lack in uniformity. So for range-bounded $\mathcal{P}$ we call $\chi_{\mathcal{P}}(\mathcal{M}, k)$ a *generalized chromatic polynomial*.

Remark 1. The restriction to coloring properties in Proposition 1 is essential. Let $\chi_{onto}(G, k)$ be the number of functions $f : V(G) \to [k]$ which are onto. Clearly, this is not a polynomial in k as for $k > \mid V(G) \mid$ it always vanishes, so it should be constantly 0.

Definability of generalized chromatic polynomials. We assume the reader is familiar with First and Second Order Logic, denoted by **FOL** and **SOL**, as defined in, for example, [EF95]. The formulas of $\mathbf{SOL}(\tau)$ are defined like the ones of **FOL**, with the addition that we allow countably many variables for n-ary relation symbols $U_{n,\alpha}$ for $\alpha \in \mathbb{N}$, for each $n \in \mathbb{N}$, and quantification over these. Monadic second order logic $\mathbf{MSOL}(\tau)$ is the restriction of $SOL(\tau)$ to unary relation variables and quantification over these.

Definition 3. *A generalized chromatic polynomial for τ-structures with coloring predicate R is definable in $\mathbf{SOL}(\tau_R)$, respectively in $\mathbf{MSOL}(\tau_R)$, if it is of the form $\chi_\phi(\mathcal{M}, \bar{\lambda})$, where $\phi \in \mathbf{SOL}(\tau_R)$, respectively in $\mathbf{MSOL}(\tau_R)$, and defines a range-bounded coloring property.*

Definition 4 (Coloring formula). *A first order (or second order) formula $\phi(\mathbf{R})$ is a* coloring formula*, if the class of its models, which are of the form $\langle \mathcal{M}, [k], R \rangle$, is a coloring property. If a coloring property $\mathcal{P}$ is definable by a coloring formula ϕ then we denote the number of generalized k-$\mathcal{P}$-colorings on R by $\chi_{\phi(R)}(\mathcal{M}, k)$.*

The coloring property of proper vertex colorings from the next example is definable by a **FOL** formula.

Examples and applications of Proposition 1. Let $\mathbf{F}$ be a unary function symbol which serves as the coloring predicate. A (not necessarily proper) vertex coloring of a graph $G = (V, E)$ is a map $F : V \to [k]$ for some k.

(i) A vertex coloring F is *proper*, if it satisfies $\forall u, v(\mathbf{E}(u, v) \to \mathbf{F}(u) \neq \mathbf{F}(v))$. Clearly, this does define a coloring property.

(ii) If we require that a vertex coloring F uses all the colors, then this is not a coloring property. It violates the extension property in Definition 1.

(iii) A vertex coloring is *pseudo-complete*, if it satisfies

$$\forall x, y \exists u, v(\mathbf{E}(u, v) \wedge \mathbf{F}(u) = x \wedge \mathbf{F}(v) = y).$$

For the same reason as above this is not a coloring property.

Connected colorings. A vertex coloring is *connected* if each monochromatic set induces a connected subgraph. We denote by $\chi_{cc}(G,k)$ the number of connected colorings of G with k colors. Clearly, coloring all connected components with the same color gives a connected coloring. However, it is not clear how difficult it is to count such colorings.

Conjecture 1. For all $k \in \mathbb{N}$ with $k \geq 2$, computing $\chi_{cc}(G,k)$ is $\sharp\mathbf{P}$ hard.

Harmonious colorings. Harmonious colorings are counterparts to complete colorings. A vertex coloring is a *harmonious coloring* if it is a proper coloring and every pair of colors appears at most once on an edge. In this case, the extension property holds and the number of harmonious colorings $\chi_{harm}(G,k)$ is a polynomial in k. Harmonious colorings are treated in [HK83], as well as [Edw97] and [EM95].

$mcc(t)$-colorings. We say a vertex coloring $f : V \to [k]$ has small monochromatic connected components if $f^{-1}(a)$ induces a graph with each of its connected components of size at most t. For a fixed t, this is a coloring property. Related graph invariants were introduced in [LMST07] and in [ADOV03].

Proposition 2. *$\chi_{cc}(G,k)$, $\chi_{harm}(G,k)$ and $\chi_{mcc(t)}(G,k)$ are generalized chromatic polynomials.*

Complete colorings. A vertex coloring is *complete*, if it is both proper and pseudo-complete. Complete colorings are studied in the context of the *achromatic number of a graph* G which is the largest number k such that G has a complete coloring with k colors. The achromatic number of G and the number of complete colorings is a function of G but not of k. In other words, to be a complete coloring is not a coloring property in our sense. Let $\chi_{complete}(G,k)$ denote the number of complete colorings of G with k colors. Using the same argument as in Remark 1 we see that $\chi_{complete}(G,k)$ ultimately vanishes for large enough k, and therefore is not a polynomial in k.

The achromatic number was introduced in [HHR67]. For a survey of recent work, cf. [HM97].

Generalized multi-colorings. To construct also graph polynomials in several variables, we extend the definition to deal with several color-sets, and also call them generalized chromatic polynomials.

Let $\mathcal{M}$ be a τ-structure with universe M. We say an $(\alpha+1)$-sorted structure $\langle \mathcal{M}, [k_1], \ldots, [k_\alpha], R\rangle$ for the vocabulary $\tau_{\alpha,R}$ with $R \subset M^m \times [k_1]^{m_1} \times \ldots \times [k_\alpha]^{m_\alpha}$ is a *generalized coloring* of $\mathcal{M}$ for colors $\bar{k}^\alpha = (k_1, \ldots, k_\alpha)$. By abuse of notation, $m_i = 0$ is taken to mean the color-set k_i is not used in R.

Definition 5 (Multi-color Coloring Property). *A class of generalized multi-colorings $\mathcal{P}$ is a coloring property if it satisfies the following conditions:*

Isomorphism property: *$\mathcal{P}$ is closed under $\tau_{\alpha,R}$-isomorphisms.*

Extension property: *For every* $\mathcal{M}$, $k_1 \leq k'_1, \ldots, k_\alpha \leq k'_\alpha$, *and* R, *if* $\langle \mathcal{M}, [k_1], \ldots, [k_\alpha], R\rangle \in \mathcal{P}$ *then* $\langle \mathcal{M}, [k'_1], \ldots, [k'_\alpha], R\rangle \in \mathcal{P}$.

Non-occurrence property: *Assume* $R \subset M^m \times [k_1]^{m_1} \times \ldots \times [k_\alpha]^{m_\alpha}$ *with* $m_i = 0$, *and* $\langle \mathcal{M}, [k_1], \ldots, [k_\alpha], R\rangle \in \mathcal{P}$, *then for every* $k'_i \in \mathbb{N}$, $\langle \mathcal{M}, [k_1], \ldots, [k'_i], \ldots, [k_\alpha], R\rangle \in \mathcal{P}$.

The extension property and the non-occurrence property require that increasing the number of colors respectively adding unused color-sets does not affect whether the generalized coloring belongs to $\mathcal{P}$.

We denote by $\chi_{\mathcal{P}}(\mathcal{M}, k_1, \ldots, k_\alpha)$ *the number of generalized* $\bar{k}^\alpha$ *–* $\mathcal{P}$*-multi-colorings* R *on* $\mathcal{M}$. *If* $\mathcal{P}$ *is definable by a formula* $\phi(\mathbf{R})$ *we also write* $\chi_{\phi(R)}(\mathcal{M}, \bar{k}^\alpha)$.

Definition 6 (Bounded multi-coloring properties)

(i) A coloring property $\mathcal{P}$ *is* bounded, *if for every* $\langle \mathcal{M}, [k_1], \ldots, [k_\alpha], R\rangle \in \mathcal{P}$ *there is a number* N_M *such that the set of colors* $\{\bar{x} \in \bar{k}^\alpha : \exists \bar{y} \in M^m R(\bar{y}, \bar{x})\}$ *has size at most* N_M, *and* N_M *does not depend on* $\bar{k}^\alpha$.

(ii) A coloring property $\mathcal{P}$ *is* range bounded, *if its range is bounded in the following sense: There is* $d \in \mathbb{N}$ *such that for every* $\langle \mathcal{M}, [k_1], \ldots, [k_\alpha], R\rangle \in \mathcal{P}$ *and* $\bar{y} \in M^m$ *the set* $\{\bar{x} \in \bar{k}^\alpha : R(\bar{y}, \bar{x})\}$ *has at most* d *elements.*

Proposition 3 (Special case of Proposition A). *Let* $\mathcal{P}$ *be a bounded multi-coloring property with coloring forumula* ϕ *and bound* N_M. *In the case of range bounded multi-colorings* $N_M \leq d \cdot \mid M_M \mid^m$. *For every* $\mathcal{M}$, $\chi_{\mathcal{P}}(\mathcal{M}, k_1, \ldots, k_\alpha)$ *is a polynomial in* $k_1, \ldots, k_\alpha$ *of the form*

$$\sum_{\bar{j}^\alpha \leq [N_M]^\alpha} c_{\phi(R)}(\mathcal{M}, \bar{j}^\alpha) \prod_{1 \leq \beta \leq \alpha} \binom{k_\beta}{j_\beta}$$

where $c_{\phi(R)}(\mathcal{M}, \bar{j}^\alpha)$ *is the number of generalized* $\bar{k}^\alpha - \phi$*-colorings* R *with fixed sets of* j_β *colors respectively.*

Proof. Similar to the one variable case.

Example 1. Recall earlier in this section we denoted by $\chi_{mcc(t)}(G, k)$ the number of vertex colorings for which no color induces a graph with a connected component larger than t. Let $\chi_{mcc}(G, k, t) = \chi_{mcc(t)}(G, k)$ be the counting function of generalized multi-colorings satisfying the above condition, where t is considered a color-set. We note for every $t \geq |V|$, every vertex coloring has only connected components of size no more then t. So, if $\chi_{mcc}(G, k, t)$ were a polynomial in t then by interpolation $\chi_{mcc}(G, k, t)$ would be the number of vertex colorings, so the function $\chi_{mcc}(G, k, t)$ is must not be a polynomial in t. The set of such vertex colorings does not satisfy the non-occurrence condition. This example shows the motivation for requiring this condition of coloring properties.

The most general case. We shall also allow several simultaneous coloring predicates $R_1, \ldots, R_s$. The notion of coloring properties for this situation extends naturally. We shall call multi-coloring properties and multi-coloring simply also coloring properties and colorings, if the situation is clear from the context. The notion of definability of multi-colorings with several coloring predicates is analogous to the simple case.

Closure properties

Proposition 4 (Sums and products). *The sum and product of two generalized chromatic polynomials* $\chi_{\phi(\mathbf{R})}(G,\lambda)$ *and* $\chi_{\psi(\mathbf{R})}(G,\lambda)$ *is again a generalized chromatic polynomial.*

Proof. For the sum we take $\chi_{\theta_1}(G,\lambda)$ with

$$\theta_1(R,R',U) = ((U=\emptyset)\wedge\phi(R)\wedge(R'=\emptyset))\vee((U=M)\wedge(R=\emptyset)\wedge\psi(R'))\,.$$

For the product we take $\chi_{\theta_2}(G,\lambda)$ with $\theta_2(\mathbf{R},\mathbf{R}') = (\phi(\mathbf{R})\wedge\psi(\mathbf{R}'))$.

More examples of generalized chromatic polynomials. We now show how many graph polynomials can be viewed as generalized chromatic polynomials.

Combinatorial polynomials. The following combinatorial polynomials can be thought of as generalized chromatic polynomials:

(i) For the polynomial λ^n we take all maps $[n]\to[k]$ for $\lambda = k$. So $\lambda^n = \chi_{true(f)}$ where $true(f)$ is $\forall v(f(v) = f(v))$. It is a **FOL** definable range-bounded coloring property.
(ii) Similarly, for $\lambda_{(n)} = \lambda\cdot(\lambda-1)\cdot\ldots\cdot(\lambda-n+1)$ we take all injective maps, which is easily expressed by a **FOL** formula which defines a range-bounded coloring property.
(iii) Finally, for $\binom{\lambda}{n}$ we take the ranges of injective maps. This is a range-bounded coloring property of a second order formula $\phi(\mathbf{P})$ which says that $P\subseteq[k]$ is the range of an injective map $f:[n]\to[k]$.

Connected components. We denote by $k(G)$ the number of connected components of G. The polynomial $\lambda^{k(G)}$ can be written as $\chi_{\phi_{connected}}(G,\lambda)$ with $\phi_{connected}(f)$ the formula $((u,v)\in E\to f(u)=f(v))$.

Matching polynomial. Let $G=(V,E)$ be a graph. A subset $M\subseteq E$ is a matching if no two edges in E have a common vertex. The matching polynomial of G is given by

$$g(G,\lambda) = \sum_j \mu(G,j)\lambda^j$$

where $\mu(G,j)$ is the number of of matchings of size j.

We look at the structure G_k and at pairs (M,F) with $M\subset E$ and $F:E\to[k]$ such that M is a matching and the domain of F is M, which can be expressed by a formula $match(M,F)$. We have

$$\chi_{match(M,F)}(G,k) = \sum_j \mu(G,j)k^j = g(G,k)$$

which shows that it is a generalized chromatic polynomial.

There are two close relatives to the matching polynomial, cf. [God93].

(i) The *acyclic polynomial*

$$m(G,k) = \sum_j (-1)^j \mu(G,j) k^{n-2j} = k^n g(G, -k^{-2})$$

(ii) The *rook polynomial*, which is defined for bipartite graphs only:

$$r(G,k) = \sum_j \mu(G,j) k^{n-j} = k^n g(G, -x^{-1}).$$

The rook polynomial does not look like a generalized chromatic polynomial, but it is a substitution instance of $g(G,k)$. Similarly, the acyclic polynomial is a product of k^n with a substitution instance of $g(G,k)$.

Tutte polynomial. We use the Tutte polynomial in the following form:

$$Z(G,q,v) = \sum_{A \subseteq E} q^{k(A)} v^{|A|}$$

where $conn(A)$ is the number of connected components of the spanning subgraph (V,A). This form of the Tutte polynomial is discussed in [Sok05]. For this purpose we look at the three-sorted structure $G_{k,l} = \langle V, [k], [l], E \rangle$ and at the triples (A, F_1, F_2) with $A \subseteq E$, $F_1 : V \to [k]$ whose interpretation depends on A, and $F_2 : A \to [l]$ such that, simultaneously, for $(u,v) \in A \to F_1(u) = F_1(v)$. This is expressed in the formula $Tutte(A, F_1, F_2)$. Now we have

$$\chi_{Tutte(A,F_1,F_2)}(G,k,l) = \sum_{A \subseteq E} k^{conn(A)} l^{|A|}$$

which is the evaluation of $Z(G,q,v)$ for $q = k, v = l$.

4 SOL-Definable Graph Polynomials

SOL(τ)-polynomials. We are now ready to introduce the **SOL**-definable polynomials. Let $\mathcal{R}$ be a commutative semi-ring, which contains the semi-ring of the integers $\mathbb{N}$. For our discussion $\mathcal{R} = \mathbb{N}$ or $\mathcal{R} = \mathbb{Z}$ suffices, but the definitions generalize. Our polynomials have a fixed finite set of variables (indeterminates, if we distinguish them from the variables of **SOL**), $\mathbf{X}$.

Definition 7 (SOL-monomials). *Let $\mathcal{M}$ be a τ-structure. We first define the* **SOL***-definable* $\mathcal{M}$-monomials*. inductively.*

(i) Elements of $\mathbb{N}$ *are* **SOL***-definable* $\mathcal{M}$*-monomials.*
(ii) Elements of $\mathbf{X}$ *are* **SOL***-definable* $\mathcal{M}$*-monomials.*
(iii) Finite products of monomials are **SOL***-definable* $\mathcal{M}$*-monomials.*

(iv) Let $\phi(\bar{a})$ be a $\tau \cup \{\bar{a}\}$-formula in **SOL***, where $\bar{a} = (a_1, \ldots, a_m)$ is a finite sequence of constant symbols not in τ. Let t be a $\mathcal{M}$-monomial. Then $\prod_{\bar{a}:\langle \mathcal{M}, \bar{a}\rangle \models \phi(\bar{a})} t$ is a* **SOL***-definable $\mathcal{M}$-monomial.*

The monomial t may depend on relation or function symbols occurring in ϕ.

Note the degree of a monomial is polynomially bounded by the cardinality of $\mathcal{M}$.

Definition 8 (SOL-polynomials). *The $\mathcal{M}$-polynomials definable in* **SOL** *are defined inductively:*

(i) $\mathcal{M}$-monomials are **SOL***-definable $\mathcal{M}$-polynomials.*
(ii) Let $\phi(\bar{a})$ be a $\tau \cup \{\bar{a}\}$-formula in **SOL** *where $\bar{a} = (a_1, \ldots, a_m)$ is a finite sequence of constant symbols not in τ. Let t be a $\mathcal{M}$-polynomial. Then $\sum_{\bar{a}:\langle \mathcal{M}, \bar{a}\rangle \models \phi(\bar{a})} t$ is a* **SOL***-definable $\mathcal{M}$-polynomial.*
(iii) Let $\phi(\bar{R})$ be a $\tau \cup \{\bar{R}\}$-formula in **SOL** *where $\bar{R} = (R_1, \ldots, R_m)$ is a finite sequence of relation symbols not in τ. Let t be a $\mathcal{M}$-polynomial definable in* **SOL***. Then $\sum_{\bar{R}:\langle \mathcal{M}, \bar{R}\rangle \models \phi(\bar{R})} t$ is a* **SOL***-definable $\mathcal{M}$-polynomial.*

The polynomial t may depend on relation or function symbols occurring in ϕ.

An $\mathcal{M}$-polynomial $p_{\mathcal{M}}(\mathbf{X})$ is an expression with parameter $\mathcal{M}$. The family of polynomials, which we obtain from this expression by letting $\mathcal{M}$ vary over all τ-structures, is called, by abuse of terminology, a $\mathbf{SOL}(\tau)$-polynomial.

Among the **SOL**-definable polynomials we find most of the known graph polynomials from the literature.

Properties of SOL-definable polynomials

Proposition 5. *(i) If we write an* **SOL***-definable polynomial as a sum of monomials, then the coefficients of the monomials are in $\mathbb{N}$.*
(ii) Let $\Psi(\mathcal{M})$ be an **SOL***-monomial viewed as a polynomial. Then $\Psi(\mathcal{M})$ is a product of a finite number s of terms of the form $\prod_{\bar{a}:\langle \mathcal{M}, \bar{a}\rangle \models \phi_i} t_i$, where $i \in [s]$, $t_i \in \mathbb{N} \cup \mathbf{X}$ and $\phi_i \in$* **SOL***.*
(iii) The product of two $\mathbf{SOL}(\tau)$-polynomials is again a $\mathbf{SOL}(\tau)$-polynomial.
(iv) The sum of two $\mathbf{SOL}(\tau)$-polynomials is again a $\mathbf{SOL}(\tau)$-polynomial.
(v) Let $\Phi(\mathcal{A}, \bar{X})$ be a **SOL***-definable monomial and $P : Str(\tau) \to \mathbb{N}[\bar{X}]$ be of form*

$$P(\mathcal{M}, \bar{X}) = \sum_{\bar{R}:\langle \mathcal{M}, \bar{R}\rangle \models \chi_R} \prod_{\bar{b}:\langle \mathcal{M}, \bar{R}, \bar{b}\rangle \models \psi} \sum_{\bar{a}:\langle \mathcal{M}, \bar{R}, \bar{a}, \bar{b}\rangle \models \phi} \Phi(\langle \mathcal{M}, R, \bar{a}, \bar{b}\rangle, \bar{X}).$$

Then $P(\mathcal{M}, \bar{X})$ is a **SOL***-definable polynomial.*

Combinatorial polynomials. As for the generalized chromatic polynomials, it is note-worthy to see which combinatorial polynomials are **SOL**-definable polynomials. The following are all **SOL**-definable generalized chromatic polynomials. We denote by $card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))$ the number of $\bar{v}$'s that satisfy φ.

Cardinality, I: The cardinality of a definable set $card_{\mathcal{M},\bar{v}}(\varphi(\bar{v})) = \sum_{v\in\varphi(v)} 1$ is an evaluation of a **SOL**-definable polynomial.

Cardinality, II: The cardinality as the exponent in a monomial $X^{card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))} = \prod_{v:\varphi(v)} X$ is an **SOL**-definable polynomial.

Cardinality, III: Exponentiation of cardinalities $card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))^{card_{\mathcal{M},\bar{v}}(\psi(\bar{v}))} = \prod_{v:\psi(v)} \sum_{u:\varphi(u)} 1$ is equivalent to an evaluation of a **SOL**-definable polynomial.

Factorials: The factorial of the cardinality of a definable set $card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))! = \sum_{\pi:\varphi(v)\to\varphi(v)} 1$ is an instance of a **SOL**-definable polynomial.

Binomial coefficients: The binomial coefficient $\binom{X}{card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))} = \frac{(X)_{|\varphi|}}{|\varphi|!}$ is not an evaluation of a **SOL**-definable polynomial. It contains terms involving division, which contradicts Proposition 5

Falling factorial: The falling factorial $(X)_{card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))} = \binom{X}{|\varphi|} \cdot |\varphi|!$ is not a **SOL**-definable polynomial, because it contains negative terms, which contradicts Proposition 5 However, if the underlying structure has a linear order, then it is an evaluation of an **SOL**-definable polynomial.

In the next sub-section we will show how the **SOL**-definable polynomials are related to the **SOL**-definable generalized chromatic polynomials via the addition of $\binom{X}{card_{\mathcal{M},\bar{v}}(\varphi(\bar{v}))}$. This example shows the motivation for this addition, as currently the **SOL**-definable polynomials are not expressible enough to include basic combinatorial functions.

Extended SOL(τ)-polynomials. Motivated by the discussion about combinatorial polynomials above we define now the *extended* **SOL**-*polynomials*.

Definition 9 (Extended SOL-polynomials)

(i) For every $\phi(\bar{v}) \in \mathbf{SOL}(\tau)$ *we define the cardinality of the set defined by* ϕ*:*

$$card_{\mathcal{M},\bar{v}}(\phi(\bar{v})) = \mid \{\bar{a} \in M^m : \langle \mathcal{M}, \bar{a}\rangle \models \phi(\bar{a})\} \mid .$$

(ii) The extended **SOL**(τ)-monomials *are defined inductively as:*

(ii.a) **SOL**(τ)*-monomials are extended* **SOL**(τ)*-monomials.*

(ii.b) For every $\phi(\bar{v}) \in \mathbf{SOL}(\tau)$ *and for every* $X \in \mathbf{X}$, $\binom{X}{card_{\mathcal{M},\bar{v}}(\phi(\bar{v}))}$ *is an extended* **SOL**(τ)*-monomial.*

(ii.c) Finite product of extended **SOL**(τ)*-monomials are extended* **SOL**(τ)*-monomial.*

Note that $\binom{X}{card_{\mathcal{M},\bar{v}}}$ *may not occur within the scope of* $\prod$*.*

(iii) The extended **SOL**(τ)*-polynomials are defined as in definition 8 with respect to extended* **SOL**(τ)*-monomials.*

(iv) Similarly, we define also extended **MSOL**-polynomials*.*

We note that the combinatorial polynomials mentioned earlier in this section are all expressible by extended **SOL**-definable polynomials.

Proof of Theorems B and C

Theorem B. *Every extended* $\mathbf{SOL}(\tau)$*-polynomial over some* τ*-structure* $\mathcal{A}$ *is a counting function of a generalized coloring definable in* $\mathbf{SOL}(\tau)$ *in a suitable expansion of* $\mathcal{A}$*.*

Theorem C. *Every generalized chromatic polynomial definable in* **SOL** *on ordered* τ*-structures* $\mathcal{M}$ *is an extended* **SOL***-definable polynomial.*

To prove Theorem B we proceed by induction, proving it first for **SOL**-monomials, and then we use the normal form for **SOL**-polynomials (Proposition 5). To prove Theorem C we code the color sets inside the graph. Using that the coloring is bounded, we can code the color sets in a fixed Cartesian product $M^{d \cdot m}$ of the domain of the structure $\mathcal{M}$.

5 Conclusions

Starting with the classical chromatic polynomial we have introduced generalized multi-colorings. We have shown that the corresponding counting functions are always polynomials, which we called generalized chromatic polynomials. We have then shown that the class of generalized chromatic polynomials is very rich and covers virtually all examples of graph polynomials which have been studied in the literature.

We presented the class of **SOL**-definable graph polynomials which were introduced in [Mak04], and extended them by allowing generalized binomial coefficients as monomials. We have then shown that the class of extended **SOL**-definable graph polynomials coincides with the class of **SOL**-definable generalized chromatic polynomials. This, along with the extensive scope of the class, suggests that the frameworks presented in this paper are natural to the study of graph polynomials.

Theorems B and C can also be used to analyze the complexity of evaluations of **SOL**-definable polynomials at integer points. They fit nicely into the framework developed by S. Toda in his unpublished thesis and in [TW92].

Acknowledgments. The second author would like to thank I. Averbouch, A. Blass, B. Courcelle, B. Godlin, E. Hrushovski, S. Shelah and M. Ziegler for valuable discussions and suggestions. We would like to thank A. Blass for allowing us to incorporate the simple proof of Proposition 1, which he suggested.

References

[ADOV03] Alon, N., Ding, G., Oporowski, B., Vertigan, D.: Partitioning into graphs with only small components. Journal of Combinatorial Theory, Series B 87, 231–243 (2003)

[Big93] Biggs, N.: Algebraic Graph Theory, 2nd edn. Cambridge University Press, Cambridge (1993)

[Bol99] Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1999)

[CGM07] Courcelle, B., Godlin, B., Makowsky, J.A.: Towards a theory of graph polynomials, I: Second order definable polynomials (in preparation, 2007)

[Cou] Courcelle, B.: A multivariate interlace polynomial (December 2006) (preprint)

[Die96] Diestel, R.: Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (1996)

[DKT05] Dong, F.M., Koh, K.M., Teo, K.L.: Chromatic Polynomials and Chromaticity of Graphs. World Scientific, Singapore (2005)

[Edw97] Edwards, K.: The harmonious chromatic number and the achromatic number. In: Bailey, R.A. (ed.) Survey in Combinatorics. London Math. Soc. Lecture Note Ser, vol. 241, pp. 13–47. Cambridge Univ. Press, Cambridge (1997)

[EF95] Ebbinghaus, H.D., Flum, J.: Finite Model Theory. Perspectives in Mathematical Logic. Springer, Heidelberg (1995)

[EFT94] Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical Logic, 2nd edn. Undergraduate Texts in Mathematics. Springer, Heidelberg (1994)

[EM95] Edwards, K., McDiarmid, C.: The complexity of harmonious colouring for trees. Discrete Appl. Math. 57(2-3), 133–144 (1995)

[God93] Godsil, C.D.: Algebraic Combinatorics. Chapman and Hall, Boca Raton (1993)

[HHR67] Harary, F., Hedetniemi, S., Rins, G.: An interpolation theorem for graphical homomorphisms. Portugal. Math. 26, 453–462 (1967)

[HK83] Hopcroft, J.E., Krishnamoorthy, M.S.: On the harmonious coloring of graphs. SIAM J. Algebraic Discrete Methods 4, 306–311 (1983)

[HM97] Hughes, F., MacGillivray, G.: The achromatic number of graphs: a survey and some new results. Bull. Inst. Combin. Appl. 19, 27–56 (1997)

[Lib04] Libkin, L.: Elements of Finite Model Theory. Springer, Heidelberg (2004)

[LMST07] Linial, N., Matoušek, J., Sheffet, O., Tardos, G.: Graph coloring with no large monochromatic components (2007) arXiv:math/0703362v1

[Mak04] Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126(1-3), 159–213 (2004)

[Mak06] Makowsky, J.A.: From a zoo to a zoology: Descriptive complexity for graph polynomials. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 330–341. Springer, Heidelberg (2006)

[MZ06] Makowsky, J.A., Zilber, B.: Polynomial invariants of graphs and totally categorical theories. MODNET Preprint No. 21 (2006), `http://www.logique.jussieu.fr/modnet/Publications/Preprint%20server`

[NW99] Noble, S.D., Welsh, D.J.A.: A weighted graph polynomial from chromatic invariants of knots. Ann. Inst. Fourier, Grenoble 49, 1057–1087 (1999)

[Sok05] Sokal, A.: The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In: Survey in Combinatorics, 2005. London Mathematical Society Lecture Notes, vol. 327, pp. 173–226 (2005)

[TW92] Toda, S., Watanabe, O.: Polynomial time 1-Turing reductions from #PH to #P. Theor. Comp. Sc. 100, 205–221 (1992)

The Descriptive Complexity of Parity Games

Anuj Dawar[1] and Erich Grädel[2]

[1] University of Cambridge Computer Laboratory, Cambridge, CB3 0FD, UK
anuj.dawar@cl.cam.ac.uk
[2] Mathematische Grundlagen der Informatik, RWTH Aachen University, Germany
graedel@logic.rwth-aachen.de

Abstract. We study the logical definablity of the winning regions of parity games. For games with a bounded number of priorities, it is well-known that the winning regions are definable in the modal μ-calculus. Here we investigate the case of an unbounded number of priorities, both for finite game graphs and for arbitrary ones. In the general case, winning regions are definable in guarded second-order logic (GSO), but not in least-fixed point logic (LFP). On finite game graphs, winning regions are LFP-definable if, and only if, they are computable in polynomial time, and this result extends to any class of finite games that is closed under taking bisimulation quotients.

1 Introduction

The question whether the winning regions (i.e. the set of positions from which a particular player has a winning strategy) in parity games can be computed efficiently is one of the most important open problems in the field of infinite games. It is equivalent to the model checking problem for the modal μ-calculus. It is known that the problem is in NP $\cap$ Co-NP, which is a simple consequence of the fact that parity games admit memoryless winning strategies. Much effort has also been put into identifying and classifying special cases of parity games which admit tractable solutions. For instance, there are deterministic polynomial-time algorithms for any class of parity games with a bounded number of priorities, and for parity games with certain restrictions on the underlying game graph, such as bounded tree width, bounded DAG-width, and others.

In this paper we investigate the *descriptive complexity* of parity games, i.e. we ask by what logical means the winning regions of parity games are definable. The descriptive complexity of a problem provides an insight into the structure of the problem, and the sources of algorithmic difficulty, as the logical resources needed to specify the problem are closely tied to its structure. In the case of parity games, the questions that naturally arise are whether the problem is definable in least fixed-point logic (LFP) and in monadic second-order logic (MSO), as these are logics with which it is closely associated. Again, the problem can be considered as solved when we just consider games with a bounded number of priorities. In that case, the winning regions are definable in the modal μ-calculus, and therefore also in monadic second-order logic MSO and in least fixed-point logic

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 354–368, 2008.
© Springer-Verlag Berlin Heidelberg 2008

LFP. However, the descriptive complexity of parity games with an unbounded number of priorities has not been settled in a satisfactory way. While it can be shown that in this case winning regions are no longer definable in the μ-calculus, the question arises whether they are definable in LFP and/or MSO.

It turns out that the descriptive complexity of parity games on finite game graphs is different from that on arbitrary ones. By making use of the strictness of the alternation hierarchy of LFP on arithmetic, together with an interpretation argument for model checking games, we are able to show that in general, winning regions of parity games are not LFP-definable. On finite game graphs, however, this may well be different. Indeed we prove that the winning regions are LFP-definable if, and only if, they are computable in polynomial-time (despite the fact that, on unordered finite structures, LFP is weaker than PTIME). Our arguments are based on bisimulation quotients of the relational structures that represent parity games and on Otto's result that the bisimulation-invariant fragment of PTIME can be captured by a multi-dimensional variant of the μ-calculus. Our analysis in fact shows that winning regions are LFP-definable on any class of finite parity games for which we have polynomial-time algorithms on their bisimulation quotients. As a consequence, we obtain LFP-definitions for the winning regions of parity games with bounded entanglement. We also show that winning regions are definable, on the class of finite game graphs in stronger fixed-point logics such as NFP and PFP.

A significant open question that remains unanswered is whether winning regions of parity games are MSO-definable. We have not been able to settle this question either on finite game graphs or on arbitrary ones. However, it is not difficult to prove definability in guarded second-order logic (GSO, which allows quantification over sets of edges in addition to sets of vertices) and in Δ^1_2 (and even Δ^1_1 on finite graphs).

We assume that the reader is familiar with the most important logics used in descriptive complexity theory and verification, such as monadic second-order logic MSO, the modal μ-calculus L_μ and the least fixed-point logic LFP. However, we will survey in Section 2 the facts about the logics that we will use.

2 Background from Logic

We assume that the reader is familiar with first-order logic (FO), second-order logic (SO) and monadic second-order logic (MSO), the extension of first-order logic by second-order quantification $\exists X$ and $\forall X$ over *sets* of elements of the structure, on which the formula is evaluated. In contrast to SO where quantification over arbitrary relations (or functions) is admitted, MSO is a much more manageable formalism; in particular it is decidable on many interesting classes of structures (on trees, in particular) and amenable to automata-based methods. We also refer below to fragments of SO defined by bounded quantifier-alternation. In particular, Σ^1_i consists of those formulae of SO in prenex-normal form with i alternations of second-order quantifiers, so that the first quantifier is existential. Π^1_i is defined analogously, for formulae starting with a universal

quantifier. Finally, Δ^1_i (which is not defined syntactically) is the class of those properties that are definabable by both a Σ^1_i and a Π^1_i formula.

Guarded second-order logic. An interesting extension of MSO is *guarded second-order logic* (GSO), which admits second-order quantification on relations of any arity, provided that they are guarded. Informally, a relation is guarded on a structure $\mathfrak{A}$ if it only consists of tuples whose elements co-exist in some atomic fact of $\mathfrak{A}$. On graphs (V, E), possibly with colourings of nodes and edges, the guarded relations are precisely the relations $R \subseteq V^k$ that only contain tuples $(a_1, \ldots, a_k)$, for which either $a_1 = a_2 = \cdots = a_k$, or $\{a_1, \ldots, a_k\} = \{u, v\}$ for some edge $(u, v) \in E$. It is not difficult to see that GSO on (coloured) graphs is equivalent to the variant of monadic second-order logic that admits second-order quantification not only over sets of nodes, but also over sets of edges. Further, guarded second-order logic, on any class of structures, is equivalent to monadic second-order logic on the associated incidence graphs [13].

An example showing that GSO is indeed more expressive than MSO, is Hamiltonicity. It is well-known (see e.g. [10]) that Hamiltonicity is not MSO-definable, but it is GSO-definable by the formula

$$(\exists H \subseteq E)\big(\forall x\,(\exists^{=1} y\, Hxy \;\wedge\; \exists^{=1} y\, Hyx) \wedge \forall X\,\big[\big(\exists x\, Xx \wedge \forall x \forall y (Hxy \wedge Xx \rightarrow Xy)\big) \rightarrow \forall x\, Xx\big]\big)$$

which, evaluated on a graph $G = (V, E)$ says that there exists an $H \subseteq E$ with unique successors and predecessors such that (V, H) is connected. This means that G has a Hamilton cycle.

Least fixed point logic. Least fixed-point logic, denoted LFP, extends first order logic by least and greatest fixed points of definable relational operators. We will briefly recall some basic definitions here. For a more extensive introduction to LFP, we refer to [12].

Every formula $\psi(R, \overline{x})$, where R is a relation symbol of arity k and $\overline{x}$ is a tuple of k variables, defines, for any structure $\mathfrak{A}$ of appropriate vocabulary, an update operator $F : \mathcal{P}(A^k) \rightarrow \mathcal{P}(A^k)$ on the class of k-ary relations over the universe A of $\mathfrak{A}$, namely $F : R \mapsto \{\overline{a} : (\mathfrak{A}, R) \models \psi(R, \overline{a})\}$. If ψ is positive in R, that is, if every occurrence of R falls under an even number of negations, this operator is monotone in the sense that $R \subseteq R'$ implies $F(R) \subseteq F(R')$. It is well known that every monotone operator has a least fixed point and a greatest fixed point, which can be defined as the intersection and union, respectively, of all fixed points, but which can also be constructed by transfinite induction.

LFP is defined by adding to the syntax of first order logic the following *fixed point formation rule:* If $\psi(R, \overline{x})$ is a formula with a relational variable R occurring only positively and a tuple of first-order variables $\overline{x}$, and if $\overline{t}$ is a tuple of terms (such that the lengths of $\overline{x}$ and $\overline{t}$ match the arity of R), then $[\mathbf{lfp}\, R\overline{x}\,.\,\psi](\overline{t})$ and $[\mathbf{gfp}\, R\overline{x}\,.\,\psi](\overline{t})$ are also formulae, binding the occurrences of the variables R and $\overline{x}$ in ψ.

The semantics of least fixed-point formulae in a structure $\mathfrak{A}$, providing interpretations for all free variables in the formula, is the following: $\mathfrak{A} \models [\mathbf{lfp}\, R\overline{x}\,.\,\psi](\overline{t})$

if $\overline{t}^{\mathfrak{A}}$ belongs to the least fixed point of the update operator defined by ψ on $\mathfrak{A}$. Similarly for greatest fixed points.

Note that in formulae $[\mathbf{lfp}\, R\overline{x}\,.\,\psi](\overline{t})$ one may allow ψ to have other free variables besides $\overline{x}$.

The duality between least and greatest fixed point implies that for any ψ,

$$[\mathbf{gfp}\, R\overline{x}\,.\,\psi](\overline{t}) \equiv \neg[\mathbf{lfp}\, R\overline{x}\,.\,\neg\psi[R/\neg R]](\overline{t}).$$

Using this duality together with de Morgan's laws, every LFP-formula can be brought into *negation normal form*, where negation applies to atoms only.

Other fixed-point logics. We briefly consider two other fixed-point logics that have been studied in the context of finite model theory: PFP, the logic of partial fixed points and NFP, the logic of nondeterministic fixed points. We omit a detailed account of their syntax and semantics here and refer the interested reader to [9]. Here we just note the following facts that will be useful. In terms of expressive power on finite structures, PFP subsumes NFP, which in turn subsumes LFP. On finite *ordered* structures, PFP captures exactly the properties decidable in PSPACE and NFP captures the polynomial hierarchy, just as LFP captures PTIME. In the absence of order, it is known that the logics are strictly weaker than the corresponding complexity classes. However, it is also known that the question of the separation of the expressive power of these logics is still equivalent to the separation of the corresponding complexity classes (see [2,1]).

3 Parity Games

Basic definitions. A parity game is specified by a directed graph $G = (V, E)$, with a partition $V = V_0 \cup V_1$ of the nodes into positions of Player 0 and positions of Player 1, and a priority function $\Omega : V \to \omega$. In case $(v, w) \in E$ we call w a successor of v and we denote the set of all successors of v by vE. A *play* in $\mathcal{G}$ is an infinite path $v_0 v_1 \ldots$ formed by the two players starting from a given initial position v_0. Whenever the current position v_i belongs to V_0, then Player 0 chooses a successor $v_{i+1} \in v_i E$, if $v_i \in V_1$, then $v_{i+1} \in v_i E$ is selected by Player 1. An infinite play $\pi = v_0 v_1 \ldots$ is won by Player 0 if the least priority appearing infinitely often in π is even or no priority appears infinitely often (which may only happen if the range of Ω is infinite). A finite play $\pi = v_0 v_1 \ldots v_n$ is won by Player 0 iff $v_n \in V_1$.

A *(deterministic) strategy* for Player σ is a partial function $f : V^* V_\sigma \to V$ that assigns to finite paths through $\mathcal{G}$ ending in a position $v \in V_\sigma$ a successor $w \in vE$. A play $v_0 v_1 \cdots \in V^\omega$ is *consistent* with f if, for each initial segment $v_0 \ldots v_i$ with $v_i \in V_\sigma$, we have that $v_{i+1} = f(v_0 \ldots v_i)$. We say that such a strategy f is winning from position v_0 if every play that starts at v_0 and that is consistent with f is won by Player σ. The *winning region* of Player σ, denoted W_σ, is the set of positions from which Player σ has a winning strategy. A game $\mathcal{G}$ is *determined* if $W_0 \cup W_1 = V$, i.e., if from each position one of the two players has a winning strategy.

Positional determinacy and complexity. Winning strategies can be rather complicated. However, for certain games, including parity games, it suffices to consider *positional strategies*, which are strategies that depend only on the current position, not on the history of the play. A game is *positionally determined*, if it is determined, and each player has a positional winning strategy on his winning region.

It has been proved independently in [11] and [19] that parity games with a finite game graph are positionally determined. The result is extended to infinite game graphs with finitely many priorities in [25]. This has been further extended in [14] to parity games with $\mathrm{rng}(\Omega) = \omega$.

Theorem 1. *Every parity game is positionally determined.*

In a parity game $\mathcal{G} = (V, V_0, V_1, E, \Omega)$, a positional strategy for Player σ, defined on $W \subseteq V$, can be represented by a subgraph $H = (W, S) \subseteq (V, E)$ such that there is precisely one outgoing S-edge from each node $v \in V_\sigma \cap W$ and $vS = vE$ for each node $v \in V_{1-\sigma} \cap W$. On a finite game graph, such a strategy is winning on W if, and only if, the least priority on every cycle in (W, S) has the same parity as σ.

Hence, given a finite parity game $\mathcal{G}$ and a positional strategy (W, S) it can be decided in polynomial time, whether the strategy is winning on W. To decide winning regions we can therefore just guess winning strategies, and verify them in polynomial time.

Corollary 2. *Winning regions of parity games (on finite game graphs) can be decided in* NP $\cap$ Co-NP.

In fact, Jurdziński [16] proved that the problem is in UP $\cap$ Co-UP, where UP denotes the class of NP-problems with unique witnesses. The best known deterministic algorithm has complexity $n^{O(\sqrt{n})}$) [18]. For parity games with a number d of priorities Jurdziński's progress measure lifting algorithm [17] computes winning regions in time $O(dm \cdot (2n/d)^{d/2})$, where m is the number of edges, giving a polynomial-time algorithm when d is bounded.

Parity games as relational structures. To represent parity games as relational structures, so as to make them amenable to a study of logical definability, we consider two different conventions.

For fixed d, we consider the class of parity games with $\mathrm{rng}(\Omega) \subseteq \{0, \ldots, d-1\}$, to be given as structures

$$(V, V_0, V_1, E, P_0, \ldots, P_{d-1})$$

where E is a binary relation and $V_0, V_1, P_0, \ldots, P_{d-1}$ are all unary, with $V = V_0 \cup V_1 = \bigcup_i P_i$, V_0 disjoint from V_1 and the P_i pairwise disjoint. Note that this class of structures (denoted $\mathcal{PG}_d$) is axiomatisable in first-order logic.

On the other hand, to consider the class of parity games with an unbounded number of priorities, we consider them as structures

$$(V, V_0, V_1, E, \prec, \mathrm{Odd})$$

where E, V_0 and V_1 are as before, where $u \prec v$ means that u has a smaller priority than v, and Odd is the set of nodes with an odd priority. We denote this class of structures by $\mathcal{PG}$.

Lemma 3. *The class $\mathcal{PG}$ is axiomatisable in* LFP*, but not in first-order logic.*

Proof. To verify that a structure $\mathcal{G} = (V, V_0, V_1, E, \prec, \mathrm{Odd})$ is a parity game, we have to check that

(1) V is the disjoint union of V_0 and V_1,
(2) $\prec$ is a pre-order on V,
(3) Odd is a union of equivalence classes with respect to the equivalence relation $\approx$ associated with $\prec$ (with $u \approx v$ if $u \not\prec v$ and $v \not\prec u$),
(4) the linear order induced by $\prec$ on the equivalence classes has some order type $\alpha \leq \omega$.

Clearly, (1) – (3) are first-order properties, and it is well-known that order types $\alpha \leq \omega$ can be distinguished from other order types in LFP, but not in first-order logic. □

In each case, when we say that winning regions are definable in a logic L, we mean that there is a formula φ of L such that for any structure $G \in \mathcal{PG}$ (resp. $\mathcal{PG}_d$), φ is true in exactly those nodes in G from which Player 0 has a winning strategy.

Parity Games and LFP. Consider a structure $\mathfrak{A}$ and an LFP-sentence ψ which we may assume to be in negation normal form, without parameters, and *well-named*, in the sense that every fixed-point variable is bound only once. The model checking game $\mathcal{G}(\mathfrak{A}, \psi)$ is a parity game whose positions are formulae $\varphi(\overline{a})$ such that $\varphi(\overline{x})$ is a subformula of ψ, and $\overline{a}$ is a tuple of elements of $\mathfrak{A}$, interpreting the free variables of φ. The initial position is ψ.

Player 0 (Verifier) moves at positions associated to disjunctions and to formulae starting with an existential quantifier. From a position $\varphi \vee \vartheta$ she moves to either φ or ϑ and from a position $\exists y\, \varphi(\overline{a}, y)$ she can move to any position $\varphi(\overline{a}, b)$ for $b \in A$. In addition, Verifier is supposed to move at atomic false positions, i.e., at positions φ of form $a = a'$, $a \neq a'$, $R\overline{a}$, or $\neg R\overline{a}$ (where R is *not* a fixed-point variable) such that $\mathfrak{A} \models \neg\varphi$. However, positions associated with these atoms do not have successors, so Verifier loses at atomic false positions. Dually, Player 1 (Falsifier) moves at conjunctions and universal quantifications, and loses at atomic true positions. In addition, there are positions associated with fixed-point formulae $[\mathbf{fp}\, T\overline{x}\,.\, \varphi(T, \overline{x})](\overline{a})$ and with fixed-points atoms $T\overline{x}$, for fixed-point variables T. At these positions there is a unique move (by Falsifier, say) to the formula defining the fixed point. For a more formal definition, recall that as ψ is well-named, for any fixed-point variable T in ψ there is a unique subformula $[\mathbf{fp}\, T\overline{x}\,.\, \varphi(T, \overline{x})](\overline{a})$. From position $[\mathbf{fp}\, T\overline{x}\,.\, \varphi(T, \overline{x})](\overline{a})$ Falsifier moves to $\varphi(T, \overline{a})$, and from $T\overline{b}$ he moves to $\varphi(T, \overline{b})$.

The priority labelling assigns even priorities to **gfp**-atoms and odd priorities to **lfp**-atoms. Further, if T, T' are fixed-point variables of different kind with

T' depending on T (which means that T occurs free in the formula defining T'), then T-atoms get lower priority than T'-atoms. All remaining positions, not associated with fixed-point variables, receive the least important priority (i.e. the one with the highest value) As a result, the number of priorities in the model checking games equals the alternation depth of the fixed-point formula plus one. For more details and explanations, and for the proof that the construction is correct, see e.g. [12,24].

Theorem 4. *Let ψ be an* LFP*-sentence and $\mathfrak{A}$ a relational structure. Then, $\mathfrak{A} \models \psi$ if, and only if, Player 0 has a winning strategy for the parity game $\mathcal{G}(\mathfrak{A}, \psi)$.*

Moreover, for every structure $\mathfrak{A}$ with at least two elements, and every formula $\varphi(\overline{x}) \in$ LFP the model checking game $\mathcal{G}(\mathfrak{A}, \varphi)$ is first-order interpretable in $\mathfrak{A}$.

Parity games and the modal μ-calculus. It is well-known that there is a very close relationship between parity games and the modal μ-calculus L_μ. First of all, parity games are the model checking games for L_μ. This means that given any formula $\psi \in L_\mu$ and a transition system $\mathcal{K}$, one can build a parity game $\mathcal{G}(\mathcal{K}, \psi)$ which is essentially the product of $\mathcal{K}$ and ψ, such that ψ is true in $\mathcal{K}$ at node v if, and only if, Player 0 has a winning strategy for $\mathcal{G}(\mathcal{K}, \psi)$ starting from the position (v, ψ).

In the other direction, for any fixed d, the winning regions of parity games in $\mathcal{PG}_d$ are definable by a μ-calculus formula

$$\mathrm{Win}_d = \nu X_0 \mu X_1 \nu X_2 \ldots \lambda X_{d-1} \bigvee_{j=0}^{d-1} \big((V_0 \wedge P_j \wedge \Diamond X_j) \vee (V_1 \wedge P_j \wedge \Box X_j) \big).$$

In this formula, the fixed-point operators alternate between ν and μ, and hence $\lambda = \nu$ if d is odd, and $\lambda = \mu$ if d is even.

As a consequence, an efficient algorithm for solving parity games would also solve the model-checking problem for L_μ.

Theorem 5. *For every $d \in \mathbb{N}$, the formula* Win_d *defines the winning region of Player 0 in parity games with priorities* $0, \ldots, d-1$.

In general, formulae of L_μ are hard to read, especially if they have many alternations between least and greatest fixed points, but here we have an elegant argument based on model checking games. Indeed, given any parity game $\mathcal{G} \in \mathcal{PG}_d$, with initial position v, let $\mathcal{G}^*$ be the model-checking game for the formula Win_d on $\mathcal{G}$ so that Player 0 has a winning strategy for $\mathcal{G}^*$ if, and only if, $\mathcal{G}, v \models \mathrm{Win}_d$. It turns out that the game $\mathcal{G}^*$ is essentially the same as the original game $\mathcal{G}$. More precisely, $\mathcal{G}^*$ reduces to $\mathcal{G}$ if we eliminate, for each player, certain moves that would give the opponent the opportunity to win immediately, and if we contract successive moves of the same player into a single move. In particular, Player 0 wins $\mathcal{G}$ if and only if she wins $\mathcal{G}^*$, which is the case if, and only if, the formula Win_d is true at $\mathcal{G}, v$. We refer to [12, Section 3.3.6] for details.

The formulae Win_d also play an important role in the study of the alternation hierarchy of the modal μ-calculus. Clearly, Win_d has alternation depth d and it has been shown that this can not be avoided. As a consequence the alternation hierarchy of the μ-calculus is strict [7,3]. We will need a slightly stronger formulation of this result, for parity games on finite and strongly connected graphs. This easily follows from the general result by the finite model property of the μ-calculus and by a straightforward reduction to strongly connected games.

Theorem 6. *Winning regions in parity games in $\mathcal{PG}_d$ are not definable by formula in the μ-calculus with alternation depth $< d$, even under the assumption that the game graphs are finite and strongly connected.*

4 Definability of Parity Games on Arbitrary Game Graphs

4.1 Definability in Fragments of Second-Order Logic

Which fragments of second-order logic winning are powerful enough to define the winning regions of parity games? In particular, are winning regions definable in monadic second-order logic? While we cannot answer this question in general, we show that it is a direct consequence of positional determinacy that winning regions are GSO-definable. By means of results due to Courcelle, this implies MSO-definability on a number of interesting classes of parity games.

Theorem 7. *Winning regions in $\mathcal{PG}$ are definable in* GSO*, and also in Δ^1_2*

Proof. The same formula can be used to demonstrate both results. Let $z \equiv p$ stand for $\neg(z \prec p) \wedge \neg(p \prec z)$ which says that z and p have the same priority Given a binary relation S, we let

$$\varphi(S,w,p) := \mathbf{gfp}\, Xx\,.\, \mathbf{lfp}\, Yy\,.\, \mathbf{gfp}\, Zz\,.\, \begin{bmatrix} (z \prec p \wedge \forall u(Szu \to Xu)) \vee \\ (z \equiv p \wedge \forall u(Szu \to Yu)) \vee \\ (p \prec z \wedge \forall u(Szu \to Zu)) \end{bmatrix} (w)(x)(y)$$

We claim that this formula defines the set of pairs (w,p) such that every infinite S-path from w either contains infinitely many nodes z with $z \prec p$, or only finitely many nodes with $z \equiv p$.

To see this, consider the model checking game for this formula, played on a graph $(V,S,\prec)$ with fixed nodes w and p. Let $\alpha(z)$ be the subformula inside the fixed point definitions. At position $\alpha(z)$ in the model checking game, Player 0 chooses the right disjunct, according to the priority of z, and then Player 1 takes the game from z to an S-successor u of z; via Xu, Yu or Zu the play then proceeds to position $\alpha(u)$.

Thus a play in the model checking game essentially amounts to the choice of an S-path from w by Player 1. Such a play is won by Player 0 if the outermost fixed point variable seen infinitely often is either X or Z. But this is the case if

either a priority $y \prec p$ is seen infinitely often or, if this is not the case, priority p occurs only finitely often on the path.

Thus, the formula

$$\psi(S, w) := \forall p(\mathrm{Odd}(p) \rightarrow \varphi(S, w, p))$$

defines those elements w such that on every infinite S-path from w the lowest priority occurring infinitely often is even. We are now ready to write a formula that defines winning regions of Player 0 for games in $\mathcal{PG}$:

$$(\exists S \subseteq E)[\forall u(V_0 u \rightarrow \exists v Suv) \wedge \forall u \forall v(V_1 u \wedge Euv \rightarrow Suv) \wedge \psi(S, w)]$$

This asserts the existence of a set S of edges which includes at least one outgoing edge for every vertex in V_0 and all the outgoing edges for every vertex in V_1 and such that for this set S, w satisfies ψ. In other words, this formula is true at a vertex w just in case Player 0 has a positional winning strategy from w. Since we know that Player 0 has a positional winning strategy whenever she has any strategy, this is sufficient.

It is easily seen that this formula is in GSO as the binary second-order quantifier on the outside is explicitly guarded and the three occurrences of a fixed-point operator are all monadic and could be therefore replaced by monadic quantifiers.

To see that this also yields a Δ^1_2 definition, we note that any formula of LFP (and hence, in particular, ψ) can be written both as a Σ^1_2 and a Π^1_2 formula (see [9] for a proof). Thus, replacing ψ with its Σ^1_2 equivalent we obtain a Σ^1_2 definition of the winning regions for Player 0. By symmetry, we can also define the winning regions for Player 1 in Σ^1_2 and thereby obtain a Π^1_2 definition for the winning regions for Player 0. □

Although GSO is more expressive than MSO in general, there are a number of classes of graphs where the two logics are equivalent. This collapse occurs in particular over graphs of bounded degree, graphs of bounded tree-width, planar graphs and graphs with an excluded minor. A general result covering and generalising all these graph classes has been proved in [8].

Theorem 8 (Courcelle). GSO *collapses to* MSO *on every class $\mathcal{C}$ of graphs that is closed under taking subgraphs and contains only k-sparse graphs, for some $k \in \mathbb{N}$, (i.e. $|E| \leq k|V|$ for every graph $G = (V, E) \in \mathcal{C}$).*

The GSO formula we construct in the proof of Theorem 7 contains monadic quantifiers and one second-order quantifier that is guarded by the relation E. In particular, the order $\prec$ does not appear as a guard. From the proof of Courcelle's theorem above, it then follows that this formula is equivalent to an MSO formula on any class of parity games satisfying the sparsity condition above with respect to the number of edges of the game. On each such class, winning regions of parity games are therefore MSO-definable. However, in general, there seems to be no way to eliminate the quantification over a set of edges and we conjecture that this winning regions are not MSO definable. It may be noted that it is possible to reduce parity games on general graphs to games on graphs with outdegree 2

(see, for instance, [17]). While this reduction can be carried out in polynomial time, it involves a quadratic blow-up in the number of vertices and it seems unlikely that it preserves MSO definability.

4.2 Non-definability in Least Fixed-Point Logic

We now show that winning regions of parity games are not definable in LFP when the game graph may be infinite.

Theorem 9. *Winning regions in $\mathcal{PG}$ are not definable in* LFP*, even under the assumptions that the game graph is countable and the number of priorities is finite.*

Proof. Suppose that $\mathrm{Win}(x) \in \mathrm{LFP}$ defines the winning region of Player 0 on $\mathcal{PG}$. We use this formula to solve the model checking problem for LFP on $\mathfrak{N} = (\omega, +, \cdot)$. Recall that, for any $\varphi(x) \in \mathrm{LFP}$, we have a parity game $\mathcal{G}(\mathfrak{N}, \varphi)$ such that, for all n

$$\mathfrak{N} \models \varphi(n) \quad \Longleftrightarrow \quad \mathcal{G}(\mathfrak{N}, \varphi) \models \mathrm{Win}(v_n)$$

(where v_n is the initial position associated with $\varphi(n)$)

Further, the model checking game $\mathcal{G}(\mathfrak{N}, \varphi)$ is first-order interpretable in $\mathfrak{N}$. Hence the formula $\mathrm{Win}(x)$ is mapped, via a first-order translation $\mathfrak{I}_\varphi$, into another LFP-formula $\mathrm{Win}_\varphi(x)$ such that

$$\mathcal{G}(\mathfrak{N}, \varphi) \models \mathrm{Win}(v_n) \quad \Longleftrightarrow \quad \mathfrak{N} \models \mathrm{Win}_\varphi(n).$$

Note that the first-order translation $\mathrm{Win}(x) \mapsto \mathrm{Win}_\varphi(x)$ depends on φ, but does not increase the alternation depth. Hence, on arithmetic, every formula $\varphi(x)$ would be equivalent to one of fixed alternation depth:

$$\mathfrak{N} \models \varphi(n) \quad \Longleftrightarrow \mathfrak{N} \models \mathrm{Win}_\varphi(n).$$

However, it is known that the alternation hierarchy of LFP on arithmetic is strict. □

5 Definability of Parity Games on Finite Graphs

On finite game graphs, the definability issues are different and closely related to complexity. First of all, we observe that winning positions in $\mathcal{PG}$ are definable in Δ^1_1 rather than just Δ^1_2. This is a simple consequence of the fact that the problem of solving parity games is in NP ∩ Co-NP, combined with Fagin's characterisation of NP as the properties that are Σ^1_1 definable.

One of the most interesting questions is whether the winning regions are definable in fixed point logics such as LFP or the μ-calculus. We first observe that the μ-calculus is not sufficient.

5.1 Non-definability in the μ-Calculus

There is a little subtlety involved in considering a μ-calculus on $\mathcal{PG}$, since the priority function is encoded via the pre-order $\prec$. In modal logics, binary relations are handled via modal operators, hence a μ-calculus on $\mathcal{PG}$ would have, for instance, formulae of form $\langle\prec\rangle\varphi$ that hold at a node x just in case φ is true at some node y with lower priority than x (we view $\prec$-edges as going downwards). Further it would be natural to admit also the reverse modality for $\succ$ and possibly others such as $\preceq, \sim, \succeq$ or the associated successor relations. In any case, the precise definition does not really matter, since no formula in any such μ-calculus can define the winning regions of parity games in $\mathcal{PG}$.

Indeed assume that such a formula ψ exists, and let m be its alternation level. For any fixed number d, we can translate ψ into a formula ψ_d of the usual μ-calculus on structures $\mathcal{PG}_d$ which is equivalent to ψ on (strongly connected) parity games with at most d priorities. To define ψ_d we just replace every subformula of form $\langle\preceq\rangle\varphi$ by

$$\bigvee_{0\leq i<j<d} P_j \wedge \mu X.((P_i \wedge \varphi) \vee \Diamond X)$$

which says, for games in $\mathcal{PG}_d$, that from the current node, there is a reachable node of lower priority at which φ is true. Similar constructions work for other modal operators. Note that this translation increases the alternation level of ψ at most by one.

Now take any strongly connected parity game $\mathcal{G} \in \mathcal{PG}$ with priorities $< d$ and let $\mathcal{G}'$ be its presentation as a structure in $\mathcal{PG}_d$. Obviously $\mathcal{G}, v \models \psi$ if, and only if $\mathcal{G}', v \models \psi_d$. But this means that, for any d, the winning regions of parity games with d priorities, on strongly connected finite graphs, can be expressed by a μ-calculus formula with alternation level $m + 1$. This contradicts Theorem 6.

Theorem 10. *Winning regions of parity games in $\mathcal{PG}$ (on finite graphs) are not definable in the modal μ-calculus.*

5.2 Definability in Fixed-Point Logics

We now turn to the least fixed point logic LFP. Clearly, a proof that winning regions of parity games in $\mathcal{PG}$ are LFP-definable would imply that parity games are solvable in polynomial time. We show that also the converse direction holds, despite the fact that LFP is weaker than PTIME. To do so, we will use a result due to Martin Otto [23] that the multi-dimensional μ-calculus, which is a fragment of LFP, captures precisely the *bisimulation-invariant* part of PTIME. See also [12, Section 3.5.3] for an exposition of this result.

Winning positions in parity games are of course invariant under the usual notion of bisimulation (e.g. as structures in $\mathcal{PG}_d$). However, to apply Otto's Theorem for parity games with an unbounded number of priorities, we have to consider bisimulation on structures of the form $\mathcal{G} = (V, V_0, V_1, E, \prec, \mathrm{Odd})$. Let $\tau = \{V_0, V_1, E, \prec, \mathrm{Odd}, v\}$ be the vocabulary of parity games with a starting

node, and let $\mathrm{Str}(\tau)$ denote the class of all structures of this vocabulary. If we have two such structures that are indeed parity games, then bisimilarity as τ-structures coincides with the usual notion of bisimilarity in $\mathcal{PG}_d$, for appropriate d. However, not all structures in $\mathrm{Str}(\tau)$ are parity games, and the class of parity games is not closed under bisimulation. An efficient procedure for deciding whether a structure is bisimilar to a parity game is to compute its quotient under bisimulation and check whether it is a parity game.

For a structure $(\mathcal{G}, v) \in \mathrm{Str}(\tau)$ consider the bisimulation relation $a \sim b$ on elements of $\mathcal{G}$ defined with respect to the binary relations E, $\prec$ and $\prec^{-1}$. That is to say $\sim$ is the largest relation satisfying:

- if $a \sim b$ then a and b agree on the unary relations V_0, V_1 and Odd;
- for every $x \in aE$ there is a $y \in bE$ such that $x \sim y$, and conversely;
- for every x with $a \prec x$ there is a y with $b \prec y$ and $x \sim y$ and conversely; and finally
- for every $x \prec a$ there is a $y \prec b$ such that $x \sim y$, and conversely.

We write $(\mathcal{G}, v)^{\sim}$ for the *bisimulation quotient* of $(\mathcal{G}, v)$, i.e. the structure whose elements are the equivalence classes in $\mathcal{G}$ with respect to $\sim$ with the relations $V_0, V_1, E, \prec, \mathrm{Odd}$ defined in the natural way and $[v]$ as the starting vertex.

Lemma 11. *A structure $(\mathcal{G}, v) \in \mathrm{Str}(\tau)$ is bisimilar to a parity game if, and only if, its bisimulation quotient is a parity game, i.e. $(\mathcal{G}, v)^{\sim} \in \mathcal{PG}$.*

Proof. The direction from right to left is obvious. For the other direction, it suffices to establish that the bisimulation quotient of a parity game is itself a parity game, since the quotient of a structure bisimilar to $(\mathcal{G}, v)$ is isomorphic to the quotient of $(\mathcal{G}, v)$. But, it is easily verified that each of the four conditions given in the proof of Lemma 3 is preserved under taking quotients. □

Theorem 12. *Let $\mathcal{C}$ be any class of parity games on finite game graphs, such that winning positions on its bisimulation quotients are decidable in polynomial time. Then, on $\mathcal{C}$, winning positions are* LFP*-definable.*

Proof. Let $\mathrm{Win}\mathcal{C}$ be the class of parity games $(\mathcal{G}, v)$, such that $(\mathcal{G}, v) \in \mathcal{C}$, and Player 0 wins from initial position v. It suffices to construct a bisimulation-invariant class X of structures $(\mathcal{H}, u)$ such that

(1) X is decidable in polynomial time.
(2) $X \cap \mathcal{C} = \mathrm{Win}\mathcal{C}$.

Indeed, by Otto's Theorem X is then definable by an LFP-formula $\psi(x)$, such that, given any parity game $(\mathcal{G}, v) \in \mathcal{C}$ we have

$$\mathcal{G}, v \in \mathrm{Win}\mathcal{C} \iff \mathcal{G}, v \in X \iff \mathcal{G} \models \psi(v).$$

By assumption, there exists a polynomial time algorithm A which, given a parity game $(\mathcal{G}, v) \in \mathcal{C}^{\sim}$, decides whether Player 0 wins $\mathcal{G}$ from v. It is not important what the algorithm returns for quotients outside $\mathcal{C}^{\sim}$, as long as it is isomorphism-invariant and halts in polynomial time. Finally, let B be the algorithm which, given

any finite structure in $\mathrm{Str}(\tau)$, first computes its bisimulation quotient, and then applies algorithm A.

Clearly B is a polynomial time algorithm, since bisimulation quotients are efficiently computable. Further the class X of structures accepted by B is invariant under bisimulation. Indeed, let $\mathcal{H}$ and $\mathcal{H}'$ be two bisimilar structures. Then their bisimulation quotients are isomorphic and are therefore either both accepted or both rejected by A. Finally, $X \cap C = \mathrm{Win}\mathcal{C}$. Indeed, given a parity game $\mathcal{G}, v \in \mathcal{C}$, then it has the same winner as its bisimulation quotient which is therefore correctly decided by the algorithm B. □

Corollary 13. *On the class $\mathcal{PG}$ of all finite parity games, winning regions are* LFP*-definable if, and only if, they are computable in polynomial time.*

We can deduce more from this construction. The proof of Otto's theorem relies on an LFP-definable interpretation of an *ordered* bisimulation quotient $\mathcal{G}^{\sim}$ within $\mathcal{G}$. Now, on ordered structures, every problem decidable in PSPACE is definable in PFP and every problem in NP is definable in NFP. Since winning regions on the class $\mathcal{PG}$ are computable in these complexity classes, we can compose the formulas of PFP and NFP with the LFP interpretation to obtain formulas which do not require an order. Thus we have the following theorem.

Theorem 14. *On the class $\mathcal{PG}$ of all finite parity games, winning regions are definable in the logics* PFP *and* NFP*.*

5.3 Restricted Classes

While the question of whether or not winning regions in finite parity games can be computed in polynomial-time remains open, many restricted classes of games have been investigated on which polynomial-time algorithms for computing winning regions (and winning strategies) are known. These include parity games with a bounded number of priorities [17], games where even and odd cycles do not intersect, solitaire games and nested solitaire games [5], and parity games of bounded tree width [20], bounded entanglement [6], bounded DAG-width [4,21], bounded Kelly-width [15], or bounded clique width [22].

In view of Theorem 12 the question arises which of these classes are closed under taking bisimulation quotients. This is not the case for games of bounded tree width or of bounded clique width, since one can easily construct trees whose bisimulation quotients contain arbitrarily large grids.

However, it is the case for games of bounded entanglement, and possibly also for other classes defined by specific measures for directed graphs.

Entanglement is a parameter for the complexity of finite directed graphs which measures to what extent the cycles of the graph are intertwined. It is defined by means of a game played by a thief against k detectives according to the following rules. Initially the thief selects an arbitrary position v_0 of the given graph $G = (V, E)$ and the detectives are outside of G. In any move the detectives may either stay where they are, or place one of them on the current position v of the thief. The thief tries to escape by moving to a successor $w \in vE$ that is not

occupied by a detective. If no such position exists, the thief is caught and the detectives have won. Note that the thief sees the move of the detectives before he decides on his own move, and that he has to leave his current position no matter whether the detectives stay where they are or not. The entanglement of G, denoted $\mathrm{ent}(G)$, is the minimal number $k \in \mathbb{N}$ such that k detectives have a strategy to catch the thief on G.

It has been proved in [5] that, for any fixed k, winning positions in parity games with entanglement at most k are computable in polynomial time. Our results complement this by showing that on such classes, winning positions are LFP-definable. By Theorem 12 it suffices to show that the entanglement cannot increase when we move from a directed graph G to its bisimulation quotient $G^{\sim}$.

Lemma 15. *For any directed graph* $\mathcal{G}$, $\mathrm{ent}(G^{\sim}) \leq \mathrm{ent}(G)$.

Proof. We show that whenever the thief has a strategy to escape against k detectives on $G^{\sim}$, he also has a similar strategy on G, by which he stays clear of not only the nodes occupied by the detectives, but also all nodes bisimilar to these. Suppose that the detectives occupy nodes $w_1, \ldots, w_k$ of G and the thief has to move from his current position u, For the corresponding situation on $G^{\sim}$ the thief can move according to his escaping strategy from the bisimulation class $[u]$ to a successor $[v] \in [u]E^{\sim}$ that is not occupied by any of the detectives, i.e. $[v] \neq [w_i]$ for all i. Since there is an edge from $[u]$ to $[v]$ in $G^{\sim}$ there exists a node $v' \sim v$ with $(u, v') \in E$. In G, the thief moves to any such node $v' \in [v] \cap uE$. Since $[v'] = [v] \neq [w_i]$ for all i, the thief maintains the property that he avoids in G all positions bisimilar to those occupied by a detective. □

Corollary 16. *In any class of parity games of bounded entanglement, the winning positions are definable in* LFP.

6 Conclusions

While the exact computational complexity of computing winning regions in finite parity games remains a much studied open question, we have addressed a related question that has not received much attention—that of the descriptive complexity of parity games, both on finite and arbitrary structures. While we are able to settle the definability questions in many interesting cases, a significant open question that remains is whether winning regions are definable in MSO. Interestingly, the question of whether winning regions are definable in LFP on finite game graphs turns out to be equivalent to the long-standing open question of whether they are computable in PTIME.

References

1. Abiteboul, S., Vardi, M.Y., Vianu, V.: Fixpoint logics, relational machines, and computational complexity. J. ACM 44(1), 30–46 (1997)
2. Abiteboul, S., Vianu, V.: Computing with first-order logic. Journal of Computer and System Sciences 50(2), 309–335 (1995)

3. Arnold, A.: The mu-calculus alternation-depth is strict on binary trees. RAIRO Informatique Théorique et Applications 33, 329–339 (1999)
4. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: Dag-width and parity games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 524–536. Springer, Heidelberg (2006)
5. Berwanger, D., Grädel, E.: Fixed-point logics and solitaire games. Theory of Computing Systems 37, 675–694 (2004)
6. Berwanger, D., Grädel, E.: Entanglement - A measure for the complexity of directed graphs with applications to logic and games. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 209–223. Springer, Heidelberg (2005)
7. Bradfield, J.: The modal μ-calculus alternation hierarchy is strict. Theoretical Computer Science 195, 133–153 (1998)
8. Courcelle, B.: The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. Theoretical Computer Science 299, 1–36 (2003)
9. Dawar, A., Gurevich, Y.: Fixed point logics. Bulletin of Symbolic Logic 8, 65–88 (2002)
10. Ebbinghaus, H.-D., Flum, J.: Finite Model Theory, 2nd edn. Springer, Heidelberg (1999)
11. Emerson, A., Jutla, C.: Tree automata, mu-calculus and determinacy. In: Proc. 32nd IEEE Symp. on Foundations of Computer Science, pp. 368–377 (1991)
12. Grädel, E., et al.: Finite Model Theory and Its Applications. Springer, Heidelberg (2007)
13. Grädel, E., Hirsch, C., Otto, M.: Back and forth between guarded and modal logics. ACM Transactions on Computational Logic 3, 418–463 (2002)
14. Grädel, E., Walukiewicz, I.: Positional determinacy of games with infinitely many priorities. Logical Methods in Computer Science (2006)
15. Hunter, P.: Complexity and Infinite Games on Finite Graphs. PhD thesis, University of Cambridge (2007)
16. Jurdziński, M.: Deciding the winner in parity games is in UP $\cap$ Co-UP. Information Processing Letters 68, 119–124 (1998)
17. 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)
18. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proceedings of ACM-SIAM Proceedings on Discrete Algorithms, SODA 2006, pp. 117–123 (2006)
19. Mostowski, A.: Games with forbidden positions. Technical Report Tech. Report 78, University of Gdansk (1991)
20. Obdrzálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 80–92. Springer, Heidelberg (2003)
21. Obdrzálek, J.: DAG-width - connectivity measure for directed graphs. In: Proceedings of ACM-SIAM Proceedings on Discrete Algorithms, SODA 2006, pp. 814–821 (2006)
22. Obdrzálek, J.: Clique-width and parity games. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 54–68. Springer, Heidelberg (2007)
23. Otto, M.: Bisimulation-invariant Ptime and higher-dimensional mu-calculus. Theoretical Computer Science 224, 237–265 (1999)
24. Stirling, C.: Bisimulation, model checking and other games. Notes for the Mathfit instructional meeting on games and computation. Edinburgh (1997)
25. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theoretical Computer Science 200, 135–183 (1998)

An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games*

Sven Schewe

Universität des Saarlandes and University of Liverpool

Abstract. This paper presents a novel strategy improvement algorithm for parity and payoff games, which is guaranteed to select, in each improvement step, an optimal combination of local strategy modifications. Current strategy improvement methods stepwise improve the strategy of one player with respect to some ranking function, using an algorithm with two distinct phases: They first choose a modification to the strategy of one player from a list of *locally* profitable changes, and subsequently evaluate the modified strategy. This separation is unfortunate, because current strategy improvement algorithms have no effective means to predict the *global effect* of the individual local modifications beyond classifying them as profitable, adversarial, or stale. Furthermore, they are completely blind towards the *cross effect* of different modifications: Applying one profitable modification may render all other profitable modifications adversarial. Our new construction overcomes the traditional separation between choosing and evaluating the modification to the strategy. It thus improves over current strategy improvement algorithms by providing the *optimal improvement* in every step, selecting the best combination of local updates from a superset of all profitable and stale changes.

1 Introduction

Solving parity games is the central and most expensive step in many model checking [1, 2, 3, 4, 5], satisfiability checking [1, 3, 6, 7], and synthesis [8, 9] algorithms. More efficient algorithms for solving parity games will therefore foster the development of performant model checkers and contribute to bringing synthesis techniques to practice. The quest for performant algorithms [1, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] for solving them has therefore been an active field of research during the last decades.

Traditional forward techniques ($\approx O(n^{\frac{1}{2}c})$ [16] for parity games with n positions and c colors), backward techniques ($\approx O(n^c)$ [10, 12, 15]), and their combination ($\approx O(n^{\frac{1}{3}c})$ [25]) provide good complexity bounds. However, these bounds are sharp, and techniques with good complexity bounds [16, 25] frequently display their worst case complexity on practical examples.

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 369–384, 2008.
© Springer-Verlag Berlin Heidelberg 2008

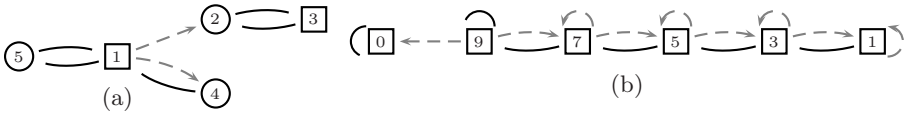


Fig. 1. The examples show situations where ignoring global effects (a) and cross effects between different updates (b) perturb the pivot rule of current strategy improvement algorithms. States of Player 0 and 1 are depicted as boxes and circles, respectively. The current strategy of Player 0 (and all options of Player 1) are depicted as full arrows, the improvement edges of Player 0 are represented by dashed arrows.

Strategy improvement algorithms [17, 18, 19, 20], on the other hand, are fast simplex style algorithms that perform well in practice. While their complexity is wide open, they are often considered the best choice for solving large scale games.

State of the Art. Strategy improvement algorithms are closely related to the simplex algorithm for solving linear programming problems. Strategy improvement algorithms assign a value to each infinite play of a parity or payoff game, and the objective of the two participating players (Player 0 and 1) is to minimize and maximize this value, respectively.

In strategy improvement algorithms for parity and payoff games [17, 18, 19, 20], the memoryless strategies of Player 0 define the corners of a simplex. For each memoryless strategy of Player 0, her opponent has an optimal counter strategy. This pair of strategies defines a pointwise ranking function that assigns to each game position p the value (or rank) $v(p)$ of the play that starts in p.

The two distinguishing differences between strategy improvement techniques compared to the simplex algorithm are a *weak pivot rule* and the option to apply *multiple modifications* in every strategy improvement step.

Weak Pivot Rule. Different to simplex techniques for linear programming problems, current strategy improvement algorithms do not take the *global effect* of a step to an adjacent corner of the simplex into account. When estimating the improvement attached to modifying the strategy locally, they presume that changing the strategy for one game position has no influence on the rank of any other game position. In the situation depicted in Figure 1a, Player 0 can choose between two improvements of her strategy from her position colored by 1; she can either move to the position with color 2, or to the position with color 4. While the latter choice is obviously better (because Player 0 asserts the parity condition), the local analysis considers only the value of the positions for the *old* strategy [17, 18, 19, 20]. A valuation function based on the old strategy, however, will favor the change to the position with color 2, because the dominating color in the infinity set for this position is 3 (for the old strategy), while the dominating color in the infinity set of the position colored by 4 is 5. In the whole, the local analysis alone does not provide much more information than a classification into locally profitable, adversarial, and stale modifications.

Current Strategy Improvement Algs	Game-Based Strategy Improvement Algorithm
1. pick initial strategy	1. pick and evaluate initial strategy
2. evaluate current strategy	2. adjust evaluation, increasing #profitable/stale mods
3. chose from profitable modifications	3. find and evaluate optimal combination of p/s mods
4. goto 2	4. goto 2

Fig. 2. Comparison between traditional strategy improvement algorithms and the proposed optimal improvement algorithm. While current techniques first choose a particular update from profitable modifications and subsequently evaluate it, our novel technique concurrently considers all combinations of profitable and stale modifications.

Multiple Modifications. An advantage of strategy improvement techniques over the simplex algorithm for linear programming problems is the option to consider several locally profitable modifications at the same time [18, 19]. This advantage, however, must be considered with care, because current strategy improvement algorithms are blind towards the *cross effect* between different local updates.

While any combination of profitable changes remains profitable, it frequently happens that applying one modification turns all remaining modifications adversarial. In the small singleton parity game depicted in Figure 1b, Player 0 is only one step away from her optimal strategy. (It suffices to update the strategy in the position with color 9.) All local changes lead to an improvement, but after updating the strategy at the position with color 9, all remaining changes become harmful. Given this shortcoming, it is unclear whether or not simultaneous updates are a step forward for current strategy improvement algorithms.

Contribution. We introduce a strategy improvement algorithm that is based on a reduction to simple update *games*, which can be solved in a single sweep. It provides substantial advantages over current strategy improvement algorithms:

1. *The reduction is more natural.* It reduces solving parity (or mean payoff) games to solving a series of simplified games, where the options of Player 0 are restricted, but not to the extreme of a singleton game. It thus preserves and exploits the game character of the problem to improve a strategy.
2. *The improvements are greater.* The game-based approach allows us to take the global and cross effects of different local modifications to the strategy into account. We thus overcome the blind spot of current strategy improvement algorithms and can make full use of simultaneous modifications.
3. *The game-based analysis is cheaper.* Reductions to graph-based algorithms need to exclude stale cycles. Both, for parity and payoff games, the codomain of the pointwise ranking function needs to be increased by a factor linear in the size n of the game, which raises the estimation for the amount of iterations by a factor of n and slows down the arithmetic operations.

From Graph-Based to Game-Based Updates. The suggested optimal strategy improvement algorithm reduces solving parity games to solving a series of simpler two player games. Turning to a game-based (rather than to a graph-based) approach allows for considering *all combinations* of profitable and stale modifications in every update step, taking all global and cross effects into account.

This advancement is achieved by a novel technique that resolves the separation between choosing and evaluating the modifications to a strategy. Where current strategy improvement algorithms first update the strategy and then evaluate the resulting singleton game, our game-based approach exploits a natural preorder for the evaluation of game positions that allows for simultaneously constructing optimal strategies for both players, such that every game position is considered only once. Following this preorder, the evaluation of each individual position can be reduced to a cheap local analysis.

The intuition for the preorder is that, in most cases, the game-based approach allows for fixing an optimal decision for a game position *after* all of its successors have been reevaluated. If all positions do have unevaluated successors, we can immediately determine the optimal choice for some position of Player 1.

The Ranking Function. We change the rules of parity games by allowing one player, say Player 0, to terminate the game in her positions. This is related to the finite unraveling of mean payoff games [13] and the controlled single source shortest path problem from the reduction of Björklund and Vorobyov [20].

The objective of Player 0 remains to assert an infinite path with even maximal priority in the infinity set. However, we add the natural secondary objective for the case that she has not yet found (or there is no) such strategy. If Player 0 cannot assert such a path, she eventually stops the unraveling of the game, optimizing the finite occurrences of the different priorities, but *disregarding* the number of positions with priority 0. (Disregarding this number leads to a coarser ranking function, and thus to an improved estimation of the number of improvement steps. It also leads to greater improvements by increasing the number of profitable or stale modifications.) Second, if the highest occurring priority is odd, there is no need to keep track of the number of occurrences of this priority. It suffices to store the information that this maximal number occurs on a finite path, resulting again in a coarser ranking function.

For parity games with n positions, m edges, and c colors, the coarser ranking function leads to an improved estimation of the number of updates from the currently best bound $O\big(n\,(\frac{n+c}{c})^{c+1}\big)$ [20] for the number of arithmetic operations needed by strategy improvement algorithms to $O\big(n\,(\frac{n+c}{c})^{c-1}\big)$ for parity games with an even number of colors, and to $O\big(n\,(\frac{n+c}{c})^{c}\big)$ if the numbers of colors is odd, reducing the bound by a factor quadratic and linear in the number of states, respectively. The bound is reduced further by decreasing the discounted cost of arithmetic operations from $O(c)$ to $O(1)$ when the number of iterations is high.

2 Parity Games

A game is composed of a finite arena and an evaluation function. We will first discuss arenas, and then turn to the evaluation functions for parity games.

Arena. A finite arena is a triple $\mathcal{A} = (V_0, V_1, E)$, where V_0 and V_1 are disjoint finite sets of positions, called the positions of Player 0 and 1, and $E \subseteq V_0 \times V_1 \cup V_1 \times V_0$ is a set of edges; that is, $(V_0 + V_1, E)$ is a bipartite directed graph. For

infinite games, the arena is also required not to contain sinks; that is, every position $p \in V = V_0 \cup V_1$ has at least one outgoing edge $(p, p') \in E$.

Plays. Intuitively, a game is played by placing a pebble on the arena. If the pebble is on a position $p_0 \in V_0$, Player 0 chooses an edge $e = (p_0, p_1) \in E$ from p_0 to a position $p_1 \in V_1$ and moves the pebble to p_1. Symmetrically, Player 1 chooses a successor if the pebble is on a position $p_1 \in V_1$. This way, they successively construct an infinite *play* $\pi = p_0 p_1 p_2 p_3 \ldots \in V^\omega$.

Strategies. For a finite arena $\mathcal{A} = (V_0, V_1, E)$, a (memoryless) *strategy* for Player 0 is a function $f : V_0 \to V_1$ that maps every position $p_0 \in V_0$ of Player 0 to a position $v_1 \in V_1$ such that there is an edge $(p_0, p_1) \in E$ from p_0 to p_1. A play is called *f-conform* if every decision of Player 0 in the play is in accordance with f. For a strategy f of Player 0, we denote with $\mathcal{A}_f = (V_0, V_1, E_f)$ the arena obtained from $\mathcal{A}$ by deleting the transitions from positions of Player 0 that are not in accordance with f. The analogous definitions are made for Player 1.

Parity Games. A *parity game* is a game $\mathcal{P} = (V_0, V_1, E, \alpha)$ with arena $\mathcal{A} = (V_0, V_1, E)$ and a surjective coloring function $\alpha : G \cup R \to \mathcal{C} \subset \mathbb{N}$ that maps each position of $\mathcal{P}$ to a natural number. $\mathcal{C}$ denotes the finite set of colors. For technical reasons we assume that the minimal color of a parity game is $0 = min\{\mathcal{C}\}$.

Each play is evaluated by the highest color that occurs infinitely often. Player 0 wins a play $\pi = p_0 p_1 p_2 p_3 \ldots$ if the highest color occurring infinitely often in the sequence $\alpha(\pi) = \alpha(p_0)\alpha(p_1)\alpha(p_2)\alpha(p_3)\ldots$ is even, while Player 1 wins if the highest color occurring infinitely often in $\alpha(\pi)$ is odd.

A strategy f of Player 0 or 1 is called *p-winning* if all f-conform plays starting in p are winning for Player 0 or 1, respectively. A position of $\mathcal{P}$ is p-winning for Player 0 or 1 if Player 0 or 1, respectively, has a p-winning strategy. We call the p-winning positions for Player 0 or 1 the *winning region* of Player 0 or 1, respectively. Parity games are memoryless determined:

Theorem 1. *[11] For every parity game $\mathcal{P}$, the game positions are partitioned into a winning region W_0 of Player 0 and a winning region W_1 of Player 1. Moreover, Player 0 and 1 have memoryless strategies that are p-winning for every position p in their respective winning region.* □

3 Escape Games

Escape games are total reward games that are tailored for the game-based improvement method. They generalize parity games by allowing Player 0 to terminate every play immediately on each of her positions. Technically this is done by extending the arena with a fresh *escape position* $\perp$, which forms a sink of the extended arena, and can be reached from every position of Player 0. Every

play of an *escape game* either eventually reaches the escape position and then terminates, or it is an infinite play in the non-extended arena.

Extended Arena. In an escape game, the finite arena $\mathcal{A} = (V_0, V_1, E)$ is extended to the directed graph $\mathcal{A}' = (V_0, V_1', E')$, which extends the arena $\mathcal{A}$ by a fresh position $\bot$ of Player 1 ($V_1' = V_1 \uplus \{\bot\}$) that is reachable from every position of Player 0 ($E' = E \cup V_0 \times \{\bot\}$). The escape position is a sink in $\mathcal{A}'$.

Finite Plays. Since the escape position $\bot$ is a sink, every play terminates when reaching $\bot$. The set of plays is therefore extended by the finite plays $\pi = p_0 p_1 p_2 p_3 \dots p_n \bot \in V^* \cdot \{\bot\}$.

Escape Games. An *escape game* is a game $\mathcal{E} = (V_0, V_1, E, \alpha)$, where $\mathcal{A} = (V_0, V_1, E)$ is a finite arena, and $\alpha : V \to \mathcal{C} \subset \mathbb{N}$ is a coloring function. An escape game is played on the extended arena $\mathcal{A}' = (V_0, V_1', E')$.

An infinite play $\pi = p_0 p_1 p_2 \dots \in V^\omega$ of an escape game is evaluated to ∞ if the highest color occurring infinitely often is even, and to $-\infty$ otherwise. A finite play $\pi = p_0 p_1 p_2 \dots p_n \bot$ is evaluated by a function $\rho(\pi) : \mathcal{C}_0 \to \mathbb{Z}$ (where $\mathcal{C}_0 = \mathcal{C} \setminus \{0\}$ is the codomain of the coloring function without 0) that maps an element c' of $\mathcal{C}_0$ to the number of positions p_i in π with $i > 0$ that are colored by $c' = \alpha(p_i)$. (Disregarding the color of the first position is technically convenient.)

The potential values of a path are ordered by the obvious alphabetic order $>$ that sets $\rho > \rho'$ if (1) the highest color c' with $\rho(c') \neq \rho'(c')$ is even and $\rho(c') > \rho'(c')$, or (2) if the highest color c' with $\rho(c') \neq \rho'(c')$ is odd and $\rho'(c') > \rho(c')$. Additionally, we define $\infty > \rho > -\infty$. The objective of Player 0 is to maximize this value, while it is the objective of Player 1 to minimize it.

We introduce an operator $\oplus$ for the evaluation of finite paths. For $\mathcal{R} = (\mathcal{C}_0 \to \mathbb{Z}) \cup \{\infty\}$, $\oplus : \mathcal{R} \times \mathcal{C} \to \mathcal{R}$ maps a function ρ and a color c' to the function ρ' that deviates from ρ only by assigning the respective successor $\rho'(c') = \rho(c') + 1$ to c' (and leaves $\rho'(d) = \rho(d)$ for $d \neq c'$). We fix $\infty \oplus c' = \infty$ and $\rho \oplus 0 = \rho$.

Estimations. We introduce *estimations* $v : V' \to \mathcal{R}$ for an escape game $\mathcal{E} = (V_0, V_1', E, \alpha)$ as witnesses for the existence of a memoryless strategy f of Player 0, which guarantees that every f-conform play π starting in some position p is evaluated to $\rho(\pi) \geq v(p)$. Formally, an estimation v has to satisfy the following side conditions:

- $v(\bot) = \mathbf{0}$ ($\mathbf{0}$ denotes the constant function that maps all colors in $\mathcal{C}_0$ to 0),
- for every $p_1 \in V_1$ and every edge $e = (p_1, p_0) \in E$, $v(p_1) \leq v(p_0) \oplus \alpha(p_0)$ holds true,
- for every position $p_0 \in V_0$ there is an edge $e = (p_0, p_1) \in E'$ such that $v(p_0) \leq v(p_1) \oplus \alpha(p_1)$ holds true, and
- Player 0 has a strategy f_∞ that maps every position $p_0 \in V_0$ with $v(p_0) = \infty$ to a position $v_1 = f_\infty(p_0)$ with $v(p_1) = \infty$, and which guarantees that every f_∞-conform play π starting in g is evaluated to $\rho(\pi) = \infty$.

A trivial estimation is simple to construct: we denote with v_0 the estimation that maps the escape position to $v_0(\bot) = \mathbf{0}$, every position $p_0 \in V_0$ to $v_0(p_0) = \mathbf{0}$, and every position $p_1 \in V_1$ of Player 1 to $v_0(p_1) = min\{\mathbf{0} \oplus \alpha(p_0) \mid (p_1, p_0) \in E\}$[1].

Lemma 1. *For every estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ there is a memoryless strategy f for Player 0 such that every f-conform play π starting in any position p satisfies $\rho(\pi) \geq v(p)$.*

Proof. We fix an arbitrary strategy f for Player 0 that agrees with f_∞ on every position $p \in V_0$ with infinite estimation ($v(p) = \infty \Rightarrow f(p) = f_\infty(p)$), and chooses some successor that satisfies $v(p) \leq v\big(f(p)\big) \oplus \alpha\big(f(p)\big)$ otherwise. Every cycle reachable in an f-conform play has nonnegative weight (that is, weight $\mathbf{0} \oplus \alpha(p_0) \oplus \ldots \oplus \alpha(p_n)$ of every cycle $p_0 \ldots p_n p_0$ is $\geq \mathbf{0}$) by construction of f; every infinite f-conform play π is therefore evaluated to $\rho(\pi) = \infty \geq v(p)$.

By induction over the length of finite f-conform plays π that start in some position p, we can show that $\rho(\pi) \geq v(p)$. □

We call an estimation v' an *improvement* of an estimation v if $v'(p) \geq v(p)$ holds for all positions $p \in V$, and we call an improvement *strict* if $v' \neq v$.

For every estimation v, we define the *improvement arena* $\mathcal{A}_v = (V_0, V_1', E_v)$ that contains an edge $e = (p, p')$ if it satisfies $v(p) \leq v(p') \oplus \alpha(p')$ (i.e., $E_v = \{(p, p') \in E' \mid v(p) \leq v(p') \oplus \alpha(p')\}$), and the 0-*arena* $\mathcal{A}_v^0 = (V_0, V_1', E_v^0)$ which contains an edge $e = (p, p') \in E_v$ of the improvement arena if it satisfies (1) $v(p) = v(p') \oplus \alpha(p')$, and, if e originates from a position $p \in V_0$ of Player 0, if additionally (2) no edge $e' = (p, p')$ with $v(p) < v(p') \oplus \alpha(p')$ originates from p ($E_v^0 = \{(p, p') \in E_v \mid v(p) = v(p') \oplus \alpha(p')$ and $p \in V_0 \Rightarrow \forall (p, p') \in E_v . v(p) = v(p') \oplus \alpha(p')\}$).

Attractors. The 0-*attractor* $A \subseteq V$ of a set $F \subseteq V$ of game positions is the set of those game positions from which Player 0 has a strategy to force the pebble into a position in F. The 0-attractor A of a set F can be defined as the least fixed point of sets that contain F, and that contain a position $p \in V_0$ of Player 0 if they contain some successor of p, and a position $p \in V_1$ of Player 1 if p has some successor, and all successors of p are contained in A. The 1-attractor is defined accordingly. Constructing this least fixed point is obviously linear in the number of positions and edges in the arena, and we can fix a memoryless strategy (the attractor strategy) for the respective player to reach F in finitely many steps during this construction.

Lemma 2. *For a given arena $\mathcal{A} = (V_0, V_1, E)$ with n positions and m edges that may contain sinks, and for a set $F \subseteq V$ of game positions, we can compute the respective attractor A of F and a memoryless strategy for Player 0 or 1, respectively, on $A \setminus F$ to reach F in finitely many steps in time $O(m + n)$.* □

[1] Having a simple-to-construct initial estimation is the reason for the restriction to bipartite games. Constructing an initial estimation for general games is not hard, and the improvement algorithm proposed in Section 4 extends to non-bipartite games.

We call an estimation *improvable* if the 1-attractor of the escape position in the 0-arena $\mathcal{A}_v^0$ does not cover all positions that are not estimated to ∞.

Theorem 2. *For every non-improvable estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ Player 1 has a memoryless strategy f' such that every f'-conform play π starting in any position p satisfies $\rho(\pi) \leq v(p)$.*

Proof. We fix a strategy f' for Player 1 that agrees on all positions $V_1 \setminus v^{-1}(\infty)$ with some 1-attractor strategy of the escape position $\perp$ in the 0-arena $\mathcal{A}_v^0$.

For plays starting in some position p that is evaluated to ∞, $\rho(\pi) \leq v(p) = \infty$ holds trivially. For plays starting in some position p that is not evaluated to ∞, we can show by induction over the length of f'-conform plays starting in p that no f'-conform play can reach a position p' that is evaluated to $v(p') = \infty$. By construction of f', every reachable cycle in an f'-conform play that does not reach a position in $v^{-1}(\infty)$ has negative weight (that is, a weight $< \mathbf{0}$), and every infinite f'-conform play which starts in a position p that is not evaluated to ∞ thus satisfies $-\infty = \rho(\pi) < v(p)$.

For every finite f'-conform play π starting in some position p, we can show by induction over the length of π that $\rho(\pi) \leq v(p)$ holds true. □

The non-improvable estimation of an escape game can be used to derive the winning regions ($v^{-1}(\infty)$ for Player 0) and the winning strategy for Player 1 on his winning region in the underlaying parity game. f_∞ defines the winning strategy of Player 0 on her winning region.

4 Solving Escape Games

In this section we introduce a game-based strategy improvement algorithm for the fast improvement of estimations of escape games. Every estimation (for example, the trivial estimation v_0) can be used as a starting point for the algorithm.

Optimal Improvement. The estimations we construct intuitively refer to strategies of Player 0 for the extended arena. (Although estimations are a more general concept; not all estimations refer to a strategy.) The edges of the improvement arena $\mathcal{A}_v = (V_0, V_1, E_v)$ of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ and an estimation v refer to all promising strategy updates, that is, all strategy modifications that *locally* lead to a—not necessarily strict—improvement (profitable and stale modifications). We call an improvement v' of v *optimal* if it is the non-improvable estimation for the restricted escape game $\mathcal{E}_v = (V_0, V_1, E_v, \alpha)$. v' is optimal in the sense that it dominates all other estimations $\widehat{v}$ that refer to strategies of Player 0 for $\mathcal{E}_v$, that is, to strategies that contain only improvement edges. Finding this optimal improvement v' thus relates to solving an *update game* $\mathcal{E}_v$, which deviates from the full escape game $\mathcal{E}$ only by restricting the choices of Player 0 to her improvement edges.

Basic Update Step. Instead of computing the optimal improvement v' of an estimation v directly, we compute the optimal update $u = v' - v$. (The operator $+ : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ maps a pair ρ, ρ' of functions to the function ρ'' that satisfies $\rho''(c') = \rho(c') + \rho'(c')$ for all $c' \in \mathcal{C}_0$. $- : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ is defined accordingly.)

For a given escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$ with estimation v, we define the *improvement potential* of an edge $e = (p, p') \in E_v$ in the improvement arena $\mathcal{A}_v$ as the value $P(e) = v(p') \oplus \alpha(p') - v(p) \geq \mathbf{0}$ by which the estimation would locally be improved when the respective player chose to turn to p' (disregarding the positive global effect that this improvement may have). To determine the optimal update, we construct the improvement arena, and evaluate the optimal update of the escape position to $u(\bot) = 0$. We then evaluate the improvement of the remaining positions successively by applying the following evaluation rule:

1. if there is a position $p \in V_1$ of Player 1 that has only evaluated successors, we evaluate the improvement of p to $u(p) = min\{u(p') + P\big((p, p')\big) \mid (p, p') \in E\}$,
2. else if there is a position $p \in V_1$ of Player 1 that has an evaluated successor p' with $u(p') = P\big((p, p')\big) = 0$, we evaluate the improvement of p to $u(p) = 0$,
3. else if there is a position $p \in V_0$ of Player 0 that has only evaluated successors, we evaluate its improvement to $u(p) = max\{u(p') + P\big((p, p')\big) \mid (p, p') \in E_v\}$[2],
4. else we choose a position $p \in V_1$ of Player 1 with minimal intermediate improvement $u'(p) = min\{u(p') + P\big((p, p')\big) \mid p'$ is evaluated and $(p, p') \in E\}$ and evaluate the improvement of p to $u(p) = u'(p)$. (Note that $min\{\emptyset\} = \infty$.)

Correctness. The basic intuition for the optimal improvement algorithm is to re-estimate the value of a position only *after* all its successors have been re-estimated. In this situation, it is easy to determine the optimal decision for the respective player. In a situation where all unevaluated positions do have a successor, we exploit that every cycle in $\mathcal{A}_v$ has non-negative weight (weight $\geq \mathbf{0}$), and every infinite play in $\mathcal{A}_v$ is evaluated to ∞. An optimal strategy of Player 1 will thus turn, for some position of Player 1, to an evaluated successor. It is safe to chose a transition such that the minimality criterion on the potential improvement u' is satisfied, because, independent of the choice of Player 1, no better potential improvement can arise at any later time during this update step. Following these evaluation rules therefore provides an optimal improvement.

Theorem 3. *For every estimation v of an escape game $\mathcal{E} = (V_0, V_1, E, \alpha)$, the algorithm computes the optimal improvement $v' = v + u$. If v is improvable, then the optimal improvement $v' \neq v$ is strictly better than v.*

Proof. During the reevaluation, we can fix optimal strategies f and f' for Player 0 and 1, respectively, by fixing $f(p)$ or $f'(p)$, respectively, to be some successor of p that satisfies the respective maximality or minimality requirement. (In Rule 2, we implicitly apply the same minimality requirement as in Rule 4.)

Every infinite f-conform play is evaluated to ∞, and for every finite f-conform play π that starts in some position p, we can show by induction over the length of π that $\rho(\pi) \geq v'(p)$ holds true.

[2] We could also choose $u(p) = max\{u(p') + P\big((p, p')\big) \mid p'$ is evaluated and $(p, p') \in E\}$.

No f'-conform play $\pi = p_0 p_1 p_2 \ldots$ in $\mathcal{A}_v$ (that is, under the restriction that Player 0 can chose only transitions in E_v), which does not start in a position p_0 that is evaluated to ∞, can contain a cycle, because p_{i+1} has been evaluated prior to p_i by construction. Thus, every such f'-conform play in $\mathcal{A}_v$ is finite. For every finite f'-conform play π in $\mathcal{A}_v$ that starts in some position p, we can show by induction over the length of π that $\rho(\pi) \leq v'(p)$ holds true.

It remains to show that the algorithm guarantees progress for improvable estimations. If at least one improvement edge e that originates from a position of Player 0 has a positive improvement potential $P(e) > \mathbf{0}$, the claim holds trivially. Let us consider the case that the improvement potential is $P(e) = \mathbf{0}$ for every improvement edge e that originates from a position of Player 0. According to the update rules, the algorithm will successively assign $u'(p) = \mathbf{0}$ to all positions in the 1-attractor of $\perp$ in the 0-arena $\mathcal{A}_v^0$. If the attractor covers all positions of $\mathcal{E}$, v is non-improvable by Theorem 2. Otherwise, $u'(p) > \mathbf{0}$ holds by definition for every remaining position $p \in V_1$ of Player 1 that is not in the 1-attractor of the escape position $\perp$. This implies $u > \mathbf{0}$ and thus $v' = v + u > v$. □

Complexity. In spite of the wide variety of strategies that are considered simultaneously, the update complexity is surprisingly low. The optimal improvement algorithm generalizes Dijkstra's single source shortest path algorithm to two player games. The critical part of the algorithm is to keep track of the intermediate update u', and the complexity of the algorithm depends on the used data structure. The default choice is to use binary trees, resulting in an update complexity of $O(m \log n)$. However, using advanced data structures like 2-3 heaps (cf. [26]) reduces this complexity slightly to $O(m + n \log n)$.

Theorem 4. *For an escape game with n positions and m edges, the optimal improvement can be computed using $O(m + \delta \log \delta)$ arithmetic operations, where $\delta \leq n$ is the number of positions of Player 1 for which the improvement is strict.*

Proof. Let us consider a run of our algorithm that, when applying rule 3, gives preference to updates of positions with improvement 0. Keeping track of these updates is cheap, and giving them preference guarantees that all positions with 0-update are removed before the remainder of the graph is treated.

Let us partition the operations occurring after these 0-updates into

1. operations needed for keeping track of the number of unevaluated successors for positions of Player 1 and for finding the direction with maximal improvement for positions of Player 0, and
2. all remaining operations.

Obviously, (1) contains only $O(m)$ operations, while the restriction to (2) coincides with a run of Dijkstra's algorithm on a subgraph of the improvement arena. (On the subgraph defined by the strategy f of Player 0 referred to in Theorem 3.) Dijkstra's algorithm can be implemented to run in $O(m + \delta \log \delta)$ arithmetic operations [26]. □

Theorem 5. *The algorithm can be implemented to solve a parity game with n positions, m edges, and c colors in time $O\big(m\left(\frac{n+c}{c}\right)^{c'}\big)$, where $c' = c-1$ if c is even, and $c' = c$ if c is odd.*

Proof. If both players follow the strategies f and f' from the proof of Theorem 3 starting in a position p_0 that is not evaluated to $\infty \neq v'(p_0)$, they reach the escape position $\perp$ on a finite acyclic path $p_0 p_1 \dots p_i \perp$. By induction over the length of this path we can show that $v(p_0) = \mathbf{0} \oplus \alpha(p_i) \oplus \dots \oplus \alpha(p_0)$. Note that, for odd highest color $c-1$ (and thus for even c), only p_i may be colored by $c-1$. Thus, the number of updates is, for each position, in $O\big(\left(\frac{n+c}{c}\right)^{c'-1}\big)$.

Let us, for the estimation of the running time, assume that only one small update occurs in every step. 'Only one' leads to a small δ (removing the $\delta \log \delta$ part from the estimation), while 'small update' can be used to reduce the discounted cost for the arithmetic operations on $\mathcal{R}$ to $O(1)$: Before computing the improvement potential P, the update u, and the intermediate update u', we first compute an *abstraction* $a : \mathcal{R} \to \mathbb{Z}$ of these values that maps a function $\rho \in \mathcal{R}$ to 0 if $\rho = \mathbf{0}$, and to $\pm$ the highest integer h with $\rho(h) \neq 0$ otherwise (+ if and only if $r > \mathbf{0}$). Computing the concrete value is then linear in the absolute value of the abstraction (rather than in c). For every edge $e = (p, p')$, updating the improvement potential $a \circ P(e)$ to its new value requires $O(max\{a \circ u(p), a \circ u(p')\})$ steps (using the old u). All other operations on abstract values are in $O(1)$.

To compute u', we proceed in two steps. In a first step, we maintain a 2-3 heap that stores only the abstraction of u', and that contains all positions where u' is above a threshold t that is initialized to $t = 0$. For positions with abstract value t, we keep a 2-3 heap with concrete values for u'. Every time we use rule 4 and find an empty concrete 2-3 heap, we increase t to the minimal value of the abstract 2-3 heap, remove all positions with this abstract value from the abstract heap, and add them (with concrete value) to the concrete heap. The required concrete arithmetic operations are linear in the value of the abstraction $a \circ u(r)$ of the concrete update (rather than in c). In the worst case scenario, 'small updates' implies that the discounted cost of the operations is in $O(1)$. □

Extended Update Step. The basic update step can be improved to an extended update step by three simple and cheap additional computation steps:

1. Recursively remove all positions from $\mathcal{E}$ that have no predecessors, and push them on a solve-me-later stack.
2. Adapt the valuation function v to v' such that the values of positions of Player 1 are left unchanged ($v'(p) = v(p) \ \forall p \in V_1$), and the values of all positions of Player 0 are maximally decreased ($v'(p) = max\{v'(p') \ominus \alpha(p) \mid (p', p) \in E\} \forall p \in V_0$). (This step again exploits that the game is bipartite.)
3. Apply a basic update step.
4. Remove the 0-attractor of all positions that are evaluated to ∞ from $\mathcal{E}$.

Step 1 simplifies the game—positions without predecessors have no impact on the value of other game positions, and their evaluation can safely be postponed

until after the remainder of the game has been evaluated—and strengthens the second step. In Step 2, we exploit the fact that the basic update step benefits from a high number of improvement edges that originate from positions of Player 0. This number is increased by changing the estimation v such that the estimations of positions of Player 1 remain unchanged, while the estimation of positions of Player 0 is decreased. The last step is again used to simplify the game.

An interesting side effect of Step 4 is that our game-based improvement algorithm behaves like standard fixed point algorithms [10, 12, 15] on Büchi and CoBüchi games (parity games with only two colors, w.l.o.g. 0 and 1). Like in these standard algorithms, we iteratively compute the set of states from which Player 0 can stay forever in positions with color 0, and then remove their 0-attractor from the game. The game-based approach described in this section can therefore also be viewed as an alternative generalization of the well accepted algorithms for Büchi and CoBüchi games to general parity games, which preserves different properties than McNaughton's generalization [10, 12, 15].

5 Benchmarks and Results

To evaluate the applicability of the game-based strategy improvement algorithm, a prototype of the algorithm was implemented and evaluated on different benchmarks, including random games with and without structure as well as other benchmarks for parity games. This section provides an overview on the results.

A first estimation of the performance of our algorithm on random games showed that the expected number of update games depends mainly on the number of colors and the outdegree, but it seems to be *constant* in the number of positions. This low expected number of updates has been confirmed by the following benchmarks. This restricts the potential competitors: The randomized subexponential algorithms of Ludwig [17], and Björklund and Vorobyov [20] change the strategy in exactly one position in every update step. It is therefore almost sure that the required number of update steps is at least linear in the size of the game. Ludwig's algorithm also has a much higher update complexity.

For the first benchmark, we therefore focused on the algorithm of Vöge and Jurdziński [19], and a (not subexponential) variant of Björklund and Vorobyov's algorithm [20] that chooses, in every step, a locally profitable modification uniformly at random for every position, for which a profitable modification exists.

The following table compares the expected number of iterations of our algorithm (*game*) with the variant of Björklund and Vorobyov (*rand*) and Vöge and Jurdziński's algorithm (VJ) for random games with 3 colors and outdegree 6.

positions	30	100	300	1000	3000	10000	30000	100000	300000
game	1.1	1.4	1.7	1.7	1.9	2.0	2.0	2.0	2.0
rand	2.5	2.9	3.1	3.0	3.0	3.1	3.2	3.7	4.0
VJ	5.3	12.2	26.1	66.1	182.0	573.1	1665.3[3]	—	—

[3] For 30000 positions, each sample took approximately four days on a 2.6 GHz Dual Core AMD Opteron machine (compared to 1.5 seconds for the game-based strategy improvement algorithm); the experiment was therefore terminated after ten samples.

The algorithm of Vöge and Jurdziński was not considered in the following benchmarks, because it took several days even for small random game with only 30000 positions and outdegree 6. This is partly due to the fact that the observed number of iterations grows linearly in the size of the game, and partly due to the much higher update complexity of $O(m\,n)$.

Different to the algorithm of Vöge and Jurdziński, the performance of the variant of Björklund and Vorobyov's algorithm (*rand*) is, on random games, close to the performance of our game-based strategy improvement algorithm. The cost of the individual updates for *rand* is slightly higher, because 0 cycles need to be excluded in their approach, which results in higher numbers (by a factor linear in the size of the game). Together with the smaller number of iterations, the running time of our algorithm improves over theirs by a factor of approximately 2 on the considered random games.

The difference between the two algorithms becomes apparent once structure is added to the game. Figure 5 compares the behavior of *game* and *rand* on different benchmarks. Benchmark 1 adds very little structure (favoring edges to

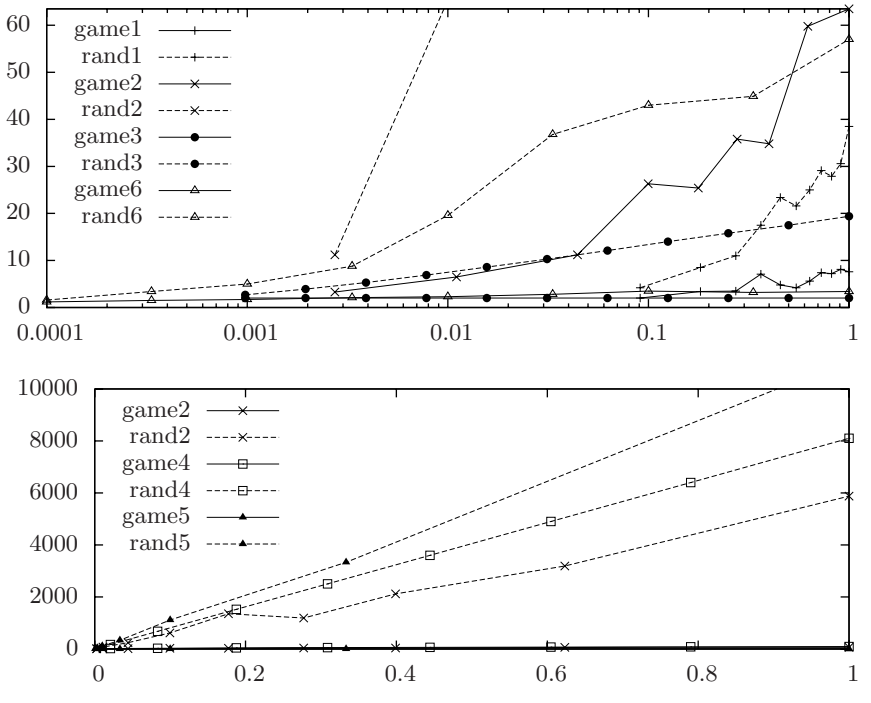


Fig. 3. The tables compare the performance of the variant of *game* (solid lines) and *rand* (dashed lines) on various benchmarks, measured in the number of iterations. The maximal size of all benchmarks is normalized to 1 (x-axis) for better readability.

For all benchmarks, the number of iterations (y-axis) needed by *game* is significant below the number needed by *rand*. The difference is particularly apparent in examples with much structure (Benchmarks 2 through 5).

'close' vertices in the randomized construction of the samples), while Benchmark 2 adds much structure (random chains of sparsely linked subgames). Benchmark 3 is a bipartite version of the games used in [16] to estimate the worst case complexity of Jurdziński's algorithm. Benchmark 4 refers to medium hard Büchi games, and Benchmark 5 is a test for the sensitivity of strategy improvement algorithms to 'traps' that lure them to a wrong direction (as in the example of Figure 1b). Finally, Benchmark 6 refers to the analysis of mean payoff games with small payoffs[4]. The results indicate that the more structure is added to the game, the greater becomes the advantage of *game* over *rand*. The detailed results are omitted due to space restrictions; they can be found in [27].

6 Discussion

The applicability of strategy improvement algorithms crucially depends on the quality of the individual improvement step, or, likewise, on the expected amount of improvement steps. Current strategy improvement techniques suffer from deficiencies in estimating the effect of strategy modifications: They cannot predict their global effect, let alone the cross effect between different modifications. The introduced game-based strategy improvement algorithm overcomes this deficiency by selecting an optimal combination of these modifications in every update step. While still greedy in nature, it allows us to make full use of the advantages attached to concurrent strategy modifications for the first time.

From a practical point of view, the amount of improvement steps used by simplex style algorithms tends to be linear in the amount of constraints that define the simplex [28]. For strategy improvement algorithms, these constraints are defined by the edges that originate from positions of Player 0. In the benchmarks, approximately 30% (at the end) to 50% (at the beginning) of these edges are improvement edges, which leads to an *exponential* number of concurrently considered improved strategies in every update game, and to a linear number of applied updates.

While the update complexity of the algorithms is low ($O(m + n \log n)$ arithmetic operations), finding a non-trivial bound on the number of updates remains an intriguing future challenge. The algorithm inherits the $\Omega(n)$ bound on the required number of updates from Büchi games, while the size of the codomain implies an $O((1 + \frac{n}{c})^c)$ upper bound, and either of these bounds may be sharp.

Understanding the complexity of the game-based strategy improvement algorithm would either lead to a proof that parity and/or mean payoff games can be solved in polynomial time, or would greatly help to understand the hardness of the problems. Hardness proofs for game-base strategy improvement, however, will not be simple. It took a quarter of a century to find a family of examples, for which the improvement complexity of the simplex algorithm is exponential [29]. These classical examples from linear programming, however, do not extend to game-based improvement methods. The Klee Minty polytope [29], for example,

[4] To extend our game-based approach to finding the 0-mean partition of mean payoff games, it suffices to replace the codomain $\mathcal{R}$ of the ranking function by $\mathbb{Z} \cup \{\infty\}$.

requires only a single update step from the origin (and at most linearly many steps from any arbitrary corner of the polytope) if we can consider all combinations of profitable and stale base changes in every improvement step.

References

1. Kozen, D.: Results on the propositional μ-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
2. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of μ-calculus. In: Proc. CAV, pp. 385–396 (1993)
3. Wilke, T.: Alternating tree automata, parity games, and modal μ-calculus. Bull. Soc. Math. Belg. 8(2) (2001)
4. de Alfaro, L., Henzinger, T.A., Majumdar, R.: From verification to control: Dynamic programs for omega-regular objectives. In: Proc. LICS, pp. 279–290. IEEE Computer Society Press, Los Alamitos (2001)
5. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM 49(5), 672–713 (2002)
6. Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)
7. Schewe, S., Finkbeiner, B.: The alternating-time μ-calculus and automata over concurrent game structures. In: Proc. CSL, pp. 591–605. Springer, Heidelberg (2006)
8. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. LICS, pp. 255–264. IEEE Computer Society, Los Alamitos (2006)
9. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: Puebla, G. (ed.) LOPSTR 2006. LNCS, vol. 4407, pp. 127–142. Springer, Heidelberg (2007)
10. Emerson, E.A., Lei, C.: Efcient model checking in fragments of the propositional μ-calculus. In: Proc. LICS, pp. 267–278. IEEE Computer Society Press, Los Alamitos (1986)
11. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proc. FOCS, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
12. McNaughton, R.: Infinite games played on finite graphs. Ann. Pure Appl. Logic 65(2), 149–184 (1993)
13. Zwick, U., Paterson, M.S.: The complexity of mean payoff games on graphs. Theoretical Computer Science 158(1–2), 343–359 (1996)
14. Browne, A., Clarke, E.M., Jha, S., Long, D.E., Marrero, W.: An improved algorithm for the evaluation of fixpoint expressions. Theoretical Computer Science 178(1–2), 237–255 (1997)
15. Zielonka, W.: Infinite games on finitely coloured graphs with applications to automata on infinite trees. Theor. Comput. Sci. 200(1-2), 135–183 (1998)
16. 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)
17. Ludwig, W.: A subexponential randomized algorithm for the simple stochastic game problem. Inf. Comput. 117(1), 151–155 (1995)
18. Puri, A.: Theory of hybrid systems and discrete event systems. PhD thesis, Computer Science Department, University of California, Berkeley (1995)

19. Vöge, J., Jurdziński, M.: A discrete strategy improvement algorithm for solving parity games. In: Proc. CAV, pp. 202–215. Springer, Heidelberg (2000)
20. Björklund, H., Vorobyov, S.: A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Appl. Math. 155(2), 210–229 (2007)
21. Obdržálek, J.: Fast mu-calculus model checking when tree-width is bounded. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 80–92. Springer, Heidelberg (2003)
22. Lange, M.: Solving parity games by a reduction to SAT. In: Proc. Int. Workshop on Games in Design and Verification (2005)
23. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: Dag-width and parity games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 524–536. Springer, Heidelberg (2006)
24. Jurdziński, M., Paterson, M., Zwick, U.: A deterministic subexponential algorithm for solving parity games. In: Proc. SODA, pp. 117–123. ACM/SIAM (2006)
25. Schewe, S.: Solving parity games in big steps. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 449–460. Springer, Heidelberg (2007)
26. Takaoka, T.: Theory of 2-3 heaps. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 41–50. Springer, Heidelberg (1999)
27. Schewe, S.: Synthesis of Distributed Systems. PhD thesis, Saarland University, Saarbrücken, Germany (2008)
28. Smale, S.: On the average number of steps of the simplex method of linear programming. Mathematical Programming 27(3), 241–262 (1983)
29. Klee, F., Minty, G.J.: How good is the simplex algorithm? Inequalities III, pp. 159–175 (1972)

Quantitative Languages*

Krishnendu Chatterjee[1], Laurent Doyen[2], and Thomas A. Henzinger[2]

[1] University of California, Santa Cruz
[2] EPFL, Lausanne, Switzerland

Abstract. Quantitative generalizations of classical languages, which assign to each word a real number instead of a boolean value, have applications in modeling resource-constrained computation. We use weighted automata (finite automata with transition weights) to define several natural classes of quantitative languages over finite and infinite words; in particular, the real value of an infinite run is computed as the maximum, limsup, liminf, limit average, or discounted sum of the transition weights. We define the classical decision problems of automata theory (emptiness, universality, language inclusion, and language equivalence) in the quantitative setting and study their computational complexity. As the decidability of language inclusion remains open for some classes of weighted automata, we introduce a notion of quantitative simulation that is decidable and implies language inclusion. We also give a complete characterization of the expressive power of the various classes of weighted automata. In particular, we show that most classes of weighted automata cannot be determinized.

1 Introduction

The automata-theoretic approach to verification is boolean. To check that a system satisfies a specification, we construct a finite automaton A to model the system and a finite (usually nondeterministic) automaton B for the specification. The language $L(A)$ of A contains all behaviors of the system, and $L(B)$ contains all behaviors allowed by the specification. The language of an automaton A can be seen as a boolean function L_A that assigns 1 (or true) to words in $L(A)$, and 0 (or false) to words not in $L(A)$. The verification problem "does the system satisfy the specification?" is then formalized as the language-inclusion problem "is $L(A) \subseteq L(B)$?", or equivalently, "is $L_A(w) \leq L_B(w)$ for all words w?". We present a natural generalization of this framework: a *quantitative language* L is a function that assigns a real-numbered value $L(w)$ to each (finite or infinite) word w. With quantitative languages, systems and specifications can be formalized more accurately. For example, a system may use a varying amount of some resource (e.g., memory consumption, or power consumption) depending on its behavior, and a specification may assign a maximal amount of available resource to each behavior, or fix the long-run average available use of the resource. The quantitative language-inclusion problem "is $L_A(w) \leq L_B(w)$ for all words w?" can then be used to check, say, if for each behavior, the peak power used by the system lies below the

* Research supported in part by the NSF grants CCR-0132780, CNS-0720884, and CCR-0225610, by the Swiss National Science Foundation, and by the European COMBEST project.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 385–400, 2008.
© Springer-Verlag Berlin Heidelberg 2008

bound given by the specification; or if for each behavior, the long-run average response time of the system lies below the specified average response requirement.

In the boolean automaton setting, the value of a word w in $L(A)$ is the maximal value of a run of A over w (if A is nondeterministic, then there may be many runs of A over w), and the value of a run is a function that depends on the class of automata: for automata over finite words, the value of a run is true if the last state of the run is accepting; for Büchi automata, the value is true if an accepting state is visited infinitely often; etc. To define quantitative languages, we use automata with weights on transitions. We again set the value of a word w as the maximal value of all runs over w, and the value of a run r is a function of the (finite or infinite) sequence of weights that appear along r. We consider several functions, such as Max and Sum of weights for finite runs, and Sup, LimSup, LimInf, limit average, and discounted sum of weights for infinite runs. For example, peak power consumption can be modeled as the maximum of a sequence of weights representing power usage; energy use can be modeled as the sum; average response time as the limit average [2,3]. Quantitative languages have also been used to specify and verify reliability requirements: if a special symbol $\perp$ is used to denote failure and has weight 1, while the other symbols have weight 0, one can use a limit-average automaton to specify a bound on the rate of failure in the long run [6]. Alternatively, the discounted sum can be used to specify that failures happening later are less important than those happening soon [8]. It should be noted that LimSup and LimInf automata generalize Büchi and coBüchi automata, respectively. Functions such as limit average (or mean payoff) and discounted sum are classical in game theory [26]; they have been studied extensively in the branching-time context of games played on graphs [12,7,3,14], and it is therefore natural to consider the same functions in the linear-time context of automata and languages.

We attempt a systematic study of quantitative languages defined by weighted automata. The main novelties concern quantitative languages of infinite words, and especially those that have no boolean counterparts (i.e., limit-average and discounted-sum languages). In the first part, we consider generalizations of the boolean decision problems of emptiness, universality, language inclusion, and language equivalence. The quantitative emptiness problem asks, given a weighted automaton A and a rational number ν, whether there exists a word w such that $L_A(w) \geq \nu$. This problem can be reduced to a one-player game with a quantitative objective and is therefore solvable in polynomial time. The quantitative universality problem asks whether $L_A(w) \geq \nu$ for all words w. This problem can be formulated as a two-player game (one player choosing input letters and the other player choosing successor states) with imperfect information (the first player, whose goal is to construct a word w such that $L_A(w) < \nu$, is not allowed to see the state chosen by the second player). The problem is PSPACE-complete for simple functions like Sup, LimSup, and LimInf, but we do not know if it is decidable for limit-average or discounted-sum automata (the corresponding games of imperfect information are not known to be decidable either). The same situation holds for the quantitative language-inclusion and language-equivalence problems, which ask, given two weighted automata A and B, if $L_A(w) \leq L_B(w)$ (resp. $L_A(w) = L_B(w)$) for all words w. Therefore we introduce a notion of quantitative simulation between weighted automata, which generalizes boolean simulation relations, is decidable, and implies

language inclusion. Simulation can be seen as a weaker version of the above game, where the first player has perfect information about the state of the game. In particular, we show that quantitative simulation can be decided in NP $\cap$ coNP for limit-average and discounted-sum automata.

In the second part of this paper, we present a complete characterization of the expressive power of the various classes of weighted automata, by comparing the classes of quantitative languages they can define. The complete picture relating the expressive powers of weighted automata is shown in Fig. 4. For instance, the results for LimSup and LimInf are analogous to the special boolean cases of Büchi and coBüchi (nondeterminism is strictly more expressive for LimSup, but not for LimInf). In the limit-average and discounted-sum cases, nondeterministic automata are strictly more expressive than their deterministic counterparts. Also, one of our results shows that nondeterministic limit-average automata are not as expressive as deterministic Büchi automata (and vice versa). It may be noted that deterministic Büchi languages are complete for the second level of the Borel hierarchy [28], and deterministic limit-average languages are complete for the third level [4]; so there is a Wadge reduction [29] from deterministic Büchi languages to deterministic limit-average languages. Our result shows that Wadge reductions are not captured by automata, and in particular, that the Wadge reduction from Büchi to limit-average languages is not regular. We sketch some details of the most interesting proofs; complete proofs are available in [5].

Other researchers have considered generalizations of languages, but as far as we know, nobody has addressed the quantitative language setting presented here. The lattice automata of [21] map finite words to values from a finite lattice. Roughly speaking, the value of a run is the meet (greatest lower bound) of its transition weights, and the value of a word w is the join (least upper bound) of the values of all runs over w. This corresponds to Min and Inf automata in our setting, and for infinite words, the Büchi lattice automata of [21] are analogous to our LimSup automata. However, the other classes of weighted automata (Sum, limit-average, discounted-sum) cannot be defined using operations on finite lattices. The complexity of the emptiness and universality problems for lattice automata is given in [21] (and implies our results for LimSup automata), while their generalization of language inclusion differs from ours. They define the *implication value* $v(A, B)$ of two lattice automata A and B as the meet over all words w of the join of $\neg L_A(w)$ and $L_B(w)$, while we use $+$ instead of join and define $v(A, B)$ as $\min_w(L_B(w) - L_A(w))$.

In classical weighted automata [25,23] and semiring automata [20], the value of a finite word is defined using the two algebraic operations $+$ and $\cdot$ of a semiring as the sum of the product of the transition weights of the runs over the word. In that case, quantitative languages are called *formal power series*. Over infinite words, weighted automata with discounted sum were first investigated in [11]. Researchers have also considered other quantitative generalizations of languages over finite words [9], over trees [10], and using finite lattices [15]. However, these works do not address the quantitative decision problems, nor do they compare the relative expressive powers of weighted automata over infinite words, as we do here. In [2], a quantitative generalization of languages is defined by discrete functions (the value of a word is an integer) and the decision problems only involve the extremal value of a language, which corresponds to emptiness.

In models that use transition weights as probabilities, such as probabilistic *Rabin automata* [24], one does not consider values of individual infinite runs (which would usually have a value, or measure, of 0), but only measurable sets of infinite runs (where basic open sets are defined as extensions of finite runs). Our quantitative setting is orthogonal to the probabilistic framework: we assign quantitative values (e.g., peak power consumption, average response time, failure rate) to individual infinite behaviors, not probabilities to finite behaviors.

2 Boolean and Quantitative Languages

We recall the classical automata-theoretic description of boolean languages, and introduce an automata-theoretic description of several classes of quantitative languages.

2.1 Boolean Languages

A *boolean language* over a finite alphabet Σ is either a set $L \subseteq \Sigma^*$ of finite words or a set $L \subseteq \Sigma^\omega$ of infinite words. Alternatively, we can view these sets as functions in $[\Sigma^* \to \{0,1\}]$ and $[\Sigma^\omega \to \{0,1\}]$, respectively.

Boolean automata. A *(finite) automaton* is a tuple $A = \langle Q, q_I, \Sigma, \delta \rangle$ where:

- Q is a finite set of states, and $q_I \in Q$ is the initial state;
- Σ is a finite alphabet;
- $\delta \subseteq Q \times \Sigma \times Q$ is a finite set of labeled transitions.

The automaton A is *total* if for all $q \in Q$ and $\sigma \in \Sigma$, there exists $(q, \sigma, q') \in \delta$ for at least one $q' \in Q$. The automaton A is *deterministic* if for all $q \in Q$ and $\sigma \in \Sigma$, there exists $(q, \sigma, q') \in \delta$ for exactly one $q' \in Q$. We sometimes call automata *nondeterministic* to emphasize that they are not necessarily deterministic.

A *run* of A over a finite (resp. infinite) word $w = \sigma_1 \sigma_2 \dots$ is a finite (resp. infinite) sequence $r = q_0 \sigma_1 q_1 \sigma_2 \dots$ of states and letters such that (i) $q_0 = q_I$, and (ii) $(q_i, \sigma_{i+1}, q_{i+1}) \in \delta$ for all $0 \leq i < |w|$. When the run r is finite, we denote by $\mathsf{Last}(r)$ the last state in r. When r is infinite, we denote by $\mathsf{Inf}(r)$ the set of states that occur infinitely many times in r. The prefix of length i of an infinite run r is the prefix of r that contains the first i states.

Given a set $F \subseteq Q$ of final (or accepting) states, the *finite-word language* defined by the pair $\langle A, F \rangle$ is $L^f_A = \{w \in \Sigma^* \mid$ there exists a run r of A over w such that $\mathsf{Last}(r) \in F\}$. The *infinite-word languages* defined by $\langle A, F \rangle$ are as follows: if $\langle A, F \rangle$ is interpreted as a Büchi automaton, then $L^b_A = \{w \in \Sigma^\omega \mid$ there exists a run r of A over w such that $\mathsf{Inf}(r) \cap F \neq \varnothing\}$, and if $\langle A, F \rangle$ is interpreted as a coBüchi automaton, then $L^c_A = \{w \in \Sigma^\omega \mid$ there exists a run r of A over w such that $\mathsf{Inf}(r) \subseteq F\}$.

Boolean decision problems. We recall the classical decision problems for automata, namely, emptiness, universality, language inclusion and language equivalence. Given a finite automaton A, the *boolean emptiness problem* asks whether $L^f_A = \varnothing$ (or $L^b_A = \varnothing$,

or $L_A^c = \varnothing$), and the *boolean universality problem* asks whether $L_A^f = \Sigma^*$ (or $L_A^b = \Sigma^\omega$, or $L_A^c = \Sigma^\omega$). Given two finite automata A and B, the *boolean language-inclusion problem* asks whether $L_A \subseteq L_B$, and the *boolean language-equivalence problem* asks whether $L_A = L_B$. It is well-known that for both finite- and infinite-word languages, the emptiness problem is solvable in polynomial time, while the universality, inclusion, and equivalence problems are PSPACE-complete [22,27].

2.2 Quantitative Languages

A *quantitative language* L over a finite alphabet Σ is either a mapping $L : \Sigma^+ \to \mathbb{R}$ or a mapping $L : \Sigma^\omega \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers.

Weighted automata. A *weighted automaton* is a tuple $A = \langle Q, q_I, \Sigma, \delta, \gamma \rangle$ where:

- $\langle Q, q_I, \Sigma, \delta \rangle$ is a total finite automaton, and
- $\gamma : \delta \to \mathbb{Q}$ is a *weight* function, where $\mathbb{Q}$ is the set of rational numbers.

Given a finite (resp. infinite) run $r = q_0\sigma_1 q_1 \sigma_2 \dots$ of A over a finite (resp. infinite) word $w = \sigma_1\sigma_2 \dots$, let $\gamma(r) = v_0 v_1 \dots$ be the sequence of weights that occur in r, where $v_i = \gamma(q_i, \sigma_{i+1}, q_{i+1})$ for all $0 \leq i < |w|$.

Given a *value function* $\mathsf{Val} : \mathbb{Q}^+ \to \mathbb{R}$ (resp. $\mathsf{Val} : \mathbb{Q}^\omega \to \mathbb{R}$), the Val-automaton A defines the quantitative language L_A such that for all words $w \in \Sigma^+$ (resp. $w \in \Sigma^\omega$), we have $L_A(w) = \sup\{\mathsf{Val}(\gamma(r)) \mid r \text{ is a run of } A \text{ over } w\}$.

In sequel we denote by n the number of states and by m the number of transitions of a given automaton. We assume that rational numbers that are given as pairs of integers, encoded in binary. All time bounds we give in this paper assume that the size of the largest integer in the input is a constant p. Without this assumption, most complexity results would involve a factor p^2, as we use only addition, multiplication, and comparison of rational numbers, which are quadratic operations.

Quantitative decision problems. We now present quantitative generalizations of the classical decision problems for automata. Given two quantitative languages L_1 and L_2 over Σ, we write $L_1 \sqsubseteq L_2$ if $L_1(w) \leq L_2(w)$ for all words $w \in \Sigma^+$ (resp. $w \in \Sigma^\omega$). Given a weighted automaton A and a rational number $\nu \in \mathbb{Q}$, the *quantitative emptiness problem* asks whether there exists a word $w \in \Sigma^+$ (resp. $w \in \Sigma^\omega$) such that $L_A(w) \geq \nu$, and the *quantitative universality problem* asks whether $L_A(w) \geq \nu$ for all words $w \in \Sigma^+$ (resp. $w \in \Sigma^\omega$). Given two weighted automata A and B, the *quantitative language-inclusion problem* asks whether $L_A \sqsubseteq L_B$, and the *quantitative language-equivalence problem* asks whether $L_A = L_B$, that is, whether $L_A(w) = L_B(w)$ for all $w \in \Sigma^+$ (resp. $w \in \Sigma^\omega$). All results that we present in this paper also hold for the decision problems defined above with inequalities replaced by strict inequalities.

Our purpose is the study of the quantitative decision problems for infinite-word languages and the expressive power of weighted automata that define infinite-word languages. We start with a brief overview of the corresponding results for finite-word languages, most of which follow from classical results in automata theory.

Finite words. For finite words, we consider the value functions Last, Max, and Sum such that for all finite sequences $v = v_1 \dots v_n$ of rational numbers,

$$\mathsf{Last}(v) = v_n, \qquad \mathsf{Max}(v) = \max\{v_i \mid 1 \le i \le n\}, \qquad \mathsf{Sum}(v) = \sum_{i=1}^{n} v_i.$$

Note that Last generalizes the classical boolean acceptance condition for finite words. One could also consider the value function $\mathsf{Min} = \min\{v_i \mid 1 \le i \le n\}$, which roughly corresponds to lattice automata [21].

Theorem 1. *The quantitative emptiness problem can be solved in linear time for* Last *and* Max*-automata, and in quadratic time for* Sum*-automata. The quantitative language-inclusion problem is PSPACE-complete for* Last- *and* Max*-automata.*

The complexity of the quantitative emptiness problem for Last and Max-automata is obtained by reduction to reachability in graphs, and for Sum-automata, by reduction to reachability of a cycle with positive value. The quantitative language-inclusion problem is undecidable for Sum-automata [19]. However, the quantitative language-inclusion problem for deterministic Sum-automata can be solved in polynomial time using a product construction. This naturally raises the question of the power of nondeterminism, which we address through translations between weighted automata.

Expressiveness. A class $\mathcal{C}$ of weighted automata *can be reduced* to a class $\mathcal{C}'$ of weighted automata if for every $A \in \mathcal{C}$ there exists $A' \in \mathcal{C}'$ such that $L_A = L_{A'}$. In particular, a class of weighted automata *can be determinized* if it can be reduced to its deterministic counterpart. All reductions that we present in this paper are constructive: when $\mathcal{C}$ can be reduced to $\mathcal{C}'$, we always construct the automaton $A' \in \mathcal{C}'$ that defines the same quantitative language as the given automaton $A \in \mathcal{C}$. We say that the *cost* of a reduction is $O(f(n,m))$ if for all automata $A \in \mathcal{C}$ with n states and m transitions, the constructed automaton $A' \in \mathcal{C}'$ has at most $O(f(n,m))$ many states. For all reductions we present, the size of the largest transition weight in A' is linear in the size p of the largest weight in A (however, the time needed to compute these weights may be quadratic in p).

It is easy to show that Last- and Max-automata can be determinized using a subset construction, while Sum-automata cannot be determinized. Results about determinizable sub-classes of Sum-automata can be found in [23,18].

Theorem 2 (see also **[23]).** Last- *and* Max*-automata can be determinized in* $O(2^n)$ *time;* Sum*-automata cannot be determinized. Deterministic* Max*-automata can be reduced to deterministic* Last*-automata in* $O(n \cdot m)$ *time; deterministic* Last*-automata can be reduced to deterministic* Sum*-automata in* $O(n \cdot m)$ *time. Deterministic* Sum*-automata cannot be reduced to* Last*-automata; deterministic* Last*-automata cannot be reduced to* Max*-automata.*

Infinite words. For infinite words, we consider the following classical value functions from $\mathbb{Q}^\omega$ to $\mathbb{R}$. Given an infinite sequence $v = v_0 v_1 \dots$ of rational numbers, define

- $\mathsf{Sup}(v) = \sup\{v_n \mid n \ge 0\}$;
- $\mathsf{LimSup}(v) = \limsup_{n\to\infty} v_n = \lim_{n\to\infty} \sup\{v_i \mid i \ge n\}$;

- $\mathsf{LimInf}(v) = \liminf_{n\to\infty} v_n = \lim_{n\to\infty} \inf\{v_i \mid i \geq n\}$;
- $\mathsf{LimAvg}(v) = \liminf_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$;
- given a discount factor $0 < \lambda < 1$, $\mathsf{Disc}_\lambda(v) = \sum_{i=0}^{\infty} \lambda^i \cdot v_i$.

For decision problems, we always assume that the discount factor λ is a rational number. Note that $\mathsf{LimAvg}(v)$ is defined using $\liminf$ and is therefore well-defined; all results of this paper hold also if the limit average of v is defined instead as $\limsup_{n\to\infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} v_i$. One could also consider the value function $\mathsf{Inf} = \inf\{v_n \mid n \geq 0\}$ and obtain results analogous to the Sup value function.

Notation. Classes of automata are sometimes denoted by acronyms of the form xyW where x is either N(ondeterministic) or D(eterministic), and y is one of the following: B(üchi), C(oBüchi), SUP, LS (LimSup), LI (LimInf), LA (LimAvg), or DI (Disc).

3 The Complexity of Quantitative Decision Problems

We study the complexity of the quantitative decision problems for weighted automata over infinite words.

Emptiness. The quantitative emptiness problem can be solved by reduction to the problem of finding the maximal value of an infinite path in a graph. This is decidable because pure memoryless strategies for resolving nondeterminism exist for all quantitative objectives that we consider [13,17,1].

Theorem 3. *The quantitative emptiness problem is solvable in $O(m+n)$ time for* Sup-, LimSup-, *and* LimInf-*automata; in $O(n \cdot m)$ time for* LimAvg-*automata; and in $O(n^2 \cdot m)$ time for* Disc-*automata.*

Language inclusion. The following theorem relies on the analogous result for finite automata.

Theorem 4. *The quantitative language-inclusion problem is PSPACE-complete for* Sup-, LimSup-, *and* LimInf-*automata.*

We do not know if the quantitative language-inclusion problem is decidable for LimAvg- or Disc-automata. The special cases of deterministic automata are easy, using a product construction.

Theorem 5. *The quantitative language-inclusion problems $L_A \sqsubseteq L_B$ for* LimAvg- *and* Disc-*automata are decidable in polynomial time when B is deterministic.*

When B is not deterministic, we make the following observation. There exist two LimAvg-automata A and B such that (i) $L_A \not\sqsubseteq L_B$ and (ii) there exist no finite words

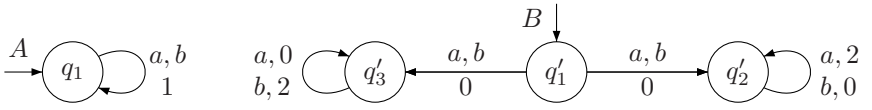


Fig. 1. Two limit-average automata A and B (nondeterministic) such that $L_A \not\sqsubseteq L_B$, but there is no word of the form $w = w_1 \cdot w_2^\omega$ with $L_A(w) > L_B(w)$.

w_1 and w_2 such that $L_A(w) > L_B(w)$ for $w = w_1 \cdot w_2^\omega$ (the word w is called a *lasso-word*). Consider the two LimAvg-automata A and B shown in Fig. 1, where B is nondeterministic. For all words $w \in \Sigma^\omega$, we have $L_A(w) = 1$. For a lasso-word of the form $w = w_1 \cdot w_2^\omega$, if in w_2 there are more b's than a's, then B chooses q'_3 from q'_1, and else chooses q'_2 from q'_1. Hence for all lasso-words $w = w_1 \cdot w_2^\omega$, we have $L_B(w) \geq 1$. However $L_A \not\sqsubseteq L_B$. Consider the word w generated inductively such that w_0 is the empty word, and w_{i+1} is generated from w_i as follows: (i) first generate a long enough sequence w'_{i+1} of a's after w_i such that the average number of b's in $w_i \cdot w'_{i+1}$ falls below $\frac{1}{3}$; (ii) then generate a long enough sequence w''_{i+1} of b's such that the average number of a's in $w_i \cdot w'_{i+1} \cdot w''_{i+1}$ falls below $\frac{1}{3}$; and (iii) let $w_{i+1} = w_i \cdot w'_{i+1} \cdot w''_{i+1}$. The infinite word w is the limit of this sequence. For the word w, we have $L_B(w) = 2 \cdot \frac{1}{3} = \frac{2}{3} < 1$, and thus $L_A \not\sqsubseteq L_B$. This observation is in contrast to the case of boolean language inclusion for, e.g., parity automata, where non-inclusion is always witnessed by a lasso-word. For the quantitative language-inclusion problem for discounted sum automata we have the following theorem.

Theorem 6. *The quantitative language-inclusion problem for* Disc*-automata is co-r.e.*

Universality and language equivalence. All of the above results about language inclusion hold for quantitative universality and language equivalence also.

4 Quantitative Simulation

As the decidability of the quantitative language-inclusion problems for limit-average and discounted-sum automata remain open, we introduce a notion of *quantitative simulation* as a decidable approximation of language inclusion for weighted automata. The quantitative language-inclusion problem can be viewed as a game of imperfect information, and we view the quantitative simulation problem as exactly the same game, but with perfect information. For quantitative objectives, perfect-information games can be solved much more efficiently than imperfect-information games, and in some cases the solution of imperfect-information games with quantitative objectives is not known. For example, perfect-information games with limit-average and discounted-sum objectives can be decided in NP $\cap$ coNP, whereas the solution for such imperfect-information games is not known. Second, quantitative simulation implies quantitative language inclusion, because it is easier to win a game when information is not hidden. Hence, as in the case of finite automata, simulation can be used as a conservative and efficient approximation for language inclusion.

Language-inclusion game. Let A and B be two weighted automata with weight function γ_1 and γ_2, respectively, for which we want to check if $L_A \sqsubseteq L_B$. The language-inclusion game is played by a *challenger* and a *simulator*, for infinitely many rounds. The goal of the simulator is to prove that $L_A \sqsubseteq L_B$, while the challenger has the opposite objective. The position of the game in the initial round is $\langle q_I^1, q_I^2 \rangle$ where q_I^1 and q_I^2 are the initial states of A and B, respectively. In each round, if the current position is $\langle q_1, q_2 \rangle$, first the challenger chooses a letter $\sigma \in \Sigma$ and a state q_1' such that $(q_1, \sigma, q_1') \in \delta_1$, and then the simulator chooses a state q_2' such that $(q_2, \sigma, q_2') \in \delta_2$. The position of the game in the next round is $\langle q_1', q_2' \rangle$. The outcome of the game is a pair (r_1, r_2) of runs of A and B, respectively, over the same infinite word. The simulator wins the game if $\mathsf{Val}(\gamma_2(r_2)) \geq \mathsf{Val}(\gamma_1(r_1))$. To make this game equivalent to the language-inclusion problem, we require that the challenger cannot observe the state of B in the position of the game.

Simulation game. The simulation game is the language-inclusion game without the restriction on the vision of the challenger, that is, the challenger is allowed to observe the full position of the game. Formally, given $A = \langle Q_1, q_I^1, \Sigma, \delta_1, \gamma_1 \rangle$ and $B = \langle Q_2, q_I^2, \Sigma, \delta_2, \gamma_2 \rangle$, a *strategy* τ for the challenger is a function from $(Q_1 \times Q_2)^+$ to $\Sigma \times Q_1$ such that for all $\pi \in (Q_1 \times Q_2)^+$, if $\tau(\pi) = (\sigma, q)$, then $(\mathsf{Last}(\pi_{|Q_1}), \sigma, q) \in \delta_1$, where $\pi_{|Q_1}$ is the projection of π on Q_1^+. A strategy τ for the challenger is *blind* if $\tau(\pi) = \tau(\pi')$ for all sequences $\pi, \pi' \in (Q_1 \times Q_2)^*$ such that $\pi_{|Q_1} = \pi'_{|Q_1}$. The set of *outcomes* of a challenger strategy τ is the set of pairs (r_1, r_2) of runs such that if $r_1 = q_0 \sigma_1 q_1 \sigma_2 \ldots$ and $r_2 = q_0' \sigma_1 q_1' \sigma_2 \ldots$, then $q_0 = q_I^1$, $q_0' = q_I^2$, and for all $i \geq 0$, we have $(\sigma_{i+1}, q_{i+1}) = \tau((q_0, q_0') \ldots (q_i, q_i'))$ and $(q_i', \sigma_{i+1}, q_{i+1}') \in \delta_2$. A strategy τ for the challenger is *winning* if $\mathsf{Val}(\gamma_1(r_1)) > \mathsf{Val}(\gamma_2(r_2))$ for all outcomes (r_1, r_2) of τ.

Theorem 7. *For all value functions and weighted automata A and B, we have $L_A \sqsubseteq L_B$ iff there is no blind winning strategy for the challenger in the language-inclusion game for A and B.*

Given two weighted automata A and B, there is a *quantitative simulation* of A by B if there exists no (not necessarily blind) winning strategy for the challenger in the simulation game for A and B. We note that for the special cases of Büchi and coBüchi automata, quantitative simulation coincides with *fair simulation* [16].

Corollary 1. *For all value functions and weighted automata A and B, if there is a quantitative simulation of A by B, then $L_A \sqsubseteq L_B$.*

Given two weighted automata A and B, the *quantitative simulation problem* asks if there is a quantitative simulation of A by B.

Theorem 8. *The quantitative simulation problem is in NP $\cap$ coNP for* LimSup-, LimInf-, LimAvg-, *and* Disc-*automata.*

The proof of Theorem 8 is obtained as follows. The quantitative simulation problems for LimSup- and LimInf-automata is reduced to perfect-information parity games; the

quantitative simulation problem for LimAvg-automata is reduced to perfect-information limit-average games; and the quantitative simulation problem for Disc-automata is reduced to perfect-information discounted-sum games. All reductions are polynomial time, and the resulting games can all be solved in NP ∩ coNP.

5 The Expressive Power of Weighted Automata

We study the expressiveness of different classes weighted automata over infinite words by comparing the quantitative languages they can define. For this purpose, we show the existence and non-existence of translations between classes of finite and weighted automata. We will use the following definition. A class $\mathcal{C}$ of finite automata *can be weakly reduced* to a class $\mathcal{C}'$ of weighted automata if for every $A \in \mathcal{C}$ there exists an $A' \in \mathcal{C}'$ such that $\inf_{w \in L_A} L_{A'}(w) > \sup_{w \notin L_A} L_{A'}(w)$.

5.1 Positive Reducibility Results

We start with the positive results about the existence of reductions between various classes of weighted automata, most of which can be obtained by generalizing corresponding results for finite automata. Our results also hold if we allow transition weights to be irrational numbers.

First, it is clear that Büchi and coBüchi automata can be reduced to LimSup- and LimInf-automata, respectively. In addition, we have the following results.

Theorem 9. Sup-*automata can be determinized in* $O(2^n)$ *time;* LimInf-*automata can be determinized in* $O(m \cdot 2^n)$ *time. Deterministic* Sup-*automata can be reduced to deterministic* LimInf-*, to deterministic* LimSup-*, and to deterministic* LimAvg-*automata, all in* $O(n \cdot m)$ *time.* LimInf-*automata can be reduced to* LimSup- *and to* LimAvg-*automata, both in* $O(n \cdot m)$ *time.*

The reduction from LimInf- to LimSup-automata (resp. to LimAvg-automata) essentially consists of guessing a position i and a transition weight v such that only weights greater than v are seen after position i. Once the guess is made, all transitions have weight v.

All reducibility relationships are summarized in Fig. 4, where the notation $^{\text{D}}_{\text{N}}$yW is used to denote the classes of automata that are determinizable.

5.2 Negative Reducibility Results

We show that all other reducibility relationships do not hold. The most important results in this section show that (i) deterministic coBüchi automata cannot be reduced to deterministic LimAvg-automata, deterministic Büchi automata cannot be reduced to LimAvg-automata, and (ii) neither LimAvg- nor Disc-automata can be determinized. Over the alphabet $\hat{\Sigma} = \{a, b\}$, we use in the sequel the boolean languages L_F, which contains all infinite words with finitely many a's, and L_I, which contains all infinite words with infinitely many a's.

The classical proof that deterministic coBüchi automata cannot reduced to deterministic Büchi automata can be adapted to show the following theorem.

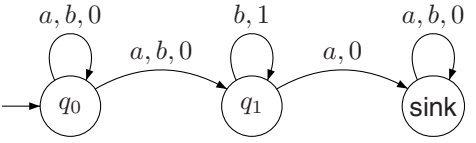


Fig. 2. A nondeterministic limit-average automaton

Theorem 10. *Deterministic coBüchi automata cannot be reduced to deterministic* LimSup*-automata.*

Since deterministic LimAvg- and deterministic Disc-automata can define quantitative languages whose range is infinite, while LimSup-automata cannot, we obtain the following result.

Theorem 11. *Deterministic* LimAvg*-automata and deterministic* Disc*-automata cannot be reduced to* LimSup*-automata.*

The next theorem shows that nondeterministic LimAvg-automata are strictly more expressive than their deterministic counterpart. Theorem 13 will show that the expressive powers of LimAvg- and LimSup-automata are incomparable.

Theorem 12. *Deterministic coBüchi automata cannot be weakly reduced to deterministic* LimAvg*-automata, and therefore they cannot be reduced to deterministic* LimAvg*-automata.* LimAvg*-automata cannot be determinized.*

Proof. Consider the language L_F of finitely many a's, which is obviously accepted by a DCW. It is also easy to see that the NLAW shown in Fig. 2 defines L_F. We show that L_F cannot be defined by any DLAW to prove the desired claims. By contradiction, assume that A is a DLAW with set of states Q and the initial state q_I that defines L_F. We assume without loss of generality that every state $q \in Q$ is reachable from q_I by a finite word w_q.

Let $\alpha = \inf_{w \in L_F} L_A(w)$. We claim that all b-cycles (a b-cycle is a cycle in A that can be executed with only b's) must be such that the average of the weights on the cycle is at least α. Indeed, if there is a b-cycle C in A with average weights less than α, then consider a state $q \in C$ and the word $w = w_q \cdot b^\omega$. We have $L_A(w) < \alpha$. Since $w = w_q \cdot b^\omega \in L_F$, this contradicts that $\alpha = \inf_{w \in L_F} L_A(w)$.

We now show that for all $\epsilon > 0$, there exists $w' \notin L_F$ such that $L_A(w') \geq \alpha - \epsilon$. Fix $\epsilon > 0$. Let $\beta = \max_{q,q' \in Q, \sigma \in \{a,b\}} |\gamma(q, \sigma, q')|$. Let $j = \lceil \frac{6 \cdot |Q| \cdot \beta}{\epsilon} \rceil$, and consider the word $w_\epsilon = (b^j \cdot a)^\omega$. A lower bound on the average of the weights in the unique run of A over $(b^j \cdot a)$ is as follows: it can have a prefix of length at most $|Q|$ whose sum of weights is at least $-|Q| \cdot \beta$, then it goes through b-cycles for at least $j - 2 \cdot |Q|$ steps with sum of weights at least $(j - 2 \cdot |Q|) \cdot \alpha$ (since all b-cycles have average weights at least α), then again a prefix of length at most $|Q|$ without completing the cycle (with sum of weights at least $-|Q| \cdot \beta$), and then weight for a is at least $-\beta$. Hence the average is at least

$$\frac{(j - 2 \cdot |Q|) \cdot \alpha - 2 \cdot |Q| \cdot \beta - \beta}{j + 1} \geq \alpha - \frac{6 \cdot |Q| \cdot \beta}{j} \geq \alpha - \epsilon;$$

we used above that $|\alpha| \leq \beta$, and by choice of j we have $\frac{6 \cdot |Q| \cdot \beta}{j} \leq \epsilon$. Hence we have $L_A(w_\epsilon) \geq \alpha - \epsilon$. Since $\epsilon > 0$ is arbitrary, and $w_\epsilon \notin L_F$, we have $\sup_{w \notin L_F} L_A(w) \geq \alpha = \inf_{w \in L_F} L_A(w)$. This establishes a contradiction, and thus A cannot exist. The desired result follows. ■

Theorem 13. *Deterministic Büchi automata cannot be weakly reduced to* LimAvg*-automata, and therefore they cannot be reduced to* LimAvg*-automata.*

Proof. We consider the language L_I of infinitely many a's, which is obviously accepted by a DBW.

By contradiction, assume that A is a NLAW with set of states Q and initial state q_I that defines L_I. We assume without loss of generality that every state $q \in Q$ is reachable from q_I by a finite word w_q.

Let $\alpha = \sup_{w \notin L_I} L_A(w)$, and $\beta = \max_{q,q' \in Q, \sigma \in \{a,b\}} |\gamma(q, \sigma, q')|$. We claim that all b-cycles C in A must have average weights at most α; otherwise, consider a state $q \in C$ and the word $w = w_q \cdot b^\omega$, we have $L_A(w) > \alpha$ which contradicts that $\alpha = \sup_{w \notin L_I} L_A(w)$.

We now show that for all $\epsilon > 0$, there exists $w \in L_I$ such that $L_A(w') \leq \alpha + \epsilon$. Fix $\epsilon > 0$. Let $j = \lceil \frac{3 \cdot |Q| \cdot \beta}{\epsilon} \rceil$, and consider the word $w_\epsilon = (b^j \cdot a)^\omega$. An upper bound on the average of the weights in any run of A over $(b^j \cdot a)$ is as follows: it can have a prefix of length at most $|Q|$ with the sum of weights at most $|Q| \cdot \beta$, then it follows (possibly nested) b-cycles[1] for at most j steps with sum of weights at most $j \cdot \alpha$ (since all b-cycles have average weights at most α), then again a prefix of length at most $|Q|$ without completing a cycle (with sum of weights at most $|Q| \cdot \beta$), and then weight for a is at most β. So, for any run of A over $w_\epsilon = (b^j \cdot a)^\omega$, the average weight is at most

$$\frac{j \cdot \alpha + 2 \cdot |Q| \cdot \beta + \beta}{j+1} \leq \alpha + \frac{3 \cdot |Q| \cdot \beta}{j} \leq \alpha + \epsilon$$

Hence we have $L_A(w_\epsilon) \leq \alpha + \epsilon$. Since $\epsilon > 0$ is arbitrary, and $w_\epsilon \in L_I$, we have $\inf_{w \in L_I} L_A(w) \leq \alpha = \sup_{w \notin L_I} L_A(w)$. The desired result follows. ■

None of the weighted automata we consider can be reduced to Disc-automata (Theorem 14), and Disc-automata cannot be reduced to any of the other classes of weighted automata (Theorem 15, and also Theorem 11).

Theorem 14. *Deterministic coBüchi automata and deterministic Büchi automata cannot be weakly reduced to* Disc*-automata, and therefore they cannot be reduced to* Disc*-automata. Also deterministic* Sup*-automata cannot be reduced to* Disc*-automata.*

The proofs of Theorem 14 and 15 are based on the property that the value assigned by a Disc-automaton to an infinite word depends essentially on a finite prefix, in the sense that the values of two words become arbitrarily close when they have sufficiently long common prefixes. In other words, the quantitative language defined by a discounted-sum automaton is a continuous function in the Cantor topology. In contrast, for the

[1] Since A is nondeterministic, a run over b^j may have nested cycles. We can decompose the run by repeatedly eliminating the innermost cycles.

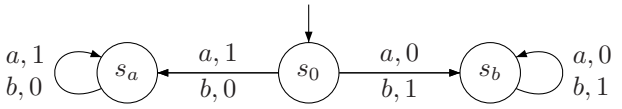


Fig. 3. The nondeterministic discounted-sum automaton N

other classes of weighted automata, the value of an infinite word depends essentially on its tail.

Theorem 15. *Deterministic* Disc*-automata cannot be reduced to* LimAvg*-automata.*

The next result shows that discounted-sum automata cannot be determinized. Consider the nondeterministic discounted-sum automaton N over the alphabet $\hat{\Sigma} = \{a, b\}$ shown in Fig. 3. The automaton N computes the maximum of the discounted sum of a's and b's. Formally, given a (finite or infinite) word $w = w_0 w_1 \ldots \in \hat{\Sigma}^* \cup \hat{\Sigma}^\omega$, let

$$v_a(w) = \sum_{i|w_i=a}^{|w|} \lambda^i \quad \text{and} \quad v_b(w) = \sum_{i|w_i=b}^{|w|} \lambda^i$$

be the λ-discounted sum of all a's (resp. b's) in w. Then $L_N(w) = \max\{v_a(w), v_b(w)\}$ for all infinite words $w \in \hat{\Sigma}^\omega$. We show that N cannot be determinized for some discount factors λ. The proof uses a sequence of intermediate lemmas.

For $\sigma \in \hat{\Sigma}$, let $\overline{\sigma} = a$ if $\sigma = b$, and $\overline{\sigma} = b$ if $\sigma = a$. We say that an infinite word $w \in \hat{\Sigma}^\omega$ *prefers* $\sigma \in \hat{\Sigma}$ if $v_\sigma(w) > v_{\overline{\sigma}}(w)$.

Lemma 1. *For all $0 < \lambda < 1$, all $w \in \hat{\Sigma}^*$, and all $\sigma \in \hat{\Sigma}$, there exists $w' \in \hat{\Sigma}^\omega$ such that $w \cdot w'$ prefers σ iff $v_\sigma(w \cdot \sigma^\omega) > v_{\overline{\sigma}}(w \cdot \sigma^\omega)$.*

We say that a finite word $w \in \hat{\Sigma}^*$ is *ambiguous* if there exist two infinite words $w'_a, w'_b \in \hat{\Sigma}^\omega$ such that $w \cdot w'_a$ prefers a and $w \cdot w'_b$ prefers b.

Lemma 2. *For all $0 < \lambda < 1$ and $w \in \hat{\Sigma}^*$, the word w is ambiguous iff $|v_a(w) - v_b(w)| < \frac{\lambda^{|w|}}{1-\lambda}$.*

Intuitively, ambiguous words are problematic for a deterministic automaton because it cannot decide which one of the two functions v_a and v_b to choose.

Lemma 3. *For all $\frac{1}{2} < \lambda < 1$, there exists an infinite word $\hat{w} \in \hat{\Sigma}^\omega$ such that every finite prefix of $\hat{w}$ is ambiguous.*

Proof. We construct $\hat{w} = w_1 w_2 \ldots$ inductively as follows. First, let $w_1 = a$ which is an ambiguous word for all $\lambda > \frac{1}{2}$ (Lemma 2). Assume that $w_1 \ldots w_i$ is ambiguous for all $1 \leq i \leq k$, that is $|x_i| < \frac{\lambda^i}{1-\lambda}$ where $x_i = v_a(w_1 \ldots w_i) - v_b(w_1 \ldots w_i)$ (Lemma 2). We take $w_{k+1} = a$ if $x_k < 0$, and $w_{k+1} = b$ otherwise. Let us show that $|x_{k+1}| < \frac{\lambda^{k+1}}{1-\lambda}$. We have $|x_{k+1}| = \big||x_k| - \lambda^k\big|$, and thus we need to show that $|x_k| - \lambda^k < \frac{\lambda^{k+1}}{1-\lambda}$ and $-|x_k| + \lambda^k < \frac{\lambda^{k+1}}{1-\lambda}$ knowing that $|x_k| < \frac{\lambda^k}{1-\lambda}$. It suffices to show that

$$\frac{\lambda^k}{1-\lambda} \leq \lambda^k + \frac{\lambda^{k+1}}{1-\lambda} \text{ and } \lambda^k - \frac{\lambda^{k+1}}{1-\lambda} < 0.$$

In other words, it suffices that $1 \leq 1 - \lambda + \lambda$ and $1 - \lambda - \lambda < 0$, which is true for all $\lambda > \frac{1}{2}$. ■

The word $\hat{w}$ constructed in Lemma 3 could be harmless for a deterministic automaton if some kind of periodicity is encountered in $\hat{w}$. We make this notion formal by defining $\mathit{diff}(w) = \frac{v_a(w) - v_b(w)}{\lambda^{|w|}}$ for all finite words $w \in \hat{\Sigma}^*$. It can be shown that if the set $R_\lambda = \{\mathit{diff}(w) \mid w \in \Sigma^*\} \cap (\frac{-1}{1-\lambda}, \frac{1}{1-\lambda})$ is finite, then the automaton N can be determinized [5], where (a, b) denotes the open interval between two reals a and b with $a < b$. Lemma 4 shows that this is also a necessary condition.

Lemma 4. *For all $0 < \lambda < 1$, if the set R_λ is infinite, then there exists no deterministic* Disc*-automaton D such that $L_D = L_N$.*

Proof. By contradiction, assume that R_λ is infinite and there exists a DDIW D such that $L_D = L_N$. For all $w \in \hat{\Sigma}^*$, let $\mathsf{Post}(w)$ be the (unique) state reached in D after reading w. We show that for all words $w_1, w_2 \in \hat{\Sigma}^*$ such that $\mathit{diff}(w_1), \mathit{diff}(w_2) \in R_\lambda$, if $\mathit{diff}(w_1) \neq \mathit{diff}(w_2)$, then $\mathsf{Post}(w_1) \neq \mathsf{Post}(w_2)$. Therefore D cannot have finitely many states.

We show this by contradiction. Assume that $\mathsf{Post}(w_1) = \mathsf{Post}(w_2)$. Then w_1 and w_2 are ambiguous by Lemma 2 since $\mathit{diff}(w_1), \mathit{diff}(w_2) \in R_\lambda$. For $i = 1, 2$, we thus have by Lemma 1

$$L_N(w_i \cdot a^\omega) = v_a(w_i) + \frac{\lambda^{|w_i|}}{1-\lambda} \quad \text{and} \quad L_N(w_i \cdot b^\omega) = v_b(w_i) + \frac{\lambda^{|w_i|}}{1-\lambda}.$$

On the other hand, since $\mathsf{Post}(w_1) = \mathsf{Post}(w_2)$, there exist $v_1, v_2, K_a, K_b \in \mathbb{R}$ such that for $i = 1, 2$,

$$L_D(w_i \cdot a^\omega) = v_i + \lambda^{|w_i|} \cdot K_a \quad \text{and} \quad L_D(w_i \cdot b^\omega) = v_i + \lambda^{|w_i|} \cdot K_b.$$

Since $L_D = L_N$, this entails that $L_D(w_i \cdot a^\omega) - L_D(w_i \cdot b^\omega) = L_N(w_i \cdot a^\omega) - L_N(w_i \cdot b^\omega)$, and therefore

$$\frac{v_a(w_1) - v_b(w_1)}{\lambda^{|w_1|}} = K_a - K_b = \frac{v_a(w_2) - v_b(w_2)}{\lambda^{|w_2|}}$$

which yields a contradiction. ■

We are now ready to prove the following theorem.

Theorem 16. Disc*-automata cannot be determinized.*

Proof. Let λ^* be a non-algebraic number in the open interval $(\frac{1}{2}, 1)$. Then, we show that the set R_{λ^*} is infinite, which establishes the theorem by Lemma 4.

By Lemma 2 and Lemma 3, there exist infinitely many finite words $w \in \hat{\Sigma}^*$ such that $\mathit{diff}(w) \in R_{\lambda^*}$. Since λ^* is not algebraic, the polynomial equation $\mathit{diff}(w_1) = \mathit{diff}(w_2)$ cannot hold for $w_1 \neq w_2$. Therefore, R_{λ^*} is infinite. ■

By a careful analysis of the shape of the family of polynomial equations in the above proof, we can show that the automaton N cannot be determinized for any rational value of λ greater than $\frac{1}{2}$ [5].

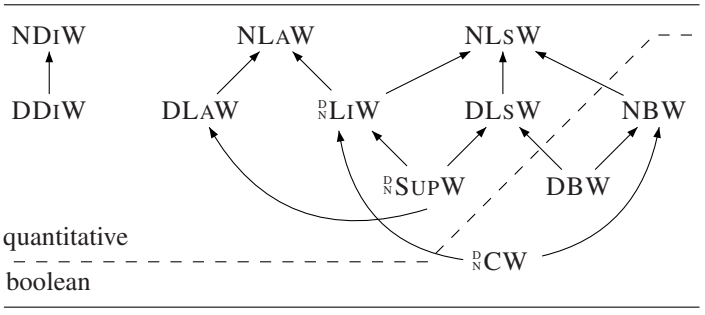


Fig. 4. Reducibility relations: a class $\mathcal{C}$ of automata can be reduced to $\mathcal{C}'$ iff $\mathcal{C} \rightarrow^* \mathcal{C}'$

References

1. Andersson, D.: An improved algorithm for discounted payoff games. In: ESSLLI Student Session, pp. 91–98 (2006)
2. Chakrabarti, A., Chatterjee, K., Henzinger, T.A., Kupferman, O., Majumdar, R.: Verifying quantitative properties using bound functions. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 50–64. Springer, Heidelberg (2005)
3. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource interfaces. In: Alur, R., Lee, I. (eds.) EMSOFT 2003. LNCS, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
4. Chatterjee, K.: Concurrent games with tail objectives. Theoretical Computer Science 388, 181–198 (2007)
5. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative languages. Technical Report MTC-REPORT-2008-003, EPFL (2008), `http://infoscience.epfl.ch/record/115228`
6. Chatterjee, K., Ghosal, A., Henzinger, T.A., Iercan, D., Kirsch, C., Pinello, C., Sangiovanni-Vincentelli, A.: Logical reliability of interacting real-time tasks. In: DATE. ACM, New York (2008)
7. Condon, A.: The complexity of stochastic games. Information and Computation 96, 203–224 (1992)
8. de Alfaro, L., Henzinger, T.A., Majumdar, R.: Discounting the future in systems theory. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1022–1037. Springer, Heidelberg (2003)
9. Droste, M., Gastin, P.: Weighted automata and weighted logics. Theoretical Computer Science 380, 69–86 (2007)
10. Droste, M., Kuich, W., Rahonis, G.: Multi-valued MSO logics over words and trees. Fundamenta Informaticae 84, 305–327 (2008)
11. Droste, M., Kuske, D.: Skew and infinitary formal power series. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 426–438. Springer, Heidelberg (2003)
12. Ehrenfeucht, A., Mycielski, J.: Positional strategies for mean payoff games. International Journal of Game Theory 8, 109–113 (1979)
13. Filar, J., Vrieze, K.: Competitive Markov Decision Processes. Springer, Heidelberg (1997)
14. Gimbert, H.: Jeux positionnels. PhD thesis, Université Paris 7 (2006)
15. Gurfinkel, A., Chechik, M.: Multi-valued model checking via classical model checking. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 263–277. Springer, Heidelberg (2003)

16. Henzinger, T.A., Kupferman, O., Rajamani, S.K.: Fair simulation. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 273–287. Springer, Heidelberg (1997)
17. Karp, R.M.: A characterization of the minimum cycle mean in a digraph. Discrete Mathematics 23(3), 309–311 (1978)
18. Kirsten, D., Mäurer, I.: On the determinization of weighted automata. Journal of Automata, Languages and Combinatorics 10, 287–312 (2005)
19. Krob, D.: The equality problem for rational series with multiplicities in the tropical semiring is undecidable. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 101–112. Springer, Heidelberg (1992)
20. Kuich, W., Salomaa, A.: Semirings, Automata, Languages. Monographs in Theoretical Computer Science. An EATCS Series, vol. 5. Springer, Heidelberg (1986)
21. Kupferman, O., Lustig, Y.: Lattice automata. In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 199–213. Springer, Heidelberg (2007)
22. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: FOCS, pp. 125–129. IEEE, Los Alamitos (1972)
23. Mohri, M.: Finite-state transducers in language and speech processing. Comp. Linguistics 23(2), 269–311 (1997)
24. Paz, A.: Introduction to probabilistic automata. Computer Science and Applied Mathematics. Academic Press, New York (1971)
25. Schützenberger, M.P.: On the definition of a family of automata. Information and control 4, 245–270 (1961)
26. Shapley, L.S.: Stochastic games. In: Proc. of the National Acadamy of Science USA, vol. 39, pp. 1095–1100 (1953)
27. Sistla, A.P., Vardi, M.Y., Wolper, P.: The complementation problem for Büchi automata with applications to temporal logic. Theoretical Computer Science 49, 217–237 (1987)
28. Thomas, W.: Languages, automata, and logic. In: Handbook of Formal Languages, ch. 7. Beyond Words, vol. 3, pp. 389–455. Springer, Heidelberg (1997)
29. Wadge, W.W.: Reducibility and Determinateness of Baire Spaces. PhD thesis, UC Berkeley (1984)

Characterization of Logics over Ranked Tree Languages

Thomas Place

LSV, ENS-Cachan, CNRS, INRIA
61, av. Pdt. Wilson, F-94230 Cachan, France
place@lsv.ens-cachan.fr

Abstract. We study the expressive power of the logics $EF + F^{-1}$, Δ_2 and boolean combinations of Σ_1 over ranked trees. In particular, we provide effective characterizations of those three logics using algebraic identities. Characterizations had already been obtained for those logics over unranked trees, but both the algebra and the proofs were dependant on the properties of the unranked structure and the problem remained open for ranked trees.

1 Introduction

Understanding the expressive power of logics over labeled trees is an important problem that can be found in many areas of Computer Science. In particular, a logic is said to have a decidable characterization if there exists a decision procedure for the following problem: given a regular language defined by its finite automaton, decide if it can be defined by a formula of the logic.

This type of problem is well known and has been extensively studied for word languages. Many word logics have been proven to have decidable characterizations using algebraic tools. Perhaps the most famous result is the characterization of $FO[<]$, the first-order logic with the order relation, which says that given a regular language L the three following properties are equivalent, L is definable by a first-order formula, L is star-free [8], the syntactic monoid of L is aperiodic [9]. Since the syntactic monoid of a regular language is a computable notion and aperiodicity a decidable property of monoids, the last property is actually a decidable characterization. This result demonstrates the importance and the relevance of the algebraic approach to obtain decidable characterizations. Using this approach, most common word logics have been proven to have decidable characterizations.

Much less results are known for tree languages, while it has been known for a long time that regular tree languages are the languages definable in monadic second order logic with the ancestor relation, until recently very few results of this type were known. For example, giving a decidable characterization for the first-order logic with the ancestor relation remains an open problem. Decidable characterizations have been issued by Walukiewicz and Bojańczyk on ranked and unranked trees for some temporal logics in [4,5], on ranked and unranked trees

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 401–415, 2008.
© Springer-Verlag Berlin Heidelberg 2008

by Benedikt and Segoufin for the first order logic with the successor relation in [1] and on ranked trees by Wilke for frontier testable languages in [11].

Recently, an algebraic formalism called Forest Algebras [5] was proposed for unranked trees. Using this formalism, decidable characterizations for several logics have been obtained. In [2], Bojańczyk, presents a characterization for $EF + F^{-1}$, the temporal logic with the ancestor and descendant modalities, in [7], Bojańczyk, Segoufin and Straubing present a characterization for boolean combinations of Σ_1, the logic of boolean combinations of purely existential first-order formulas, and in [3], Bojańczyk and Segoufin present a characterization for Δ_2, the languages definable by boolean combination of first order formulas with only one quantifier alternation.

Our aim in this paper is to present decidable characterizations for $EF + F^{-1}$, Δ_2 and Σ_1 over ranked trees using an algebraic formalism that is close to both Forest Algebras and the formalism used by Wilke in [11]. Both the statements and the proofs of the characterizations for unranked trees rely on the specific structure of unranked trees and have no obvious extension on ranked trees. For example, languages definable in $EF + F^{-1}$ are closed under bisimulation, the characterization of [2] reflects this property with an identity stating that those languages are closed under the action of duplicating a subtree within a tree, this property obviously does not make sense for ranked trees since each node has a fixed arity. Another simple example is the proof of the characterization of Σ_1, which uses the fact that there is a natural way to supress a node from an unranked tree to form a new unranked tree, once again the fixed arity of nodes on ranked trees makes this operation more delicate and technical. Therefore, since the arity of nodes cannot be expressed in those logics, giving a decidable characterization for ranked tree languages definable in them remained an open problem.

Because of space limitation, some proofs are missing and will appear in the journal version of the paper.

2 Notations

In this paper, we work with binary trees. Classic binary trees only have nodes of arity two or zero, meaning that every node has either zero or two sons, in the model we use, we also allow nodes of arity one (nodes that have exactly one son). Our motivation is that structures of arity one play a central role in our characterizations. However, this assumption is by no means restrictive since we can always suppose that the set of labels of arity one is empty.

Trees are defined over a finite alphabet (A, B, C), where A is a set of leaves symbols, B a set of unary symbols and C a set of binary symbols. The notion of tree is defined inductively as follows: Any $a \in A$ is a tree, if t is a tree bt is tree for $b \in B$ and if t and t' are trees, $c(t, t')$ is a tree for $c \in C$. The notion of nodes of a tree is defined in the usual way. A tree t' is a subtree of a tree t if there is a node x of t such that t' is the tree we get by keeping only the nodes of t that are below x.

A set of trees L over an alphabet (A, B, C) is called a tree language. As usual a regular language is a language recognized by a finite bottom-up automata. All the languages we consider in this paper are regular.

A context over an alphabet (A, B, C) is a tree that has exactly one leaf labeled by a special symbol $*$, that we call the hole. Notice that there exists a natural composition operation on contexts, if we take two contexts p and q, we get a new context pq by replacing $*$ in p by q. Another natural operation is to attach a tree t to replace $*$, forming a new tree pt.

We also consider objects that we call bi-contexts. They are trees with a special subtree $c(*, *)$ with $c \in C$. Notice that we also have natural operations with this type of object: we can attach a bi-context below a context to get a new bi-context, or attach a tree to replace the left or right $*$ in a bi-context to get a context.

We are interested in three tree logics: $EF + F^{-1}$, Δ_2 and Σ_1. We see a tree as a logical structure, we take the set of its nodes as domain and consider unary predicates P_d for $d \in A, B, C$, which hold true in x if x is labeled by d, and a binary predicate for the ancestor relation $<$.

A formula φ of $EF+F^{-1}$ over an alphabet (A, B, C) is defined by the following grammar:

$$\varphi = P_d \ d \in A, B, C \mid EF\varphi \mid F^{-1}\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi$$

Given a tree t and a node x of t, we say that $t, x \models \varphi$ if and only if:

- $\varphi = P_d$ and x is labeled by d
- $\varphi = EF\varphi'$ and x has a proper descendant x' such that $t, x' \models \varphi'$
- $\varphi = F^{-1}\varphi'$ and x has a proper ancestor x' such that $t, x' \models \varphi'$
- The semantics of the boolean operators are defined in the usual way

We say that $t \models \varphi$ if $t, x \models \varphi$ with x the root of t. A formula φ defines a tree language over (A, B, C) which is the set of trees t such that $t \models \varphi$.

Δ_2 is a restriction of the first-order logic on trees. A Σ_2 formula is of the form: $\exists x_1 ... \exists x_n \forall y_1 ... \forall y_m \varphi(x_1, ..., x_n, y_1, ..., y_m)$ with φ quantifier free. A Π_2 formula is the negation of a Σ_2 formula. We say a language is Δ_2 if it is definable by both a Σ_2 and a Π_2 formula.

Σ_1 is another restriction of the first-order logic on trees. We consider formulas that are boolean combinations of formulas of the form $\exists x_1 ... \exists x_n \varphi(x_1, ..., x_n)$ with φ quantifier free. A language is in Σ_1 if it is definable by such a formula.

Our main goal is to give decidable characterizations for those three logics on binary trees. More formaly, given $\mathcal{L}$ a tree logic, we want to be able to decide the following problem:

INPUT: A regular tree language L
PROBLEM: Can we define L with a formula of $\mathcal{L}$

Notice that the two logics $EF + F^{-1}$ and Δ_2 are related. On words, they have the same expressive power [10,6]. This result no longer holds on trees, for

example if we fix a $a \in A$ and consider L the language of trees with two different leaves labeled by a, L is definable in Δ_2 but not in $EF + F^{-1}$. However, we will see they remain closely related.

For both words and unranked trees the characterizations of those logics are stated and proven using an algebraic formalism called Forest Algebras. We also define our own algebraic formalism. Our definition is intended to be close to the definition of Forest Algebras in order to be able to follow and compare the ranked and unranked cases.

3 Binary Algebras

We defined three types of objects, trees, contexts and bi-contexts, our algebra reflects that by being constitued of three sets. The set corresponding to trees is actually very close to the set of states of an automaton and the set corresponding to bi-context very close to a transition function. However, we see them as algerbaic objets in order to stay close to the notions introduced for the characterizations over unranked trees.

We first give the formal definition of Binary Algebras, then move on with the definitions of morphisms of Binary Algebras, recognition of a language by a Binary Algebra and syntactic Binary Algebra of a language.

A Binary Algebra is a tuple (H, V, W) where H and W are sets and V is a monoid, we note $\cdot$ its operation. The idea is that each set represents one of the three objects we defined, H represents trees, V contexts and W bi-contexts. Several operations are defined on this tuple, each one reflecting an operation we described on trees, contexts and bi-contexts. The operations of contexts over trees and bi-contexts are reflected by actions of V on H and W. An action of a monoid V on H is function $f : V \times H \to H$ such that $f(v \cdot v', h) = f(v, f(v', h))$. We abusively write both actions $\cdot$ ($f(v, h) = v \cdot h$), we also want those actions to be faithful, meaning that for each v the function $h \to f(v, h)$ must be different. Finally, we reflect the bi-context tree operations by having two operations $\diamond_l$ and $\diamond_r$, from $W \times H$ onto V, such that $(w \diamond_l h) \cdot h' = (w \diamond_r h') \cdot h$ (the order in which we attach trees under bi-contexts is not important). Given this property we will sometimes write $w(h, h')$ instead of $(w \diamond_l h) \cdot h'$.

We use the usual notion of morphism: A morphism of binary algebras $\alpha : (H_1, V_1, W_1) \to (H_2, V_2, W_2)$ is actually composed of three applications $\beta : H_1 \to H_2$, $\gamma : V_1 \to V_2$ and $\delta : W_1 \to W_2$, such that γ is a morphism and $\gamma(v)\beta(h) = \beta(vh)$, $\gamma(v)\delta(h) = \delta(vh)$, $\delta(w) \diamond_l \beta(h) = \gamma(v \diamond_l h)$ and $\delta(w) \diamond_r \beta(h) = \gamma(v \diamond_r h)$.

Given an alphabet (A, B, C), the Free Binary Algebra $(A, B, C)^\Delta$ is the binary algebra $(H_\Delta, V_\Delta, W_\Delta)$ such that H_Δ is the set of trees that can be built using (A, B, C), V_Δ is the set of contexts that can be built using (A, B, C) and W_Δ is the set of bi-contexts that can be built using (A, B, C). The operation $\cdot$ is the natural context operation. $w \diamond_l h$ ($w \diamond_r h$) is the context obtained by filling the left (right) hole of w with h.

We say a tree language L is recognized by a Binary Algebra (H, V, W) if and only if there is a morphism $\alpha : (A, B, C)^\Delta \to (H, V, W)$ and a subset G of H

such that $L = \alpha^{-1}(G)$. It is simple to show that a tree language is regular if and only if it is recognized by a finite binary algebra.

Given a tree language we define an equivalence relation on H_Δ as follows $t \sim_L t'$ if and only if for all context $p \in V_\Delta$, pt is in L if and only if pt' is in L. We extend this equivalence on V_Δ as follows $p \sim_L p'$ if and only if for all trees $t \in H_\Delta$, pt is in L if and only if $p't$ is in L. Finally we extend it on W_Δ as follows $q \sim_L q'$ if and only if for all pairs of trees $t, t' \in H_\Delta$, $q(t, t')$ is in L if and only if $q'(t, t')$ is in L. Those relations define a congruence over $(A, B, C)^\Delta$ and we call the quotient, the syntactic binary algebra of L. The syntactic binary algebra of a regular language is a finite object that we can compute from the automata of the language.

In this paper we only consider finite binary algebras. Given any finite monoid V there is a computable number $\omega(V)$ such that for every element v of V, v^ω is an idempotent ($v^\omega = v^\omega v^\omega$). We write this number ω when V is understood from the context.

4 Links between Δ_2 and $EF + F^{-1}$

Recall that we said that over words the logics $F + F^{-1}$ and Δ_2 have the same expressive power and that while this property is no longer true on trees, both the statements and proofs of the characterizations remained closely related. In this section we describe a subclass of regular languages that contains both the languages definable in $EF + F^{-1}$ and Δ_2, and verifies those common properties.

Definition 1. *We say a Binary Algebra (H, V, W) belongs to TDA if it verifies the following identities:*

$$\forall u, v \in V \qquad (uv)^\omega = (uv)^\omega u (uv)^\omega \tag{1}$$

$$\forall h \in H \ \forall w \in W \qquad w \diamond_l h = w \diamond_r h \tag{2}$$

$$\begin{gathered} \forall w, w' \in W \ \forall h, h' \in H \quad \text{and for } e = ((w \diamond_l h)(w' \diamond_l h'))^\omega \\ e = e(w \diamond_l h')(w' \diamond_l h)e \end{gathered} \tag{3}$$

Notice that over words, the languages definable in $F + F^{-1}$ are exactly the languages whose syntactic monoid verifies Equation (1) (called the DA equation), therefore, this definition is indeed an extension of the word case. Also notice that (1) and (2) have clear equivalents stated in the charcaterization of Δ_2 and $EF + F^{-1}$ over unranked trees on Forest Algebras. However, this is not the case for (3), which is a new equation. Over unranked trees, the properties of Forest Algebras would allow us to derive a similar result from (1), in our case, its statement is needed in order to prove properties that are not consequences of (1). Finally (2) means that $\diamond_r = \diamond_l$, therefore, from now on, we only write $\diamond$.

We prove our characterizations by induction on orders over binary algebras. Those orders are significant because of the properties of TDA, we define them here and state those properties. We define two orders, one on H and one on V, we begin with the order on H.

Given a binary algebra (H, V, W) and $h, h' \in H$, we say that $h \sqsubseteq h'$ if and only if there exists $v \in V$ such that $h = vh'$. We write $h \sim h'$ if and only if $h \sqsubseteq h'$ and $h' \sqsubseteq h$. We show that the $\sim$-classes of a binary algebra of TDA verify useful properties.

Definition 2. *Given $h \in H$ we call $stab(h)$ a tuple of three sets:*

$$stab_H(h) = \{g \in H \mid \exists w \in W \quad w(h,g) \sim h\} \; stab_V(h) = \{v \in V \mid vh \sim h\}$$
$$stab_W(h) = \{w \in W \mid \exists g \in H \quad w(h,g) \sim h\}$$

The following proposition shows that in Binary Algebras of TDA, $stab(h)$ depends only on the $\sim$-class of h. Therefore, we will sometimes write $stab(G)$ if G is a $\sim$-class. This result makes $\sqsubseteq$ a relevant order to use for the inductions in our proofs.

Proposition 1. *If (H, V, W) verifies the TDA identities, the following properties hold:*

- $h \sim h' \Rightarrow stab(h) = stab(h')$.
- $stab_V(h)$ *is a submonoid of* V.
- $stab_W(h) \diamond stab_H(h) \subseteq stab_V(h)$
- $stab_V(h) stab_W(h) \subseteq stab_W(h)$

This proposition is proven using classical algebraic techniques. Notice that Equation (3) is used in order to prove the third item.

The order $\sqsubseteq$ has a second property that we will use in our proofs, there is a single minimal class of $\sim$ relatively to $\sqsubseteq$. We call it H_{min}.

Lemma 1. *Given (H, V, W) a Binary Algebra, there is a minimal $\sim$-class in H relatively to $\sqsubseteq$ and if $(H, V, W) \in TDA$, for all minimal h, there exists $u_h \in V$ such that $\forall h' \in H \; u_h h' = h$.*

Proof. Let $g \in H_{min}$. We write $H = \{h_1, ..., h_n\}$. We take $w \in W$. Consider:

$$h_{min} = ((w \diamond h_1)...(w \diamond h_n)^\omega g$$

$h_{min} \in H_{min}$, hence, we have $g = vh_{min}$. We take:

$$u_g = v((w \diamond h_1)...(w \diamond h_n)^\omega w(*, ((w \diamond h_1)...(w \diamond h_n))^\omega g)$$

Thanks to equation (1) we have $\forall h \; u_g h = g$. □

Over V we use the classic Green relation $\mathcal{R}$ on monoids, $v \leq_{\mathcal{R}} v'$ if and only if $\exists u \in V \mid v = v'u$. $\mathcal{R}$-classes verify properties that are very similar to the ones verified by $\sim$-classes, if (H, V, W) belongs to TDA.

Definition 3. *Given $v \in V$ we call $stab(v)$ a tuple of three sets:*

$$stab_H(v) = \{g \in H \mid \exists w \in W \quad v(w \diamond g) \; \mathcal{R} \; v\} \; stab_V(v) = \{u \in V \mid vu \; \mathcal{R} \; v\}$$
$$stab_W(v) = \{w \in W \mid \exists g \in H \quad v(w \diamond g) \; \mathcal{R} \; v\}$$

Again, we show in the following proposition that in Binary Algebras of TDA, $stab(v)$ depends only on the $\mathcal{R}$-class of v. Therefore, we will sometimes write $stab(U)$ if U is a $\mathcal{R}$-class. The order $\leq_{\mathcal{R}}$ is also relevant to be used for the inductions in our proofs.

Proposition 2. *If (H, V, W) verifies the TDA identities, the following properties hold:*

- $v \ \mathcal{R} \ v' \Rightarrow stab(v) = stab(v')$.
- $stab_V(v)$ *is a submonoid of* V.
- $stab_W(v) \diamond stab_H(v) \subseteq stab_V(v)$
- $stab_V(v) stab_W(v) \subseteq stab_W(v)$

5 Characterization of $EF + F^{-1}$

In this section we give a decidable characterization for $EF + F^{-1}$. One of the identities we state in our characterization uses a relation over V. This relation can be seen as a more powerful binary variant of a relation used in [2] over Forest Algebras. In [2], the relation compared two contexts, from a context a smaller context can be built by suppressing subtrees that are off the path from the root to the hole. This notion does not have any obvious extension to binary trees since we cannot supress subtrees in a binary tree without changing the arity of a node. We use an alternate relation, instead of suppressing subtrees, we ask that all the subtrees that are off the path from the root to the hole in the smaller context can be found in the same path in the bigger context.

Definition 4. *Given $u, v \in V$, we say that $u \dashv v$ if and only if we can write:*

- $u = (u_0 \diamond h_0)...(u_n \diamond h_n)$
- $v = (u_0 \diamond g_0)...(u_n \diamond g_n)$
- $\{h_0, ..., h_n\} \subseteq \{g_0, ..., g_n\}$

With $u_0, ..., u_n \in W$ and $g_0, ..., g_n \in H$.

Theorem 1. *Let L be a regular tree language on an alphabet (A, B, C), L is definable in $EF + F^{-1}$ if and only if its syntactic binary algebra belongs to TDA and verifies the two following identities:*

$$\begin{gathered} \forall w \in W \ \forall h, g \in H \\ (w \diamond h)^\omega g = (w \diamond h)^\omega w((w \diamond h)^\omega g, (w \diamond h)^\omega g) \end{gathered} \tag{4}$$

$$\begin{gathered} \forall u_1, u_2, v_1, v_2 \in V \quad \text{such that } u_1 \dashv u_2 \text{ and } v_1 \dashv v_2 \\ (u_1 v_1)^\omega (u_2 v_2)^\omega = (u_1 v_1)^\omega v_2 (u_2 v_2)^\omega \end{gathered} \tag{5}$$

Notice that identities (2), (3), (4) and (5) are sufficient, and that identity (1) is redundant since it a consequence of (5) when $u_1 = u_2$ and $v_1 = v_2$.

The characterization proposed in [2] for unranked tree languages shares some similarities with our definition. This characterization can be seen as divided in

two parts, an horizontal one and a vertical one. Horizontally, it states closure under bisimulation, while we still state commutativity with (2), as we said closure under duplication of subtrees has no sense on binary trees, we replace it with (4). We will use (4) to solve in an alternate way the cases where closure under duplication of subtrees is used on unranked trees.

Vertically, the characterization of [2] used two identities, the classic DA equation (1), and another equation similar to our equation (5). Since (1) is a consequence of (5), (5) replaces those two equations and plays a similar role in the proof. Recall however, that we had to redifine the relation $\dashv$, which will lead to differents uses of this equation. Finally, we need to state a last equation (3) which is used for proving the propreties of the *stab* sets in the TDA section. This equation is needed in order to solve problems related to the lack of malleability of the binary tree structure.

Before we prove this proposition, we prove that it fulfills our decidability goal, given a regular language we compute its syntactic binary algebra and then, since the $\dashv$ relation is computable via a fix point algorithm, we can decide whether it satisifes the identities or not, which decides our problem.

We proceed with the proof, we prove both directions using an Ehrenfeucht Fraïssé approach. A k rounds Ehrenfeucht Fraïssé game for $EF + F^{-1}$ on two trees t and t' is played as follows. There are two players called Spoiler and Duplicator with two pebbles each, when the game begins, each player has a pebble on the root of t or t'. A round is played as follows, with a pebble at position x_1 on t and at position x_2 on t':

- Spoiler chooses one of the two trees, say t, and he chooses a node y_1 which is either a proper descendant or a proper ancestor of x_1 and puts his pebble on it.
- Duplicator chooses y_2 in t' with the same label as y_1 and which is a proper descendant of x_2 if y_1 was a proper descendant of x_1 and a proper ancestor of x_2 if y_1 was a proper ancestor of x_1. If she can't play Spoiler wins.
- Both players take back their pebble in x_1, x_2 and they move on to the next round with y_1, y_2 playing the role of x_1, x_2.

If Duplicator can survive k rounds she is declared the winner and we write $t \cong_k t'$. We state a Lemma that links the notion of definability in $EF + F^{-1}$ with the notion of game we just defined, this Lemma is proven using classical Ehrenfeucht Fraïssé techniques. The rank of a formula is its nesting depth of modalities.

Lemma 2. *We have $t \cong_k t'$ if and only if t and t' satisfy the same $EF + F^{-1}$ formulas of rank k.*

The easier direction of Theorem 1 is proven using classic Ehrenfeucht Fraïssé techniques, if a language L is definable in $EF+F^{-1}$, the syntactic binary algebra of L must verify the identities of Theorem 1.

We prove the hardest direction of Theorem 1, if L is a language whose syntactic algebra verifies (2), (3), (4) and (5), it is definable in $EF + F^{-1}$. Let

(H, V, W) be the syntactic binary algebra of L and α the corresponding morphism. We suppose that $\forall h \in H\ \exists a \in A$ such that $\alpha(a) = h$ (notice that this assumption is not restrictive as we will not consider the size of A in the proof). Given a subset X of H we say that a tree t is X-trimmed if and only if the only subtrees of t that have their image under α in X are leaves. Instead of directly proving the proposition we prove a slightly more general Proposition.

Proposition 3. *Let X be a union of $\sim$-classes of H and $v \in V$. There exists $k \in \mathbb{N}$ such that for all X-trimmed trees t and t', we have:*

$$t \cong_k t' \quad \Rightarrow \quad v\alpha(t) = v\alpha(t')$$

This proposition is very similar to the one proved in [2], the main difference in the statement being our usage of Ehrenfeucht Fraïssé games. We also use a similar proof structure to prove it, the inner proofs however, are different because of our new equations and of the constraints related to the ranked tree structure.

We first show that Theorem 1 is a consequence of this Proposition 3. Take $X = \emptyset$ and $v = 1_V$ (the neutral element of V) we get:

$$\exists k \text{ such that } \forall t, t' \quad t \cong_k t' \quad \Rightarrow \quad \alpha(t) = \alpha(t')$$

It means that for some k, L is the union of classes of the Ehrenfeucht Fraïssé game. Since the classes of the Ehrenfeucht Fraïssé game are definable in $EF + F^{-1}$ (see Lemma 2), it follows that L is definable in $EF + F^{-1}$. The rest of this section is devoted to the proof of Proposition 3. We show that it holds for:

$$k \geq (2^{2(|B|+|C|)})(2^{2dp(v)})(4 \times 2^{2|H-X|})((2^{2|V|})^{|H|+1})(3 \times 2^{2|H|})$$

We proceed by induction on the four following parameters:

1. $|H|$
2. The number of $\sim$-classes left in $H - X$
3. $|B| + |C|$
4. The number of $\mathcal{R}$-classes left below v

We consider three cases. First, we suppose that there are at least two $\sim$-classes in $H - X$, we will decrease the first and second induction parameters to conclude by induction. In the second case, we suppose that there are still labels in t and t' that decrease the $\mathcal{R}$-class of v and we will decrease the third and fourth parameters to conclude by induction. The third case is the complement of the two first cases, we will show that if neither of the assumptions we state in those cases hold, we are able to conclude the proof using our identities.

5.1 First Case

In this case we suppose that there is more than one $\sim$-class left in $H - X$, we will conclude by induction, adding one $\sim$-class to X. We take $G = \{g_1, ..., g_n\}$, a maximal $\sim$-class in $H - X$ relatively to $\sqsubseteq$, let $a_1, ..., a_n$ be leaf representatives for the $g_1, ..., g_n$ $(\alpha(a_i) = g_i)$. We say a tree is a twig if its only node which is not a leaf is its root.

Definition 5. *From t and t', we construct new trees:*

1. *s and s' by replacing all twigs of type g_i by a_i, for all i.*
2. *r and r' by replacing all maximal subtrees of s and s' with type $g_i \in G$ (maximal in the sense that they are not subtrees of bigger subtrees of type in G) with the leaf a_i. Notice that r and r' are $X \cup G$ trimmed.*

Notice that by construction, we have $v\alpha(t) = v\alpha(s) = v\alpha(r)$ and $v\alpha(t') = v\alpha(s') = v\alpha(r')$. The following Lemma, which is the central point of this case, is a consequence of the assumption we made about $H - X$ containing more than one $\sim$-class and about G being a maximal one.

Lemma 3. $r \cong_{\frac{k}{2}-4} r'$

Before we prove it, we show how it can be used to conclude this case. We have:

1. $r \cong_{\frac{k}{2}-4} r'$
2. r and r' are $X \cup G$ trimmed
3. $\frac{k}{2} - 4 \geq (4 \times 2^{2|H-X \cup G|})K$, since $k \geq (4 \times 2^{2|H-X|})K$

Therefore using the induction hypothesis $v\alpha(r) = v\alpha(r')$ and by construction, it follows that $v\alpha(t) = v\alpha(t')$.

We prove Lemma 3 in two steps, the first one is that we can detect subtrees of type in G under α in the game. The second one is that we can then detect precisely their type in G. The first step is mainly a consequence of the properties of *stab* described in Section 4. This step uses bisimulation in the unranked case, since we do not have bisimulation we use Equation (4) instead. Notice that this first step, because it relies on the results of Section 4, is also using Equations (1) and (3).

Claim. Let f be a subtree of s and f' a subtree of s' such that $\alpha(f) \in G$ and $\alpha(f') \notin G$. Spoiler wins the two rounds game on f and f'.

Proof. We claim that an X-Trimmed tree t has type outside G if and only if one the following conditions holds:

1. t is a leaf and it has type outside G
2. t has a twig subtree with type outside G
3. t has non-twig unary node with label b such that $\alpha(b) \notin stab_V(G)$
4. t has non-twig binary node with label c such that $\alpha(c) \notin stab_W(G)$
5. t has an inner node with a leaf of label a as brother such that $\alpha(a) \notin stab_H(G)$

Since G is a maximal $\sim$-class in $H - X$, it follows from the properties of *stab* that the conditions are sufficient. We prove that they are necessary.

In the unranked case, closure under bisimulation entails that $G \subseteq stab_H(G)$, this is no longer true in our case. However, (4) entails a weaker result that is sufficient in order to prove that the conditions are necessary.

Claim. If $stab_H(G) \neq \emptyset$, $G \subseteq stab_H(G)$

Proof. Because there exists an $h \in stab_H(G)$, $\exists w$ such that $w \diamond h \in stab_V(G)$. We have $(w \diamond h)^\omega \in stab_V(G)$ (recall that $stab_V(G)$ is a submonoid), therefore:

$$(w \diamond h)^\omega g = g' \in G$$

Using Equation (4) we get:

$$(w \diamond h)^\omega w(g', g') = g'$$

Therefore $g' \in stab_H(G)$, hence $w(g, g') \sim g$, which also means that $g \in stab_H(G)$. □

We say that (t_1, t_2) is a good pair if it belongs to $stab_H(G) \times G$ or $G \times stab_H(G)$ (notice that if $stab_H(G) = \emptyset$, there is no good pair). Let t be a tree of type outside G, if t is a leaf (1) holds, otherwise let t' be a minimal subtree of t of type outside G if it is a twig (2) holds, otherwise we are in one of the three following cases:

- $t' = b(t_1)$, t_1 has type in G so $\alpha(b) \notin stab_V(G)$, (3) holds
- $t' = c(t_1, t_2)$ and (t_1, t_2) is a good pair. Then, we have $\alpha(t_1) \in stab_H(G)$ and $\alpha(t_2) \in G$. Then, since t' has type outside G, $\alpha(c) \diamond \alpha(t_1) \notin stab_V(G)$, which implies that $\alpha(c) \notin stab_W(G)$ (recall that $stab_W(G) \diamond stab_H(G) \subseteq stab_V(G)$), (4) holds.
- $t' = c(t_1, t_2)$ and (t_1, t_2) is not a good pair, we consdier two cases. If $stab_H(G) = \emptyset$, we have by definition $stab_W(G) = \emptyset$, therefore (4) holds. Otherwise, we have $G \subseteq stab_H(G)$, since (t_1, t_2) is not a good pair, t_1 or t_2 is outside $stab_H(G)$ and it is necessarily a leaf since it cannot have type in G, (5) holds.

So f verifies none of the conditions and f' verifies at least one. It is clear that the four first conditions are detectable in two rounds, the fifth is because f is a subtree of s of type in G so by construction of s it has no leaf with type outside $stab_H(G)$ in twigs. □

The second step is proven using the induction hypothesis. We prove that if Duplicator can win the game on two X-trimmed trees of type in G, they have the same type. This is done by building a smaller algebra than (H, V, W) from *stab* which coincides with (H, V, W) on X-trimmed trees of type in G and using the induction hypothesis on that smaller algebra. Aside from technical details, this proof is similar to the one used in the unranked case. Using these two steps, we are able to derive a winning strategy for Duplicator on r and r' from her winning strategy on t and t', proving Lemma 3.

5.2 Second Case

In this case we suppose that there is a b in B such that $\alpha(b) \notin stab_V(v)$ or a $c \in C$ such that $\alpha(c) \notin stab_W(v)$. Let $B_\downarrow = \{b_1, ..., b_n\}$ be the set of all such

$b \in B$ and $C_\downarrow = \{c_1, ..., c_m\}$ be the set of all such $c \in C$. We reduce the size of the alphabet by suppressing all such b and c. For two trees s, s', we write:

$$s \equiv s' \text{ when } \forall u <_{\mathcal{R}} v \; u\alpha(s) = u\alpha(s')$$

Notice that $\equiv$ is an equivalence relation of finite index. Let $L_1, ..., L_\iota$ be the classes of this equivalence relation and $a_1, ..., a_\iota$ be leaf representatives of those classes. We write $a_{i,l}$ the leaves that have the same type as the trees $b_l(a_i)$ and $a_{i,j,l}$ leaves that have the same type as the trees $c_l(a_i, a_j)$. We modify t and t' in two steps:

1. First we consider each minimal subtree of t which has its root labeled by a $b_l \in B_\downarrow$ (minimal in the sense that it is not subtree of a tree of root in $B_\downarrow$ or $C_\downarrow$). We look at the subtree rooted in b_l, we compute its class L_i of $\equiv$, and we replace it by a_i. We do the same with the minimal subtrees that have their root labeled by $c_l \in C_\downarrow$. We look at the two subtrees that are rooted to the node and we compute their class L_i and L_j of $\equiv$, and we replace them by a_i and a_j. We obtain a new tree s. We modify t' the same way and we obtain s'.
2. Finally we replace the subtrees $b_l(a_i)$ by $a_{i,l}$ and $c_l(a_i, a_j)$ by $a_{i,j,l}$. We have now a pair of trees that we call r and r'.

We state two lemmas that are central to this case. The first one is a consequence of the definition of the relation $\equiv$ and of the sets $B_\downarrow$ and $C_\downarrow$.

Lemma 4. *$v\alpha(t) = v\alpha(s) = v\alpha(r)$ and $v\alpha(t') = v\alpha(s') = v\alpha(r')$*

Lemma 5. $\bar{r} \cong_{\frac{k}{2}-1} \bar{r}'$

Before proving those results, we use them to conclude this case. We have:

1. No node of label in $B_\downarrow$ or $C_\downarrow$ is present in r and r' so the size of the node alphabets have decreased in r and r'.
2. $\frac{k}{2} - 1 \geq 2^{2(|B|+|C|-1)}K$ because $k \geq 2^{2(|B|+|C|)}K$.

Therefore using Lemma 5 by induction we get $v\alpha(r) = v\alpha(r')$, using Lemma 4 we conclude $v\alpha(t) = v\alpha(t')$.

Lemma 5 is proven in two steps. The first step is that Spoiler can detect in t and t' the subtrees that are to be deleted in order to build r and r'. The second step is that he is able to detect for each of those subtrees the label of the leaf that will be used in order to replace it in r and r'. The first step is a consequence of the fact that those subtrees are rooted with nodes whose label do not appear above. The second step is a consequence of the fact that if Duplicator wins the EF game on two trees it means that they have the same type under all contexts that are strictly smaller than v relatively to $\leq_{\mathcal{R}}$ (induction hypothesis), which means that by definition they are equivalent under $\equiv$. Given these two steps, we are able to derive a winning strategy for Duplicator on r and r' from her strategy on t and t'. While technically more difficult those proofs are similar in spirit to the unranked case.

5.3 Third Case

We suppose that we are not in one of the two previous cases, meaning that for all $b \in B$, $\alpha(b) \in stab_V(v)$, for all $c \in C$, $\alpha(c) \in stab_W(v)$ and $H - X = H_{min}$. If t is a leaf, so is t' and vice versa. Therefore we can suppose that t and t' are not leaves. In this case, t and t' have their types in H_{min} (they are X-trimmed). We prove that for all $h, g \in H_{min}$ $vg = vh$, in particular $v\alpha(t) = v\alpha(t')$ which concludes this case.

Let u_h and u_g be as defined in the Lemma 1. We assume that:

$$u_h = \alpha(c_1 \diamond a_1)...(c_n \diamond a_n)$$
$$u_g = \alpha(c_{n+1} \diamond a_{n+1})...(c_m \diamond a_m)$$

We supposed that there were no unary symbols in order to simplify the expressions, this does not affect the proof. We also supposed that all subtrees off the main path were leaves, this is not restrictive since we supposed that all types were reachable with a leaf. By hypothesis, $\forall i$ $c_i \in stab_W(v)$, so we have an a'_i such that $\alpha(c_i \diamond a'_i) \in stab_V(v)$. We write:

$$w_h = \alpha(c_1 \diamond a'_1)...(c_n \diamond a'_n)$$
$$w_g = \alpha(c_{n+1} \diamond a'_{n+1})...(c_m \diamond a'_m)$$

By construction of w_g and w_h:

$$vw_hw_hw_gw_g \mathcal{R} v$$
$$vw_hw_hw_gw_gv' = v \text{ for some v'}$$
$$v(w_hw_hw_gw_gv')^\omega = v$$

Hence by definition of u_h:

$$v(w_hw_hw_gw_gv')^\omega(u_hw_hu_gw_gv')^\omega g = vh$$

Notice that we have $w_gw_g \dashv u_gw_g$ and $w_hw_h \dashv u_hw_h$, which allows us to use Equation (5):

$$vh = v(w_hw_hw_gw_gv')^\omega u_gw_gv'(u_hw_hu_gw_gv')^\omega g$$
$$vh = v(w_hw_hw_gw_gv')^\omega g \qquad \text{(by definition } \forall g' \; u_gg' = g)$$
$$vh = vg$$

This completes the proof.

6 Characterization of Δ_2

In this section we give a decidable characterization of Δ_2. Like the characterization for unranked trees in [3], our characterization use the piece relation. However, our definition of piece is more restrictive than the one used in [3], a piece of a tree is obtained by suppressing some nodes and attaching the remaining nodes such that the ancestor relation is preserved. With this definition, every piece of an unranked tree is an unranked tree, but a piece of a binary tree need not be a binary tree. Therefore, our piece relation only considers binary pieces.

Definition 6. *Over an alphabet (A, B, C), we say that a tree t is a piece of a tree s if and only if there is an injective morphism of the nodes of t to the nodes of s that preserves labels and the ancestor relation, we write $s \preceq t$. This relation is naturaly extended on contexts (recall that a context is a tree with a special node $*$).*

Given $u, v \in V$ we say that u is a piece of v and write $u \preceq v$ if and only if there is some morphism $\alpha : (A, B, C)^{\Delta} \to (H, V, W)$ with $\alpha(p) = u$ and $\alpha(q) = v$ such that $p \preceq q$.

Theorem 2. *A binary tree language L is definable in Δ_2 if and only if it belongs to TDA and verifies the following identity:*

$$u^{\omega} = u^{\omega} v u^{\omega} \quad \forall u, v \in V \text{ such that } v \preceq u \tag{6}$$

Notice that it is sufficient to consider only identities (2) and (6), since identities (1) and (3) of TDA are direct consequences of (6).

Those identities are almost identical to the ones stated in the unranked characterization given in [3], the only difference being the restriction we made by considering only binary pieces in our piece relation $\preceq$. The impact of this restriction on the proof remains minimal in this case. However, technical differences related to the ranked tree structure do appear in the proof. Of the three logics we consider in this paper, Δ_2 is the one whose characterization has the closest proof to its unranked variant. The main differences reside in the common part with the $EF+F^{-1}$ logic in which we defined the *stab* sets and proved their properties. The ranked tree structure actually makes the rest of the proof technically simpler in this case.

Since the identities are almost identical to the ones of the unranked case, the proof of the easy direction of the Theorem is exactly the same as in the unranked case.

7 Characterization of Boolean Combinations of Σ_1

In this section we give a decidable characterization of boolean combinations of Σ_1. We use the notion of piece we defined when we stated the characterization of Δ_2.

Theorem 3. *A binary tree language L is definable in Σ_1 if and only if its syntactic binary algebra verifies the following properties:*

$$u^{\omega} v = u^{\omega} = v u^{\omega} \quad v \preceq u \tag{7}$$

for all $u, v \in V$.

Notice that once again the identity is very close to the one used in the characterization over unranked trees. Like the characterization of Δ_2, the only difference in the statement, is the restriction to binary pieces. However, while this restriction was anecdotic in Δ_2, it leads to many problems for this logic. An example

is that for unranked trees, if you take a piece of a tree and split the tree into a context and another tree, the piece is also naturaly splited into a piece of the context and a piece of the new tree. This property which is extensively used in the proof of the unranked characterization is not true on ranked trees, which forces us to be careful when we split trees.

Since, the identity is similar to the one stated in [7] for unranked trees, the easy direction of Theorem 3 is identical, if a language is definable by a boolean combination of Σ_1 formulas, its syntactic algebra verifies (7).

8 Discussion

We gave characterizations for $EF + F^{-1}$, Δ_2 and Boolean Combinations of Σ_1 for binary trees. However, these results could easily be extended to trees of rank k for a fixed k by extending the algebraic framework. For example, trees of rank three would be characterized by adding an other set representing tri-contexts, (contexts with three holes which are all siblings).

A relevant question would be to consider extensions of these logics. The only relation we considered is the order $<$, but what about other relations? We could add an order between siblings or a vertical successor relation.

References

1. Benedikt, M., Segoufin, L.: Regular tree languages definable in FO. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 327–339. Springer, Heidelberg (2005)
2. Bojańczyk, M.: Two-way unary temporal logic over trees. In: 22nd IEEE Symposium on Logic in Computer Science, pp. 121–130 (2007)
3. Bojańczyk, M., Segoufin, L.: Tree languages defined in first-order logic with one quantifier alternation (2008)
4. Bojańczyk, M., Walukiewicz, I.: Characterizing EF and EX tree logics. Theoritical Compututer Science 358(2-3), 255–272 (2006)
5. Bojańczyk, M., Walukiewicz, I.: Forest algebras. In: Automata and Logic: History and Perspectives, pp. 107–132. Amsterdam University Press, Amsterdam (2007)
6. Vardi, M.Y., Etessami, K., Wilke, T.: First-order logic with two variables and unary temporal logic. In: 12th IEEE Symposium on Logic in Computer Science, pp. 228–235 (1997)
7. Segoufin, L., Bojańczyk, M., Straubing, H.: Piecewise testable tree languages (2008)
8. McNaughton, R., Papert, S.: Counter-Free Automata. MIT Press, Cambridge (1971)
9. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Information and Control 8, 190–194 (1965)
10. Thérien, D., Wilke, T.: Over words, two variables are as powerful as one quantifier alternation. In: 30th ACM Symposium on Theory of Computing, pp. 234–240 (1998)
11. Wilke, T.: An algebraic characterization of frontier testable tree languages. Theoretical Computer Science 154(1), 85–106 (1996)

The Nesting-Depth of Disjunctive μ-Calculus for Tree Languages and the Limitedness Problem

Thomas Colcombet[1] and Christof Löding[2]

[1] Liafa, Cnrs and Université Paris Diderot, Case 7014, 75205 Paris Cedex 13, France
thomas.colcombet@liafa.jussieu.fr
[2] RWTH Aachen, Informatik 7, 52056 Aachen, Germany
loeding@cs.rwth-aachen.de

Abstract. In this paper we lift the result of Hashiguchi of decidability of the restricted star-height problem for words to the level of finite trees. Formally, we show that it is decidable, given a regular tree language L and a natural number k whether L can be described by a disjunctive μ-calculus formula with at most k nesting of fixpoints. We show the same result for disjunctive μ-formulas allowing substitution. The latter result is equivalent to deciding if the language is definable by a regular expression with nesting depth at most k of Kleene-stars.

The proof, following the approach of Kirsten in the word case, goes by reduction to the decidability of the limitedness problem for nondeterministic nested distance desert automata over trees. We solve this problem in the more general framework of alternating tree automata.

1 Introduction

For regular languages of finite words, the star-height problem is to determine for a given language the minimal number of nestings of Kleene-stars required in a regular expression describing this language. The star-height can be seen as a measure for the complexity of a regular language. This problem was raised by Eggan in [6] who showed that the star-height of languages induces a strict hierarchy.

The problem was solved 25 years later by Hashiguchi [7] with a very complex proof. Recently, Kirsten has given a new proof [8] using the notion of nested distance desert automata. These automata compute a value (a natural number) for each accepted input. Kirsten showed that the star-height problem can be reduced to the limitedness problem for nested distance desert automata. A nested distance desert automaton is called limited if the mapping it computes is bounded. This problem is solved in [8] using an algebraic approach.

In this paper we lift Kirsten's result from words to trees. The theory of regular languages of finite trees has been initiated in [9,5] and since then it has turned out that a lot of concepts can be generalized from words to the more general setting of trees (see [4] for an overview). Similar to the case of words, regular languages of finite trees can be described by several formalisms, including finite automata and regular expressions. The definition of regular expressions for trees

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 416–430, 2008.
© Springer-Verlag Berlin Heidelberg 2008

is very similar to the one for words using the operations of union, concatenation, and iteration (Kleene-star). So the star-height problem for regular tree languages is posed in the same way as for words. However it is more convenient to refer to the nesting-depth problem for disjunctive μ-calculus. Disjunctive μ-calculus is another way of defining regular languages of trees. This formalism uses a fix point operator μ in place of the Kleene-star, and comes in two variants: with substitution and without substitution. In both cases, the number of nestings of μ-fix-points induces a strict hierarchy of languages. In the case with substitution, this hierarchy coincides with the star-height hierarchy.

As mentioned above, the proof of Kirsten consists of two main steps: a reduction of the star-height problem to the limitedness problem for nested distance desert automata, and the decidability proof for the limitedness problem. Our proof is along the same lines. The first step is very similar to the one presented in [8]. We define a natural extension of distance desert automata from words to trees (but we prefer to call them cost automata) and reduce the nesting-depth problem (both with and without substitution) for tree languages to the limitedness problem for cost automata. One should note here that in [8] a specific construction is used to obtain an automaton for the given language that has some good properties making the reduction work. We introduce here the notion of subset automata as an abstract notion for capturing the properties of an automaton required for the reduction. Then we show that every regular language of trees can be accepted by such a subset automaton.

The main obstacle for the second step of the proof is that the algebraic objects required for dealing with languages of trees are more complex than the ones that can be used when dealing with words. To describe the behavior of a tree automaton one has to capture the fact that it runs in parallel over many branches of the tree. We solve this problem by reducing the limitedness problem for cost automata to the same problem for the much simpler class of purely non-deterministic automata. These automata, started at the root of the tree, only move to one of the successors of the current node. In this way each of their computations corresponds to a single path through the tree. This reduction uses game-theoretic arguments and even works if we start from an alternating cost automaton instead of a non-deterministic one. Then, proving the decidability of the limitedness problem for purely non-deterministic automata is a technical work that relies on algebraic arguments similar to the proof of Kirsten. New ideas inspired from [2] are required for making the proof compatible with the branching nature of trees.

To sum up, the contributions of this paper are the following:

1. We show the reduction of the nesting-depth problem for tree languages to the limitedness problem for cost automata (Lemma 5). This reduction is done twice, in the case of disjunctive μ-calculus with substitution, as well as without substitution. The new notion of subset automata – that we believe to be of independent interest – is used.
2. We prove the decidability of the limitedness problem in the more general framework of alternating cost automata (Theorem 2). This requires new

game-theoretic arguments for a first reduction to the case of purely nondeterministic automata, as well as an involved variant of the proof of Kirsten.

Combining these results we obtain our main theorem:

Theorem 1. *Given a regular language L of trees the following values and corresponding formulas or expressions can be computed.*

1. *The minimal nesting-depth of a disjunctive μ-calculus formula for L.*
2. *The minimal nesting-depth of a disjunctive μ-calculus formula with substitution for L.*
3. *The minimal star-height of a regular expression for L.*

The paper is organized as follows. In Section 2 we give the main definitions concerning tree automata, disjunctive μ-calculus, and cost automata. Section 3 presents the reduction of the nesting-depth problem to the limitedness problem for cost automata on trees. In Section 4 we show that the limitedness problem for alternating cost automata on trees is decidable.

2 Definitions

In this section we introduce the basics on trees, tree automata, μ-calculus, and regular expressions. The reader not familiar with the subject of regular tree languages is referred to [4] for an introduction.

2.1 Trees and Patterns

We fix from now a *ranked alphabet* A, i.e., a finite set of symbols, together with an arity $|a|$ for every $a \in A$. The set of (finite) *trees* $\mathcal{T}$ is the least set such that for all $a \in A$ and all $t_1, \ldots, t_{|a|} \in \mathcal{T}$, $a(t_1, \ldots, t_{|a|}) \in \mathcal{T}$. In case of $|a| = 0$ we simply write a instead of $a()$. If we want to make the alphabet of labels explicit we also refer to a tree as an A-tree. Given a finite set X of *variables* disjoint from A, the set of X-*patterns* $\mathcal{T}[X]$ is the least set containing X and such that for all $a \in A$ and all $t_1, \ldots, t_{|a|} \in \mathcal{T}[X]$, $a(t_1, \ldots, t_{|a|}) \in \mathcal{T}[X]$. Hence trees are just $\emptyset$-patterns. Sets of trees and sets of patterns are called *languages*. For languages $L_1, \ldots, L_{|a|}$ we denote by $a(L_1, \ldots, L_a)$ the language $\{a(t_1, \ldots, t_{|a|}) \ : \ t_1 \in L_1, \ldots, t_{|a|} \in L_{|a|}\}$.

As usual, the *domain* $dom(t) \subseteq \mathbb{N}^*$ of a tree or pattern $t = a(t_1, \ldots, t_{|a|})$ is defined inductively as $dom(t) = \{\varepsilon\} \cup \bigcup_{i=1}^{|a|} i \cdot dom(t_i)$, and we view t as a mapping from its domain to the alphabet A (and X in the case of patterns). The elements of the domain are called *nodes*. We refer to nodes that are labeled by some $x \in X$ as *variables nodes*. Nodes that are labeled with symbols of arity 0 or variables are called *leaves*. The other ones are called *inner nodes*.

Given two sets X, Y disjoint from A, a mapping v from X to languages of Y-patterns, and an X-pattern t, we denote by $t[v]$ the language of Y-patterns obtained by replacing every $x \in X$ appearing in t by some $t' \in v(x)$. We lift this notation to languages of X-patterns, $K[v] = \bigcup_{t \in K} t[v]$. By $K[x := K']$ we denote the set of patterns that is obtained by taking a pattern from K and replacing each x with some pattern from K'.

2.2 Automata

A *non-deterministic tree automaton* is of the form $\mathcal{A} = (Q, A, In, \Delta)$, where Q is a finite set of *states* disjoint from A, A is the ranked alphabet, $In \subseteq Q$ is the set of *initial states*, $\Delta \subseteq \cup_{a \in A} Q \times \{a\} \times Q^{|a|} \cup Q \times \{\epsilon\} \times Q$ is the *transition relation*. Transitions of the form (q, ϵ, r) are called *ϵ-transitions*, and transitions for symbols of arity 0 are written as (q, a). Given a state q of $\mathcal{A}$ we denote by $\mathcal{A}_q$ the automaton $\mathcal{A}$ with q as only initial state.

To define acceptance of an automaton we use the notion of *run*. If the automaton does not have ϵ-transitions, then a run ρ on a tree t is a Q-tree (in which states can have any arity) with the same domain as t that starts in an initial state and respects the transitions. But as we are working with automata with ϵ-transitions, the domain of a run can be different from the domain of the input tree. Therefore we adopt an inductive definition of runs. Let t be an A-tree.

- If $t = a$ and $(q, a) \in \Delta$, then q is a run of $\mathcal{A}$ on t.
- If $t = a(t_1, \ldots, t_{|a|})$ and $(q, a, q_1, \ldots, q_{|a|}) \in \Delta$, then $q(\rho_1, \ldots, \rho_{|a|})$ is a run of $\mathcal{A}$ on t, where each ρ_i is a run of $\mathcal{A}$ on t_i with state q_i at the root.
- If $(p, \epsilon, q) \in \Delta$ and ρ' is a run of $\mathcal{A}$ on t with state q at the root, then $p(\rho')$ is also a run of $\mathcal{A}$ on t.

A tree t is accepted by $\mathcal{A}$ if there is a run of $\mathcal{A}$ on t that starts in an initial state. The language $L(\mathcal{A})$ is the set of all trees that are accepted by $\mathcal{A}$. A regular language is a language that is accepted by some automaton.

If we want an automaton $\mathcal{A}$ to read X-patterns instead of solely trees, then we explicitly specify the transitions that the automaton can use at leaves labeled with a variable. This is done by giving a relation between states and variables. If R is such a relation, then $\mathcal{A}[R]$ denotes the automaton $\mathcal{A}$ with the additional transitions (q, x) for $(q, x) \in R$.

2.3 Disjunctive μ-Calculus

In this section, we introduce two other formalisms for describing regular languages of trees, namely the disjunctive μ-calculus and regular expressions.

A *disjunctive μ-calculus formula with substitution* (simply *μ-formula with substitution* from now) has the following syntax:

$$\phi ::= \quad \bot \quad | \quad a(\underbrace{\phi, \ldots, \phi}_{|a|}) \quad | \quad \phi + \phi \quad | \quad x \quad | \quad \mu x.\phi \quad | \quad \phi[x := \phi] \ ,$$

in which $a \in A$, and x is a *variable*. If $|a| = 0$ we just write a instead of $a()$. If a μ-formula with substitution does not use the rule $\phi[x := \phi]$, it is simply called a *μ-formula*. One defines the *free variables* of a μ-formula as usual. A μ-formula with no free variables is *closed*.

The semantics $[\![\phi]\!]$ of a μ-formula is the language of patterns defined by:

- $[\![\bot]\!] = \emptyset$, $[\![x]\!] = \{x\}$,
- $[\![a(\psi_1, \ldots, \psi_{|a|})]\!] = a([\![\psi_1]\!], \ldots, [\![\psi_{|a|}]\!])$, $[\![\psi + \psi']\!] = [\![\psi]\!] \cup [\![\psi']\!]$,

– $[\![\phi[x := \psi]]\!] = [\![\phi]\!][x := [\![\psi]\!]]$,
– $[\![\mu x.\psi]\!] = \bigcup_{n\in\mathbb{N}} L_n$, in which $L_0 = \emptyset$ and $L_{n+1} = L_n \cup [\![\psi]\!][x := L_n]$.

A closed μ-formula with substitution defines a regular language of trees. Reciprocally, every regular language of trees is the semantics of a μ-formula.

The *nesting-depth* of a μ-formula ϕ (with or without substitution) is the maximal number of nestings of fixpoint operators:

– $nd(\bot) = nd(x) = 0$,
– $nd(\phi + \phi') = nd(\phi[x := \phi']) = \max(nd(\phi), nd(\phi'))$,
– $nd(a(\phi_1, \ldots, \phi_{|a|})) = \max(nd(\phi_1), \ldots, nd(\phi_{|a|}))$,
– $nd(\mu x.\phi) = 1 + nd(\phi)$.

Remark 1. Regular expressions over words have be extended to trees, see e.g., [4], Chapter 2. Star-height can be defined in this framework as for word languages. This parameter is linked to the nesting-depth as follows: Each regular tree language can be defined by a μ-formula with substitution of nesting-depth k iff it can be defined by a regular expression of star-height k. Hence solving the nesting-depth problem also solves the star-height problem.

2.4 Cost Automata

We now extend the model of tree automata such that trees are not just accepted or rejected but furthermore a cost is computed. For this purpose, we add a function that assigns to each state a priority from a set D. This set D is totally ordered and partitioned into *increments* and *resets*. We can view such an automaton as having as many counters as there are increments. Whenever a state is visited that is assigned an increment, then the corresponding counter is incremented, and all counters for smaller increments (recall that D is ordered) are reset. If the automaton visits a state that is assigned a reset, then all counters for increments that are smaller than this reset are set to 0. If we consider a run of such an automaton, then the cost along a path through the run corresponds to the maximal value of one of the counters. The cost of a run is the maximal cost of all the paths. The cost of a tree is the minimal cost over all runs for this tree.

Formally, a cost tree automaton is of the form $\mathcal{A} = (Q, A, In, \Delta, pri)$, where the first four components are as before, and $pri : Q \to D$ is a priority function. The set D of priorities is totally ordered, and the elements of D are referred to as increments and resets. Usually, it is of the form $D = \{I_1, R_1, \ldots, I_k, R_k\}$ where the I_i are increments, the R_i are resets, and the order is $I_1 < R_1 < \cdots < I_k < R_k$. The same notation is used in [1] for hierarchical B-automata, which work in the same way as our cost automata but on words and not on trees. In [8] the increments are called $\angle_i$ (péage) and the resets $\curlyvee_i$ (source).

A run ρ on a tree t is defined as for standard tree automata. The language $L(\mathcal{A})$ is also defined as if no counters where involved. The difference is that a cost is associated to each run, each tree, and each language by the automaton. We start by defining the cost of a sequence of states. Let $\sigma = p_0 p_1 \cdots p_n \in Q^*$. If all

priorities in σ are at most I_i, then we write $|\sigma|_{I_i}$ for the number of states with priority I_i in σ:

$$|\sigma|_{I_i} = \begin{cases} 0 \text{ if } pri(p_j) > I_i \text{ for some } j, \\ |\{j \ : \ pri(p_j) = I_i\}| \text{ otherwise.} \end{cases}$$

Intuitively, this corresponds to the number of increments to the counter for I_i. The condition that no priority higher that I_i occurs means that the counter is not reset. The cost of σ is defined as

$$val(\sigma) = \max\{|\sigma'|_{I_i} \ : \ i \in [k] \text{ and } \sigma' \text{ is a factor of } \sigma\},$$

where σ' is a factor of σ if $\sigma = \sigma_1\sigma'\sigma_2$ for some σ_1, σ_2. The cost $val(\rho)$ of a run ρ is defined as the maximal cost of all the state sequences along paths through ρ. The cost assigned to a tree t by the automaton $\mathcal{A}$ is:

$$\mathcal{A}(t) = \min\{val(\rho) \ : \ \rho \text{ is a run of } \mathcal{A} \text{ on } t\} \qquad (\text{and } \omega \text{ if } t \notin L(\mathcal{A})).$$

Given a language of trees K, we define its cost for the automaton $\mathcal{A}$ as:

$$\mathcal{A}(K) = \sup\{\mathcal{A}(t) \ : \ t \in K\} \qquad (\text{and } 0 \text{ if } K = \emptyset).$$

This value $\mathcal{A}(K)$ can be ω, and this for two reasons: either if $K \not\subseteq L(\mathcal{A})$, or $K \subseteq L(\mathcal{A})$ but K contains trees of arbitrary high costs.

A cost automaton $\mathcal{A}$ is *limited* if $\mathcal{A}(L(\mathcal{A})) < \omega$. It is *uniformly universal* if $\mathcal{A}(\mathcal{T}) < \omega$. Those two notions are tightly related as follows:

Remark 2. A cost automaton is uniformly universal iff it is both universal (as a standard tree automaton) and limited. Conversely, given a tree automaton $\mathcal{C}$ accepting the language complement of $L(\mathcal{A})$, one can see it as a cost tree automaton of single priority R_1. Then $\mathcal{A}$ is limited iff $\mathcal{A}+\mathcal{C}$ is uniformly universal, in which $\mathcal{A}+\mathcal{C}$ is the disjoint union of the automata $\mathcal{A}$ and $\mathcal{C}$ (as the standard construction for the union of languages).

When we reduce the problems of determining the star-height or the nesting-depth of a language to the uniform universality problem for cost automata, then we use automata with ϵ-transitions. The solution of the uniform universality problem is presented for automata without ϵ-transitions. To justify this we now present a result that allows to remove ϵ-transitions while preserving uniform universality. The proof uses a construction that replaces sequences of ϵ-transitions by a single state whose priority is the maximal priority occurring in this sequence.

Lemma 1. *For each cost automaton $\mathcal{A}$ there exists a cost automaton $\mathcal{B}$ without ϵ-transitions such that for each language K of trees we have $\mathcal{A}(K) < \omega$ iff $\mathcal{B}(K) < \omega$.*

3 From Nesting-Depth to Limitedness

In this section we describe how it is possible to reduce the nesting-depth problem for regular tree languages to the the limitedness problem for non-deterministic cost automata. We present this reduction for the case of μ-formulas with substitution. The case without substitution follows the same lines.

The reduction consists of two parts. In the first one we present subset automata, which are non-deterministic tree automata that have special properties with respect to the subset ordering of languages. In the second part we construct a cost tree automaton and present our main Lemma 6 relating the nesting-depth to the limitedness problem for this automaton.

3.1 Subset Automata

We define in this section the notion of a subset-automaton. Though we do not develop this aspect in the present abstract, we point out that this notion is purely driven by algebraic considerations.

Given a tree automaton $\mathcal{A} = (Q_\mathcal{A}, A, In_\mathcal{A}, \Delta_\mathcal{A})$, the transitions in $\Delta_\mathcal{A}$ are partitioned into ϵ-transitions $\Delta^\epsilon_\mathcal{A}$ and non-ϵ-transitions $\Delta^{\neg\epsilon}_\mathcal{A}$. The automaton is a *subset-automaton* if it satisfies the following items:

1. $\mathcal{A}^{\neg\epsilon} = (Q_\mathcal{A}, A, In_\mathcal{A}, \Delta^{\neg\epsilon}_\mathcal{A})$ is (bottom-up) deterministic and complete, i.e., for all $a \in A$ and $p_1, \dots, p_{|a|} \in Q_\mathcal{A}$, the set $\{p \ : \ (p, a, p_1, \dots, p_{|a|}) \in \Delta^{\neg\epsilon}_\mathcal{A}\}$ is a singleton; we denote by $a_\mathcal{A}(p_1, \dots, p_{|A|})$ its sole element.
2. The relation $p \leq q$ if $(q, \epsilon, p) \in \Delta^\epsilon_\mathcal{A}$ equips $Q_\mathcal{A}$ with a complete sup-semilattice structure, i.e, $\leq$ is an order, and every subset P of $Q_\mathcal{A}$ has a least upper bound $\vee P$ for the order $\leq$ (in particular, there exists a minimum element $\bot_\mathcal{A} = \vee\emptyset$ and a maximal element $\top_\mathcal{A} = \vee Q_\mathcal{A}$).
3. For all $a \in A$, the mapping $a_\mathcal{A}$ is continuous with respect to the sup-semilattice $(Q_\mathcal{A}, \leq)$, i.e., for every $P_1, \dots, P_{|a|} \subseteq Q_\mathcal{A}$,

$$a_\mathcal{A}(\vee P_1, \dots, \vee P_{|a|}) = \vee\{a_\mathcal{A}(p_1, \dots, p_{|a|}) \ : \ p_1 \in P_1, \dots, p_{|a|} \in P_{|a|}\} \ .$$

4. $In_\mathcal{A} = \{q \in Q_A \ : \ q \leq q_0\}$ for some q_0 in $Q_\mathcal{A}$.

Since $\mathcal{A}^{\neg\epsilon}$ is deterministic and complete, given a tree t, there exists a unique state $f(t)$ such that $\mathcal{A}^{\neg\epsilon}_{f(t)}$ has a run over t. Remark that in an algebraic framework, Item 1 means that the mappings $a_\mathcal{A}$ equip $Q_\mathcal{A}$ with a tree algebra structure. The mapping f is nothing but the unique tree algebra morphism from the free algebra of trees to this algebra. As f depends on the automaton $\mathcal{A}$, it should rather be called $f_\mathcal{A}$. However, we drop the subscript to simplify the notation.

Consider now a tree language K, define:

$$f(K) = \bigvee_{t \in K} f(t) \ , \qquad \text{and} \quad F(K) = L(\mathcal{A}_{f(K)}) \ .$$

This extended mapping f is nothing but the extension of the previous morphism to the setting of tree algebras equipped with a complete sup-semi-lattice structure. The corresponding free algebra is the set of tree languages equipped with the inclusion ordering. With this equivalence in mind the following lemmas are natural. Our first lemma makes the morphism properties of f explicit.

Lemma 2. *For all sets of tree languages $Z \subseteq 2^{\mathcal{T}}$, $f\left(\bigcup_{Y\in Z} Y\right) = \bigvee_{Y\in Z} f(Y)$. For all $a \in A$ and tree languages $K_1, \ldots, K_{|a|}$,*

$$f(a(K_1, \ldots, K_{|a|})) = a_{\mathcal{A}}(f(K_1), \ldots, f(K_{|a|})) \ .$$

The second lemma shows how f is related to the semantics of the automaton $\mathcal{A}$.

Lemma 3. *For all states $q \in Q_{\mathcal{A}}$, all trees t, and all tree language K:*

- $t \in L(\mathcal{A}_q)$ *iff* $f(t) \leq q$,
- $K \subseteq L(\mathcal{A}_q)$ *iff* $F(K) \subseteq L(\mathcal{A}_q)$ *iff* $f(K) \leq q$.

The first item of the lemma characterizes the language accepted from state q. The second statement shows that this equivalence can be raised to the level of languages. More precisely, there is a very simple way to check if a language K is included in some $L(\mathcal{A}_q)$ (this is not symmetric and does not work for the superset relation), hence the name of a subset-automaton.

Finally, we need the following:

Lemma 4. *Every regular language is accepted by a subset-automaton.*

There are several ways for proving this lemma, each one of different interest. Here we sketch two possibilities. Let L the tree language for which we want to find a subset automaton.

An adapted version of the construction used by Kirsten [8] starts from an automaton $\mathcal{C}$ (with state set $Q_{\mathcal{C}}$) accepting the *complement* language of L. Then one constructs a non-deterministic automaton $\mathcal{A}$ with state set $2^{Q_{\mathcal{C}}}$ in such a way that for all $P \subseteq Q_{\mathcal{C}}$ and for all trees t: $t \in L(\mathcal{A}_P)$ iff $\forall q \in P,\ t \notin L(\mathcal{C}_q)$. Such an automaton accepts L with initial states $\{P \subseteq Q_{\mathcal{C}} \ : \ In_{\mathcal{C}} \subseteq P\}$ (in which $In_{\mathcal{C}}$ is the set of initial states of $\mathcal{C}$). The states are equipped with a complete sup-semi-lattice structure by $P \leq R$ iff $R \subseteq P$. The automaton obtained by adding the corresponding transitions (R, ϵ, P) to $\mathcal{A}$ yields a subset automaton accepting L. This construction yields an exponential upper bound in the size of an automaton that accepts the complement of the language.

Second, the language theoretic construction consists in considering the set of residuals of L (a language is a residual if it is of the form $\{t \ : \ s[x := t] \in L\}$ in which s is an $\{x\}$-pattern with a single unique occurrence of x). It is classical that every regular tree language L has finitely many residuals. One constructs an automaton that has intersections of residuals as states. Those intersections induce a complete sup-semi-lattice structure for the inclusion. The property of residuals makes the remaining of the construction unique from this point. Once more this construction yields a subset automaton; more precisely, the minimal one.

3.2 Reduction of the Nesting-Depth Problem to Limitedness

The reduction is stated in the following lemma.

Lemma 5. *Given a regular tree language L and a natural number k, there exists effectively a cost tree automaton that is limited iff L can be defined by a μ-formula of nesting-depth at most k. The same statement holds for μ-formulas with substitution.*

We sketch the proof here for the case with substitution. Consider a regular tree language L, a subset automaton $\mathcal{A}$ for the language L, and the corresponding mapping f. We construct for every $k \in \mathbb{N}$ a cost automaton $\mathcal{B}^k = (Q_k, A, In_k, \Delta_k, pri_k)$ and a mapping π from Q_k to $Q_\mathcal{A}$ as follows:

- Q_k is the set of nonempty words over $Q_\mathcal{A}$ of length up to $k+1$, we set $\pi(u)$ to be the last letter of u.
- In_k is $In_\mathcal{A}$, i.e., words consisting of a single initial state of $\mathcal{A}$.
- Δ_k contains all transitions of the form:
  1. $(up, a, up_1, \ldots, up_{|a|})$ whenever $(p, a, p_1, \ldots, p_{|a|}) \in \Delta_A$, (up, ϵ, ur) whenever $(p, \epsilon, r) \in \Delta_\mathcal{A}$,
  2. (up, ϵ, upp),
  3. (uqp, ϵ, up).
- $pri_k(u) = \begin{cases} R_{k+2-|u|} & \text{if } u = vpp \text{ for some } p \in Q_\mathcal{A} \\ I_{k+2-|u|} & \text{else.} \end{cases}$

It should be rather clear that $L(\mathcal{B}^k) = L(\mathcal{A}) = L$. Indeed, every run of $\mathcal{A}$ can be seen as a run of $\mathcal{B}^k$, and conversely every run of $\mathcal{B}^k$ is mapped by π to a run of $\mathcal{A}$ with the same initial state. The key lemma is the following:

Lemma 6. *The cost automaton $\mathcal{B}^k$ is limited iff L is the evaluation of a μ-formula with substitution of nesting-depth at most k. In this case, the μ-formula with substitution can be effectively given.*

Let us give some ideas about the proof. From left to right: this part does not require $\mathcal{A}$ to be a subset automaton. Assuming that $\mathcal{B}^k$ is limited, one obtains a value $N = \mathcal{B}^k(L) < \omega$. The principle is to construct a μ-formula with substitution that is able to 'simulate' the behavior of the automaton $\mathcal{B}^k$ up to the value N of counters.

From right to left. The idea is to prove that for all μ-formulas with substitution ϕ of nesting-depth at most k, every tree in $[\![\phi]\!]$ is accepted by a run of $\mathcal{B}^k$ of cost at most $|\phi|$ (i.e., the size of ϕ) from state $f([\![\phi]\!])$. In practice, this is done via an induction on the structure of ϕ. This means that one has to deal with non-closed formulas and free variables. Hence, our induction hypothesis is more technical: given a μ-formula with substitution ϕ of nesting-depth at most k and free variables X, given a mapping v from X to tree languages,

$$\mathcal{B}^k_{f([\![\phi]\!][v])}[\{(x, f(v(x))) \ : \ x \in X\}]([\![\phi]\!]) \leq |\phi|.$$

There is no special difficulty in the proof itself. It of course relies heavily on the properties of $\mathcal{A}$ and f that we have presented above.

If ϕ has no free variables and evaluates to L, we get that $\mathcal{B}^k_{f(L)}(L) \leq |\phi| < \omega$. Recall that $\mathcal{B}^k$ accepts the language L. Since all trees in L are accepted by $\mathcal{B}^k$, this means by Lemma 3 that $f(L) \in In_k$. Hence $\mathcal{B}^k(L(\mathcal{B}^k)) < \omega$: the cost automaton $\mathcal{B}^k$ is limited.

4 Decidability of the Limitedness Problem

The aim of this section is to show the following theorem.

Theorem 2. *The limitedness for alternating cost tree automata is decidable.*

In our proof we work with the uniform universality problem according to Remark 2. The proof goes in two steps: first we show how to reduce the uniform universality problem of alternating cost tree automata to the one of purely non-deterministic automata, a very weak form of non-deterministic automata (Lemma 10). We then show the decidability of the latter problem (Lemma 11). Among those two parts, we emphasize on the first one which is completely new, while the first one, though more involved, roughly follows the algebraic arguments of [8] in combination with ideas from [2] for handling trees.

The rest of this section is divided as follows. We first introduce in Section 4.1 cost games (a game theoretic counterpart to cost automata) and establish a result on positional strategies for them (Lemma 7). We then use in Section 4.2 cost games in a proof for Lemma 10. In Section 4.3 we present Lemma 11, the decidability of uniform universality for purely non-deterministic automata.

4.1 Cost Games

The semantics of alternating cost automata, which we introduce below, is defined by means of a game. For this reason we first introduce the general terminology for games that we need later.

A *cost game* is of the form $\mathfrak{G} = (V_E, V_A, v_0, E, pri, F)$ with the following components:

- $V := V_E \cup V_A$ is the finite set of vertices, where V_E are the vertices of Eva and V_A are the vertices of Adam (the sets V_E and V_A are disjoint).
- v_0 is the initial vertex.
- $E \subseteq V \times V$ is the set of edges. We require that the graph (V, E) is acyclic.
- $pri : V \to D$ is a priority function where D is as for cost tree automata.
- F is a subset of states that Eva should avoid.

A play σ is a finite sequence of vertices such that successive vertices in the sequence are connected by an edge (note that we consider finite and acyclic games and therefore only finite plays are possible). The cost $val(\sigma)$ of a play σ is defined as for automata with the only difference that the cost is ω if the play contains a vertex from F:

$$val(\sigma) = \begin{cases} \omega \text{ if } \sigma \text{ contains a vertex from } F, \\ \max\{|\sigma'|_{I_i} \ : \ i \in [k] \text{ and } \sigma' \text{ is a factor of } \sigma\} \text{ otherwise.} \end{cases}$$

The notion of strategy for Eva or Adam is defined as usual, it is a function that takes a play ending in a node of the respective player and maps it to one of the possible moves (if such a move exists, otherwise it is undefined). Given two strategies f_E for Eva and f_A for Adam, they define a unique play $\sigma(f_E, f_A)$ from the initial vertex v_0.

The goal of Eva is to minimize the cost of the play while Adam tries to maximize it. If we fix a threshold for the cost, then we can talk about winning strategies: We call a strategy f_E for Eva a winning strategy in $(\mathfrak{G}, val, N)$ if $\max_{f_A} val(\sigma(f_E, f_A)) \leq N$ (where f_A ranges over strategies for Adam). Similarly, a strategy f_A for Adam is called a winning strategy in $(\mathfrak{G}, val, N)$ if $\min_{f_E} val(\sigma(f_E, f_A)) > N$. As the plays of $\mathfrak{G}$ are of finite duration, for each N the game $(\mathfrak{G}, val, N)$ is determined.

Proposition 1. *For each N, either Adam or Eva has a winning strategy in $(\mathfrak{G}, val, N)$.*

From this proposition one can easily deduce that the following equality holds:

$$\min_{f_E} \max_{f_A} val(\sigma(f_E, f_A)) = \max_{f_A} \min_{f_E} val(\sigma(f_E, f_A))$$

where f_E and f_A range over strategies for Eva and Adam. We call the corresponding value the value of the game.

In the reduction from the uniform universality problem for alternating automata to the one for purely non-deterministic ones we want to annotate input trees with strategies of Adam. For this purpose we need positional strategies, i.e., strategies that make their choice only depending on the current vertex and not on the whole history of the play.

Unfortunately, positional strategies are not sufficient for Adam. But for our reduction it is enough if we can guarantee a positional winning strategy for a smaller value. In the following we show that this is indeed possible.

Formally, a *positional strategy* for Adam is a function $f_A : V_A \to V$ such that $(v, f_A(v)) \in E$ for all $v \in V_A$ that have an E-successor, and $f_A(v)$ is undefined otherwise.

Lemma 7. *Let $\mathfrak{G}$ be a cost game with k increments, and let $N \geq 1$. If Adam has a winning strategy in $(\mathfrak{G}, val, N^k - 1)$, then Adam has a positional winning strategy in $(\mathfrak{G}, val, N - 1)$.*

To prove the lemma we want to compute optimal values for Adam at each vertex of the game, and at the same time construct positional strategies in a bottom-up fashion, starting at the nodes without successor. The problem is that the way the value of a play is defined, the optimal choice for Adam at a position might depend on how the play arrived at this position. To avoid this problem we first define a new valuation *pval* ("p" for positional), show that *val* and *pval* are related as indicated in Lemma 7, and prove that this new valuation allows optimal positional strategies for Adam.

The formal definition of *pval* is parameterized by N, i.e., the function should be called $pval_N$ to be more precise. To avoid the subscript we fix some value N for the remainder of this section.

The idea for *pval* is the following. To allow a backward construction (starting from the leaves of the game) we evaluate the plays by reading them right to left. As before, when reading an increment I_i, the corresponding counter is increased and all the smaller ones are reset. But now we view the sequence of the counters as the digits of a single number encoded in base N. In particular, if a counter reaches value N, then it is set back to 0 and the next higher digit (counter) is increased by 1. The goal of Adam is to reach the value N^k, i.e., to exceed the highest value that can be represented with k digits in base N.

Formally, *pval* is a function $pval : V^* \to \{0, \ldots, N-1\}^k \cup \{\omega\}$. The set $\{0, \ldots, N-1\}^k \cup \{\omega\}$ is denoted as N^k_ω. We define *pval* by induction on the length of a play using the operator $\oplus : D \times N^k_\omega \to N^k_\omega$ defined as follows (using infix notation) according to the informal description above:

$$c \oplus (n_k, \ldots, n_1) = \begin{cases} (n_k, \ldots, n_{i+1}, 0, \ldots, 0) \text{ if } c = R_i, \\ \omega \text{ if } c = I_i \text{ and } n_j = N-1 \text{ for all } j \geq i, \\ (n_k, \ldots, n_{j+1}, n_j + 1, 0, \ldots, 0) \text{ for } c = I_i \text{ and the} \\ \qquad\qquad \text{smallest } j \geq i \text{ with } n_j < N-1. \end{cases}$$

and $c \oplus \omega = \omega$. Now we can define *pval* inductively by $pval(\varepsilon) = (0, \ldots, 0)$ and

$$pval(v\sigma) = \begin{cases} \omega & \text{if } v \in F, \\ pri(v) \oplus pval(\sigma) & \text{otherwise.} \end{cases}$$

Our first lemma relates the values computed by *val* and *pval*. The proof is based on a simple analysis of the definitions of the the two measures.

Lemma 8. *Let σ be a play.*

(a) If $val(\sigma) \geq N^k$, then $pval(\sigma) = \omega$.
(b) If $pval(\sigma) = \omega$, then $val(\sigma) \geq N$.

Having established this relation we now look at strategies for Adam when he tries to reach the *pval*-value of ω. We say that a strategy f_A for Adam in $\mathfrak{G}$ is a winning strategy in $(\mathfrak{G}, pval)$ if $\min_{f_E} pval(\sigma(f_E, f_A)) = \omega$. Note that according to Lemma 8 such a winning strategy is also a winning strategy in $(\mathfrak{G}, val, N-1)$.

For $(\mathfrak{G}, pval)$ a positional strategy can be constructed inductively starting at the leaves by always picking the optimal successor.

Lemma 9. *If Adam has a winning strategy in $(\mathfrak{G}, pval)$, then Adam has a positional winning strategy for $(\mathfrak{G}, pval)$.*

Lemma 7 is then a direct consequence of Lemmas 8 and 9.

4.2 Alternating and Purely Non-deterministic Automata

An alternating automaton in general does not send exactly one state to each successor of a node in a tree but it can send several states into the same direction and also no state at all in certain directions. For the formal definition we let

$\Gamma = \{1, \ldots, r\}$, where r is the maximal rank of a letter in the alphabet A. An *alternating cost automaton* is of the form $\mathcal{A} = (Q, A, In, \delta, pri)$, where

- Q is a finite set of *states*,
- A is the alphabet,
- $In \subseteq Q$ is the set of initial states,
- $\delta : Q \times A \to B^+(\Gamma \times Q)$ is the transition function, mapping $Q \times A$ to positive boolean combinations over $\Gamma \times Q$, such that $\delta(q, a)$ is an element of $B^+(\{1, \ldots, |a|\} \times Q)$. If $|a| = 0$, then this set contains only the formulas *true* and *false*.
- $pri : Q \to D$ is a *priority function*.

A cost automaton is called *purely non-deterministic* if the formulas in the transition function only use disjunctions for symbols of arity at least 1.

Often it is convenient to view the transition function of an alternating automaton as a mapping $\delta : Q \times A \to 2^{2^{\Gamma \times Q}}$, where $\delta(q, a) = \{P_1, \ldots, P_n\}$ corresponds to the following formula in DNF: $\bigvee_{i=1}^{n} \bigwedge_{(h,p) \in P_i} (h, p)$. Whenever we write $P \in \delta(q, a)$, then we refer to this representation of automata.

The semantics of a cost automaton is defined by means of a cost game $\mathfrak{G}_{\mathcal{A},t}$ that we describe in the following.

- The vertices or positions of the game are $V = V_E \cup V_A$ with

  $V_E = (Q \times dom(t)) \cup \{v_0\}$ and
  $V_A = \{(u, P) \ : \ u$ is an inner node of t and $P \in \delta(q, t(u))$ for some $q \in Q\}$.

- The initial position is v_0.
- The edges are defined as follows:
  - From v_0 there are edges to (q, ε) for all $q \in In$.
  - From a position (q, u) where u is not a leaf, Eva can move to all positions (u, P) with $P \in \delta(q, t(u))$, i.e., she chooses one of the sets specified by the transition function for state q at node u. (At positions (q, u) where u is a leaf the game stops.)
  - From position (u, P) Adam can move to all positions (p, uh) with $(h, p) \in P$, i.e., Adam chooses one pair of direction and state from P and then moves correspondingly in the tree.
- The priority function is defined by extending pri of $\mathcal{A}$ to the vertices of the game by setting $pri(q, u) = pri(q)$ and $pri(u, P)$ to be some reset smaller than all other elements of D (the vertices of the form (u, P) do not have any influence on the cost of the play).
- The set F contains all vertices (q, u) such that u is a leaf and $\delta(q, t(u)) = false$.

The cost of t is defined as the value of the game:

$$\mathcal{A}(t) = \min_{f_E} \max_{f_A} val(\sigma(f_E, f_A)) \left[= \max_{f_A} \min_{f_E} val(\sigma(f_E, f_A)) \right]$$

where f_E and f_A range over strategies for Eva and Adam. One can note that for the case of non-deterministic automata this value is the same as the one defined using runs. Strategies of Eva correspond to runs and strategies of Adam select a path through the run. As before we extend the definition to languages of trees: $\mathcal{A}(K) = \sup\{\mathcal{A}(t) \ : \ t \in K\}$.

We now come to the reduction from alternating to purely non-deterministic automata, which is based on the following idea: An alternating automaton $\mathcal{A}$ is uniformly universal if there exists an N such that for each t Eva has a winning strategy in $(\mathfrak{G}_{\mathcal{A},t}, val, N)$. If the game is won by Eva this means that Adam does not have a winning strategy. The purely non-deterministic automaton $\mathcal{B}$ that we construct works over trees that are annotated with strategies for Adam. The aim is to check that the strategy of Adam fails, which can be done in a purely non-deterministic way. If $\mathcal{A}$ is uniformly universal, then all strategies of Adam fail and hence $\mathcal{B}$ is also uniformly universal.

Lemma 10. *For each alternating cost automaton $\mathcal{A}$ one can construct a purely non-deterministic cost automaton $\mathcal{B}$ such that $\mathcal{A}$ is uniformly universal iff $\mathcal{B}$ is uniformly universal.*

4.3 Uniform Universality of Purely Non-deterministic Tree Automata

What remains to be shown is the following lemma:

Lemma 11. *It is possible, given a purely non-deterministic cost tree automaton, to decide whether it is uniformly universal or not.*

The reduction done so far, to purely non-deterministic automata, has led us to an almost word-theoretic problem. The proof of Lemma 11 relies on word-related considerations. In particular the proof heavily relies on the theory of semigroups, in a way similar to the proof of Kirsten [8]. The principal difficulty is to make the proof of Kirsten compatible with the tree nature of the problem. This is done using ideas originating from [2]. The proof itself is long and technical.

5 Conclusion

We have shown that the problems of nesting-depth of the disjunctive μ-calculus (with and without substitution) for regular tree languages are decidable. The proof uses cost tree automata, a tree version of the model of nested distance desert automata used by Kirsten in [8] for deciding the star-height of regular word languages. The main new contributions are the notion of subset automata, an abstract description of tree automata that allow the reduction of the star-height or nesting-depth problem to the limitedness of cost automata, the reduction of the uniform universality problem for alternating cost automata to the same problem for purely non-deterministic automata, and the adaption of the algebraic methods from [8] to the tree setting.

Possible future work includes the study of other complexity measures for tree languages. The most natural one is the nesting-depth for μ-calculus (that is for formulas allowing furthermore the intersection). This problem is open in the word case, and is referred to as the semi-restricted star-height in the framework of word languages (the restricted star-height being the one posed by Eggan and studied by Hashigushi, Kirsten, and in this work, and the generalized star-height corresponding to regular expressions with complementation, for which we do not even know if the hierarchy is strict). Another complexity measure concerns the number of distinct variables used in μ-formulas or regular expressions, and more precisely the number of variables used in fix-points (as opposed to variables used for substitutions). For those different problems, reduction to limitedness questions seems the natural path to follow.

Furthermore, we hope to be able to adapt the game-theoretic framework presented in Section 4 also to the setting of infinite trees (the remaining of the proof of limitedness being easy to adapt to this framework). This would be a major step for solving the problem of parity rank for regular languages of infinite trees [3], i.e., the problem of finding the minimal number of priorities required for a parity automaton that accepts a given regular language of infinite trees (a parameter also known as Mostowski index and tightly connected to the Rabin index).

References

1. Bojanczyk, M., Colcombet, T.: Bounds in ω-regularity. In: Proceedings of LICS 2006, pp. 285–296. IEEE Computer Society Press, Los Alamitos (2006)
2. Colcombet, T.: A combinatorial theorem for trees. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 901–912. Springer, Heidelberg (2007)
3. Colcombet, T., Löding, C.: The non-deterministic Mostowski hierarchy and distance-parity automata. In: Proceedings of ICALP 2008. LNCS, vol. 5126, pp. 398–409. Springer, Heidelberg (2008)
4. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Löding, C., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), `http://tata.gforge.inria.fr`
5. Doner, J.: Tree acceptors and some of their applications. Journal of Computer and System Sciences 4, 406–451 (1970)
6. Eggan, L.C.: Transition graphs and the star-height of regular events. Michigan Math. J. 10(4), 385–397 (1963)
7. Hashiguchi, K.: Algorithms for determining relative star height and star height. Inf. Comput. 78(2), 124–169 (1988)
8. Kirsten, D.: Distance desert automata and the star height problem. RAIRO 3(39), 455–509 (2005)
9. Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1), 57–81 (1968)

Upper Bounds on the Automata Size for Integer and Mixed Real and Integer Linear Arithmetic* (Extended Abstract)**

Jochen Eisinger

Albert-Ludwigs-Universität Freiburg, Germany
eisinger@informatik.uni-freiburg.de

Abstract. Automata-based decision procedures have proved to be a particularly useful tool for infinite-state model checking, where automata are used to represent sets of real and integer values. However, not all theoretical aspects of these decision procedures are completely understood. We establish triple exponential upper bounds on the automata size for $\mathrm{FO}(\mathbb{Z}, +, <)$ and $\mathrm{FO}(\mathbb{R}, Z, +, <)$. While a similar bound for Presburger Arithmetic, i.e., $\mathrm{FO}(\mathbb{Z}, +, <)$ was obtained earlier using a quantifier elimination based approach, the bound for $\mathrm{FO}(\mathbb{R}, Z, +, <)$ is novel. We define two graded back-and-forth systems, and use them to derive bounds on the automata size by establishing a connection between those systems and languages that can be described by formulas in the respective logics. With these upper bounds that match the known lower bounds, the theoretical background for automata-based decision procedures for linear arithmetics becomes more complete.

1 Introduction

Automata-theoretic methods have long been a useful mathematical tool to understand the decidability of various logics. Büchi observed, in the 1960s, that automata over finite and infinite words can be used to study arithmetical theories [5, 6]. In addition to being a theoretical tool, automata-based methods are increasingly employed as the basis for implementations of decision procedures. An important example of such a decision procedure is Presburger Arithmetic, i.e., $\mathrm{FO}(\mathbb{Z}, +, <)$, which can be decided using deterministic finite automata (DFA) [4]. The elements of the domain are represented by finite words and for a given formula, and an automaton is constructed recursively over the formula structure that accepts precisely the words that represent the integers that satisfy the formula. A similar approach works for mixed real and linear arithmetic, i.e., $\mathrm{FO}(\mathbb{R}, Z, +, <)$, which can be decided using weak deterministic Büchi automata (WDBA) [3]. To represent reals, one uses infinite words. Note that WDBAs can

* This work was supported by the Deutsche Forschungsgemeinschaft (German Research Foundation).

** Due to space limitations, proofs are omitted. Details are in the technical report [10].

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 431–445, 2008.
© Springer-Verlag Berlin Heidelberg 2008

be handled algorithmically almost as efficient as DFA, i.e., they can be efficiently complemented and minimized [19].

Although there exist efficient implementations of these automata-based decision procedures [2, 8, 18], many research questions are still only partially answered. Such questions regard an upper bound on the size of the automata for deciding $\mathrm{FO}(\mathbb{R}, Z, +, <)$. For instance, the results presented in [11], where certain systematical redundancies in WDBAs are exploited to decrease the automata sizes, suggest that the automata-based decision procedure for $\mathrm{FO}(\mathbb{R}, Z, +, <)$ might not be optimal. A naive analysis of the size of the automata yields one exponent for each nested negation. In contrast, Fischer and Rabin established in [13] a lower bound for any decision procedure for $\mathrm{FO}(\mathbb{R}, Z, +, <)$ [13], namely double exponential in non-deterministic time.

In this paper, we establish two tight upper bounds, first on the size of a minimal DFA for deciding $\mathrm{FO}(\mathbb{Z}, +, <)$, and second on the size of a minimal WDBA for deciding $\mathrm{FO}(\mathbb{R}, Z, +, <)$. We employ a method recently developed by Klaedtke to establish an upper bound on the automata size for $\mathrm{FO}(\mathbb{R}, +, <)$ [16]. Roughly speaking, the states of a minimal automaton for a formula are related to equivalence classes of an appropriately chosen refinement of the equivalence relation defined by Ehrenfeucht-Fraissé games. Such refinements are called *graded back-and-forth systems* or sometimes Ehrenfeucht-Fraissé relations [12, 14]. It is often easier to reason about such a refinement, and to show an upper bound on the index of these relations and therefore on the index of the equivalence relation defined by Ehrenfeucht-Fraissé games.

We first define two graded back-and-forth systems for $\mathrm{FO}(\mathbb{Z}, +, <)$ and $\mathrm{FO}(\mathbb{R}, Z, +, <)$, which characterize sets of integers and reals, respectively, that can be expressed using integer and mixed real and integer linear arithmetic. These systems are parameterized by the dimension of the sets and the quantifier rank of the formulas. Upper bounds on the indices of the equivalence relations of the graded back-and-forth systems are proved. In the second part of this paper, we then show that these graded back-and-forth systems can be used to refine the Nerode congruence relation on words. By this, we yield upper bounds on the size of the minimal DFA for deciding $\mathrm{FO}(\mathbb{Z}, +, <)$ and the minimal WDBA for deciding $\mathrm{FO}(\mathbb{R}, Z, +, <)$. The triple exponential upper bounds given here match the known lower bounds for [13, 15], and thus make the theoretical picture for those automata-based decision procedures more complete.

Related to this work is the result by Klaedtke [16], where a double exponential upper bound on the size of a minimal WDBA for deciding $\mathrm{FO}(\mathbb{R}, +, <)$ is established. We use this result and combine it with our result for $\mathrm{FO}(\mathbb{Z}, +, <)$ to establish an upper bound for $\mathrm{FO}(\mathbb{R}, Z, +, <)$. From an implementation point of view, the difference between an automata-based decision procedure for $\mathrm{FO}(\mathbb{R}, +, <)$ and $\mathrm{FO}(\mathbb{R}, Z, +, <)$ is minimal, since no algorithms tailored for $\mathrm{FO}(\mathbb{R}, +, <)$ are known. Therefore, existing implementations only support the more general theory $\mathrm{FO}(\mathbb{R}, Z, +, <)$ [2, 18]. Also related to this is the work by Klaedtke [15], where a triple exponential bound on the size of a minimal DFA for deciding $\mathrm{FO}(\mathbb{Z}, +, <)$ is established. In contrast to our approach, this bound is established by quantifier

elimination. The quantifier elimination approach relies on a specific quantifier elimination procedure, and several upper bounds for quantifier-free formulas. Note that $\mathrm{FO}(\mathbb{Z}, +, <)$ does not admit quantifier elimination directly but needs to be augmented with divisibility predicates and constant symbols. On the other hand, our approach is more direct and independent of how the DFA is actually constructed. While the resulting bound is identical in both cases, our approach allows for a straight-forward adoption of the results for showing an upper bound for $\mathrm{FO}(\mathbb{R}, Z, +, <)$.

The rest of the paper is structured as follows. In section 2, we recall preliminaries and basic definitions. In section 3, we define equivalence relations on tuples of reals and integers, which are then (section 4 and section 5) used to establish upper bounds on the automata size for integer linear arithmetic and mixed real and integer linear arithmetic respectively. Finally, in section 6 we summarize our results and draw conclusions.

2 Preliminaries

We assume that the reader is familiar with logic and automata theory. For the sake of completeness and to fix notation, we state basic definitions and well-known facts from these areas.

2.1 Words and Languages

Let Σ be an alphabet. We denote the set of all finite words over Σ by Σ^* and by Σ^+ the set $\Sigma^* \setminus \{\varepsilon\}$, where ε is the empty word. Σ^ω is the set of all ω-words over Σ. The *concatenation* of words is written as juxtaposition. We write $|u|$ for the *length* of $u \in \Sigma^*$. We often write a word $u \in \Sigma^*$ of length $\ell \geq 0$ as $u(0) \dots u(\ell - 1)$ and an ω-word $\gamma \in \Sigma^\omega$ as $\gamma(0)\gamma(1)\gamma(2)\dots$, where $u(i)$ and $\gamma(i)$ denote the $(i+1)$th letter of u and γ, respectively. For $L \subseteq \Sigma^*$, we define the (*Nerode*) congruence relation $\sim_L$ on $\Sigma^* \times \Sigma^*$ as $u \sim_L v$ iff $uw \in L \Leftrightarrow vw \in L$, for all $w \in \Sigma^*$. For ω-languages, the congruence relation $\sim_L$ is defined analogously.

Additional notation. Let $r \geq 1$ and $1 \leq i \leq r$. We denote the ith component of $b \in \Sigma^r$ by $b_{\restriction i}$ and we write $u_{\restriction i}$ for the ith *track* of $u \in (\Sigma^r)^*$, i.e., $u_{\restriction i}$ is the word $v \in (\Sigma)^*$ defined as $v(j) = u_{\restriction i}(j)$ for $0 \leq j < |u|$. We use the same notation for ω-words.

2.2 First-Order Logic

We define first-order logic as usual and assume familiarity with the basic notions of signatures and first-order formulas (see e.g. [9]). In the following, we restrict ourselves to relational signatures, i.e., signatures without function symbols and constants.

We write $\varphi(x_1, \dots, x_r)$ for a formula φ with free variables from $x_1, \dots, x_r$. The *quantifier rank* of a formula φ is denoted as $\mathrm{qr}(\varphi)$. A first-order structure $\mathfrak{A}$

over a relational signature defines a domain $\mathrm{dom}(\mathfrak{A})$ and for each relation symbol of the signature with arity r, $\mathfrak{A}$ defines a relation over $\mathrm{dom}(\mathfrak{A})^r$. We use $\mathfrak{Z}$, $\mathfrak{R}$, and $\mathfrak{M}$ to denote the structures $(\mathbb{Z}, +, <)$, $(\mathbb{R}, +, <)$, and $(\mathbb{R}, Z, +, <)$, where $+$ is the ternary addition relation, $<$ is the binary order predicate over the integers and the reals respectively, and Z is an unary predicate such that $Z(x)$ is true iff x is an integer.

For $a_1, \ldots, a_r \in \mathrm{dom}(\mathfrak{A})$ and a formula $\varphi(x_1, \ldots, x_r)$, we write $\mathfrak{A} \models \varphi[a_1, \ldots, a_r]$ if φ is satisfied in $\mathfrak{A}$ with x_i interpreted as a_i for all $1 \leq i \leq r$. We often write $\bar{x}$ and $\bar{a}$ for $x_1, \ldots, x_r$ and $a_1, \ldots, a_r$ when r is obvious from the context.

For $m, r \in \mathbb{N}$ and $\bar{a}, \bar{b} \in \mathrm{dom}(\mathfrak{A})^r$ we write $\bar{a} \equiv_m^r \bar{b}$ iff $\mathfrak{A} \models \varphi[\bar{a}] \Leftrightarrow \mathfrak{A} \models \varphi[\bar{b}]$ for all formulas $\varphi(x_1, \ldots, x_r)$ with $\mathrm{qr}(\varphi) \leq m$. Note that $\equiv_m^r$ partitions $\mathrm{dom}(\mathfrak{A})^r$ such that the elements of an equivalence class cannot be distinguished by any formula of quantifier rank less than or equal to m. The equivalence classes of $\equiv_m^r$ can be game-theoretically characterized using so-called Ehrenfeucht-Fraissé games (see e.g. [14]). Since it is often difficult to reason about $\equiv_m^r$ directly, we will define relations that refine $\equiv_m^r$, and which are easier to reason about.

2.3 Representing Sets of Reals and Integers

In the remainder, let $\varrho \in \mathbb{N}$ with $\varrho > 1$ and $\Sigma = \{0, \ldots, \varrho - 1\}$ be fixed. ϱ is called the *base*.

We will use the following well-known mapping from words and ω-words to reals [4], to define languages corresponding to sets of integers and reals. Note that this encoding of reals as words is based on the ϱ's complement, most significant bit first representation. In this representation, the first letter of a word determines the sign of the value. Also, the first letter can be repeated arbitrarily often, so that we can assume that for a vector of values, each track of the corresponding word has the same length, even if a single component could be encoded with less letters. The symbol $\star$ plays the role of a decimal point, separating the integer part from the fractional part. In the following, let $r \in \mathbb{N}$ with $r \geq 1$.

1. V_r denotes the set of all ω-words over the alphabet $\Sigma^r \cup \{\star\}$ of the form $u \star \gamma$, where $u \in (\Sigma^r)^+$ and $\gamma \in (\Sigma^r)^\omega$. The word u is called the integer part and γ the fractional part.
2. An ω-word $u \star \gamma \in \mathsf{V}_r$ represents the vector of reals with r components

$$\langle u \star \gamma \rangle = \sum_{0<i<|u|} \varrho^{|u|-i-1} \cdot u(i) + \sum_{i \geq 0} \varrho^{-i-1} \cdot \gamma(i) + \begin{cases} \bar{0} & \text{if } u(0) = \bar{0}, \\ -\varrho^{|u|-1} & \text{otherwise,} \end{cases}$$

   where vector addition and scalar multiplication are componentwise.[1] For $u \in (\Sigma^r)^+$ and $u' \in (\Sigma^r)^*$, we define $\langle u \rangle = \langle u \star \bar{0}^\omega \rangle$ and $\langle u \star u' \rangle = \langle u \star u' \bar{0}^\omega \rangle$.
3. For a formula $\varphi(x_1, \ldots, x_r)$, we define $L(\varphi) = \{\gamma \in \mathsf{V}_r : \mathfrak{M} \models \varphi[\langle \gamma \rangle]\}$, and $L^*(\varphi) = \{u \in (\Sigma^r)^+ : \mathfrak{Z} \models \varphi[\langle u \rangle]\}$.

[1] Note that we do not distinguish between vectors and tuples.

Additional notation. For $a \in \mathbb{R}$, $\lfloor a \rfloor$ denotes the largest integer that is less than or equal to a, $\lceil a \rceil$ denotes the smallest integer that is greater than or equal to a, and $\{a\}$ denotes the *fractional part* of a, i.e., $a - \lfloor a \rfloor$. We use the same notation for tuples of values where $\lfloor \cdot \rfloor$, $\lceil \cdot \rceil$, and $\{\cdot\}$ are applied componentwise.

3 Characterization of Sets Definable in Linear Arithmetic

In this section, we introduce three families of relations, which refine $\equiv_m^r$ on the structures $\mathfrak{Z}$, $\mathfrak{R}$, and $\mathfrak{M}$ respectively. Such relations are commonly known as *graded back-and-forth systems* or *Ehrenfeucht-Fraissé relations* [12, 14]. For $\mathfrak{Z}$, we define a family of relations which refine the relations used by Ferrante and Rackoff in [12], and for $\mathfrak{R}$, we will use the relations defined by Klaedtke in [16]. The latter is a refinement of the relations given by Kozen in [17]. With the refinements of $\equiv_m^r$ on the structure $\mathfrak{Z}$ and $\mathfrak{R}$, we can then define a family of relations on $\mathfrak{M}$.

In all three cases, the definitions of the relations closely resemble quantifier elimination methods for the respective structures [7, 21, 22].

3.1 Integer Linear Arithmetic

We will define relations that equate tuples of integers that cannot be distinguished by linear functions with bounded coefficients. First, we define sets of possible coefficients. The definition of the coefficients is technical, but it is required to establish a tight upper bound on the indices of the relations. Roughly speaking, we allow only double exponential many integer values out of the set of integer values whose absolute value is triple-exponentially bounded in m. For an understanding of this paper, it is sufficient to consider the sets B_m defined in the following as the set of integer values whose absolute value is triply exponentially bounded in m. In this section, all formulas are over the structure $\mathfrak{Z}$.

For $m \in \mathbb{N}$, we inductively define the sets B_m as $B_0 = \{-2, -1, 0, 1, 2\}$, $B'_m = \{\delta v / v' : \delta = \mathrm{lcm}(B_m); v, v' \in B_m; v' \neq 0\}$, and $B_{m+1} = \{v + v' : v, v' \in B'_m\}$, where $\mathrm{lcm}(A)$ denotes the least positive common multiple of all non-zero members of $A \subseteq \mathbb{Z}$.

Next, we define sets of linear functions, where the coefficients are taken from the sets B_m.

For $r, m \in \mathbb{N}$, let B_m^r be the set of functions of the form $f(\bar{x}) = c_0 + \sum_{i=1}^r c_i x_i$, where $c_1, \ldots, c_r \in B_m$, $c_0 \in \mathbb{Z}$, and $|c_0| \leq (r+1)(\mathrm{lcm}(B_m))^2$. For a function $f \in B_m^r$ with $f(\bar{x}) = c_0 + \sum_{i=1}^r c_i x_i$, we use $f^*(\bar{x})$ to denote the function $f(\bar{x}) - c_0$. Note that also $f^* \in B_m^r$.

The difference between our definition and the definition given by Ferrante and Rackoff can be pin-pointed to the set of functions B_m^r, where Ferrante and Rackoff define a smaller bound for the constant element c_0 of a function, namely $|c_0| \leq (\mathrm{lcm}(B_m))^2$. Consequently, the family of relations defined here refines the family of relations given by Ferrante and Rackoff. We will need this refinement for Lemma 14 where we establish a connection between this family of relations and languages. This lemma would not hold when using the original definition.

Definition 1. *For $\bar{a}, \bar{b} \in \mathbb{Z}^r$ we define the equivalence relation E^r_m as*

$\bar{a} E^r_m \bar{b}$ *iff*

(1) $f(\bar{a}) \geq 0 \Leftrightarrow f(\bar{b}) \geq 0$ *for every function* $f \in B^r_m$, *and*

(2) $a_i = b_i \pmod{(\mathrm{lcm}(B_m))^2}$, *for all* $1 \leq i \leq r$.

Observe that E^r_{m+1} refines E^r_m. We will use this fact in the following without explicitely referencing it.

The next three lemmas state properties of the relations E^r_m. The first two lemmas state that E^r_m refines $\equiv^r_m$ on $\mathfrak{Z}$, while the third lemma establishes an upper bound on the index of E^r_m. Although our definition deviates from the one given by Ferrante and Rackoff, we omit the proof details due to space limitations and refer the reader to our technical report [10].

Lemma 2. *For $\bar{a}, \bar{b} \in \mathbb{Z}^r$ with $\bar{a} E^r_{m+1} \bar{b}$, it holds that for all $a_{r+1} \in \mathbb{Z}$ there is a $b_{r+1} \in \mathbb{Z}$ such that $(\bar{a}, a_{r+1}) E^{r+1}_m (\bar{b}, b_{r+1})$.*

Lemma 3. *For $m, r \in \mathbb{N}$, it holds that $\bar{a} E^r_m \bar{b}$ implies $\bar{a} \equiv^r_m \bar{b}$.*

Lemma 4. *There is a constant $c \in \mathbb{N}$ such that the index of E^r_m is bounded by $2^{2^{2^{c(m+r)}}}$.*

3.2 Real Linear Arithmetic

In this section, all formulas are over the structure $\mathfrak{R}$. Similar to the previous section, we will now introduce a family of relations F^r_m over the reals, as used by Klaedtke in [16]. Since divisibility cannot be expressed using first-order formulas over $\mathfrak{R}$, the relations F^r_m are only defined over the sign of certain functions with bounded coefficients.

For $r, m \in \mathbb{N}$, let C^r_m be the set of functions of the form $f(\bar{x}) = c_0 + \sum_{i=1}^r c_i x_i$, where $c_0, \ldots, c_r \in \mathbb{Z}$, $|c_0| \leq rm$, and $|c_i| \leq m$ for all $1 \leq i \leq r$.

Definition 5. *For $\bar{a}, \bar{b} \in \mathbb{R}^r$, we define the equivalence relation F^r_m as $\bar{a} F^r_m \bar{b}$ iff $f(\bar{a}) \geq 0 \Leftrightarrow f(\bar{b}) \geq 0$, for all $f \in C^r_m$.*

Next, we state properties of the relations F^r_m, which we will use later in our proofs. Due to space limitations we refer the reader to [16] for the proof details.

Lemma 6. *For $\bar{a}, \bar{b} \in \mathbb{R}^r$ with $\bar{a} F^r_{4m^2} \bar{b}$, it holds that for all $a_{r+1} \in \mathbb{R}$, there is a $b_{r+1} \in \mathbb{R}$ such that $(\bar{a}, a_{r+1}) F^{r+1}_m (\bar{b}, b_{r+1})$.*

Lemma 7. *For all $\bar{a}, \bar{b} \in \mathbb{R}^r$ it holds that $\bar{a} F^r_{2^{3 \cdot 2^m - 2}} \bar{b}$ implies $\bar{a} \equiv^r_m \bar{b}$.*

Lemma 8. *There is a constant $c \in \mathbb{N}$ such that the index of $F^r_{2^{2^m}}$ is bounded by $2^{2^{c(m+r)}}$.*

3.3 Mixed Real and Integer Linear Arithmetic

Using the relations defined in the previous two sections, we now define a family of relations Gm, n^r that refine $\equiv_m^r$ on $\mathfrak{M}$. Roughly speaking, $G_{m,n}^r$ relates two tuples of real values, if their integer parts are in the same equivalence class of E_m^r, and their fractional part are in the same equivalence class of F_n^r. Note that both $\mathfrak{Z}$ and $\mathfrak{R}$ can be interpreted in $\mathfrak{M}$. In this section, all formulas will be over the structure $\mathfrak{M}$.

Definition 9. *For $\bar{a}, \bar{b} \in \mathbb{R}^r$ we define the equivalence relation $G_{m,n}^r$ as*

$$\bar{a} G_{m,n}^r \bar{b} \text{ iff } \lfloor \bar{a} \rfloor E_m^r \lfloor \bar{b} \rfloor \text{ and } \{\bar{a}\} F_n^r \{\bar{b}\}.$$

The next lemmas state that this family of relations refines $\equiv_m^r$ on the structure $\mathfrak{M}$, and establish an upper bound on the index of $G_{m,n}^r$. We will use a standard technique from the field of model theory. First, we show that $G_{m,n}^r$ has the back-and-forth property.

Lemma 10. *For $\bar{a}, \bar{b} \in \mathbb{R}^r$ with $\bar{a} G_{m+1,4n^2}^r \bar{b}$, it holds that for all $a_{r+1} \in \mathbb{R}$, there is a $b_{r+1} \in \mathbb{R}$ with $(\bar{a}, a_{r+1}) G_{m,n}^{r+1} (\bar{b}, b_{r+1})$.*

Proof. Given $\bar{a}$, $\bar{b}$, and a_{r+1} with $\bar{a} G_{m+1,4n^2}^r \bar{b}$. Because $\bar{a} G_{m+1,4n^2}^r \bar{b}$ implies $\lfloor \bar{a} \rfloor E_{m+1}^r \lfloor \bar{b} \rfloor$, Lemma 2 states that for $\lfloor a_{r+1} \rfloor$ there is a $b'_{r+1} \in \mathbb{Z}$ such that $\lfloor \bar{a}, a_{r+1} \rfloor E_m^{r+1} \lfloor \bar{b}, b'_{r+1} \rfloor$ holds. Similarly, $\bar{a} G_{m+1,4n^2}^r \bar{b}$ implies $\{\bar{a}\} F_{4n^2}^r \{\bar{b}\}$, so from Lemma 6 it follows that for $\{a_{r+1}\}$, there is a $b''_{r+1} \in \mathbb{R}$ such that $\{\bar{a}, a_{r+1}\} F_n^{r+1} \{\bar{b}, b''_{r+1}\}$ holds.

Choose $b_{r+1} = b'_{r+1} + b''_{r+1}$, then for $(\bar{a}, a_{r+1}) G_{m,n}^{r+1} (\bar{b}, b_{r+1})$ it remains to show that $0 \leq \{a_{r+1}\} < 1$ implies $0 \leq b''_{r+1} < 1$. Note that the functions x_{r+1} and $-x_{r+1} + 1$ from C_n^{r+1} are both positive for $\{\bar{a}, a_{r+1}\}$, and so $\{\bar{a}, a_{r+1}\} F_n^{r+1} \{\bar{b}, b''_{r+1}\}$ implies $0 \leq b''_{r+1} < 1$. □

Next, we state that $G_{m,n}^r$ refines $\equiv_m^r$. by an inductive argument, using the back-and-forth property for the induction step.

Lemma 11. *For $m, r \in \mathbb{N}$, it holds that $G_{m,2^{3 \cdot 2^m - 2}}^r$ refines $\equiv_m^r$.*

Proof. Given $\bar{a}, \bar{b} \in \mathbb{R}^r$ with $\bar{a} G_{m,2^{3 \cdot 2^m - 2}}^r \bar{b}$. We prove the claim by induction over $m \in \mathbb{N}$. For $m = 0$, it suffices to show that $\bar{a}$ and $\bar{b}$ satisfy the same atomic formulas, namely $x + y = z$, $x = y$, $x < z$, and $Z(x)$. Observe that $\bar{a} G_{0,2^{3 \cdot 2^0 - 2}}^r \bar{b}$ implies both $\lfloor \bar{a} \rfloor E_0^r \lfloor \bar{b} \rfloor$ and $\{\bar{a}\} F_{2^{3 \cdot 2^0 - 2}}^r \{\bar{b}\}$ which refine $\equiv_0^r$ on $\mathfrak{Z}$ and $\mathfrak{R}$ respectively. It is easy to see that $a_i + a_j = a_k$ iff $b_i + b_j = b_k$ for all $1 \leq i, j, k \leq r$, and similar for $x = y$ and $x < y$. For $Z(x)$, observe that $\{a_i\} = 0$ iff $\{b_i\} = 0$ for all $1 \leq i \leq r$, otherwise there is a function $f \in C_{2^{2^{cm}}}^r$ with $f(\bar{x}) = -x_i$ and $f(\{\bar{a}\}) = 0$ and $f(\{\bar{b}\}) < 0$ contradicting $\{\bar{a}\} F_{2^{3 \cdot 2^0 - 2}}^r \{\bar{b}\}$. Therefore, $Z(a_i)$ iff $Z(b_i)$ holds for all $1 \leq i \leq r$.

Now assume that the claim is true for some $m \geq 0$. We have to show that $G_{m+1,2^{3 \cdot 2^{m+1} - 2}}^r$ refines $\equiv_{m+1}^r$. Observe that all formulas φ with $\mathrm{qr}(\varphi) = m+1$ are

equivalent to a Boolean combination of formulas of the form $\exists x\psi$ with $\mathrm{qr}(\psi) \leq m$. So it suffices to show that $\mathfrak{M} \models \exists x\psi[\bar{a}]$ iff $\mathfrak{M} \models \exists x\psi[\bar{b}]$, where ψ is a formula with $\mathrm{qr}(\psi) \leq m$. For reasons of symmetry, it is enough to show one direction. Assume $\mathfrak{M} \models \exists x\psi[\bar{a}]$, then there is a $a_{r+1} \in \mathbb{R}$ such that $\mathfrak{M} \models \psi[\bar{a}, a_{r+1}]$.

Assume that $\bar{a} G^r_{m+1,2^{3\cdot 2^{m+1}-2}} \bar{b}$, and observe that $4(2^{3\cdot 2^m-2})^2 = 2^{3\cdot 2^{m+1}-2}$. We can conclude from Lemma 10 that there is a $b_{r+1} \in \mathbb{R}$ such that $(\bar{a}, a_{r+1}) G^{r+1}_{m,2^{3\cdot 2^m-2}} (\bar{b}, b_{r+1})$. By the induction hypothesis, it follows that $\mathfrak{M} \models \psi[\bar{b}, b_{r+1}]$, and therefore $\mathfrak{M} \models \exists x\psi[\bar{b}]$. □

Finally, we establish an upper bound on the index of $G^r_{m,n}$.

Lemma 12. *There is a constant $c \in \mathbb{N}$ such that the index of $G^r_{m,2^{2^m}}$ is bounded by $2^{2^{2^{c(m+r)}}}$.*

Proof. The index of $G^r_{m,2^{2^m}}$ is bounded by the product of the upper bounds on the index of E^r_m and $F^r_{2^{2^m}}$, so with Lemmas 4 and 8, there is a constant c such that the claimed bound holds. □

4 Integer Linear Arithmetic

In this section, we establish a connection between the relations E^r_m defined in Definition 1 and the Nerode relation $\sim_{L^*_\varphi}$ for first-order formulas φ over the structure $\mathfrak{Z}$. We achieve this by showing that E^r_m has certain congruence properties with regard to word concatenation. This property is the heart of our proof for an upper bound to the index of the Nerode relation of a given language, and under the assumption that Lemma 14 has been shown, it is easy to see that the following theorem holds.

Theorem 13. *Let $\varphi(x_1, \ldots, x_r)$ be a formula with $\mathrm{qr}(\varphi) \leq m$. There is a constant $c \in \mathbb{N}$ such that the index of $\sim_{L^*_\varphi}$ is at most $2^{2^{2^{cn}}}$, where n is the size of φ, i.e., the number of symbols in φ.*

This theorem follows immediately from the definition of the Nerode relation, the upper bound on the index of E^r_m in Lemma 4, and the following lemma.

Lemma 14. *For $u, v \in (\Sigma^r)^+$, if $\langle u\rangle E^r_m \langle v\rangle$, then $\langle uw\rangle E^r_m \langle vw\rangle$ for all $w \in (\Sigma^r)^*$.*

Proof. Recall that for a linear function f, the linear function f^* is defined as $f^*(\bar{x}) = f(\bar{x}) - f(\bar{0})$. We will use the two facts that for $u \in (\Sigma^r)^+$, $w \in (\Sigma^r)^*$, and for any integer linear function f, in particular for $f \in B^r_m$, the following holds:

$$\langle uw\rangle = \langle u\rangle \varrho^{|w|} + \langle \bar{0} w\rangle \text{ and}$$
$$f(\langle uw\rangle) = f(\bar{0}) + f^*(\langle u\rangle)\varrho^{|w|} + f^*(\langle \bar{0} w\rangle).$$

Assume that the claim is false, i.e., let $u, v \in (\Sigma^r)^+$ with $\langle u \rangle E_m^r \langle v \rangle$, and assume that there is a word $w \in (\Sigma^r)^*$ such that $\langle uw \rangle E_m^r \langle vw \rangle$ does not hold. Let $\delta = \mathrm{lcm}(B_m)$. Obviously, $\langle uw_i \rangle = \langle vw_i \rangle \pmod{\delta^2}$, so we can conclude that there exists a linear function $f \in B_m^r$ and either (1) $f(\langle uw \rangle) \geq 0$ and $f(\langle vw \rangle) < 0$ or (2) $f(\langle uw \rangle) < 0$ and $f(\langle vw \rangle) \geq 0$. Because (2) can be reduced to (1) with a function $g(\bar{x}) = -f(\bar{x})$, we will restrict ourselves to (1).

We will argue that $f^*(\langle u \rangle)\varrho^{|w|}$ and $f^*(\langle v \rangle)\varrho^{|w|}$ are large enough such that the addition of $f^*(\langle w \rangle)$ is negligible. Hence, if the sign of $f(\langle uw \rangle)$ and $f(\langle vw \rangle)$ is different, already the sign of $f(\langle u \rangle)$ and $f(\langle v \rangle)$ has to be different. But this is a contradiction to $\langle u \rangle E_m^r \langle v \rangle$.

We continue with the proof by establishing bounds on $f^*(\langle u \rangle)$, $f^*(\langle v \rangle)$ and $f^*(\langle w \rangle)$. Because the words u and v are in the same equivalence class of E_m^r, and $f^*(\langle u \rangle) \neq f^*(\langle v \rangle)$ holds, we can conclude that $|f^*(\langle u \rangle)|$ and $|f^*(\langle v \rangle)|$ are large. Indeed, we can assume that $|f^*(\langle u \rangle)|, |f^*(\langle v \rangle)| \geq (r+1)\delta^2$. If $|f^*(\langle u \rangle)| < (r+1)\delta^2$ or $|f^*(\langle v \rangle)| < (r+1)\delta^2$, then there is a function $g \in B_m^r$ that just shifts $f^*(x)$ by a value smaller than or equal to $(r+1)\delta^2$, such that $g(\langle u \rangle) \geq 0$ and $g(\langle v \rangle) < 0$. But this is a contradiction to $\langle u \rangle E_m^r \langle v \rangle$.

We can also find an upper bound for $|f^*(\langle \bar{0}w \rangle)|$, which is a sum of r products between a value from B_m, which is surely less or equal to δ^2, and a value which can be written in base ϱ with $|w|$ letters, so $|f^*(\langle \bar{0}w \rangle)| \leq r\delta^2(\varrho^{|w|} - 1)$.

With this, we can now continue our proof. There are two possible cases:

(i) $f^*(\langle u \rangle) \leq -(r+1)\delta^2$, but then $f(\langle uw \rangle) \leq (r+1)\delta^2 - (r+1)\delta^2\varrho^{|w|} + r\delta^2(\varrho^{|w|} - 1) < 0$, contradicting our assumption that $f(\langle uw \rangle) \geq 0$, and
(ii) $f^*(\langle u \rangle) \geq (r+1)\delta^2$, from which follows that $f^*(\langle v \rangle) \geq (r+1)\delta^2$ and we get a similar contradiction to $f(\langle vw \rangle) < 0$.

We have to conclude that such a linear function does not exist but $\langle uw \rangle E_m^r \langle vw \rangle$ is true for all $w \in (\Sigma^r)^*$. □

It becomes clear now, why we needed the finer graded back-and-forth system E_m^r. When using the original definition from Ferrante and Rackoff, we can only conclude that $|f^*(\langle u \rangle)|$ and $|f^*(\langle v \rangle)|$ are larger than δ^2, while the bound for $|f^*(\langle \bar{0}w \rangle)|$ also depends on r. Therefore, we defined B_m^r to include functions where the constant element c_0 is not bounded by δ^2 but by $(r+1)\delta^2$.

With this lemma at hand, it is clear that E_m^r refines the Nerode relation and hence that Theorem 13 holds. Note that for a non-trivial formula φ, the empty word is in its own equivalence class of the Nerode relation $\sim_{L_\varphi^*}$, but this does not affect our upper bound. Because for any formula φ, the language L_φ^* is regular, the equivalence classes of $\sim_{L_\varphi^*}$ determine the number of states of the minimal DFA accepting L_φ^*. Thus, Theorem 13 establishes a triple exponential bound with respect to the formula length on the number of states of the minimal DFA accepting L_φ^*.

Note that a similar result was already obtained by Klaedtke in [15] using a quantifier elimination approach. However, the structure $\mathfrak{Z}$ does not allow for quantifier elimination but needs to be augmented with divisibility predicates and constant symbols [7]. Therefore, the method presented there depends on the

quantifier elimination method and relies on several other bounds both for the generated quantifier free formulas and automata accepting the languages defined by these formulas. On the other hand, the proof presented here is more direct and does not depend on the way the automaton is actually constructed from the formula. Also, we can reuse it to establish an upper bound on the automata size for deciding mixed linear integer and real arithmetic in the next section.

Also note that the upper bound on the automata size is tight, i.e., there exists a family of formulas φ_n such that the index of $\sim_{L^*_{\varphi_n}}$ is at least triple exponential in n [15].. These formulas are derived from the proof of a lower bound for any decision procedure for FO$(\mathbb{Z}, +, <)$ by Fischer and Rabin [13].

5 Mixed Linear Arithmetic

In this section, we will establish a connection between the relations $G^r_{m,n}$ and the relation $\sim_{L_\varphi}$ for first-order formulas φ over the structure $\mathfrak{M}$. While the general structure of this section is the same as for the previous section, the proofs are more involved. This has mainly two reasons.

First, the value encoded by a finite word $u \star u'$ with $u \in (\Sigma^r)^+$ and $u' \in (\Sigma^r)^*$ does not change when appending the letter $\bar{0}$. This means, the word $u \star u'$ and the words $u \star u'\bar{0}^+$ all encode a single value which in turn is member of a single equivalence class of $G^r_{m,n}$. However, when appending an ω-word $\gamma \in (\Sigma^r)^\omega$, the ω-words $u \star u'\gamma$ and $u \star u'\bar{0}^+\gamma$ might encode different real values. Obviously, these different values might be in different equivalence classes of $G^r_{m,n}$. They potentially encode different real values, because the fractional parts u' and $u'\bar{0}^+$ are of different length. Therefore, the letters of γ are encoding for digits at different positions of the fractional parts. In general, this problem occurs for all finite words $u \star u'$ and $v \star v'$ which encode real values in the same equivalence class of $G^r_{m,n}$, but u' and v' are of different length.

Second, we cannot treat the integer and the fractional part of a value encoded by an ω-word $u \star u'\gamma$ separately by just looking at $\langle u \rangle$ and $\langle \bar{0} \star u'\gamma \rangle$, since, e.g., the ω-word $\langle 0 \star (\varrho - 1)^\omega \rangle$ encodes the integer value 1. So when examining the fractional part of the values encoded by $u \star u'$ and $v \star v'$, we have to make sure that $u'_{\restriction i} \in (\varrho - 1)^*$ iff $v'_{\restriction i} \in (\varrho - 1)^*$ for $1 \leq i \leq r$. It is then the case that if the fractional part of a track in $u \star u'\gamma$ encodes 1, the fractional part of the same track in $v \star v'\gamma$ also encodes 1, and vice versa.

5.1 Relationship to Languages

Despite the two problems mentioned above, the following lemmas establish certain properties of $G^r_{m,n}$ with regard to word concatenation.

First, we recall a result for the family of relations F^r_m on $\mathfrak{R}$ from [16].

Lemma 15. *For $u, v \in (\Sigma^r)^+$ and $u', v' \in (\Sigma^r)^*$ the following fact holds. If $\langle u \star u' \rangle F^r_{2m} \langle v \star v' \rangle$ with $|u'| \geq |v'|$, then for all $\gamma \in (\Sigma^r)^\omega$, it holds that $\langle u \star u'\gamma \rangle F^r_m \langle v \star v'\bar{0}^k\gamma \rangle$ with $k = \min(\{|u'| - |v'|\} \cup \{k \in \mathbb{Z} : \varrho^k \geq rm\})$.*

Roughly speaking, this lemma states that, if two words $u \star u'$ and $v \star v'$ encode values that are in the same equivalence class of F^r_m, but u' and v' are of different length, we can extend the shorter one with zeros such that F^r_m has a congruence property with regard to word concatenation. We only need a bounded number of zeros. Because either after appending zeros to the shorter word, both words are that long that by appending an arbitrary ω-word, the change in the values encoded is negligible. Or, both words are of the same size after padding the shorter one with zeros.

With this result, and with Lemma 14, we can now proceed to show that $G^r_{m,n}$ also has a certain congruence property with regard to word concatenation. However, as mentioned earlier, this is not a direct result, since we cannot treat the integer part and the fractional part separately.

In the following, we will use the functions $\mathsf{D}(u) = \{i \in \mathbb{N} : 1 \leq i \leq r \text{ and } u_{\restriction i} \in (\varrho - 1)^*\}$ for $u \in (\Sigma^r)^*$, and $\mathsf{D}_\omega(\gamma) = \{i \in \mathbb{N} : 1 \leq 1 \leq r \text{ and } \gamma_{\restriction i} = (\varrho - 1)^\omega\}$ for $\gamma \in (\Sigma^r)^\omega$. We use these functions to identify tracks where the fractional part potentially encodes 1 instead of a strictly smaller value.

Lemma 16 states three properties of $G^r_{m,n}$ with regard to word concatenation, namely: (1) $G^r_{m,n}$ has the same properties as E^r_m for words without a fractional part; (2) $G^r_{m,n}$ has a similar property as F^r_n, if avoiding words which might result in the fractional part encoding for an integer value; And (3) $G^r_{m,n}$ has a congruence property with regard to word concatenation, if we restrict ourselves to words where the same tracks of the fractional part potentially encode the value 1.

Lemma 16. *For $u, v \in (\Sigma^r)^+$ and $u', v' \in (\Sigma^r)^*$, the following three facts hold.*

(i) If $\langle u \rangle G^r_{m,n} \langle v \rangle$ then for all $w \in (\Sigma^r)^*$ it holds that $\langle uw \rangle G^r_{m,n} \langle vw \rangle$.
(ii) If $\langle u \star u' \rangle G^r_{m,2n} \langle v \star v' \rangle$ with $|u'| \geq |v'|$, then for all $\gamma \in (\Sigma^r)^\omega$ with $\mathsf{D}_\omega(\gamma) \cap (\mathsf{D}(u') \cup \mathsf{D}(v')) = \emptyset$, it holds that $\langle u \star u'\gamma \rangle G^r_{m,n} \langle v \star v' \bar{0}^k \gamma \rangle$ with $k = \min(\{|u'| - |v'|\} \cup \{k \in \mathbb{Z} : \varrho^k \geq rn\})$.
(iii) If $\langle u \star u' \gamma \rangle G^r_{m+1,n} \langle v \star v' \gamma \rangle$ for all $\gamma \in (\Sigma^r)^\omega$ with $\mathsf{D}_\omega(\gamma) \cap (\mathsf{D}(u') \cup \mathsf{D}(v')) = \emptyset$, and $\mathsf{D}(u') = \mathsf{D}(v')$, then $\langle u \star u'\gamma \rangle G^r_{m,n} \langle v \star v' \gamma \rangle$ holds for all $\gamma \in (\Sigma^r)^\omega$.

Proof. (i) This follows directly from Lemma 14 and Definition 9.

(ii) The restriction $\mathsf{D}_\omega(\gamma) \cap (\mathsf{D}(u') \cup \mathsf{D}(v')) = \emptyset$ enforces that by appending γ to either $u \star u'$ or $v \star v' \bar{0}^k$, the integer value of the encoded values cannot change. Therefore, this property follows from Lemma 15 and Definition 9.

(iii) Due to space limitations, we will only give a proof sketch.

Let $\gamma \in (\Sigma^r)^\omega$ with $\mathsf{D}_\omega(\gamma) \cap (\mathsf{D}(u') \cup \mathsf{D}(v')) \neq \emptyset$, i.e., certain tracks of the words $\bar{0} \star u' \gamma$ and $\bar{0} \star v' \gamma$ encode the value 1. Since $\mathsf{D}(v') = \mathsf{D}(u')$, the integer value of the values encoded by the words $u \star u'$ and $v \star v'$ change by the same amount when appending γ. Therefore, we can proceed to proof the third property separately for the integer part and for the fractional part. Recall that $\bar{a} G^r_{m,n} \bar{b}$ is defined as $\lfloor \bar{a} \rfloor E^r_m \lfloor \bar{b} \rfloor$ and $\{\bar{a}\} F^r_n \{\bar{b}\}$.

First, we consider the integer part. Since E^r_{m+1} refines E^r_m, and the change in the integer part is bounded (at most r components are changed by 1), it is easy to show that $\lfloor \langle u \star u' \gamma \rangle \rfloor E^r_m \lfloor \langle v \star v' \gamma \rangle \rfloor$ holds.

For the fractional part, we can just ignore those tracks of $\bar{0} \star u'\gamma$ (and also those of $\bar{0} \star v'\gamma$) that encode (the integer value) 1. But then we can find an ω-word $\gamma' \in (\Sigma^r)^\omega$ such that $f(\{\langle u \star u'\gamma' \rangle\}) = f(\{\langle u \star u'\gamma' \rangle\})$ for all $f \in C_n^r$ (and similar for $v \star v'$).

From these two facts, and together with the precondition of (iii), we can now conclude that $\langle u \star u'\gamma \rangle G_{m,n}^r \langle v \star v'\gamma \rangle$ holds for all $\gamma \in (\Sigma^r)^\omega$. □

5.2 Upper Bounds

Using the results from this and the previous section, we will establish an upper bound on the index of $\sim_{L_\varphi}$.

Theorem 17. *Let $\varphi(x_1, \ldots, x_r)$ be a formula with $\mathrm{qr}(\varphi) \leq m$. There is a constant $c \in \mathbb{N}$ such that the index of $\sim_{L_\varphi}$ is at most $2^{2^{2^{cn}}}$, where n is the size of φ, i.e., the number of symbols in φ.*

In contrast to the integer linear arithmetic case, $G_{m,n}^r$ does not directly refine $\sim_{L_\varphi}$. However, we can use $G_{m,n}^r$ to define a family of relations that refines $\sim_{L_\varphi}$ using the results established in the previous lemma. The general idea is as follows. We consider all words separately that encode values of a single equivalence class of $G_{m,n}^r$. Then we partition these words by their length using Lemma 16(ii). We partition these new classes again into words with similar fractional parts as in the precondition of Lemma 16(iii). Then, after refining $G_{m,n}^r$ twice, we find a partition that refines $\sim_{L_\varphi}$, and we can use the upper bound on the index of $G_{m,n}^r$ to deduce an upper bound on the index of $\sim_{L_\varphi}$.

Proof. We now define such an equivalence relation S_m^r on $(\Sigma^r)^* \times (\Sigma^r)^*$. For all $u, v \in (\Sigma^r)^+$ we define $uS_m^r v$ iff $\langle u \rangle G^r_{m+1,2^{2^{2m+2}+1}} \langle v \rangle$ (cf. Lemma 16(i)). The empty word ε is in its own equivalence class, and all words $u \in (\Sigma^r \cup \{\star\})^*$ with two or more occurrences of $\star$, or where $u(0) = \star$, i.e., words not encoding for a value, are in one equivalence class.

It remains to define the equivalence classes of S_m^r on the words $u \star u'$ with $u \in (\Sigma^r)^+$ and $u' \in (\Sigma^r)^*$. For such a word $u \star u'$ consider the set $X = \{v \star v' : \langle u \star u' \rangle G^r_{m+1,2^{2^{2m+2}+1}} \langle v \star v' \rangle\}$. Assume that $|u'| \leq |v'|$ for all $v \star v' \in X$. Then we partition X into classes Y_k (cf. Lemma 16(ii)) such that

- for $k \in \{0, \ldots, \lceil \log_\varrho r 2^{2^{2m+2}+1} \rceil - 1\}$
  let $Y_k = \{v \star v' : v \star v' \in X \wedge |u'| + k = |v'|\}$, and
- for $k = \lceil \log_\varrho r 2^{2^{2m+2}+1} \rceil$
  let $Y_k = \{v \star v' : v \star v' \in X \wedge |u'| + k \leq |v'|\}$.

S_m^r then refines this partition of X such that $u \star u' S_m^r v \star v'$ iff both $\mathsf{D}(u') = \mathsf{D}(v')$ and $u \star u', v \star v' \in Y_k$ for some k (cf. Lemma 16(iii)).

It follows immediately that for all $u, v \in (\Sigma^r)^+$ and $u', v' \in (\Sigma^r)^*$

(i) $uS_m^r v$ implies $u \sim_{L_\varphi} v$, and
(ii) $u \star u' S_m^r v \star v'$ implies $u \star u' \sim_{L_\varphi} v \star v'$.

It remains to show an upper bound on the index of S^r_m. We defined S^r_m to partition the equivalence classes of $G^r_{m+1,2^{2^{2m+2}+1}}$ into $\lceil \log_\varrho r 2^{2^{2m+2}+1} \rceil$ sets and each of these sets into 2^r classes. From Lemma 12, we know that the index of $G^r_{m+1,2^{2^{2m+2}+1}}$ is bounded by $2^{2^{2^{d(m+r)}}}$ for some constant $d \in \mathbb{N}$. Observe that $1 \leq m + r \leq n$. Hence, there is a constant $c \in \mathbb{N}$ such that the index of S^r_m is bounded by

$$2^{2^{2^{d(m+r)}}} \cdot \lceil \log_\varrho r 2^{2^{2m+2}+1} \rceil \cdot 2^r \leq 2^{2^{2^{dn}}} \cdot 2^{3+n} \cdot 2^n \leq 2^{2^{2^{cn}}}. \qquad \square$$

Note that for any formula φ, the language L_φ is in the Borel class $F_\sigma \cap G_\delta$ [3], which exactly captures the expressive power of weak deterministic Büchi automaton (WDBA) [20]. Therefore, the equivalence classes of $\sim_{L_\varphi}$ determine the number of states of the minimal WDBA accepting L_φ [19]. Thus, Theorem 17 establishes a triple exponential bound with respect to the formula length on the number of states of the minimal WDBA accepting L_φ. Note that the upper bound on the automata size is tight, i.e., there exists a family of formulas φ_n such that the index of $\sim_{L_{\varphi_n}}$ is at least triple exponential in n. This is the same family of formulas as for the structure $\mathfrak{Z}$, because $\mathfrak{Z}$ can be interpreted in $\mathfrak{M}$.

6 Conclusion

We have established a triple exponential bound on the size of the minimal deterministic finite automata for deciding $\mathrm{FO}(\mathbb{Z}, +, <)$, and on the size of the minimal weak deterministic Büchi automata for deciding $\mathrm{FO}(\mathbb{R}, Z, +, <)$. We used in both cases a most significant bit first (MSB) encoding. We defined two graded back-and-forth systems and showed that they refine the equivalence relation defined by Ehrenfeucht-Fraissé games. By establishing certain congruence properties of these systems with regard to word concatenation, we could use them to derive these bounds. The relation between the graded back-and-forth systems and the languages describing sets of values definable in linear arithmetic allows for interesting insights into the structure of the automata. For instance when interpreting $\mathrm{FO}(\mathbb{Z}, +, <)$ in $\mathrm{FO}(\mathbb{R}, Z, +, <)$, the size of the minimal WDBA for a formula is (in the number of variables) exponentially larger than the size of a minimal DFA. This overhead, which also appears in the proof of the upper bound for $\mathrm{FO}(\mathbb{R}, Z, +, <)$, is due to the ambiguous encoding of real values as ω-words.

Both theories, integer linear arithmetic as well as mixed real and integer linear arithmetic, are prominent examples for automata-based decision procedures, and are implemented in several tools used in, e.g., infinite state-space model checkers [1, 2, 3]. Our results make the theoretical background for these approaches more complete. The question, whether it is possible to construct the minimal automaton for a given formula in time polynomial to the size of the resulting minimal automaton remains open.

As future work, we plan to investigate the automata-based approach for deciding $\mathrm{FO}(\mathbb{Z}, +, <)$ using the least significant bit first (LSB) encoding. To the best of our knowledge, no tight upper bounds are known for this case. Note that

the family of relations E_m^r presented here does not refine the Nerode relation for the LSB encoding. Additionally, we plan to identify further redundancies in the encoding of sets of values as languages and try to leverage them to achieve asymptotically smaller automata, similar to the "don't care" language approach presented in [11].

Acknowledgements. The author thanks Bernd Becker, Felix Klaedtke, Moritz Müller, Stefan Wölfl, and the anonymous reviewers for their comments on earlier versions of this paper.

References

1. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Fast acceleration of symbolic transition systems. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 118–121. Springer, Heidelberg (2003)
2. Becker, B., Dax, C., Eisinger, J., Klaedtke, F.: LIRA: Handling constraints of linear arithmetics over the integers and the reals. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 307–310. Springer, Heidelberg (2007)
3. Boigelot, B., Jodogne, S., Wolper, P.: An effective decision procedure for linear arithmetic over the integers and reals. ACM Trans.Comput. Log. 6, 614–633 (2005)
4. Boigelot, B., Wolper, P.: Representing arithmetic constraints with finite automata: An overview. In: Stuckey, P.J. (ed.) ICLP 2002. LNCS, vol. 2401, pp. 1–19. Springer, Heidelberg (2002)
5. Büchi, J.: Weak second-order arithmetic and finite automata. Zeitschrift der mathematischen Logik und Grundlagen der Mathematik 6, 66–92 (1960)
6. Büchi, J.: On a decision method in restricted second order arithmetic. In: Logic, Methodology and Philosophy of Science (Proc. 1960 Internat. Congr.), pp. 1–11. Stanford University Press (1962)
7. Cooper, D.C.: Theorem proving in arithmetic without multiplication. In: Meltzer, B., Michie, D. (eds.) Proceedings of the 7th Annual Machine Intelligence Workshop, pp. 91–100. Edinburgh University Press (1972)
8. Couvreur, J.-M.: A BDD-like implementation of an automata package. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 310–311. Springer, Heidelberg (2005)
9. Ebbinghaus, H.-D., Flum, J., Thomas, W.: Mathematical Logic, 2nd edn. Springer, Heidelberg (1994)
10. Eisinger, J.: Upper bounds on the automata size for integer and mixed real and integer linear arithmetic, Tech. Report 239, Institut für Informatik, Universität Freiburg (2008)
11. Eisinger, J., Klaedtke, F.: Don't care words with an application to the automata-based approach for real addition. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 67–80. Springer, Heidelberg (2006)
12. Ferrante, J., Rackoff, C.: The Computational Complexity of Logical Theories. LNM, vol. 718. Springer, Heidelberg (1979)
13. Fischer, M.J., Rabin, M.O.: Super-exponential complexity of presburger arithmetic, tech. report, Massachusetts Institute of Technology, Cambridge, MA, USA (1974)
14. Hodges, W.: A shorter model theory. Cambridge University Press, New York (1997)

15. Klaedtke, F.: On the automata size for Presburger arithmetic. In: LICS 2004, pp. 110–119. IEEE Computer Society Press, Los Alamitos (2004)
16. Klaedtke, F.: Ehrenfeucht-Fraïssé goes automatic for real addition. In: STACS 2008. IBFI Schloss Dagstuhl, pp. 445–456 (2008)
17. Kozen, D.: Theory of Computation. Springer, New York (2006)
18. LASH, The Liège Automata-based Symbolic Handler, `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`
19. Löding, C.: Efficient minimization of deterministic weak ω-automata. Information Processing Letters 79, 105–109 (2001)
20. Maler, O., Staiger, L.: On syntactic congruences for omega-languages. Theoretical Comput. Sci. 181, 93–112 (1997)
21. Weispfenning, V.: Mixed real-integer linear quantifier elimination. In: ISSAC 1999, pp. 129–136. ACM, New York (1999)
22. Weispfenning, V., Loos, R.: Applying linear quantifier elimination. The Computer Journal 36, 450–462 (1993)

Syntactic Metatheory of Higher-Order Subtyping

Andreas Abel and Dulma Rodriguez

Department of Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{andreas.abel|dulma.rodriguez}@ifi.lmu.de

Abstract. We present a new proof of decidability of higher-order subtyping in the presence of bounded quantification. The algorithm is formulated as a judgement which operates on beta-eta-normal forms. Transitivity and closure under application are proven directly and syntactically, without the need for a model construction or reasoning on longest beta-reduction sequences. The main technical tool is hereditary substitution, i.e., substitution of one normal form into another, resolving all freshly generated redexes on the fly. Hereditary substitutions are used to keep types in normal-form during execution of the subtyping algorithm. Termination of hereditary substitutions can be proven in an elementary way, by a lexicographic induction on the kind of the substituted variable and the size of the expression substituted into—this is what enables a purely syntactic metatheory.

Keywords: Higher-order subtyping, bounded quantification, algorithmic subtyping, hereditary substitution.

1 Introduction

Higher-order subtyping with bounded quantification has been used to model aspects of object-oriented programming languages [Pie02, Ch. 32]. Decidability is non-trivial and has been studied extensively in the past. Both Compagnoni [Com95] and Pierce and Steffen [PS97] have provided an algorithm for deciding subtyping for Kernel System $F^{\omega}_{<:}$ and proven its completeness by establishing a strong normalization theorem, while Compagnoni and Goguen [CG03, CG06] have studied the more general system $\mathcal{F}^{\omega}_{\leq}$ and proved completeness by constructing a Kripke model.

The cited works are impressive, but the complexity of the proofs is a bit overwhelming when it comes to formalizing them in a theorem prover like Coq, Isabelle, or Twelf. The reason is that strong normalization theorems or models are laborious to mechanize and little is known about automating the involved proofs. However, recently there have been successes in formalizing purely syntactical developments of metatheory of programming languages, most notably SML [LCH07]. Such formalizations use only first-order inductive judgements over syntactical objects and proofs by induction over these judgements or simple arithmetical measures.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 446–460, 2008.
© Springer-Verlag Berlin Heidelberg 2008

A modern technique to treat the metatheory of systems which rely on normalization is *hereditary substitution* [WC+03]. Hereditary substitution provides an algorithm for bottom-up normalization whose termination can be proven by a simple lexicographic induction—provided the proof-theoretical strength of the language does not exceed that of the simply-typed lambda-calculus. The technique of hereditary substitution simplifies the metatheory considerably. Hereditary substitution has been successfully used for the normalization of the Concurrent Logical Framework [WC+03], the Edinburgh LF [HL07], and the type language of SML [LCH07].

In this article, we present a purely syntactic metatheory of $F^{\omega}_{<:}$ using the technique of hereditary substitution, affirming that it is flexible enough to account for bounded quantification, which is similar to lazy let-binding or singleton types. The result is a major simplification of the metatheory of $F^{\omega}_{<:}$ and a proof structure that is ready for formalization in a proof assistant even with low proof-theoretical complexity such as Twelf.

Detailed proofs for the theorems of this paper can be found in the diploma thesis of the second author [Rod07].

Contents. In Section 2, we recapitulate System $F^{\omega}_{<:}$ with kinding, equality and declarative subtyping and present a subtyping algorithm which works on η-long β-normal type constructors. In Section 3 we present algorithms for hereditary substitution and normalization and prove their correctness; this section contains the main technical work. Afterwards, in Section 4, we show soundness, completeness, and termination of the subtyping algorithm, which is in essence a consequence of the results of Section 3. Finally, we discuss related and further work in the conclusions.

Judgements. In this article, the following judgements will be defined inductively:

$\Gamma \vdash F : \kappa$	constructor F has kind κ in context Γ
$\Gamma \vdash F = F' : \kappa$	F and F' of kind κ are $\beta\eta$-equal
$\Gamma \vdash F \leq F' : \kappa$	F is a higher-order subtype of F'
$\Gamma \vdash V \Uparrow \kappa$	V is hereditarily normal of kind κ
$\Gamma \vdash_{\mathsf{a}} V \leq V'$	V is a subtype of V', algorithmically

When we write $\mathcal{D} :: J$, we mean that judgement J is established by derivation $\mathcal{D}$. Then, $|\mathcal{D}|$ denotes the height of this derivation. We consider derivations ordered by their height: $\mathcal{D}_1 \leq \mathcal{D}_2$ iff $|\mathcal{D}_1| \leq |\mathcal{D}_2|$.

2 System $F^{\omega}_{<:}$

This section introduces the syntax and the rules of kinding, equality, and subtyping of system $F^{\omega}_{<:}$, a typed λ-calculus of polymorphic functions, subtyping, and type operators.

2.1 Constructors and Kinds

System $F^{\omega}_{<:}$ consists of terms (programs), type constructors and kinds. This article is dedicated to the decidability of subtyping alone, thus, we ignore terms and typing completely. Note, however, that decidability of typing follows from decidability of subtyping (via the concept of *promotion* [Pie02, Ch. 28.1] [PS97]).

Constructors are classified by their kind, i.e., as types, functions on types, etc. Kinds are given by the grammar:

$$\kappa ::= * \mid \kappa_1 \to \kappa_2$$

Let $\boldsymbol{\kappa} \to \kappa'$ be an abbreviation for $\kappa_1 \to \kappa_2 \to \ldots \to \kappa_n \to \kappa'$ where $|\boldsymbol{\kappa}| = n$. Let $|\kappa| \in \mathbb{N}$ denote a measure on kinds with $|\kappa'| \leq |\kappa \to \kappa'|$ and $|\kappa| < |\kappa \to \kappa'|$. An example of such a measure is the *rank*, which is defined recursively as $\mathsf{rk}(\boldsymbol{\kappa} \to *) = \max\{1 + \mathsf{rk}(\kappa_i) \mid 1 \leq i \leq |\boldsymbol{\kappa}|\}$. In particular, $\mathsf{rk}(*) = 0$.

Constructors are given by the following Church-style type-level λ-calculus. The meta-variable X ranges over a countable infinite set of constructor variables.

$$A, B, F, G ::= X \mid \lambda X{:}\kappa.\, F \mid F\, G \mid A \to B \mid \forall X \leq G{:}\kappa.\, A \mid \top$$

As usual, $\lambda X : \kappa.\, F$ binds variable X in F. We identify constructors up to α-equivalence, i.e., up to renaming of bound variables. Sometimes, when we want to stress syntactic identity of constructors, we use $\equiv$ for α-equivalence. The letters U, V, W denote β-normal constructors and A, B constructors of kind $*$ (types). We define $\top_{\boldsymbol{\kappa} \to *} = \lambda \boldsymbol{Y} : \boldsymbol{\kappa}.\, \top$ (meaning $\lambda Y_1 : \kappa_1. \ldots \lambda Y_{|\boldsymbol{Y}|} : \kappa_{|\boldsymbol{\kappa}|}.\, \top$) where the lengths of $\boldsymbol{Y}$ and $\boldsymbol{\kappa}$ coincide. A vector notation is also used for application: $F\, \boldsymbol{G}$ means $F\, G_1 \ldots G_{|\boldsymbol{G}|}$, where application associates to the left as usual.

Contexts follow the grammar $\Gamma ::= \diamond \mid \Gamma, X \leq G : \kappa$. We refer to G as the *bound* of X. We assume all variables bound in Γ to be distinct and define the notation $\Gamma, X{:}\kappa$ as an abbreviation for $\Gamma, X \leq \top_{\kappa} : \kappa$.

2.2 Kinding and Well-Formed Contexts

Well-formed contexts $\Gamma \vdash$, defined mutually with the kinding judgement $\Gamma \vdash F : \kappa$ in Figure 1, are constructed from the empty context by adding well-kinded type variable declarations. Kinding is decidable and unique, since constructor variables are annotated with their kinds.

The extra assumption $\Gamma \vdash G : \kappa$ in rule (K-Var) is due to Pierce and Steffen [PS97] and simplifies the proof of Theorem 1, which entails termination of the subtyping algorithm.

We maintain the invariant that kinding statements are only derivable in well-formed contexts (see Lemma 1.1). Bounds do not matter for kinding, i.e., if a constructor F has kind κ in a context Γ , and Γ' is the same context as Γ but with different, well-kinded bounds, then F has kind κ in context Γ' as well.

$\boxed{\Gamma \vdash}$

$$\text{(C-Empty)}\ \frac{}{\diamond \vdash} \qquad \text{(C-Bound)}\ \frac{\Gamma \vdash \qquad \Gamma \vdash G : \kappa}{\Gamma, X \leq G : \kappa \vdash}$$

$\boxed{\Gamma \vdash F : \kappa}$

$$\text{(K-Var)}\ \frac{(X \leq G : \kappa) \in \Gamma \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash X : \kappa} \qquad \text{(K-Top)}\ \frac{\Gamma \vdash}{\Gamma \vdash \top : *}$$

$$\text{(K-Abs)}\ \frac{\Gamma, X : \kappa \vdash F : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F : \kappa \to \kappa'} \qquad \text{(K-App)}\ \frac{\Gamma \vdash F : \kappa \to \kappa' \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash F\, G : \kappa'}$$

$$\text{(K-Arr)}\ \frac{\Gamma \vdash A : * \qquad \Gamma \vdash B : *}{\Gamma \vdash A \to B : *} \qquad \text{(K-All)}\ \frac{\Gamma \vdash G : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A : *}{\Gamma \vdash \forall X \leq G : \kappa.\, A : *}$$

$\boxed{\Gamma \vdash F = F' : \kappa}$

$$\text{(Eq-}\beta\text{)}\ \frac{\Gamma, X : \kappa \vdash F : \kappa' \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash (\lambda X : \kappa.\, F)\, G = [G/X]F : \kappa'} \qquad \text{(Eq-}\eta\text{)}\ \frac{\Gamma \vdash F : \kappa \to \kappa' \qquad X \notin \mathsf{FV}(F)}{\Gamma \vdash \lambda X : \kappa.\, FX = F : \kappa \to \kappa'}$$

$$\text{(Eq-Var)}\ \frac{(X \leq G : \kappa) \in \Gamma \qquad \Gamma \vdash G : \kappa}{\Gamma \vdash X = X : \kappa} \qquad \text{(Eq-Abs)}\ \frac{\Gamma, X : \kappa \vdash F = F' : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F = \lambda X : \kappa.\, F' : \kappa \to \kappa'}$$

$$\text{(Eq-App)}\ \frac{\Gamma \vdash F = F' : \kappa \to \kappa' \qquad \Gamma \vdash G = G' : \kappa}{\Gamma \vdash F\, G = F'\, G' : \kappa'}$$

$$\text{(Eq-Top)}\ \frac{\Gamma \vdash}{\Gamma \vdash \top = \top : *} \qquad \text{(Eq-Arr)}\ \frac{\Gamma \vdash A = A' : * \qquad \Gamma \vdash B = B' : *}{\Gamma \vdash A \to B = A' \to B' : *}$$

$$\text{(Eq-All)}\ \frac{\Gamma \vdash G = G' : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A = A' : *}{\Gamma \vdash \forall X \leq G : \kappa . A = \forall X \leq G' : \kappa . A' : *}$$

$$\text{(Eq-Sym)}\ \frac{\Gamma \vdash F = F' : \kappa}{\Gamma \vdash F' = F : \kappa} \qquad \text{(Eq-Trans)}\ \frac{\Gamma \vdash F_1 = F_2 : \kappa \qquad \Gamma \vdash F_2 = F_3 : \kappa}{\Gamma \vdash F_1 = F_3 : \kappa}$$

$\boxed{\Gamma \vdash F \leq F' : \kappa}$

$$\text{(S-Var)}\ \frac{\Gamma \vdash G : \kappa \qquad X \leq G : \kappa \in \Gamma}{\Gamma \vdash X \leq G : \kappa} \qquad \text{(S-App)}\ \frac{\Gamma \vdash F \leq F' : \kappa \to \kappa' \qquad \Gamma \vdash H : \kappa}{\Gamma \vdash FH \leq F'H : \kappa'}$$

$$\text{(S-Top)}\ \frac{\Gamma \vdash A : *}{\Gamma \vdash A \leq \top : *} \qquad \text{(S-Abs)}\ \frac{\Gamma, X : \kappa \vdash F \leq F' : \kappa'}{\Gamma \vdash \lambda X : \kappa.\, F \leq \lambda X : \kappa.\, F' : \kappa \to \kappa'}$$

$$\text{(S-Arr)}\ \frac{\Gamma \vdash A' \leq A : * \qquad \Gamma \vdash B \leq B' : *}{\Gamma \vdash A \to B \leq A' \to B' : *}$$

$$\text{(S-All)}\ \frac{\Gamma \vdash G : \kappa \qquad \Gamma, X \leq G : \kappa \vdash A \leq A' : *}{\Gamma \vdash \forall X \leq G : \kappa . A \ \leq\ \forall X \leq G : \kappa . A' : *}$$

$$\text{(S-Eq)}\ \frac{\Gamma \vdash F = G : \kappa}{\Gamma \vdash F \leq G : \kappa} \qquad \text{(S-Trans)}\ \frac{\Gamma \vdash F \leq G : \kappa \qquad \Gamma \vdash G \leq H : \kappa}{\Gamma \vdash F \leq H : \kappa}$$

Fig. 1. Declarative presentation of $F^{\omega}_{<:}$

Lemma 1 (Admissible rules for kinding)

1. *Validity: If* $\Gamma \vdash F : \kappa$ *then* $\Gamma \vdash$.
2. *Weakening: If* $\Gamma, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, X \leq G : \kappa, \Gamma' \vdash F : \kappa'$.
3. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F : \kappa'$.

Since bounds do not matter for kinding, we do not require $\Gamma \vdash G \leq H : \kappa$ in the substitution property.

2.3 Equality

In contrast to previous presentations of System $F^{\omega}_{<:}$ [Pie02, Ch. 31], we consider constructors equivalent modulo β *and* η. Instead of an untyped equality we define an equality judgement $\Gamma \vdash F = F' : \kappa$, since this is more robust w. r. t. extensions, e. g., by polarities [Ste98, Abe06b]. The judgement is given by the axioms (Eq-β) and (Eq-η) plus congruence and equivalence rules (see Figure 1).

The equality judgement has the usual properties.

Lemma 2 (Admissible rules for equality)

1. *Reflexivity: If* $\Gamma \vdash F : \kappa$ *then* $\Gamma \vdash F = F : \kappa$.
2. *Validity: If* $\Gamma \vdash G = G' : \kappa$ *then* $\Gamma \vdash G : \kappa$ *and* $\Gamma \vdash G' : \kappa$.
3. *Weakening: If* $\Gamma, \Gamma' \vdash F = F' : \kappa$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, X \leq G : \kappa, \Gamma' \vdash F = F' : \kappa$.
4. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F = F' : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F = [G/X]F' : \kappa'$.

Proof Each by induction on the first derivation. In the proof of item 2, case (Eq-All), we use the fact that bounds do not matter for kinding. □

2.4 Subtyping

The $F^{\omega}_{<:}$ subtyping relation $\Gamma \vdash F \leq F' : \kappa$ (see Figure 1) extends the subtyping relation of system $F_{\leq}$ [CW85]. Subtyping for type operators of higher kind is defined pointwise, see (S-Abs) and (S-App).

Decidability requires using the Kernel-Fun rule (S-All). *Full* subtyping would allow a bound $G' \leq G$ on the right hand side, losing decidability [Pie92]. We do not treat antisymmetry here. Reflexivity is inherited from equality.

Lemma 3 (Admissible rules for subtyping)

1. *Validity: If* $\Gamma \vdash F \leq F' : \kappa$ *then* $\Gamma \vdash F : \kappa$ *and* $\Gamma \vdash F' : \kappa$.
2. *Weakening: If* $\Gamma, \Gamma' \vdash F \leq F' : \kappa$ *and* $\Gamma \vdash G : \kappa'$ *then* $\Gamma, X \leq G : \kappa', \Gamma' \vdash F \leq F' : \kappa$.
3. *Substitution: If* $\Gamma, X \leq H : \kappa, \Gamma' \vdash F \leq F' : \kappa'$ *and* $\Gamma \vdash G \leq H : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F \leq [G/X]F' : \kappa'$ *(not needed in the following).*

2.5 Algorithmic Subtyping

The declarative definition of subtyping contains two rules that correspond to a logical *cut*: on the level of constructors, the transitivity rule (S-Trans), if one views types as predicates and subtyping as predicate inclusion; and on the level of kinds, the application rule (S-App), if one views kinds as implicational propositions and constructors as their proofs.[1] In an algorithmic version of subtyping, both kinds of cuts have to be eliminated [Com95, PS97]. Application is eliminated by considering constructors in normal form V only, in our case η-long β-normal form. Transitivity is incorporated into the variable rule (SA-Bound), which is a fusion of (S-Var), (S-App), (S-Eq), and (S-Trans). It looks up the bound U of the head X of a *neutral* constructor $X\boldsymbol{V}$ and recursively checks subtyping for $U\,\boldsymbol{V}$. To keep everything in normal form, a *normalizing application* $U \mathbin{@} \boldsymbol{V}$ is employed which will be defined in Section 3.1.

The algorithm receives as input a context Γ and two η-long constructors V, V' such that $\Gamma \vdash V : \kappa$ and $\Gamma \vdash V' : \kappa$ and decides $\Gamma \vdash V \leq V' : \kappa$.

$\boxed{\Gamma \vdash_{\mathsf{a}} V \leq V'}$

$$\text{(SA-Top)}\ \frac{}{\Gamma \vdash_{\mathsf{a}} V \leq \top} \qquad \text{(SA-Refl)}\ \frac{}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq X\boldsymbol{V}}$$

$$\text{(SA-Bound)}\ \frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq W}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq W}$$

$$\text{(SA-Arr)}\ \frac{\Gamma \vdash_{\mathsf{a}} W \leq V : * \qquad \Gamma \vdash_{\mathsf{a}} V' \leq W'}{\Gamma \vdash_{\mathsf{a}} V \to V' \leq W \to W'}$$

$$\text{(SA-All)}\ \frac{\Gamma, X \leq V : \kappa \vdash_{\mathsf{a}} W \leq W'}{\Gamma \vdash_{\mathsf{a}} \forall X \leq V : \kappa . W \leq \forall X \leq V : \kappa . W'}$$

$$\text{(SA-Abs)}\ \frac{\Gamma, X : \kappa \vdash_{\mathsf{a}} V \leq V'}{\Gamma \vdash_{\mathsf{a}} \lambda X : \kappa . V \leq \lambda X : \kappa . V'}$$

Observe that since we are dealing with η-long forms, the only constructors of higher kind are λ-abstractions (SA-Abs). The rules specify a deterministic algorithm if one checks applicability of rules earlier in the list prior to rules mentioned later. In particular, (SA-Top) is checked first, and (SA-Refl) before (SA-Bound).

Our algorithmic subtyping rules correspond to Compagnoni's [Com95] and Pierce and Steffen's [PS97]—except that they consider Church-style β-normal constructors, hence, their algorithm does not validate η-equality. Thus, one cannot claim our algorithm is new, but in the remainder of this article we will present a novel, very direct and concise correctness proof with purely syntactic methods, which has a realistic chance of being mechanized in a theorem prover such as Coq, Isabelle, or Twelf.

[1] More precisely, (S-App) is the rule of modus ponens, but in unrestricted form it allows non-normal proofs.

3 Normalization of Constructors

In last section we have described a decision procedure for subtyping which works on η-long β-normal forms. In this section, we will describe a normalization algorithm $\mathsf{nf}_{\_}(-)$ which is correct w. r. t. judgmental equality:

1. Sound: If $\Gamma \vdash F : \kappa$ then $\Gamma \vdash \mathsf{nf}_\Gamma(F) = F : \kappa$.
2. Complete: If $\Gamma \vdash F = F' : \kappa$ then $\mathsf{nf}_\Gamma(F) \equiv \mathsf{nf}_\Gamma(F')$.

3.1 Hereditary Substitution

There are abundant strategies to compute β-normal forms; we use *bottom-up normalization*, because its termination can be shown directly in a simply-typed setting—in our case it is a *simply-kinded* setting.

The bottom-up strategy $\mathsf{nf}(H)$ normalizes the immediate subterms of H and then puts them back together. For an application $H = F\,G$, normalization $\mathsf{nf}(F)$ of the function part may yield an abstraction $\lambda Y : \kappa.\, F'$. In this case, a β-normal form can be recovered by substituting $\mathsf{nf}(G)$ for Y in F'. New redexes may be created in turn which are resolved immediately by another substitution etc. The iteration of this process, which we call *hereditary substitution*, terminates since the kind of the substituted variable decreases with each iteration.

Hereditary substitutions are implicit in combinatorial normalization proofs, e. g., Prawitz [Pra65], Levy [Lév76], Girard, Lafont, and Taylor [GLT89, Ch. 4], and Amadio and Curien [AC97, Thm. 2.2.9]. They were first made explicit by Watkins et. al. [WC+03] and Adams [Ada05] for dependent types. We follow the presentation of the first author [Abe06a].

Hereditary substitution (see Figure 2) is given as a 4-ary function $[G/X]^\kappa H$, whose result $\hat{F}$ is either just a constructor F or a constructor annotated with a kind, written F^κ. If G and H are β-normal (and well-kinded) then the result will also be β-normal (and well-kinded). Results can be coerced back to constructors via an erasure operation given by $\underline{F^\kappa} = F$ and $\underline{F} = F$.

It is easy to see that if $[G/X]^\kappa H = F^{\kappa_2}$ then $|\kappa_2| \leq |\kappa|$. This invariant ensures the termination of hereditary substitution. For correctness we need to show that, modulo β-equality, hereditary substitution is conventional substitution. In the following, we leave the coercion implicit.

Lemma 4 (Termination and soundness of hereditary substitutions). *If* $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma, [G/X]\Gamma' \vdash [G/X]F = [G/X]^\kappa F : \kappa'$.

Proof. By lexicographical induction on $(|\kappa|, \mathcal{D})$. The proof is a straightforward extension of the soundness proof in [Abe06a]. □

Corollary 1. *If* $\Gamma \vdash F : \kappa \to \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\Gamma \vdash F \mathbin{@} G = F\,G : \kappa'$.

$\boxed{[G/X]^\kappa F}$

$$
\begin{array}{lll}
[G/X]^\kappa Y & := G^\kappa & \text{if } X = Y \\
 & \quad Y & \text{otherwise} \\
 & & \\
[G/X]^\kappa(\lambda Y{:}\kappa'.\, F) & := \lambda Y{:}\kappa'.\, \underline{[G/X]^\kappa F} & \text{where } Y \text{ fresh for } X, G \\
 & & \\
[G/X]^\kappa(F H) & := (\underline{\underline{[\hat{H}/Y]^{\kappa_1} F'}})^{\kappa_2} & \text{if } \hat{F} = (\lambda Y{:}\kappa_1'.\, F')^{\kappa_1 \to \kappa_2} \\
 & \quad \underline{\hat{F}}\ \underline{\hat{H}} & \text{otherwise} \\
 & \quad \text{herein, } \hat{F} = [G/X]^\kappa F & \\
 & \qquad\quad \hat{H} = [G/X]^\kappa H & \\
 & & \\
[G/X]^\kappa(A \to B) & := \underline{[G/X]^\kappa A} \to \underline{[G/X]^\kappa B} & \\
 & & \\
[G/X]^\kappa(\forall Y \le H{:}\kappa.A) & := \forall Y \le \underline{[G/X]^\kappa H}{:}\kappa. & \\
 & \quad \underline{[G/X]^\kappa A} & \text{where } Y \text{ fresh for } X, G \\
 & & \\
[G/X]^\kappa \top & := \top &
\end{array}
$$

$\boxed{F \mathbin{@} G}$

$$
\begin{array}{lll}
(\lambda X{:}\kappa.\, F) \mathbin{@} G & := \underline{[G/X]^\kappa F} & \\
F \mathbin{@} G & := F\, G & \text{if } F \neq \lambda X{:}\kappa.\, F'
\end{array}
$$

$\boxed{F \mathbin{@} \boldsymbol{G}}$

$$
F \mathbin{@} \boldsymbol{G} := ((F \mathbin{@} G_1) \mathbin{@} G_2 \ldots) \mathbin{@} G_{|\boldsymbol{G}|}
$$

Fig. 2. Hereditary substitution

Lemma 5 (Commutativity of hereditary substitutions). *Let* $\Gamma \vdash U : \kappa$ *and* $\Gamma, X{:}\kappa, \Gamma' \vdash V : \kappa'$ *with* $\kappa' = \boldsymbol{\kappa} \to \kappa_0$.

1. *If* $\Gamma, X{:}\kappa, \Gamma' \vdash W_i : \kappa_i$ *for* $1 \le i \le |\boldsymbol{\kappa}|$ *then*

$$[U/X]^\kappa(V \mathbin{@} \boldsymbol{W}) \equiv [U/X]^\kappa V \mathbin{@} [U/X]^\kappa \boldsymbol{W}$$

2. *If* $\Gamma, X{:}\kappa, \Gamma', Y{:}\kappa', \Gamma'' \vdash W : \kappa''$ *then:*

$$[U/X]^\kappa([V/Y]^{\kappa'} W) \equiv [[U/X]^\kappa V/Y]^{\kappa'}([U/X]^\kappa W)$$

Proof. Simultaneously by simultaneous induction[2] on $\{\kappa, \kappa'\}$, first 1 and then 2. The proof of 2 proceeds by a local induction on W. □

[2] Simultaneous order is defined by $\{X, Y\} < \{X', Y'\}$ if $X < X'$ and $Y \le Y'$.

3.2 Computing the Long Normal Form

Well-kinded constructors are β-normalizing, and we can compute their η-long β-normal form. First, let us define the η-expansion $\eta_\kappa(N)$ of a neutral constructor N at kind κ by $\eta_*(N) = N$ and $\eta_{\kappa\to\kappa'}(N) = \lambda X{:}\kappa.\, \eta_{\kappa'}(N\, \eta_\kappa(X))$.

Lemma 6 (η-expansion is sound). *If $\Gamma \vdash N : \kappa$ then $\Gamma \vdash N = \eta_\kappa(N) : \kappa$.*

Normalization is given by a function $\mathsf{nf}_\Gamma(F)$, defined by recursion on F.

$$
\begin{array}{lll}
\mathsf{nf}_\Gamma(X) & := \eta_\kappa(X) & \text{if } (X \leq \_ : \kappa) \in \Gamma \\
\mathsf{nf}_\Gamma(\top) & := \top & \\
\mathsf{nf}_\Gamma(\lambda X{:}\kappa.\, F) & := \lambda X{:}\kappa.\, \mathsf{nf}_{\Gamma, X:\kappa}(F) & \\
\mathsf{nf}_\Gamma(A \to B) & := \mathsf{nf}_\Gamma(A) \to \mathsf{nf}_\Gamma(B) & \\
\mathsf{nf}_\Gamma(\forall X \leq G{:}\kappa.A) & := \forall X \leq \mathsf{nf}_\Gamma(G){:}\kappa.\, \mathsf{nf}_{\Gamma, X\leq G:\kappa}(A) & \\
\mathsf{nf}_\Gamma(F\, G) & := \mathsf{nf}_\Gamma(F)\, @\, \mathsf{nf}_\Gamma(G) &
\end{array}
$$

The algorithm η-expands the variables, this way producing η-long β-normal constructors. For well-kinded constructors $\mathsf{nf}()$ returns the normal form (see Theorem 1), but not for all ill-kinded constructors, e.g., $\mathsf{nf}(\Omega) = \Omega$ for $\Omega = (\lambda X{:}*.\, X\, X)\, (\lambda X{:}*.\, X\, X)$.

We omit the subscript Γ if clear from the context of discourse. Normalization can be extended to contexts in the obvious way: $\mathsf{nf}(\Gamma)$ computes a context where all bounds have been normalized.

Lemma 7 (Soundness and termination of normalization). *If $\Gamma \vdash F : \kappa$ then $\Gamma \vdash F = \mathsf{nf}_\Gamma(F) : \kappa$.*

Proof. By induction on the kinding derivation, using Lemma 6 in the variable case, and Corollary 1 in the application case. □

3.3 Characterization of Long Normal Forms

We will now define a judgement $\Gamma \vdash V \Uparrow \kappa$, read "*$V$ is hereditarily normal of kind κ in context Γ*", that classifies the β-normal constructor V as a possible input for the subtyping algorithm. In particular, V must be η-long, and new redexes which might be created by (SA-Bound) during the execution of the algorithm must be normalizable. We call such redexes *hidden*, an example is $\forall X \leq (\lambda Y{:}*.\, V){:}* \to *.\, X\, W$ which contains the hidden redex $(\lambda Y{:}*.\, V)\, W$.

$\boxed{\Gamma \vdash V \Uparrow \kappa}$

$$\text{(LN-Bound)}\ \frac{(X \leq U{:}\boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash V_i \Uparrow \kappa_i \qquad \Gamma \vdash U\, @\, \boldsymbol{V} \Uparrow * \ (\star)}{\Gamma \vdash X\, \boldsymbol{V} \Uparrow *}$$

$$\text{(LN-Abs)}\ \frac{\Gamma, X{:}\kappa \vdash V \Uparrow \kappa'}{\Gamma \vdash \lambda X{:}\kappa.\, V \Uparrow \kappa \to \kappa'} \qquad \text{(LN-Arr)}\ \frac{\Gamma \vdash V \Uparrow * \qquad \Gamma \vdash W \Uparrow *}{\Gamma \vdash V \to W \Uparrow *}$$

$$\text{(LN-All)}\ \frac{\Gamma, X \leq U{:}\kappa \vdash V \Uparrow *}{\Gamma \vdash \forall X \leq U{:}\kappa.\, V \Uparrow *} \qquad \text{(LN-Top)}\ \frac{\Gamma \Uparrow}{\Gamma \vdash \top \Uparrow *}$$

Remark 1. The third hypothesis ($\star$) in (LN-Bound) ensures normalization of hidden redexes, hence $\Gamma \vdash V \Uparrow \kappa$ can be used as a termination measure for the algorithm. Even without ($\star$) the judgement characterizes the η-long normal forms.

A context is normal, $\Gamma \Uparrow$, if all bounds in Γ are hereditarily normal. In the following, we establish that normalization is the identity on long normal forms. First we prove it for variables.

Lemma 8

1. *If* $\Gamma \vdash V \Uparrow \kappa$ *then* $[V/X]^\kappa \eta_\kappa(X) \equiv V$.
2. *If* $\mathcal{D} :: \Gamma, X:\kappa, \Gamma' \vdash W \Uparrow \kappa'$ *then* $[\eta_\kappa(X)/X]^\kappa W \equiv W$.

Proof. Simultaneously by induction on κ, and a local induction on $\mathcal{D}$ in 2. □

Lemma 9 (Idempotency of nf). *If* $\mathcal{D} :: \Gamma \vdash U \Uparrow \kappa$ *then* $\mathsf{nf}(U) \equiv U$.

Proof. By induction on $\mathcal{D}$, using Lemma 8 in the variable case. □

We now build up to a normalization theorem: we will show that nf produces a hereditarily normal form from each well-kinded constructor. The following lemma, which can be seen as the heart of our technical development, proves normalization of application.

Lemma 10 (Substitution and application for normal forms)
Let $\mathcal{E} :: \Gamma \vdash U \Uparrow \kappa$.

1. *Assume* $\mathcal{D} :: \Gamma, X:\kappa, \Gamma' \Uparrow$. *Then* $\Gamma, [U/X]^\kappa \Gamma' \Uparrow$.
2. *Assume* $\mathcal{D} :: \Gamma, X:\kappa, \Gamma' \vdash V \Uparrow \kappa'$. *Then* $\Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa V \Uparrow \kappa'$.
3. *Assume* $\mathcal{D} :: \Gamma \vdash V \Uparrow \kappa \to \kappa'$. *Then* $\Gamma \vdash V \mathbin{@} U \Uparrow \kappa'$.

Proof. Simultaneously by lexicographical induction on $(|\kappa|, \mathcal{D})$. The interesting case of item 2 is (LN-Bound) with $V \equiv X\boldsymbol{V}$, $\kappa = \boldsymbol{\kappa} \to *$.

$$\frac{\Gamma, X:\kappa, \Gamma' \vdash V_i \Uparrow \kappa_i \qquad \Gamma, X:\kappa, \Gamma' \vdash \top_\kappa \mathbin{@} \boldsymbol{V} \Uparrow *}{\Gamma, X:\kappa, \Gamma' \vdash X\boldsymbol{V} \Uparrow *}$$

By induction hypothesis (2), we have $\mathcal{D}_i :: \Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa V_i \Uparrow \kappa_i$ for $i = 1..|\boldsymbol{\kappa}|$. Moreover, we have by induction hypothesis (3):

$$\mathcal{E}_1 :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} [U/X]^\kappa V_1 \Uparrow \kappa_2 \to \ldots \to \kappa_n \to *$$

because $(\kappa_1, \mathcal{E}) < (\kappa, \mathcal{D})$. We again have by induction hypothesis (3):

$$\mathcal{E}_2 :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} ([U/X]^\kappa V_1) \mathbin{@} ([U/X]^\kappa V_2) \Uparrow \kappa_3 \to \ldots \to \kappa_n \to *$$

because $(\kappa_2, \mathcal{E}_1) < (\kappa, \mathcal{D})$. Continuing this schema, we get

$$\mathcal{E}_n :: \Gamma, [U/X]^\kappa \Gamma' \vdash U \mathbin{@} [U/X]^\kappa \boldsymbol{V} \Uparrow *$$

which is equivalent to $\Gamma, [U/X]^\kappa \Gamma' \vdash [U/X]^\kappa (X\,\boldsymbol{V}) \Uparrow *$ and we are finished.

In case $V \equiv Y\boldsymbol{V}$, with $Y \neq X$, we use Lemma 5. □

Theorem 1 (Normalization)

1. *If* $\mathcal{D} :: \Gamma \vdash$ *then* $\mathsf{nf}(\Gamma) \Uparrow$.
2. *If* $\mathcal{D} :: \Gamma \vdash F : \kappa$ *then* $\mathsf{nf}(\Gamma) \vdash \mathsf{nf}(F) \Uparrow \kappa$.

Proof. Simultaneously by induction on $\mathcal{D}$, using the previous lemma in case of application. □

3.4 Completeness of Normalization

In this section, we prove completeness of the normalization function, i.e., that the normal forms of judgmentally equal constructors are identical.

Lemma 11. *If* $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ *and* $\Gamma \vdash G : \kappa$ *then* $\mathsf{nf}([G/X]F) \equiv [\mathsf{nf}(G)/X]^{\kappa}\mathsf{nf}(F)$.

Proof. By induction on $\mathcal{D}$. In the application case we use Lemma 5. □

Theorem 2 (Completeness of the normalization). *If* $\mathcal{D} :: \Gamma \vdash F = F' : \kappa$ *then* $\mathsf{nf}(F) \equiv \mathsf{nf}(F')$.

Proof. The proof is by induction on $\mathcal{D}$, in case (Eq-η) we use the characterization of η-long β-normal forms (Theorem 1 and Lemma 9) and in case (Eq-β) we use the previous lemma. □

Corollary 2 (Uniqueness of normal forms). *If* $\Gamma \vdash V = V' : \kappa$ *and* $\Gamma \vdash V, V' \Uparrow \kappa$ *then* $V \equiv V'$.

Proof. Directly, using Theorem 2 and Lemma 9. □

4 Verification of Algorithmic Subtyping

In this section, we show that the subtyping algorithm is sound and complete for the declarative rules in Section 2. The difficult part, namely establishing the necessary properties of hereditary substitution and normalization and constructing a termination measure $\Gamma \vdash V \Uparrow \kappa$, has been completed in the last section. The actual properties of algorithmic subtyping are now easy to verify.

Soundness of the algorithm is straightforward because the algorithmic rules are less permissive than the declarative ones.

Lemma 12. *Let* $\Gamma \vdash V, V' : \kappa$. *If* $\mathcal{D} :: \Gamma \vdash_{\mathsf{a}} V \leq V'$, *then* $\Gamma \vdash V \leq V' : \kappa$.

Proof. By induction on $\mathcal{D}$. □

Theorem 3 (Soundness of algorithmic subtyping). *Let* $\Gamma \vdash F, F' : \kappa$. *If* $\mathcal{D} :: \mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$, *then* $\Gamma \vdash F \leq F' : \kappa$.

Proof. Combining lemmata 12 and 7. To account for normalization of Γ, we establish that declarative equality and subtyping remain valid if we replace bounds in the context by judgmentally equal ones. □

Completeness of the algorithm means that any derivable statement in the declarative system is also derivable in the algorithmic system. This is more difficult to show than soundness, because we have eliminated declarative rules like (S-Trans) or (S-App). Thus, we need to show that these rules are admissible in the algorithmic system.

Lemma 13 (Reflexivity). *If V is β-normal then $\Gamma \vdash_{\mathsf{a}} V \leq V$.*

Proof. By induction on V. □

Lemma 14 (Transitivity). *$\mathcal{D}_1 :: \Gamma \vdash_{\mathsf{a}} V_1 \leq V_2$ and $\mathcal{D}_2 :: \Gamma \vdash_{\mathsf{a}} V_2 \leq V_3$ imply $\Gamma \vdash_{\mathsf{a}} V_1 \leq V_3$.*

Proof. By induction on $\mathcal{D}_1$. □

The following substitution lemma is the key step in showing that algorithmic subtyping is closed under application, i.e., complete for rule (S-App).

Lemma 15 (Substitution). *Let $\Gamma, X:\kappa, \Gamma' \vdash V, V' \Uparrow \kappa'$ and $\Gamma \vdash U \Uparrow \kappa$. If $\mathcal{D} :: \Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} V \leq V'$ then $\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa V \leq [U/X]^\kappa V'$.*

Proof. By induction on $\mathcal{D}$. The interesting case is (SA-Bound) for $V \equiv Y\boldsymbol{V}$ with Y bound later in the context than X.

$$\frac{(Y \leq U' : \boldsymbol{\kappa} \to *) \in \Gamma' \qquad \Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} U' \mathbin{@} \boldsymbol{V} \leq W}{\Gamma, X:\kappa, \Gamma' \vdash_{\mathsf{a}} Y\boldsymbol{V} \leq W}$$

By induction hypothesis we have

$$\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa (U' \mathbin{@} \boldsymbol{V}) \leq [U/X]^\kappa W$$

which by Lemma 5, part (1), is equivalent to

$$\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} [U/X]^\kappa U' \mathbin{@} [U/X]^\kappa \boldsymbol{V} \leq [U/X]^\kappa W.$$

The goal $\Gamma, [U/X]^\kappa \Gamma' \vdash_{\mathsf{a}} Y\, [U/X]^\kappa \boldsymbol{V} \leq [U/X]^\kappa W$ follows by (SA-Bound). □

Completeness can now be shown by induction on derivations, using the previous lemmas: reflexivity, transitivity and substitution.

Theorem 4 (Completeness).
If $\mathcal{D} :: \Gamma \vdash F \leq F' : \kappa$ then $\mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$.

Proof. By induction on $\mathcal{D}$, using Lemma 15 in case (S-App). □

Termination of algorithmic subtyping is now proven via the inductive termination predicate $\Gamma \vdash V \Uparrow \kappa$ (which by Theorem 1 holds for all normal forms of well-kinded constructors). It is a considerable simplification of Compagnoni's [Com95] termination measure which features intersection types as well. It has to be investigated whether the simplicity of our measure can be kept in the presence of intersection types.

Theorem 5 (Termination of algorithmic subtyping). *If* $\mathcal{D}_1 :: \Gamma \vdash V \Uparrow \kappa$ *and* $\mathcal{D}_2 :: \Gamma \vdash V' \Uparrow \kappa$ *then the query* $\Gamma \vdash_{\mathsf{a}} V \leq V'$ *terminates.*

Proof. By simultaneous induction on $\{\mathcal{D}_1, \mathcal{D}_2\}$. We detail the case that $\mathcal{D}_1$ ends with (LN-Bound).

$$\frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash V_i \Uparrow \kappa_i \text{ for all } i \qquad \Gamma \vdash U \mathbin{@} \boldsymbol{V} \Uparrow *}{\Gamma \vdash X\boldsymbol{V} \Uparrow *}$$

In case $V' \neq X\boldsymbol{V}$ we apply the rule (SA-Bound)

$$\frac{(X \leq U : \boldsymbol{\kappa} \to *) \in \Gamma \qquad \Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq V'}{\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq V'}$$

By the induction hypothesis, the query $\Gamma \vdash_{\mathsf{a}} U \mathbin{@} \boldsymbol{V} \leq V'$ terminates, thus, the query $\Gamma \vdash_{\mathsf{a}} X\boldsymbol{V} \leq V'$ terminates as well. □

Corollary 3 (Decidability). *If* $\Gamma \vdash F, F' : \kappa$ *then* $\Gamma \vdash F \leq F' : \kappa$ *is decidable.*

Proof. By Theorem 2 $\mathsf{nf}(\Gamma) \vdash \mathsf{nf}(F), \mathsf{nf}(F') \Uparrow \kappa$. Hence, the query $\mathsf{nf}(\Gamma) \vdash_{\mathsf{a}} \mathsf{nf}(F) \leq \mathsf{nf}(F')$ terminates by Theorem 5. If it succeeds then $\Gamma \vdash F \leq F' : \kappa$ by soundness (Theorem 3), if it fails then $\Gamma \not\vdash F \leq F' : \kappa$ by completeness (Theorem 4).

5 Conclusions and Related Work

In this article we have proven decidability of subtyping for $F^{\omega}_{<:}$ with $\beta\eta$-equality in a purely syntactical way, with only first-order inductive definitions and simple induction measures. The proofs have been organized with a formalization in mind and can be mechanized in proof assistants such as Coq, Isabelle, or Twelf.

Related work. Foundations of $F^{\omega}_{<:}$ and the type-theoretic investigation of object-oriented languages have been laid by Cardelli [Car88], Mitchell [Mit90], and Abadi and Cardelli [AC96].

Most similar to the present work is the one of Compagnoni [Com95] both in the design of the subtyping algorithm and the organization of the proofs of soundness, completeness, and termination. She also treats intersection types but not η-equality of constructors. The main differences in the proof are: She inserts the notion of *normal subtyping* between the declarative and the algorithmic one, which we do not require, she refers to strong normalization of reduction, which adds some complexity to the proof, and she has a complicated termination measure which involves the longest reduction sequences of constructors classified by a judgement similar to our hereditary normal forms. Our approach has allowed considerable simplifications in comparison with Compagnoni's.

Pierce and Steffen [PS97] justify algorithmic subtyping using rewriting theory. They consider an extension of β-reductions by Γ-reduction, which replaces a head variable by its bound, and $\top$-reduction which witnesses that $\top$ is defined

pointwise for higher kinds. The confluence and strong normalization theorems shed light on the combination of these notions of reduction, however, they are not the shortest path to the verification of the subtyping algorithm.

Compagnoni and Goguen [CG03] construct a Kripke model based on typed operational semantics to analyze the metatheory of $\mathcal{F}^{\omega}_{\leq}$, an extension of $F^{\omega}_{<:}$ by bounded abstraction $\lambda X \leq F.\, G$. They use this model to prove anti-symmetry [CG06] of subtyping which allows replacing judgmental equality by bi-inclusion. Recently they have investigated Church-style subtyping [CG07], where each variable is annotated by its bound, hence, the complicated Kripke-model is replaced by a non-Kripke logical relation. Their subtyping algorithm works with weak-head normal forms, only the decision whether the bound-lookup rule (SA-Bound) should fire demands full normalization in some cases. The structure of their proof, with logical relation and strong normalization theorem, seems very robust w. r. t. extensions of the type language; however, it cannot compete with us in terms of proof economics.

The first author [Abe06b] considers $\mathsf{F}^{\widehat{}}_{\omega}$, the higher-order polymorphic lambda-calculus with sized types and polarized subtyping, but without bounded quantification. His subtyping algorithm uses weak head normal forms and is proven complete purely syntactically as in this work. The lexicographic induction on kinds and constructors, which in this article justifies hereditary substitution, is used by the first author to prove a substitution property for algorithmic subtyping, which entails completeness. It remains open whether this argument scales to bounded quantification.

Future work. Steffen has extended his work to $F^{\omega}_{<:}$ with polarities, i. e., co-, contra-, and mixed-variant type constructors are distinguished. For this extension, only backtracking algorithms exist [Ste98, p. 102]; we would like to see whether our proof technique can simplify its metatheory as well. The same question arises for the extension $\mathcal{F}^{\omega}_{\leq}$ by bounded abstraction.

Acknowledgments. The research has been partially supported by the EU coordination action TYPES (510996).

References

[Abe06a] Abel, A.: Implementing a normalizer using sized heterogeneous types. In: McBride, C., Uustalu, T. (eds.) Wksh. on Mathematically Structured Functional Programming, MSFP 2006 (2006)

[Abe06b] Abel, A.: Polarized subtyping for sized types. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 381–392. Springer, Heidelberg (2006)

[AC96] Abadi, M., Cardelli, L.: A Theory of Objects. Springer, Heidelberg (1996)

[AC97] Amadio, R., Curien, P.-L.: Domains and Lambda Calculi. Cambridge University Press, Cambridge (1997)

[Ada05] Adams, R.: A Modular Hierarchy of Logical Frameworks. Ph.D. thesis, University of Manchester (2005)

[Car88] Cardelli, L.: Structural subtyping and the notion of power type. In: Proc. of the 15th ACM Symp. on Principles of Programming Languages, POPL 1988, pp. 70–79 (1988)

[CG03] Compagnoni, A.B., Goguen, H.: Typed operational semantics for higher-order subtyping. Inf. Comput. 184(2), 242–297 (2003)

[CG06] Compagnoni, A., Goguen, H.: Anti-symmetry of higher-order subtyping and equality by subtyping. Math. Struct. in Comput. Sci. 16, 41–65 (2006)

[CG07] Compagnoni, A., Goguen, H.: Subtyping à la Church. In: Barendsen, E., Capretta, V., Geuvers, H., Niqui, M. (eds.) Reflections on Type Theory, λ-calculus, and the Mind. Essays dedicated to Henk Barendregt on the Occasion of his 60th Birthday. Radboud University Nijmegen (2007)

[Com95] Compagnoni, A.B.: Decidability of higher-order subtyping with intersection types. In: Pacholski, L., Tiuryn, J. (eds.) CSL 1994. LNCS, vol. 933, pp. 46–60. Springer, Heidelberg (1995)

[CW85] Cardelli, L., Wegner, P.: On understanding types, data abstraction, and polymorphism. ACM Computing Surveys 17(4), 471–522 (1985)

[GLT89] Girard, J.-Y., Lafont, Y., Taylor, P.: Proofs and Types. Cambridge Tracts in Theoretical Computer Science, vol. 7. Cambridge University Press, Cambridge (1989)

[HL07] Harper, R., Licata, D.: Mechanizing metatheory in a logical framework. J. Func. Program 17(4–5), 613–673 (2007)

[LCH07] Lee, D.K., Crary, K., Harper, R.: Towards a mechanized metatheory of Standard ML. In: Hofmann, M., Felleisen, M. (eds.) Proc. of the 34th ACM Symp. on Principles of Programming Languages, POPL 2007, pp. 173–184. ACM Press, New York (2007)

[Lév76] Lévy, J.-J.: An algebraic interpretation of the $\lambda\beta$K-calculus; and an application of a labelled λ-calculus. Theor. Comput. Sci. 2(1), 97–114 (1976)

[Mit90] Mitchell, J.C.: Toward a typed foundation for method specialization and inheritance. In: Proc. of the 17th ACM Symp. on Principles of Programming Languages, POPL 1990, pp. 109–124 (1990)

[Pie92] Pierce, B.C.: Bounded quantification is undecidable. In: POPL, pp. 305–315 (1992)

[Pie02] Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)

[Pra65] Prawitz, D.: Natural Deduction. Almqvist & Wiksell, Stockholm, 1965. Republication by Dover Publications Inc., Mineola (2006)

[PS97] Pierce, B.C., Steffen, M.: Higher order subtyping. Theor. Comput. Sci. 176(1,2), 235–282 (1997)

[Rod07] Rodriguez, D.: Algorithmic Subtyping for Higher Order Bounded Quantification. Diploma thesis, LMU Munich (2007)

[Ste98] Steffen, M.: Polarized Higher-Order Subtyping. Ph.D. thesis, Technische Fakultät, Universität Erlangen (1998)

[WC+03] Watkins, K., Cervesato, I., Pfenning, F., Walker, D.: A concurrent logical framework I: Judgements and properties. Tech. rep., School of Computer Science. Carnegie Mellon University, Pittsburgh (2003)

On Isomorphisms of Intersection Types

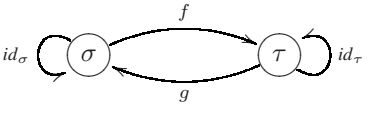
Mariangiola Dezani-Ciancaglini[1], Roberto Di Cosmo[2],
Elio Giovannetti[1], and Makoto Tatsuta[3]

[1] Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy
[2] Université Paris Diderot, PPS, UMR 7126, case 7014, 2 place Jussieu, 75005 Paris, France
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, 101-8430 Tokyo, Japan

Abstract. The study of type isomorphisms for different λ-calculi started over twenty years ago, and a very wide body of knowledge has been established, both in terms of results and in terms of techniques. A notable missing piece of the puzzle was the characterization of type isomorphisms in the presence of intersection types. While at first thought this may seem to be a simple exercise, it turns out that not only finding the right characterization is not simple, but that the very notion of isomorphism in intersection types is an unexpectedly original element in the previously known landscape, breaking most of the known properties of isomorphisms of the typed λ-calculus. In particular, types that are equal in the standard models of intersection types may be non-isomorphic.

1 Introduction

The notion of *type isomorphism* is a particularization of the general notion of isomorphism as defined, for example, in category theory. Two objects σ and τ are *isomorphic* iff there exist two morphisms $f\colon \sigma \to \tau$ and $g\colon \tau \to \sigma$ such that $f \circ g = id_\tau$ and $g \circ f = id_\sigma$:



Analogously, two *types* σ and τ in some (abstract) programming language, like the typed λ-calculus, are *isomorphic* if the same diagram holds, with f and g functions of types $\sigma \to \tau$ and $\tau \to \sigma$ respectively.

In the early 1980s, some interest started to develop in the problem of finding *all* the domain equations (type isomorphisms) that must hold in *every* model of a given language, or *valid isomorphisms of types*, as they were called in [5].

There are essentially two families of techniques for addressing this question: it is possible to work *syntactically* to characterize those programs f that possess an inverse g making the above diagram commute, or one can work *semantically* trying to find some specific model that captures the isomorphisms valid in all models (see [9] for a recent survey).

Each approach has its own difficulty: finding the syntactic characterization of the invertible terms can be very hard, while the rest follows then rather straightforwardly; finding the right specific model and showing that the only isomorphisms holding in it

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 461–477, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Table 1. Type isomorphisms in typed lambda calculi

(**swap**) $\sigma \to (\tau \to \gamma) = \tau \to (\sigma \to \gamma)$	Th^1
1. $\sigma \times \tau = \tau \times \sigma$	$Th^1_{\times T}$
2. $\sigma \times (\tau \times \gamma) = (\sigma \times \tau) \times \gamma$	
3. $(\sigma \times \tau) \to \gamma = \sigma \to (\tau \to \gamma)$	
4. $\sigma \to (\tau \times \gamma) = (\sigma \to \tau) \times (\sigma \to \gamma)$	
5. $\sigma \times \mathbf{T} = \sigma$	
6. $\sigma \to \mathbf{T} = \mathbf{T}$	
7. $\mathbf{T} \to \sigma = \sigma$	
8. $\forall X.\forall Y.\sigma = \forall Y.\forall X.\sigma$	$+ \mathbf{swap} = Th^2$
9. $\forall X.\sigma = \forall Y.\sigma[Y/X]$	
10. $\forall X.(\sigma \to \tau) = \sigma \to \forall X.\tau$	
11. $\forall X.\sigma \times \tau = \forall X.\sigma \times \forall X.\tau$	$Th^2_{\times T}$
12. $\forall X.\mathbf{T} = \mathbf{T}$	
split $\forall X.\sigma \times \tau = \forall X.\forall Y.\sigma \times (\tau[Y/X])$	$-10, 11 = Th^{ML}$

N.B.: in equation 8, X must be free for Y in σ and $Y \notin FTV(\sigma)$; in equation 10, $X \notin FTV(\sigma)$.

are those holding in all models can be very hard too, even if the advent of game semantics has a bit blurred the distinction between these approaches, by building models which are quite syntactical in nature [11].

In our work, we started along the first line (as we already know the shape of the invertible terms), so here we only recall the relevant literature for the syntactic approach.

Type isomorphisms and invertible terms

In [7], Dezani fully characterized the *invertible λ-terms* as the *finite hereditary permutators*, a class of terms which can be easily defined inductively, and which can be seen as a family of generalized η-expansions.

Definition 1 (Invertible term). *A λ-term M is* invertible *if there exists a term M^{-1} such that $M \circ M^{-1} = M^{-1} \circ M =_{\beta\eta} \mathbf{I}$ (where $\circ$ denotes, as usual, functional composition, and $\mathbf{I}$ is the identity $\lambda x.x$). Obviously, M^{-1} is called an* inverse *of M.*

Definition 2 (Finite Hereditary Permutator). *A* finite hereditary permutator (f.h.p.) *is a λ-term whose β-normal form is $\lambda z x_1 \ldots x_n \,.\, z Q_1 \ldots Q_n$ ($n \geq 0$) and is such that, for a permutation π of $1 \ldots n$, the λ-terms $\lambda x_{\pi(1)}.Q_1, \ldots, \lambda x_{\pi(n)}.Q_n$ are finite hereditary permutators.*

Theorem 1. *[7] A λ-term is invertible iff it is a finite hereditary permutator.*

Observe that f.h.p.'s are closed terms: so, by the above theorem, invertible λ-terms are closed terms.The proof of Theorem 1 shows that every f.h.p. has a unique inverse

modulo $\beta\eta$-conversion. We use P to range over β-normal forms of f.h.p.'s. Thus P^{-1} denotes the unique (modulo η-conversion) inverse of P.

While the result of [7] was obtained in the framework of the untyped λ-calculus, it turned out that this family of invertible terms *can be typed* in the simply typed λ-calculus, and this allowed Bruce and Longo [5] to prove by a straightforward induction on the structure of the f.h.p.'s that in the simply typed λ-calculus the only type isomorphisms w.r.t. $\beta\eta$-equality are those induced by the *swap* equation

$$\sigma \to (\tau \to \rho) = \tau \to (\sigma \to \rho).$$

Notice that the type isomorphisms which correspond to invertible terms (called *definable isomorphisms of types in [5]*) are *a priori* not the same as the *valid isomorphisms of types*: a definable isomorphism seems to be a stronger notion, demanding that not only a given isomorphism holds in all models, but that it also holds in all models *uniformly*. Nevertheless, in all the cases studied in the literature, it is easy to build a free model out of the calculus, and to prove that valid and definable isomorphisms coincide, so this distinction has gradually disappeared in time, and in this work we will use the following definition of type isomorphism.

Definition 3 (Type isomorphism). *Given a λ-calculus along with a type system, two types σ and τ (in the system's type language) are* isomorphic*, and we write $\sigma \approx \tau$, if in the calculus there exists an invertible term, i.e., by the above theorem, a f.h.p. P, such that $\vdash \mathsf{P} : \sigma \to \tau$ and $\vdash \mathsf{P}^{-1} : \tau \to \sigma$ hold in the system. Following a standard nomenclature, we say that the term P proves the isomorphism $\sigma \approx \tau$, and we write $\sigma \approx_{\mathsf{P}} \tau$. Of course, $\sigma \approx_{\mathsf{P}} \tau$ iff $\sigma \approx_{\mathsf{P}^{-1}} \tau$.*

An immediate observation is that

Theorem 2. *Isomorphism is an equivalence relation.*

Observe that transitivity holds because invertible terms are closed under functional composition by definition. So if the f.h.p. P_1 proves $\sigma \approx \tau$ and the f.h.p. P_2 proves $\tau \approx \rho$, then $\mathsf{P}_2 \circ \mathsf{P}_1$ is a f.h.p. that proves $\sigma \approx \rho$.

By extending Dezani's original technique to the invertible terms in typed calculi with additional constructors (like product and unit type) or with higher order (System F or Core-ML), it has been possible to pursue this line of research to the point of getting a full characterization of isomorphisms in a whole set of typed λ-calculi, from $\lambda^1\beta\eta$, which corresponds to $IPC(\Rightarrow)$, the intuitionistic positive calculus with implication, whose isomorphisms are described by Th^1 [13,5], to $\lambda^1\beta\eta\pi*$, which corresponds to Cartesian Closed Categories and $IPC(\mathbf{True}, \wedge, \Rightarrow)$, for which $Th^1_{\times T}$ is complete [4][1], to $\lambda^2\beta\eta$ (System F), which corresponds to $IPC(\forall, \Rightarrow)$, and whose isomorphisms are given by Th^2 [5], to $\lambda^2\beta\eta\pi*$ (System F with products and unit type), which corresponds to $IPC(\forall, \mathbf{True}, \wedge, \Rightarrow)$, whose isomorphisms are given by $Th^2_{\times T}$ [8]. A summary of the axioms in these theories is given in Table 1.

Hence, in this line of research, the standard approach has been to find all the type isomorphisms for a given language (λ-calculus) and a given notion of equality on terms

[1] But this result had been proved earlier by Soloviev using model-theoretic techniques [15].

(which almost always contains extensional rules like η, as otherwise no nontrivial invertible term exists [7]) as a consequence of an inductive characterization of the invertible terms. The general schema in all the known cases is the same: first guess an equational theory for the isomorphisms (this is the hard part), then by induction on the structure of the invertible terms show the completeness of the equational theory (the easy part).

One notable missing piece in the table summarizing the theory of isomorphisms of types is the case of intersection types. At first sight, it should be an easy exercise to deal with it: we already know the form of the invertible terms, as they are again the f.h.p.'s, and it should just be a matter of guessing the right equational theory and proving it complete by induction.

But it turns out that with intersection types all the intuitions that one has formed in the other systems fail: the intersection type discipline can give many widely different typings for the same term, so that the simple proof technique originated in [5] does not apply, and we are in for some surprises.

In this paper, we explore the world of type isomorphisms with intersection types, establishing a series of results that are quite unexpected: on the one hand, we will see in Section 2 that in the presence of intersection types the theory of isomorphisms is no longer a congruence, so that there is no hope to capture these isomorphisms via an equational theory, and the theory does not even include equality in the standard models; yet, decidability can be easily established, though with no simple bound on its complexity. On the other hand, we will be able to provide a very precise characterization of isomorphisms, via a special notion of similarity for type normal forms.

2 Basic Properties of Isomorphisms with Intersection Types

In this section we establish the basic properties of intersection types that show their deep difference with respect to the other cases studied in the literature, before tackling, in the later sections, their precise characterization.

Isomorphisms of intersection types are not a congruence
In all the cases known in the literature, the isomorphism equivalence relation is a *congruence*, as the type constructors explored so far (arrow, cartesian product, universal quantification, sum) all preserve isomorphisms.

Intersection, by contrast, does not preserve isomorphism: from $\sigma \approx \sigma'$ and $\tau \approx \tau'$ it does not follow, in general, that $\sigma \cap \tau \approx \sigma' \cap \tau'$. The intuitive reason is that the existence of two separate (invertible) functions that respectively transform all values of type σ into values of type σ' and all those of type τ into values of type τ', does not ensure that there is a function mapping any value that is both of type σ and of type τ to a value that is both of type σ' and of type τ'.

For example, though the isomorphism $\alpha \to \beta \to \gamma \approx \beta \to \alpha \to \gamma$ is given by the f.h.p. $\lambda xyz\,.\,xzy$, the two types $\varphi \cap (\alpha \to \beta \to \gamma)$ and $\varphi \cap (\beta \to \alpha \to \gamma)$ are not isomorphic, since the term $\lambda yz\,.\,xzy$ cannot be typed (from the assumption $x\colon \varphi$) with an atomic type φ, which can only be transformed into itself by the identity.

Therefore we have the following result:

Theorem 3. *The theory of isomorphisms for intersection types* is not a congruence.

In particular, this theory *cannot be described* with a standard equational theory: a non-trivial equivalence relation has to be devised[2].

Isomorphisms do not contain equality in the standard intersection models
Another quite unconventional fact is that

Theorem 4. *Types* equality *in the standard models* [3] *of intersection types does not entail type isomorphisms.*

Proof. Take for example the two isomorphic types $\alpha\to\gamma$ and $(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)$. They are semantically coincident, because the type $\alpha\cap\beta\to\gamma$ is greater than $\alpha\to\gamma$, and therefore its presence in the intersection is useless.

Now, if we just add to both a seemingly innocent intersection with an atomic type, we obtain the two types $(\alpha\to\gamma)\cap\varphi$ and $(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)\cap\varphi$, which also have identical meanings but are not isomorphic: if they were, the isomorphism would be given by the f.h.p. $\lambda xy.xy$ because, while the identity is trivially able to map any intersection to each of its components (i.e., $\vdash \lambda x.x\colon \sigma\cap\tau\to\sigma$, $\vdash \lambda x.x\colon \sigma\cap\tau\to\tau$), the mapping in the opposite direction, from $\alpha\to\gamma$ to $(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)$, requires an *$\eta$-expansion* of the identity, as can be seen from the following derivation, where $\Gamma = x\colon \alpha\to\gamma, y\colon \alpha\cap\beta$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma\vdash x\colon\alpha\to\gamma \qquad \dfrac{\Gamma\vdash y\colon\alpha\cap\beta}{\Gamma\vdash y\colon\alpha}(\cap\mathrm{E})}{\Gamma\vdash xy\colon\gamma}(\to\mathrm{E})}{x\colon\alpha\to\gamma\vdash\lambda y.xy\colon\alpha\cap\beta\to\gamma}(\to\mathrm{I}) \qquad \dfrac{\dfrac{\dots}{x\colon\alpha\to\gamma,\, y\colon\alpha\vdash xy\colon\gamma}(\to\mathrm{E})}{x\colon\alpha\to\gamma\vdash\lambda y.xy\colon\alpha\to\gamma}(\to\mathrm{I})}{x\colon\alpha\to\gamma\vdash\lambda y.xy\colon(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)}(\cap\mathrm{I})}{\vdash\lambda xy.xy\colon(\alpha\to\gamma)\to(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)}(\to\mathrm{I})$$

An η-expansion of the identity, however, cannot map an atomic type to itself; in particular, the judgment $x\colon(\alpha\to\gamma)\cap\varphi\vdash\lambda y.xy\colon\varphi$ cannot be derived, hence the term $\lambda xy.xy$ cannot be assigned the type $(\alpha\to\gamma)\cap\varphi\to(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)\cap\varphi$.

We could establish an isomorphism relation including the pair of types $(\alpha\to\gamma)\cap\varphi$ and $(\alpha\cap\beta\to\gamma)\cap(\alpha\to\gamma)\cap\varphi$ only by assuming, as in some models, that all atomic types are arrow types.

One could simply see this fact as a proof that the universal model – traditionally hard to find – where all and only the valid isomorphisms hold is not a standard model; but it is quite unconventional that *equality* in the standard models is not included in the isomorphism relation, and this really comes from the strong intensionality of intersection types.

[2] Notice that even in the very tricky case of the sum types, isomorphism is a congruence [10].

[3] The standard models of intersection types are the models in which the arrow is interpreted as function space constructor and the intersection as set theoretic intersection.

Decidability
Despite the weird nature of isomorphisms with intersection types, it is easy to establish the following decidability result.

Theorem 5. *Isomorphisms of intersection types are decidable.*

Proof. Given two types σ and τ, a f.h.p. of type $\sigma \to \tau$ may have a number of top-level abstractions at most equal to the number of top-level arrows, and also every subterm of the f.h.p. cannot have, at each nesting level, more abstractions than the corresponding number of arrows nested at that level. The number of f.h.p.'s that are candidate to prove the isomorphism $\sigma \approx \tau$ is therefore finite, and each of them can be checked whether it can be assigned the type $\sigma \to \tau$ [14].

3 The Type System and the Reduction to Type Normal Form

In order to keep the theory sufficiently manageable, we restrict arrow types to the ones ending with an atomic type: $\sigma := \alpha \cap \alpha \cdots \cap \alpha$, with $\alpha := \sigma \to \cdots \to \sigma \to \varphi$, since it is well known that such restriction does not alter the set of typeable terms [1].

The formal syntax of types therefore is:

$$\begin{array}{l} \sigma := \alpha \mid \sigma \cap \sigma \quad \text{types} \\ \alpha := \varphi \mid \sigma \to \alpha \text{ atomic and arrow types} \end{array}$$

where φ denotes an atomic type. We use σ, τ, ρ to range over types, α, β, γ to range over arrow types, and $\varphi, \chi, \psi, \vartheta, \xi$ to range over atomic types. We will occasionally use roman letters to denote atomic types in complex examples.

Also, we consider types modulo idempotence, commutativity and associativity of $\cap$, so we can write $\bigcap_{i \in I} \sigma_i$ with finite I. We write $\sigma \equiv \tau$ if σ coincides with τ modulo idempotence, commutativity and associativity of $\cap$.

The type assignment system is the standard simple system with intersection types for the ordinary λ-calculus [6].

$$(Ax) \qquad \Gamma, x : \sigma \vdash x : \sigma$$

$$(\to I)\ \frac{\Gamma, x : \sigma \vdash M : \alpha}{\Gamma \vdash \lambda x.M : \sigma \to \alpha} \qquad (\to E)\ \frac{\Gamma \vdash M : \sigma \to \alpha \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \alpha}$$

$$(\cap I)\ \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau} \qquad (\cap E)\ \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \tau}$$

Since we do not allow an arrow type to have an intersection on the right-hand side, we must modify the formal definition of isomorphism. To this purpose, the following notation is useful.
Notation If $\tau = \bigcap_{i \in I} \alpha_i$, then $\vdash \mathsf{P} : \sigma \mapsto \tau$ is short for $\vdash \mathsf{P} : \sigma \to \alpha_i$ for all $i \in I$.

Definition 3 is then replaced by

Definition 4 (Isomorphism for the intersection type system). *Two intersection types σ and τ are* isomorphic *($\sigma \approx \tau$) if there exists a f.h.p. P such that $\vdash \mathsf{P} : \sigma \mapsto \tau$ and $\vdash \mathsf{P}^{-1} : \tau \mapsto \sigma$.*

Adopting a technique similar to one used by [8], we introduce a notion of *type normal form* along with an isomorphism-preserving reduction, and then we give the syntactic characterization of isomorphism on normal types only. We use reduction to eliminate redundant (arrow) types in intersections, i.e., those that are intersected with types intuitively included in them. For example, $(\sigma \to \alpha) \cap (\sigma \cap \tau \to \alpha)$ reduces to $(\sigma \to \alpha)$. The reduction relation is expressed with the help of some preliminary definitions.

Definition 5 (Intersection Occurrence). An intersection occurrence *of* α *in* τ *is defined inductively as follows (always considering, as stated at the beginning,* $\cap$ *modulo commutativity and associativity):*

- *if* $\tau = \alpha \cap \sigma$*, then the showed occurrence of* α *is an intersection occurrence;*
- *if* $\tau = \rho_1 \to \ldots \to \rho_n \to \tau' \to \beta$*, and* α *is an intersection occurrence in* τ'*, then* α *is an intersection occurrence in* τ*;*
- *if* $\tau = \beta \cap \sigma$ *and* α *is an intersection occurrence in* β*, then* α *is an intersection occurrence in* τ*.*

Definition 6 (Erasure). *If* α *is an intersection occurrence in* τ*, then the* erasure of α in τ *(notation* $|\tau|_\alpha$*) is defined by:*

- $|\tau|_\alpha = \sigma$ *if* $\tau = \alpha \cap \sigma$*;*
- $|\tau|_\alpha = \sigma_1 \to \ldots \to \sigma_n \to |\tau'|_\alpha \to \beta$ *if* $\tau = \sigma_1 \to \ldots \to \sigma_n \to \tau' \to \beta$ *and* α *is an intersection occurrence in* τ'*;*
- $|\tau|_\alpha = |\beta|_\alpha \cap \sigma$ *if* $\tau = \beta \cap \sigma$ *and* α *is an intersection occurrence in* β*.*

Definition 7 (Finite Hereditary Identity). *A* finite hereditary identity (f.h.i.) *is a* β*-normal form obtained from* $\lambda x.x$ *through a finite (possibly zero) number of* η*-expansions. We use* Id *to range over f.h.i.'s (Note that f.h.i.'s are particular forms of f.h.p.'s).*

We are now able to state the reduction rule.

Definition 8 (Reduction). *The* reduction rule *is:*

$$\tau \rightsquigarrow |\tau|_\alpha$$

if there are a type sub-expression α *and two f.h.i.s* Id, Id' *such that* $\vdash \mathsf{Id} : |\tau|_\alpha \mapsto \tau$ *and* $\vdash \mathsf{Id}' : \tau \mapsto |\tau|_\alpha$.

It is immediate to see that reduction is confluent and terminating, thus defining a *type normal form*. Also, a type and its normal form are isomorphic by definition, since a f.h.i. is a f.h.p. and all f.h.i.'s are inverse of one another.

Observe that, as noted in Section 1, redundant arrow types cannot be erased if they occur in intersections with atomic types, which prevent η-expansions of the identity to provide the isomorphism between the original type and the simplified type: thus, while we have $(\alpha \cap \beta \to \gamma) \cap (\alpha \to \gamma) \rightsquigarrow \alpha \to \gamma$, the type $(\alpha \cap \beta \to \gamma) \cap (\alpha \to \gamma) \cap \varphi$ does not reduce to $(\alpha \to \gamma) \cap \varphi$. For any type σ, the type $\sigma \cap \varphi$ (with φ atomic) is in normal form, since the atom φ blocks any reduction.

On the other hand, the type $\sigma = ((\alpha \cap \beta \to \psi) \to \varphi) \cap ((\alpha \to \psi) \cap \chi \to \varphi)$ reduces to the type $\gamma = (\alpha \cap \beta \to \psi) \to \varphi$ through the f.h.i. $\lambda xy.x(\lambda v.yv)$. Note that,

as pointed out in Section 1, the mapping from σ to γ only needs the simple identity (we have $\vdash \lambda x.x\colon \sigma \to \gamma$), but the opposite mapping requires an η-*expansion* of the identity, so as to have the typing $\vdash \lambda xy.x(\lambda v.yv)\colon \gamma \mapsto \sigma$.

We may now introduce the key notion of our work, i.e., a *similarity* between types, which we will prove to be the desired syntactic counterpart of the notion of isomorphism.

Definition 9 (Similarity). *The* similarity *between two sequences of types* $\langle \sigma_1, \ldots, \sigma_m \rangle$ *and* $\langle \tau_1, \ldots, \tau_m \rangle$*, written* $\langle \sigma_1, \ldots, \sigma_m \rangle \sim \langle \tau_1, \ldots, \tau_m \rangle$*, is the smallest equivalence relation such that:*

1. $\langle \sigma_1, \ldots, \sigma_m \rangle \sim \langle \sigma_1, \ldots, \sigma_m \rangle$*;*
2. *if* $\langle \sigma_1, \ldots, \sigma_i, \sigma_{i+1}, \ldots, \sigma_m \rangle \sim \langle \tau_1, \ldots, \tau_i, \tau_{i+1}, \ldots, \tau_m \rangle$*, then* $\langle \sigma_1, \ldots, \sigma_i \cap \sigma_{i+1}, \ldots, \sigma_m \rangle \sim \langle \tau_1, \ldots, \tau_i \cap \tau_{i+1}, \ldots, \tau_m \rangle$*;*
3. *if* $\langle \sigma_i^{(1)}, \ldots, \sigma_i^{(m)} \rangle \sim \langle \tau_i^{(1)}, \ldots, \tau_i^{(m)} \rangle$ *for* $1 \leq i \leq n$*, then* $\langle \sigma_1^{(1)} \to \ldots \to \sigma_n^{(1)} \to \alpha^{(1)}, \ldots, \sigma_1^{(m)} \to \ldots \to \sigma_n^{(m)} \to \alpha^{(m)} \rangle \sim \langle \tau_{\pi(1)}^{(1)} \to \ldots \to \tau_{\pi(n)}^{(1)} \to \alpha^{(1)}, \ldots, \tau_{\pi(1)}^{(m)} \to \ldots \to \tau_{\pi(n)}^{(m)} \to \alpha^{(m)} \rangle$*, where* π *is a permutation of* $1, \ldots, n$*.*

Similarity between types *is trivially defined as similarity between unary sequences:* $\sigma \sim \tau$ *if* $\langle \sigma \rangle \sim \langle \tau \rangle$.

The reason is that, for two intersection types to be isomorphic, it is not sufficient that they coincide modulo permutations of types in the arrow sequences, as in the case of cartesian products: the permutation must be the same for all the corresponding type pairs in an intersection. The notion of similarity exactly expresses such property.

For example, the two types $(\varphi_1 \to \varphi_2 \to \varphi_3 \to \chi) \cap (\psi_1 \to \psi_2 \to \psi_3 \to \vartheta)$ and $(\varphi_3 \to \varphi_2 \to \varphi_1 \to \chi) \cap (\psi_2 \to \psi_3 \to \psi_1 \to \vartheta)$ are not *similar* and thus (as we will prove) not isomorphic, while the corresponding types with cartesian product instead of intersection are. The reason is that, owing to the semantics of intersection, the same f.h.p. must be able to map all the conjuncts of one intersection to the corresponding conjuncts in the other intersection. In the example, there is obviously not one f.h.p. that maps both $\varphi_1 \to \varphi_2 \to \varphi_3 \to \chi$ to $\varphi_3 \to \varphi_2 \to \varphi_1 \to \chi$ and at the same time $\psi_1 \to \psi_2 \to \psi_3 \to \vartheta$ to $\psi_2 \to \psi_3 \to \psi_1 \to \vartheta$.
On the other hand, the two types

$$(\rho_1 \to \rho_2 \to \rho_3 \to \alpha) \cap (\sigma_1 \to \sigma_2 \to \sigma_3 \to \beta),$$
$$(\rho_2 \to \rho_3 \to \rho_1 \to \alpha) \cap (\sigma_2 \to \sigma_3 \to \sigma_1 \to \beta)$$

are similar (and therefore isomorphic), since the permutation is the same in the two components of the intersection.

A type like $(\sigma_1 \to \ldots \to \sigma_n \to \alpha) \cap \varphi$ may only be similar (and thus isomorphic) to itself, since the presence of the atom φ in the intersection blocks the possibility of any permutation other than the identity in the conjunct type subexpression $\sigma_1 \to \ldots \to \sigma_n \to \alpha$.

A more complex example of similar types is the following:

$\alpha_1 \cap \alpha_2 \sim \beta_1 \cap \beta_2$,
where (with roman letters indicating atomic types):
$\alpha_1 = (e \to f) \to (a \cap b \to c \to d) \cap (g \to b \to c) \to s \to t$
$\alpha_2 = (h \to k) \cap (p \to q) \to (u \to v \to w) \to q \cap r \to (a \cap b \to z)$
$\beta_1 = (c \to a \cap b \to d) \cap (b \to g \to c) \to s \to (e \to f) \to t$
$\beta_2 = (v \to u \to w) \to q \cap r \to (h \to k) \cap (p \to q) \to (a \cap b \to z)$.

Note that the introduction of type sequences in the definition of similarity is needed in order to keep the correspondence between types in intersections. Consider, for example, the following two types:

$$\rho_1 = (\sigma_1 \cap \alpha \to \sigma_2 \to \beta) \cap (\tau_1 \to \tau_2 \to \gamma),$$
$$\rho_2 = (\sigma_2 \to \sigma_1 \to \beta) \cap (\tau_2 \to \alpha \cap \tau_1 \to \gamma).$$

They are not isomorphic, and are also not similar since the sequences $\langle \sigma_1 \cap \alpha, \tau_1 \rangle$, $\langle \sigma_1, \alpha \cap \tau_1 \rangle$ are not. If, however, the definitions were given directly through intersection, owing to the associativity of $\cap$ the two sequences would be represented by the same intersection $\sigma_1 \cap \alpha \cap \tau_1$, and the two types ρ_1, ρ_2 would therefore be similar.

An equivalent, slightly more algorithmic, definition of *similarity* may be given through a notion of *permutation tree*.

Definition 10 (Permutation Tree)

- *The empty tree* $\varnothing$ *is a permutation tree.*
- $\langle \pi, [\Pi_1, \ldots, \Pi_n] \rangle$ *is a permutation tree if* π *is a permutation of* $1, \ldots, n$ *and* $\Pi_1, \ldots, \Pi_n$ *are permutation trees.*

An example of a permutation tree is the tree $\Pi_0 = \langle (2,3,1), [\langle (2,1), [\varnothing, \varnothing] \rangle, \varnothing, \varnothing] \rangle$. A more complex example is the tree Π defined as follows:

$\Pi = \langle (2,3,1), [\Pi_1, \varnothing, \Pi_3] \rangle$
where
$\Pi_1 = \left\langle \big(3,1,4,2\big), \left[\varnothing, \varnothing, \langle (2,1), [\varnothing, \varnothing] \rangle, \langle (1,3,2), [\varnothing, \varnothing, \varnothing] \rangle\right] \right\rangle$
$\Pi_3 = \left\langle \big(1,2,3\big), \left[\langle (2,1), [\varnothing, \varnothing] \rangle, \varnothing, \langle (3,2,1,4), [\varnothing, \varnothing, \varnothing, \varnothing] \rangle\right] \right\rangle$

A permutation tree is nothing but an abstract representation of a f.h.p. One may easily build the concrete f.h.p. corresponding to a permutation tree, by creating as many fresh variables as is the cardinality of the permutation and by recursively creating subterms that respectively have those variables as head variables, in the order specified by the permutation.

In the following definition **trm** is the recursive mapping: it takes a permutation tree and the name z of a fresh variable, and creates a term with free head variable z, which is the β-reduct of the corresponding f.h.p. applied to z. The top-level mapping **fhp** merely abstracts the head variable so as to transform the term into a f.h.p. proper.

Definition 11 (F.h.p. corresponding to a permutation tree)
The f.h.p. corresponding to a permutation tree Π *is:*

$\mathbf{fhp}(\Pi) = \lambda z.\mathbf{trm}(\Pi, z)$, *with* z *fresh variable;*
$\mathbf{trm}(\varnothing, z) = z$;
$\mathbf{trm}(\langle \pi, [\Pi_1, \ldots, \Pi_n] \rangle, z) = \lambda x_1 \ldots x_n \,.\, z\, \mathbf{trm}(\Pi_1, x_{\pi(1)}) \ldots \mathbf{trm}(\Pi_n, x_{\pi(n)})$
with $x_1 \ldots x_n$ *fresh variables.*

Examples.
The f.h.p. corresponding to the permutation tree $\Pi_0 = \langle (2,3,1), [\langle (2,1), [\varnothing, \varnothing] \rangle, \varnothing, \varnothing] \rangle$ is the term $\lambda z x_1 x_2 x_3 . z(\lambda u_1 u_2 . x_2 u_2 u_1) x_3 x_1$.

The f.h.p. corresponding to the permutation tree $\Pi = \langle (2,3,1), [\Pi_1, \varnothing, \Pi_3] \rangle$ of the example above is the term $P = \lambda z x_1 x_2 x_3 . z P_1 P_2 P_3$, where

$$
\begin{aligned}
P_1 &= \lambda u_1 u_2 u_3 u_4 \,.\, x_2 u_3 u_1 (\lambda v_1 v_2 \,.\, u_4 v_2 v_1)(\lambda w_1 w_2 w_3 \,.\, u_2 w_1 w_3 w_2) \\
P_2 &= x_3 \\
P_3 &= \lambda y_1 y_2 y_3 \,.\, x_1 (\lambda s_1 s_2 \,.\, y_1 s_2 s_1) y_2 (\lambda t_1 t_2 t_3 t_4 \,.\, y_3 t_3 t_2 t_1 t_4)
\end{aligned}
$$

A permutation tree represents a tree of nested permutations: if we *apply* it to a type having a homologous tree structure, i.e., if we (are able to) recursively perform on the type all the permutations at all levels, we obtain a new type which is clearly *similar* to the original one. We therefore give the following natural definition.

Definition 12 (Application of a permutation tree to a type)
Application of a permutation tree is a partial map from types to types:

- $\varnothing(\sigma) = \sigma$
- $\langle \pi, [\Pi_1, \ldots, \Pi_n] \rangle (\sigma_1 \to \cdots \to \sigma_n \to \alpha) = \Pi_1(\sigma_{\pi(1)}) \to \cdots \to \Pi_n(\sigma_{\pi(n)}) \to \alpha$
- $\Pi(\sigma \cap \tau) = \Pi(\sigma) \cap \Pi(\tau)$
- $\Pi(\sigma) = \text{undefined}$ *otherwise.*

Taking again one of the examples above, if

$$
\begin{aligned}
\alpha_1 &= (e \to f) \to (a \cap b \to c \to d) \cap (g \to b \to c) \to s \to t \\
\alpha_2 &= (h \to k) \cap (p \to q) \to (u \to v \to w) \to q \cap r \to (a \cap b \to z) \\
\Pi_0 &= \langle (2,3,1), [\langle (2,1), [\varnothing, \varnothing] \rangle, \varnothing, \varnothing] \rangle
\end{aligned}
$$

then we have $\Pi_0(\alpha_1 \cap \alpha_2) = \beta_1 \cap \beta_2$, where

$$
\begin{aligned}
\beta_1 &= (c \to a \cap b \to d) \cap (b \to g \to c) \to s \to (e \to f) \to t \\
\beta_2 &= (v \to u \to w) \to q \cap r \to (h \to k) \cap (p \to q) \to (a \cap b \to z)
\end{aligned}
$$

With the other example, if we have:

$$
\begin{aligned}
&\sigma = \gamma_1 \to \gamma_2 \to \xi_3 \to \xi \\
&\text{where} \\
&\gamma_1 = (\varphi_{11} \to \varphi_{12} \to \chi_1) \to \chi_2 \to (\varphi_{31} \to \varphi_{32} \to \varphi_{33} \to \varphi_{34} \to \chi_3) \to \chi \\
&\gamma_2 = \vartheta_1 \to (\psi_{21} \to \psi_{22} \to \psi_{23} \to \vartheta_2) \to \vartheta_3 \to (\psi_{41} \to \psi_{42} \to \vartheta_4) \to \vartheta
\end{aligned}
$$

then $\Pi(\sigma) = \tau$, where

$$\tau = \gamma'_2 \to \xi_3 \to \gamma'_1 \to \xi$$
where
$$\gamma'_2 = \vartheta_3 \to \vartheta_1 \to (\psi_{42} \to \psi_{41} \to \vartheta_4) \to (\psi_{21} \to \psi_{23} \to \psi_{22} \to \vartheta_2) \to \vartheta$$
$$\gamma'_1 = (\varphi_{12} \to \varphi_{11} \to \chi_1) \to \chi_2 \to (\varphi_{33} \to \varphi_{32} \to \varphi_{31} \to \varphi_{34} \to \chi_3) \to \chi$$

Two types can then be defined as equivalent when one can be obtained from the other (modulo idempotence, commutativity and associativity, as usual) by applying a permutation tree.

Definition 13 (Type permutation-equivalence)
Two types σ and τ are permutation-equivalent*, notation $\sigma \doteq \tau$, if* $\exists \Pi \,.\, \Pi(\sigma) \equiv \tau$.

It is trivial to see that if $\Pi(\sigma) \equiv \tau$, then there also exists an *inverse* permutation tree Π^{-1} such that $\Pi^{-1}(\tau) \equiv \sigma$.

It is easy to prove that $\sigma \sim \tau$ if and only if $\sigma \doteq \tau$, so that the latter equivalence merely is an alternative definition of the previously defined similarity. We will therefore always use the first notation.
As an immediate consequence of Definition 12, we have the following lemma.

Lemma 1. *If $\Pi(\sigma) \equiv \tau$, with $\Pi = \langle \pi, [\Pi_1, \ldots, \Pi_n] \rangle$, then there exists a set I of indices such that σ and τ have the forms:*

$$\sigma \equiv \bigcap_{i \in I} (\sigma_1^i \to \ldots \to \sigma_n^i \to \alpha^i), \quad \tau \equiv \bigcap_{i \in I} ({\tau_1}^i \to \ldots \to {\tau_n}^i \to \alpha^i)$$

and for all $i \in I$, for $k = 1, \ldots, n$, one has $\Pi_k(\sigma^i_{\pi(k)}) \equiv \tau^i_k$, *therefore* $\sigma^i_{\pi(k)} \sim \tau^i_k$.

Note that the above definitions of *similarity* are *not* equivalent to stating that, in the inductive case:

$$\bigcap_{i \in I} (\sigma_1^i \to \ldots \to \sigma_n^i \to \alpha^i) \;\sim\; \bigcap_{i \in I} ({\tau_1}^i \to \ldots \to {\tau_n}^i \to \alpha^i)$$

if there exists a permutation π such that

$$\forall i \in I \,.\, \tau^i_k \sim \sigma^i_{\pi(k)} \quad \text{and} \quad \bigcap_{i \in I} \tau^i_k \sim \bigcap_{i \in I} \sigma^i_{\pi(k)} \quad \text{for } k = 1, \ldots, n.$$

A counterexample is given by the following pair of types:

$$\sigma = (\beta_1 \to \alpha_1) \cap (\beta_2 \to \alpha_2) \cap (\beta_3 \to \alpha_3)$$
$$\tau = (\gamma_1 \to \alpha_1) \cap (\gamma_2 \to \alpha_2) \cap (\gamma_3 \to \alpha_3)$$

where

$$\beta_1 = \varphi \to \chi \to \psi \to \vartheta = \gamma_2$$
$$\beta_2 = \varphi \to \psi \to \chi \to \vartheta = \gamma_3$$
$$\beta_3 = \chi \to \varphi \to \psi \to \vartheta = \gamma_1$$

We have $\Pi_1(\beta_1) \equiv \gamma_1$, $\Pi_2(\beta_2) \equiv \gamma_2$, $\Pi_3(\beta_3) \equiv \gamma_3$, with

$$\Pi_1 = \langle (2,1,3), [\varnothing, \varnothing, \varnothing] \rangle, \quad \Pi_2 = \langle (1,3,2), [\varnothing, \varnothing, \varnothing] \rangle,$$
$$\Pi_3 = \langle (3,1,2), [\varnothing, \varnothing, \varnothing] \rangle,$$

and therefore $\beta_1 \sim \gamma_1$, $\beta_2 \sim \gamma_2$, $\beta_3 \sim \gamma_3$; also, $\beta_1 \cap \beta_2 \cap \beta_3 \sim \gamma_1 \cap \gamma_2 \cap \gamma_3$ since trivially $\beta_1 \cap \beta_2 \cap \beta_3 \equiv \gamma_1 \cap \gamma_2 \cap \gamma_3$. This, however, does not allow us to conclude that $\sigma \sim \tau$, since there exists no permutation tree Π such that $\Pi(\sigma) = \tau$ (because $\Pi(\sigma) = \Pi(\sigma_1) \cap \Pi(\sigma_2) \cap \Pi(\sigma_3)$ should hold), or, equivalently, since – following the first definition of similarity – the two sequences $\langle \beta_1, \beta_2, \beta_3 \rangle$, $\langle \gamma_1, \gamma_2, \gamma_3 \rangle$ $(= \langle \beta_3, \beta_1, \beta_2 \rangle)$ are not similar. Accordingly, the two types σ and τ are not similar ($\sigma \not\sim \tau$), and thus, as will be proved by Theorem 8, not isomorphic ($\sigma \not\approx \tau$).

4 Standard Properties of the Type System

Our system, being a trivial restriction of the simple intersection type system, obviously has the well-known standard properties of the unrestricted system [2]. In particular, the Lemma 2, a generation lemma and the subject reduction property hold, and the proofs are standard.

Lemma 2. *If $\Gamma \vdash \lambda x.M : \sigma$, then σ cannot be an atomic type.*

Lemma 3 (Generation Lemma)

1. *If $x : \bigcap_{i \in I} \alpha_i \vdash x : \bigcap_{j \in J} \beta_j$, then $\{\beta_j \mid j \in J\} \subseteq \{\alpha_i \mid i \in I\}$.*
2. *If $\Gamma \vdash \lambda x.M : \bigcap_{i \in I} (\sigma_i \to \alpha_i)$, then for all $i \in I$: $\Gamma, x : \sigma_i \vdash M : \alpha_i$.*
3. *If $\Gamma \vdash MN : \alpha$, then there exists a type σ such that $\Gamma \vdash M : \sigma \to \alpha$ and $\Gamma \vdash N : \sigma$.*

Proof. The proof is by induction on derivations.

Theorem 6 (Subject Reduction). *If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_\beta N$, then $\Gamma \vdash N : \sigma$.*

Proof. Standard.

The Lemmata 4, 5 and 6 state some useful properties of η-expansions of the identity and of permutators. In particular, Lemma 4 expresses a necessary condition for a f.h.i. to be typeable with an arrow type, and gives the forms of the types of its subterms. Lemma 5.1 says that a f.h.i. is able to map an intersection $\alpha \cap \beta$ to one of its components, for example α, only if it is able to map such component α to itself (which is not always the case, since the number of top-level arrows in α cannot be less than the number of top-level abstractions of the f.h.i.). Lemma 5.2 states the rather obvious fact that if a f.h.i. is able to map both the type σ to itself and the type τ to itself, then it also maps their intersection to itself.

Finally, Lemma 6 states that a f.h.p. P maps an intersection $\bigcap_{i \in I} \alpha_i$ to another intersection $\bigcap_{j \in J} \beta_j$, i.e., $\vdash \mathsf{P} : \bigcap_{i \in I} \alpha_i \mapsto \bigcap_{j \in J} \beta_j$, if and only if every component β_j in the target intersection is obtained by P from some component α_i in the source intersection.

In such lemmata and in the following we write judgments of the form $x : \sigma \vdash \mathsf{P}x : \tau$ (where P may also be Id) instead of $\vdash \mathsf{P} : \sigma \mapsto \tau$, in order to simplify the proofs. The two kinds of judgments are equivalent not because of a property of subject expansion, which does not hold in general, but because a f.h.p. P is an abstraction, and therefore $\vdash \mathsf{P} : \sigma \mapsto \tau$ if and only if $x : \sigma \vdash \mathsf{P}x : \tau$, as can be easily seen: if $x : \sigma \vdash \mathsf{P}x : \tau$, with $\mathsf{P} = \lambda x.M$, then by β-reduction and subject reduction one has $x : \sigma \vdash M : \tau$, whence,

Table 2. Proof of Lemma 6

$$
\begin{array}{ll}
x:\bigcap_{i\in I}\alpha_i \vdash \mathsf{P}x:\bigcap_{j\in J}\beta_j & \Longrightarrow x:\bigcap_{i\in I}\alpha_i \vdash \lambda z_1\ldots z_n.x(\mathsf{P}_1 z_{\pi(1)})\ldots(\mathsf{P}_n z_{\pi(n)}):\bigcap_{j\in J}\beta_j \\
& \quad \text{by Theorem 6} \\
& \Longrightarrow \forall j\in J.\ x:\bigcap_{i\in I}\alpha_i \vdash \lambda z_1\ldots z_n.x(\mathsf{P}_1 z_{\pi(1)})\ldots(\mathsf{P}_n z_{\pi(n)}):\beta_j \\
& \quad \text{by rule } (\cap E) \\
& \Longrightarrow \forall j\in J.\ \beta_j = \sigma_1^{(j)}\to\ldots\to\sigma_n^{(j)}\to\gamma^{(j)} \\
& \quad \text{for some } \sigma_1^{(j)},\ldots,\sigma_n^{(j)},\gamma^{(j)} \text{ by Lemma 2} \\
& \Longrightarrow \forall j\in J.\ \Gamma\vdash x(\mathsf{P}_1 z_{\pi(1)})\ldots(\mathsf{P}_n z_{\pi(n)}):\gamma^{(j)} \\
& \quad \text{where } \Gamma = x:\bigcap_{i\in I}\alpha_i, z_1:\sigma_1^{(j)},\ldots z_n:\sigma_n^{(j)} \text{ by Lemma 3(2)} \\
& \Longrightarrow \forall j\in J.\ x:\bigcap_{i\in I}\alpha_i \vdash x:\tau_1^{(j)}\to\ldots\to\tau_n^{(j)}\to\gamma^{(j)} \ \&\ \\
& \quad z_{\pi(1)}:\sigma_{\pi(1)}^{(j)}\vdash \mathsf{P}_1 z_{\pi(1)}:\tau_1^{(j)} \ \&\ \ldots \\
& \quad \&\ z_{\pi(n)}:\sigma_{\pi(n)}^{(j)}\vdash \mathsf{P}_n z_{\pi(n)}:\tau_n^{(j)} \\
& \quad \text{for some } \tau_1^{(j)},\ldots,\tau_n^{(j)} \text{ by Lemma 3(3)} \\
& \Longrightarrow \forall j\in J.\ \exists i_j\in I.\ \alpha_{i_j}=\tau_1^{(j)}\to\ldots\to\tau_n^{(j)}\to\gamma^{(j)} \\
& \quad \text{by Lemma 3(1)} \\
& \Longrightarrow \forall j\in J.\ \exists i_j\in I.\ x:\alpha_{i_j}\vdash \mathsf{P}x:\beta_j \\
& \quad \text{by rules } (\to E) \text{ and } (\to I).
\end{array}
$$

by ($\cap$ E) and ($\to$I), $\vdash \mathsf{P}:\sigma\mapsto\tau$ (rule ($\cap$ E) is needed since ($\to$I) can only build arrow types). More generally, this is a consequence of the property of subject expansion for intersection types in the λI-calculus. The opposite implication, from $\vdash \mathsf{P}:\sigma\mapsto\tau$ to $x:\sigma\vdash \mathsf{P}x:\tau$, trivially follows by ($\to$E) and ($\cap$ I).

Lemma 4. *1. If $n\neq m$ or $\varphi\neq\varphi'$, then there is no* Id *such that*

$$\vdash \mathsf{Id}:(\sigma_1\to\ldots\to\sigma_n\to\varphi)\to\tau_1\to\ldots\to\tau_m\to\varphi'.$$

2. If $\vdash \mathsf{Id}:(\sigma_1\to\ldots\to\sigma_n\to\varphi)\to\tau_1\to\ldots\to\tau_n\to\varphi$ and $\sigma_m\neq\tau_m$ and $\sigma_q=\tau_q$ for $m+1\leq q\leq n$, then $\mathsf{Id}\ {}_\beta\!\longleftarrow \lambda y z_1\ldots z_p.y(\mathsf{Id}_1 z_1)\ldots(\mathsf{Id}_p z_p)$ for some p and some $\mathsf{Id}_1,\ldots,\mathsf{Id}_p$ such that and $m\leq p\leq n$ and $\vdash \mathsf{Id}_l:\tau_l\mapsto\sigma_l$ for $1\leq l\leq p$.

Proof. Easy, using the Generation Lemma.

Lemma 5. *1. If $x:\sigma\cap\alpha\vdash \mathsf{Id}\,x:\alpha$, then $x:\alpha\vdash \mathsf{Id}\,x:\alpha$.*
2. If $x:\sigma\vdash \mathsf{Id}\,x:\sigma$ and $x:\tau\vdash \mathsf{Id}\,x:\tau$, then $x:\sigma\cap\tau\vdash \mathsf{Id}\,x:\sigma\cap\tau$.

Proof. Easy.

Lemma 6. $x:\bigcap_{i\in I}\alpha_i\vdash \mathsf{P}x:\bigcap_{j\in J}\beta_j$ *iff* $\forall j\in J.\ \exists i_j\in I.\ x:\alpha_{i_j}\vdash \mathsf{P}x:\beta_j$.

Proof. The right-to-left direction easily follows by application of the $(\cap E)$ rule and then of the $(\cap I)$ rule. For the left-to-right direction the proof is given in Table 2, where $\mathsf{P}\ {}_\beta\!\longleftarrow \lambda y z_1\ldots z_n.y(\mathsf{P}_1 z_{\pi(1)})\ldots(\mathsf{P}_n z_{\pi(n)})$.

5 Isomorphism Characterisation

Having established an isomorphism-preserving reduction in Section 3, we can now restrict ourselves to normal types, for which we show that the similarity relation is a (sound and complete) characterization of isomorphism.

If we only consider normal types, we can strengthen the Lemma 6 by Lemma 8, which states that if a f.h.p. P has the type $\bigcap_{i\in I}\alpha_i \mapsto \bigcap_{j\in J}\beta_j$, then not only $\forall j\in J\,.\,\exists i_j\in I.\vdash \mathsf{P}\colon \alpha_{i_j}\to\beta_j$, but its inverse P^{-1} precisely maps each component β_j of the target intersection to its corresponding α_{i_j} in the source intersection. This is the key lemma that allows us to prove the main theorem, which states the coincidence between the two relations $\sim$ and $\approx$ for normal types.

Lemma 7 is instrumental to the proof of Lemma 8, and expresses the fact that in an intersection in normal form there are no redundant components, i.e., there cannot exist an η-expansion of the identity that "adds" one of the conjunct types starting from the others.

Lemma 7. *If $\tau\cap\alpha$ is normal, then there is no* Id *such that $x\colon\tau\vdash\mathsf{Id}\,x\colon\tau\cap\alpha$.*

Proof. Let $\tau=\bigcap_{i\in I}\alpha_i$. Towards a contradiction assume $x:\tau\vdash\mathsf{Id}\,x:\tau\cap\alpha$. Since $x\colon\tau\cap\alpha\vdash x\colon\tau$ we get $\tau\cap\alpha\rightsquigarrow\tau$.

Lemma 8. *If $\bigcap_{j\in J}\beta_j$ is a normal type, and $x\colon\bigcap_{i\in I}\alpha_i\vdash\mathsf{P}x\colon\bigcap_{j\in J}\beta_j$, and $x\colon\bigcap_{j\in J}\beta_j\vdash\mathsf{P}^{-1}x\colon\bigcap_{i\in I}\alpha_i$, and $x\colon\alpha_{i_0}\vdash\mathsf{P}x\colon\beta_{j_0}$, then $x\colon\beta_{j_0}\vdash\mathsf{P}^{-1}x\colon\alpha_{i_0}$.*

Proof. By Lemma 6 there is $j_1\in J$ such that $x\colon\beta_{j_1}\vdash\mathsf{P}^{-1}x\colon\alpha_{i_0}$. We assume $j_0\neq j_1$ towards a contradiction. From $x:\beta_{j_1}\vdash\mathsf{P}^{-1}x:\alpha_{i_0}$ and $x:\alpha_{i_0}\vdash\mathsf{P}x:\beta_{j_0}$ we get $x:\beta_{j_1}\vdash\mathsf{P}(\mathsf{P}^{-1}x):\beta_{j_0}$, which implies $x:\bigcap_{j\in J,j\neq j_0}\beta_j\vdash(\mathsf{P}\circ\mathsf{P}^{-1})x:\bigcap_{j\in J}\beta_j$ by Lemma 5. This is, by Lemma 7, impossible, since $\mathsf{P}\circ\mathsf{P}^{-1}$ is β-reducible to a f.h.i.

Theorem 7 (Soundness of $\sim$). *If σ and τ are arbitrary types, then $\sigma\sim\tau$ implies $\sigma\approx\tau$.*

Proof. By induction on the definition of $\sim$ we show that $\langle\sigma_1,\dots,\sigma_m\rangle\sim\langle\tau_1,\dots,\tau_m\rangle$ implies that there is a f.h.p. P such that $\vdash\mathsf{P}:\sigma_j\mapsto\tau_j$ for $1\le j\le m$.
The only interesting case is

$$\langle\sigma_1,\dots,\sigma_m\rangle=\langle\sigma_1^{(1)}\to\dots\to\sigma_n^{(1)}\to\alpha^{(1)},\dots,\sigma_1^{(m)}\to\dots\to\sigma_n^{(m)}\to\alpha^{(m)}\rangle$$
$$\langle\tau_1,\dots,\tau_m\rangle=\langle\tau_{\pi(1)}^{(1)}\to\dots\to\tau_{\pi(n)}^{(1)}\to\alpha^{(1)},\dots,\tau_{\pi(1)}^{(m)}\to\dots\to\tau_{\pi(n)}^{(m)}\to\alpha^{(m)}\rangle,$$

since $\langle\sigma_i^{(1)},\dots,\sigma_i^{(m)}\rangle\sim\langle\tau_i^{(1)},\dots,\tau_i^{(m)}\rangle$ for $1\le i\le n$.
By induction, there is a P_i such that $\vdash\mathsf{P}_i:\sigma_i^{(j)}\mapsto\tau_i^{(j)}$ for $1\le j\le m$. We can then choose P as the β-normal form of $\lambda yz_1\dots z_n.y(\mathsf{P}_1z_{\pi^{-1}(1)})\dots(\mathsf{P}_nz_{\pi^{-1}(n)})$.

The opposite implication does not hold: two isomorphic types are not necessarily similar. For example, the type $\sigma=((\alpha\cap\beta\to\psi)\to\varphi)\cap((\alpha\to\psi)\cap\chi\to\varphi)$ and its normal form $\gamma=(\alpha\cap\beta\to\psi)\to\varphi$, already considered in Section 3, are isomorphic but not similar, simply because they are intersection types of different arities: γ consists of only one arrow type, while σ is an intersection of two arrow types, though one of them is redundant. On the other hand, the double implication holds for normal types.

Theorem 8 (Main Theorem). *If σ and τ are normal types, then $\sigma \approx \tau$ iff $\sigma \sim \tau$.*

Proof. We have to prove that $\sigma \approx \tau \implies \sigma \sim \tau$ (the opposite implication is established by Theorem 7).

We show by structural induction on P that if $\vdash \mathsf{P} : \sigma_j \mapsto \tau_j$ and $\vdash \mathsf{P}^{-1} : \tau_j \mapsto \sigma_j$ for $1 \leq j \leq m$, then $\langle \sigma_1, \ldots, \sigma_m \rangle \sim \langle \tau_1, \ldots, \tau_m \rangle$.
Let $\sigma_j = \bigcap_{1 \leq i \leq n_j} \alpha_i^{(j)}$ and $\tau_j = \bigcap_{1 \leq i \leq p_j} \beta_i^{(j)}$.
By Lemma 8 we get $n_j = p_j$ and $\vdash \mathsf{P} : \alpha_i^{(j)} \to \beta_i^{(j)}$ and $\vdash \mathsf{P}^{-1} : \beta_i^{(j)} \to \alpha_i^{(j)}$. Let $\mathsf{P} \underset{\beta}{\longleftarrow} \lambda y z_1 \ldots z_n . y(\mathsf{P}_1 z_{\pi(1)}) \ldots (\mathsf{P}_n z_{\pi(n)})$. As in the proof of Lemma 6, we get

$$\alpha_i^{(j)} = \tau_1^{(i,j)} \to \ldots \to \tau_n^{(i,j)} \to \gamma^{(i,j)} \text{ and } \beta_i^{(j)} = \sigma_1^{(i,j)} \to \ldots \to \sigma_n^{(i,j)} \to \gamma^{(i,j)}$$

and $\vdash \mathsf{P}_l : \sigma_{\pi(l)}^{(i,j)} \mapsto \tau_l^{(i,j)}$ and $\vdash \mathsf{P}_l^{-1} : \tau_l^{(i,j)} \mapsto \sigma_{\pi(l)}^{(i,j)}$ for $1 \leq l \leq n$. By induction we have

$$\begin{array}{c} \langle \sigma_{\pi(l)}^{(1,1)}, \ldots, \sigma_{\pi(l)}^{(n_1,1)}, \ldots, \sigma_{\pi(l)}^{(1,m)}, \ldots, \sigma_{\pi(l)}^{(n_m,m)} \rangle \\ \sim \\ \langle \tau_l^{(1,1)}, \ldots, \tau_l^{(n_1,1)}, \ldots, \tau_l^{(1,m)}, \ldots, \tau_l^{(n_m,m)} \rangle \end{array}$$

for $1 \leq l \leq n$, which implies

$$\langle \alpha_1^{(1)}, \ldots, \alpha_{n_1}^{(1)}, \ldots, \alpha_1^{(m)}, \ldots, \alpha_{n_m}^{(m)} \rangle \sim \langle \beta_1^{(1)}, \ldots, \beta_{n_1}^{(1)}, \ldots, \beta_1^{(m)}, \ldots, \beta_{n_m}^{(m)} \rangle$$

and then $\langle \sigma_1, \ldots, \sigma_m \rangle \sim \langle \tau_1, \ldots, \tau_m \rangle$.

Of course, the characterization of isomorphisms immediately extends, via normalization, to all types of our system, as stated by the following corollary of the main theorem.

Theorem 9. *For any two types σ and τ, $\sigma \approx \tau \iff \sigma\downarrow \sim \tau\downarrow$, where $\sigma\downarrow$ and $\tau\downarrow$ are the normal forms respectively of σ and τ.*

Proof. Since a type is isomorphic to its normal form we have that:

1. for the $\Rightarrow$-direction, if $\sigma \approx \tau$, then $\sigma\downarrow \approx \sigma \approx \tau \approx \tau\downarrow$, whence, by the Main Theorem in the $\Rightarrow$-direction, $\sigma\downarrow \sim \tau\downarrow$;
2. for the opposite direction, if $\sigma\downarrow \sim \tau\downarrow$, then by the Main Theorem in the $\Leftarrow$-direction we have $\sigma\downarrow \approx \tau\downarrow$, whence: $\sigma \approx \sigma\downarrow \approx \tau\downarrow \approx \tau$, i.e., $\sigma \approx \tau$.

A prototypal isomorphism checker, directly obtained by the permutation-tree definition of similarity, has been realized in Prolog, and a simple web interface for it is available at the address `http://lambda.di.unito.it/iso/index.html`.

6 How to Normalise Types

The application of the type reduction rule, as defined in Section 3, suffers from combinatorial explosion in the search for the erasable type subexpression α, thus possibly making the normalization impractical. However, the search space can be considerably reduced with a more accurate formulation of the algorithm.

As explained in Section 3, the reduction may only simplify an intersection by erasing a type that is greater – according to the standard semantics – than one of the other

conjuncts. We can then formally introduce a preorder relation on types, whose axioms and rules correspond to the view of "$\to$" as a function space constructor and of "$\cap$" as set intersection:

$$\sigma \leq \sigma \qquad \sigma \leq \tau,\ \tau \leq \rho \ \Rightarrow \sigma \leq \rho$$

$$\sigma \cap \tau \leq \sigma \qquad \sigma \cap \tau \leq \tau$$

$$\sigma \leq \tau,\ \sigma \leq \rho \Rightarrow \sigma \leq \tau \cap \rho$$

$$\sigma \leq \tau,\ \alpha \leq \beta \Rightarrow \tau \to \alpha \leq \sigma \to \beta$$

Then, when reducing a type σ to normal form, the search for a redundant type within σ may be limited to an outermost search for a type α that is greater than a type β in an intersection, followed by the testing whether there exist two f.h.i.'s Id, Id' with the appropriate types, i.e., such that $\vdash \mathsf{Id}\colon |\sigma|_\alpha \mapsto \sigma$ and $\vdash \mathsf{Id}'\colon \sigma \mapsto |\sigma|_\alpha$. This can be performed through a mapping $\mathcal{I}$ which, applied to two types σ and τ, builds the set of all f.h.i.'s Id such that $\vdash \mathsf{Id}\colon \sigma \mapsto \tau$.

$$\mathcal{I}(\sigma_1 \to \ldots \to \sigma_n \to \varphi, \tau_1 \to \ldots \to \tau_m \to \varphi') = \varnothing \quad \text{if } n \neq m \text{ or } \varphi \neq \varphi'$$

$$\begin{aligned}\mathcal{I}(\sigma_1 \to \ldots \to \sigma_n \to \varphi, \tau_1 \to \ldots \to \tau_n \to \varphi) \ = \\ \{\mathsf{Id} \mid \lambda y z_1 \ldots z_p.y(\mathsf{Id}_1 z_1)\ldots(\mathsf{Id}_p z_p) \longrightarrow_\beta \mathsf{Id} \ \&\ \mathsf{Id}_l \in \mathcal{I}(\tau_l, \sigma_l) \text{ for } 1 \leq l \leq p\} \\ \text{if } \sigma_m \neq \tau_m \text{ and } \sigma_q = \tau_q \text{ for } m+1 \leq q \leq n \text{ and } m \leq p \leq n\end{aligned}$$

$$\mathcal{I}(\textstyle\bigcap_{i\in I} \alpha_i, \bigcap_{j\in J} \beta_j) = \{\mathsf{Id} \mid \forall j \in J\ \exists i \in I.\ \mathsf{Id} \in \mathcal{I}(\alpha_i, \beta_j)\}$$

The correctness of the mapping $\mathcal{I}$ easily follows from Lemmas 4 and 6.

7 Conclusions and Future Work

In this paper we have investigated for the first time the type isomorphisms for intersection types, and we have provided, by means of a fine analysis of the invertible terms, a precise characterization of their structure, despite the unexpected fact that isomorphism with intersection types is not a congruence.

Even if the isomorphism relation is decidable, we have shown that it is weaker than type equality in the standard models of intersection types, where arrows are interpreted as sets of functions, and intersections as set intersections; such equality is a congruence, consisting of the equality theory given by the axioms of commutativity, associativity and swap (i.e, the first line and the axioms 1 and 2 of Table 1 with $\times$ replaced by $\cap$) and by the order relation induced by the preorder reported in Section 6. This means that the *universal model for type isomorphisms* is not a standard model of intersection types, while Cartesian Closed Categories build a universal model for the simply typed lambda calculus with surjective pairing and terminal object; the existence of such natural universal model for intersection types is an open question.

Finally, we recall that since types may in general be interpreted – owing to the well-known Curry-Howard correspondence – as propositions in some suitable logic, a characterization of type isomorphisms may immediately become a characterization of strong logical equivalences between propositions. In the case of intersection types, however, this is a problematic issue, since it is well known that intersection is an intensional

operator, with no direct logical counterpart in the Curry-Howard sense. Recently, new kinds of logics have been proposed which give a logical meaning to the intersection operator [3], [12]. It might therefore be interesting to explore the role of intersection type isomorphisms in such contexts.

Acknowledgments. We would like to thank the anonymous referees for their detailed remarks and helpful comments.

References

1. van Bakel, S.: Complete restrictions of the intersection type discipline. Theoretical Computer Science 102(1), 135–163 (1992)
2. Barendregt, H., Coppo, M., Dezani-Ciancaglini, M.: A filter lambda model and the completeness of type assignment. The Journal of Symbolic Logic 48(4), 931–940 (1983)
3. Bono, V., Venneri, B., Bettini, L.: A typed lambda calculus with intersection types. Theoretical Computer Science 398(1-3), 95–113 (2008)
4. Bruce, K., Di Cosmo, R., Longo, G.: Provable isomorphisms of types. Mathematical Structures in Computer Science 2(2), 231–247 (1992)
5. Bruce, K., Longo, G.: Provable isomorphisms and domain equations in models of typed languages. In: Sedgewick, R. (ed.) STOC 1985, pp. 263–272. ACM, New York (1985)
6. Coppo, M., Dezani-Ciancaglini, M.: An extension of the basic functionality theory for the λ-calculus. Notre Dame Journal of Formal Logic 21(4), 685–693 (1980)
7. Dezani-Ciancaglini, M.: Characterization of normal forms possessing an inverse in the $\lambda\beta\eta$-calculus. Theoretical Computer Science 2, 323–337 (1976)
8. Di Cosmo, R.: Second order isomorphic types. A proof theoretic study on second order λ-calculus with surjective pairing and terminal object. Information and Computation, pp. 176–201 (1995)
9. Di Cosmo, R.: A short survey of isomorphisms of types. Mathematical Structures in Computer Science 15, 825–838 (2005)
10. Fiore, M., Di Cosmo, R., Balat, V.: Remarks on isomorphisms in typed lambda calculi with empty and sum types. Annals of Pure and Applied Logic 141(1–2), 35–50 (2006)
11. Laurent, O.: Classical isomorphisms of types. Mathematical Structures in Computer Science 15, 969–1004 (2005)
12. Liquori, L., Ronchi Della Rocca, S.: Intersection types à la Church. Information and Computation 205(9), 1371–1386 (2007)
13. Martin, C.F.: Axiomatic bases for equational theories of natural numbers. Notices of the American Mathematical Society 19(7), 778 (1972)
14. Ronchi Della Rocca, S.: Principal type scheme and unification for intersection type discipline. Theoretical Computer Science 59, 1–29 (1988)
15. Soloviev, S.: A complete axiom system for isomorphism of types in closed categories. In: Voronkov, A. (ed.) LPAR 1993. LNCS, vol. 698, pp. 360–371. Springer, Heidelberg (1993)

Undecidability of Type-Checking in Domain-Free Typed Lambda-Calculi with Existence

Koji Nakazawa[1,⋆], Makoto Tatsuta[2],
Yukiyoshi Kameyama[3], and Hiroshi Nakano[4]

[1] Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
[2] National Institute of Informatics, Japan
[3] Department of Computer Science, University of Tsukuba, Japan
[4] Department of Applied Mathematics and Informatics, Ryukoku University, Japan

Abstract. This paper shows undecidability of type-checking and type-inference problems in domain-free typed lambda-calculi with existential types: a negation and conjunction fragment, and an implicational fragment. These are proved by reducing type-checking and type-inference problems of the domain-free polymorphic typed lambda-calculus to those of the lambda-calculi with existential types by continuation passing style translations.

Keywords: undecidability, existential type, CPS-translation, domain-free type system.

1 Introduction

Existential types correspond to second-order existence in logic by the Curry-Howard isomorphism, so they are a natural notion from the point of view of logic. They have been also actively studied from the point of view of computer science since Mitchell and Plotkin [7] showed that abstract data types are existential types.

Existential types are also important since, together with negation and conjunction, it gives a suitable target calculus for continuation-passing style (CPS) translations. Thielecke showed that the negation ($\neg$) and conjunction ($\wedge$) fragment of a λ-calculus suffices for a CPS calculus [14] as the target of various first-order calculi. Recent studies on CPS translations for polymorphic calculi have shown that the $\neg \wedge \exists$-fragment of λ-calculus is an essence of a target calculus of CPS translations for various systems, such as the polymorphic typed λ-calculus [4], the $\lambda\mu$-calculus [3,5], and delimited continuations. [6] showed that a $\neg \wedge \exists$-fragment is even more suitable as a target calculus of a CPS translation for delimited continuations such as `shift` and `reset` [2].

Domain-free type systems, which are in an intermediate style between Church- and Curry-style, are useful for having the subject reduction property. In domain-free style λ-calculus, the type of a bound variable is not explicit in $\lambda x.M$ as in

⋆ knak@kuis.kyoto-u.ac.jp

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 478–492, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Curry-style, while as in Church-style, terms may contain type information for second-order quantifiers, such as a type abstraction $\lambda X.M$ for $\forall$-introduction rule, and a term $\langle A, M\rangle$ with a witness A for $\exists$-introduction rule. Domain-free type systems are introduced for a study on the $\lambda\mu$-calculus. [9] showed the Curry-style call-by-value $\lambda\mu$-calculus does not enjoy the subject reduction property, and [3] introduced a domain-free $\lambda\mu$-calculus $\lambda_V\mu$ to have the subject reduction. In addition, the $\neg \wedge \exists$-fragment of the domain-free typed λ-calculus works as a target calculus of a CPS translation for $\lambda_V\mu$.

Type-inhabitation (INH) is a problem that asks whether there exists M such that $\vdash M : A$ is derivable for given A. INH corresponds to provability of the formula A. The other properties of typed λ-calculi are decidability of type-checking and type-inference. Type-checking (TC) is a problem that asks whether $\Gamma \vdash M : A$ is derivable for given Γ, M, and A. Type-inference (TI) is a problem that asks whether there exist Γ and A such that $\Gamma \vdash M : A$ is derivable for given M. These three questions are fundamentally important in computer science.

Although λ-calculi with existential types are important as computational systems, their properties have not been studied enough yet. It is only recent that INH in the $\neg \wedge \exists$-fragment was proved to be decidable in [13]. TC and TI in typed λ-calculi with existential types remained unknown until this paper.

This paper proves undecidability of the type-checking and the type-inference problems in domain-free typed λ-calculi with existential types: (1) a $\neg \wedge \exists$-fragment $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, (2) another $\neg \wedge \exists$-fragment $\mathsf{DF}\text{-}\lambda_g^{\neg\wedge\exists}$ with a generalized $\wedge$-elimination rule, and (3) an $\rightarrow\exists$-fragment $\mathsf{DF}\text{-}\lambda^{\rightarrow\exists}$.

Our results show that the system $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ is marginal and interesting, because Tatsuta et al [13] showed the decidability of its INH, while ours shows the undecidability of its TC and TI. So far we know few type systems that have this property.

In order to prove undecidability of TC and TI in $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, $\mathsf{DF}\text{-}\lambda_g^{\neg\wedge\exists}$, and $\mathsf{DF}\text{-}\lambda^{\rightarrow\exists}$, we reduce it to undecidability of TC and TI in the domain-free polymorphic typed λ-calculus $\mathsf{DF}\text{-}\lambda 2$. For $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, we define a negative translation $(\cdot)^\bullet$ from types of $\mathsf{DF}\text{-}\lambda 2$ to types of $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, and a translation $[\![\cdot]\!]$ from terms of $\mathsf{DF}\text{-}\lambda 2$ to terms of $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, which is a variant of call-by-name CPS translations inspired by [4]. We will show that $\Gamma \vdash M : A$ is derivable in $\mathsf{DF}\text{-}\lambda 2$ if and only if $\neg\Gamma^\bullet \vdash [\![M]\!] : \neg A^\bullet$ is derivable in $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$. By this fact, we can reduce TC of $\mathsf{DF}\text{-}\lambda 2$ to that of $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$, which concludes undecidability of TC of $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$.

The key of the proof is as follows. For a term M, a type derivation of $\neg\Gamma^\bullet \vdash [\![M]\!] : \neg A^\bullet$ in $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ may contain a type B which is not any CPS type, where a CPS type is defined as a type of the form $\neg C^\bullet$ for some type C in $\mathsf{DF}\text{-}\lambda 2$. If a derivation contains such a type B, it does not correspond to any derivation in $\mathsf{DF}\text{-}\lambda 2$. However, in fact, we can define a contraction transformation that maps a type to a CPS type so that by the contraction transformation, from any type derivation of $\neg\Gamma^\bullet \vdash [\![M]\!] : \neg A^\bullet$, we can construct another type derivation of the same judgment in which every type is a CPS type. By this we can pull back it into a derivation in $\mathsf{DF}\text{-}\lambda 2$.

Systems	TC	TI	INH
Curry-$\lambda 2$	no[16]	no[16]	no
DF-$\lambda 2$	no[1]	no[1]	
Curry-$\lambda^{\neg\wedge\exists}$	?	?	yes[13]
DF-$\lambda^{\neg\wedge\exists}$	NO	NO	
Curry-$\lambda^{\to\exists}$	?	?	?
DF-$\lambda^{\to\exists}$	NO	NO	

Fig. 1. Decidability of TC, TI and INH

We summarize related results about decidability of TC, TI, and INH in several systems in Figure 1, where DF means domain-free, and NO denotes the main results of this paper.

Section 2 introduces the domain-free typed λ-calculus DF-$\lambda^{\neg\wedge\exists}$ with negation, conjunction and existence. Section 3 gives our main theorem which states undecidability of TC and TI in DF-$\lambda^{\neg\wedge\exists}$. Section 4 proves the main theorem, and applies the proof method to DF-$\lambda_g^{\neg\wedge\exists}$. Section 5 discusses CPS-translations for various systems to show that DF-$\lambda^{\neg\wedge\exists}$ is an essence of a target of CPS translations. Section 6 shows undecidability of TC and TI in a domain-free typed λ-calculus DF-$\lambda^{\to\exists}$ with implication and existence.

2 Typed λ-Calculus with Negation, Conjunction and Existence

In this section, we introduce the negation ($\neg$), conjunction ($\wedge$), and existence ($\exists$) fragment DF-$\lambda^{\neg\wedge\exists}$ of domain-free typed λ-calculus.

Definition 1 (DF-$\lambda^{\neg\wedge\exists}$). *(1) The types (denoted by* A, B,*... , and called* $\neg\wedge\exists$*-types) and the terms (denoted by* M, N,*...) of* DF-$\lambda^{\neg\wedge\exists}$ *are defined by*

$$\mathsf{A} ::= X \mid \bot \mid \neg\mathsf{A} \mid \mathsf{A}\wedge\mathsf{A} \mid \exists X.\mathsf{A},$$
$$M ::= x \mid \lambda x.M \mid \langle M, M\rangle \mid \langle \mathsf{A}, M\rangle \mid MM \mid M\pi_1 \mid M\pi_2 \mid M[Xx.M],$$

where X *and* x *denote a type variable and a term variable, respectively. In the type* $\exists X.\mathsf{A}$*, the variable* X *is bound in* A*. In the term* $\lambda x.M$*, the variable* x *is bound in* M*. In the term* $N[Xx.M]$*, the variables* X *and* x *are bound in* M*. We use* $\equiv$ *to denote syntactic identity modulo renaming of bound variables.*

(2) Γ *denotes a context, which is a finite set of type assignments in the form of* $(x : \mathsf{A})$*. We suppose that if both* $(x : \mathsf{A})$ *and* $(x : \mathsf{B})$ *are in* Γ*,* $\mathsf{A} \equiv \mathsf{B}$ *holds. We write* $\Gamma, x : \mathsf{A}$ *for* $\Gamma \cup \{x : \mathsf{A}\}$*, and* Γ_1, Γ_2 *for* $\Gamma_1 \cup \Gamma_2$*.* $\neg\Gamma$ *is defined as* $\{(x : \neg\mathsf{A}) | (x : \mathsf{A}) \in \Gamma\}$*. The typing rules of* DF-$\lambda^{\neg\wedge\exists}$ *are the following.*

$$\frac{}{\Gamma, x : \mathsf{A} \vdash x : \mathsf{A}}\ (\mathsf{Ax})$$

$$\frac{\Gamma, x : \mathsf{A} \vdash M : \bot}{\Gamma \vdash \lambda x.M : \neg\mathsf{A}}\ (\neg\mathsf{I}) \qquad \frac{\Gamma_1 \vdash M : \neg\mathsf{A} \quad \Gamma_2 \vdash N : \mathsf{A}}{\Gamma_1, \Gamma_2 \vdash MN : \bot}\ (\neg\mathsf{E})$$

$$\frac{\Gamma_1 \vdash M : \mathsf{A} \quad \Gamma_2 \vdash N : \mathsf{B}}{\Gamma_1, \Gamma_2 \vdash \langle M, N\rangle : \mathsf{A} \wedge \mathsf{B}}\ (\wedge\mathsf{I}) \qquad \frac{\Gamma \vdash N : \mathsf{A}[X := \mathsf{B}]}{\Gamma \vdash \langle \mathsf{B}, N\rangle : \exists X.\mathsf{A}}\ (\exists\mathsf{I})$$

$$\frac{\Gamma \vdash M : \mathsf{A}_1 \wedge \mathsf{A}_2}{\Gamma \vdash M\pi_1 : \mathsf{A}_1}\ (\wedge\mathsf{E1}) \qquad \frac{\Gamma \vdash M : \mathsf{A}_1 \wedge \mathsf{A}_2}{\Gamma \vdash M\pi_2 : \mathsf{A}_2}\ (\wedge\mathsf{E2})$$

$$\frac{\Gamma_1 \vdash M : \exists X.\mathsf{A} \quad \Gamma_2, x : \mathsf{A} \vdash N : \mathsf{C}}{\Gamma_1, \Gamma_2 \vdash M[Xx.N] : \mathsf{C}}\ (\exists\mathsf{E})$$

$\mathsf{A}[X := \mathsf{B}]$ *is the ordinary capture-avoiding substitution for types. In the rule* $(\exists\mathsf{E})$*,* Γ_2 *and* C *must not contain* X *freely. We write* $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} M : \mathsf{A}$ *to denote that* $\Gamma \vdash M : \mathsf{A}$ *is derivable by the typing rules above.*

In Section 5, we will show this calculus is useful for a target of CPS translations. In addition, $\lambda^{\neg\wedge\exists}$ represents every function representable in the polymorphic typed λ-calculus, because of a CPS-translation from the polymorphic typed λ-calculus to this calculus.

3 Type-Checking and Type-Inference

Type-inhabitation (INH) is a problem that asks whether there exists M such that $\vdash M : A$ is derivable for given A, which corresponds to provability of the formula A. In [13], INH of $\lambda^{\neg\wedge\exists}$ was proved to be decidable. Moreover, it immediately implies decidability of INH in DF-$\lambda^{\neg\wedge\exists}$.

Type-checking (TC) is a problem that asks whether $\Gamma \vdash M : A$ is derivable for given Γ, M, and A. Type-inference (TI) is a problem that asks whether there exist Γ and A such that $\Gamma \vdash M : A$ is derivable for given M.

Theorem 1. *Type-checking and type-inference of* DF-$\lambda^{\neg\wedge\exists}$ *are undecidable.*

This theorem is proved in the Section 4.

4 Proof of Undecidability of TC and TI in DF-$\lambda^{\neg\wedge\exists}$

This section will prove Theorem 1. The subsection 4.1 will give a definition of a domain-free polymorphic typed λ-calculus DF-$\lambda 2$. The subsection 4.2 will define a CPS translation from that calculus to DF-$\lambda^{\neg\wedge\exists}$. We will also define an inverse CPS translation from the image DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$ of the CPS translation to DF-$\lambda 2$. The subsection 4.3 will show our main lemma, which states that DF-$\lambda^{\neg\wedge\exists}$ is conservative over DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$. The subsection 4.4 will finish our undecidability proof. Our proof method will be applied to a variant DF-$\lambda^{\neg\wedge\exists}_{g}$ with general elimination rules in the subsection 4.5.

4.1 Domain-Free Polymorphic Typed λ-Calculus

In this subsection, we introduce the domain-free variant DF-$\lambda 2$ of the polymorphic typed λ-calculus, for which TC and TI have been already known to be undecidable [1].

Definition 2 (DF-$\lambda 2$). *(1) The types (denoted by A, B,..., and called $\rightarrow\forall$-types), and the terms (denoted by M, N,...) of* DF-$\lambda 2$ *are defined by*

$A ::= X \mid A \rightarrow A \mid \forall X.A$,

$M ::= x \mid \lambda x.M \mid \lambda X.M \mid MM \mid MA$.

(2) The typing rules of DF-$\lambda 2$ *are the following.*

$$\frac{}{\Gamma, x : A \vdash x : A}\ (\mathsf{Ax})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \rightarrow B}\ (\rightarrow\mathsf{I}) \qquad \frac{\Gamma_1 \vdash M : A \rightarrow B \quad \Gamma_2 \vdash N : A}{\Gamma_1, \Gamma_2 \vdash MN : B}\ (\rightarrow\mathsf{E})$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \lambda X.M : \forall X.A}\ (\forall\mathsf{I}) \qquad \frac{\Gamma \vdash M : \forall X.A}{\Gamma \vdash MB : A[X := B]}\ (\forall\mathsf{E})$$

In the rule $(\forall\mathsf{I})$*, the lower sequent must not contain X freely.*

Theorem 2 ([1]). *Type-checking and type-inference of* DF-$\lambda 2$ *are undecidable.*

4.2 CPS Translation

We give a CPS translation for DF-$\lambda 2$ in this subsection. Our translation is inspired by Fujita's translation in [4], but since it is in Church-style, we cannot use it directly for domain-free calculi, and we modified it appropriately.

Definition 3 (CPS Translation). *(1) The negative translation from $\rightarrow\forall$-types to $\neg \wedge \exists$-types is defined by*

$X^\bullet \equiv X$, $(A \rightarrow B)^\bullet \equiv \neg A^\bullet \wedge B^\bullet$, $(\forall X.A)^\bullet \equiv \exists X.A^\bullet$.

$\Gamma^\bullet$ is defined as $\{(x : \mathsf{A}^\bullet) | (x : \mathsf{A}) \in \Gamma\}$.

(2) The CPS translation from terms in DF-$\lambda 2$ *to terms in* DF-$\lambda^{\neg\wedge\exists}$ *is defined by*

$[\![x]\!] \equiv \lambda k.xk$,

$[\![\lambda x.M]\!] \equiv \lambda k.(\lambda x.[\![M]\!](k\pi_2))(k\pi_1)$,

$[\![\lambda X.M]\!] \equiv \lambda k.k[Xk'.[\![M]\!]k']$,

$[\![MN]\!] \equiv \lambda k.[\![M]\!]\langle [\![N]\!], k\rangle$,

$[\![MA]\!] \equiv \lambda k.[\![M]\!]\langle A^\bullet, k\rangle$,

where variables k and k' are supposed to be fresh.

Proposition 1. *$\Gamma \vdash_{\lambda 2} M : A$ implies $\neg\Gamma^\bullet \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!] : \neg A^\bullet$.*

Definition 4 (DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$). *(1) The continuation types (denoted by $\mathcal{A}$, $\mathcal{B}$,...) and the CPS terms (denoted by P, Q,...) are defined as the image of the negative translation and that of the CPS translation, respectively. These are inductively defined by*

$\mathcal{A} ::= X \mid \neg\mathcal{A} \wedge \mathcal{A} \mid \exists X.\mathcal{A}$,

$P ::= \lambda k.xk \mid \lambda k.(\lambda x.P(k\pi_2))(k\pi_1) \mid \lambda k.k[Xk'.Pk'] \mid \lambda k.P\langle P, k\rangle \mid \lambda k.P\langle \mathcal{A}, k\rangle$,

where occurrences of k and k' denote those of the same variable, for example, $\lambda k.xk$ denotes $\lambda k_1.xk_1$ but does not denote $\lambda k_1.xk_2$ for $k_1 \not\equiv k_2$. The CPS types are defined as types of the form $\neg\mathcal{A}$.

(2) We define the subsystem $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$ *of* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ *by restricting terms and types to CPS terms and CPS types, respectively. We write* $\neg\Gamma \vdash_{\mathsf{cps}} P : \neg\mathcal{A}$ *to denote that the judgment is derivable in* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$.

Definition 5 (Inverse CPS Translation). *The inverse translation* $(\cdot)^\circ$ *from continuation types to* $\to\forall$*-types is defined by*

$$X^\circ \equiv X,\ \ (\neg\mathcal{A}\wedge\mathcal{B})^\circ \equiv \mathcal{A}^\circ\to\mathcal{B}^\circ,\ \ (\exists X.\mathcal{A})^\circ \equiv \forall X.\mathcal{A}^\circ.$$

The inverse translation $(\cdot)^\#$ *from CPS terms to terms of* $\mathsf{DF}\text{-}\lambda 2$ *is defined by*

$$(\lambda k.xk)^\# \equiv x,$$
$$(\lambda k.(\lambda x.P(k\pi_2))(k\pi_1))^\# \equiv \lambda x.P^\#,$$
$$(\lambda k.k[Xk'.Pk'])^\# \equiv \lambda X.P^\#,$$
$$(\lambda k.P\langle Q,k\rangle)^\# \equiv P^\#Q^\#,$$
$$(\lambda k.P\langle \mathcal{A},k\rangle)^\# \equiv P^\#\mathcal{A}^\circ.$$

Lemma 1. *(1) For any* $\to\forall$*-type* A, $A^{\bullet}$ *is a continuation type, and* $A^{\bullet\circ} \equiv A$.
(2) For any $\mathsf{DF}\text{-}\lambda 2$*-term* M, $[\![M]\!]$ *is a CPS term, and* $[\![M]\!]^\# \equiv M$ *holds.*

Proposition 2. *(1) If* $\neg\Gamma \vdash_{\mathsf{cps}} P : \neg\mathcal{A}$ *holds, then* $\Gamma^\circ \vdash_{\lambda 2} P^\# : \mathcal{A}^\circ$ *holds.*
(2) If $\neg\Gamma^\bullet \vdash_{\mathsf{cps}} [\![M]\!] : \neg A^\bullet$, *then* $\Gamma \vdash_{\lambda 2} M : A$ *holds.*

Proof. (1) By induction on the derivation.
(2) By (1), we have $\Gamma^{\bullet\circ} \vdash_{\lambda 2} [\![M]\!]^\# : A^{\bullet\circ}$. By Lemma 1, we have the claim. □

4.3 Typing for CPS Terms in $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$

Proposition 1 shows that, for any typable term M in $\mathsf{DF}\text{-}\lambda 2$, $[\![M]\!]$ has a CPS type. In fact, its converse can be also proved. In order to prove that, in this subsection, we will show that $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ is conservative over $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps}}$.

A type derivation of a CPS term in $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ may contain a non CPS type. For example, a CPS term $Q \equiv \lambda k'.xk'$ has an arbitrary negation type $\neg\mathsf{A}$ under a context $\{x : \neg\mathsf{A}\}$, and then $P \equiv \lambda k.(\lambda x.Q(k\pi_2))(k\pi_1)$ has a type $\neg(\neg\mathsf{A}\wedge\mathsf{A})$ as

$$\dfrac{\dfrac{\dfrac{x:\neg\mathsf{A}\vdash Q:\neg\mathsf{A}\quad \dfrac{}{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash k:\neg\mathsf{A}\wedge\mathsf{A}}}{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash k\pi_2:\mathsf{A}}}{k:\neg\mathsf{A}\wedge\mathsf{A},x:\neg\mathsf{A}\vdash Q(k\pi_2):\bot}}{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash\lambda x.Q(k\pi_2):\neg\neg\mathsf{A}}\qquad \dfrac{\dfrac{}{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash k:\neg\mathsf{A}\wedge\mathsf{A}}}{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash k\pi_1:\neg\mathsf{A}}$$
$$\dfrac{k:\neg\mathsf{A}\wedge\mathsf{A}\vdash(\lambda x.Q(k\pi_2))(k\pi_1):\bot}{\vdash\lambda k.(\lambda x.Q(k\pi_2))(k\pi_1):\neg(\neg\mathsf{A}\wedge\mathsf{A})},$$

where the type A may not be a continuation type, for example, A may be $X\wedge Y$. However, as we can see in the example, such a type A cannot be consumed in the type derivation of a CPS term, so we can replace A by any type without changing the form of the derivation. In general, we can define a translation $(\cdot)^{\mathsf{c}}$ from $\neg\wedge\exists$-types to CPS types such that, for any CPS term P and any type derivation of $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} P : \mathsf{A}$, we have $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} P : \mathsf{A}^{\mathsf{c}}$. We call the translation $(\cdot)^{\mathsf{c}}$ the contraction translation. Moreover, we have $(\neg A^\bullet)^{\mathsf{c}} \equiv \neg A^\bullet$.

Definition 6 (Contraction Translation). *Let $\mathcal{S}$ be a fixed closed continuation type, such as $\exists X.X$. The contraction translation $(\cdot)^{\mathsf{c}}$ from $\neg \wedge \exists$-types to CPS types is defined by*

$$(\neg \mathsf{A})^{\mathsf{c}} \equiv \neg \mathsf{A}^{\mathsf{d}}, \qquad \mathsf{A}^{\mathsf{c}} \equiv \neg \mathcal{S} \quad (\mathsf{A} \textit{ is not a negation}),$$

$$X^{\mathsf{d}} \equiv X, \quad \bot^{\mathsf{d}} \equiv \mathcal{S}, \quad (\neg \mathsf{A})^{\mathsf{d}} \equiv \mathcal{S}, \quad (\mathsf{A} \wedge \mathsf{B})^{\mathsf{d}} \equiv \mathsf{A}^{\mathsf{c}} \wedge \mathsf{B}^{\mathsf{d}}, \quad (\exists X.\mathsf{A})^{\mathsf{d}} \equiv \exists X.\mathsf{A}^{\mathsf{d}}.$$

Γ^{c} is defined as $\{(x : \mathsf{A}^{\mathsf{c}}) | (x : \mathsf{A}) \in \Gamma\}$.

Lemma 2. *(1) For any continuation type $\mathcal{A}$, $(\neg \mathcal{A})^{\mathsf{c}} \equiv \neg \mathcal{A}$ and $\mathcal{A}^{\mathsf{d}} \equiv \mathcal{A}$ hold.*

(2) For any continuation type $\mathcal{A}$ and any $\neg \wedge \exists$-type B, $(\mathsf{B}[X := \mathcal{A}])^{\mathsf{c}} \equiv \mathsf{B}^{\mathsf{c}}[X := \mathcal{A}]$ and $(\mathsf{B}[X := \mathcal{A}])^{\mathsf{d}} \equiv \mathsf{B}^{\mathsf{d}}[X := \mathcal{A}]$ hold.

Proof. (1) By induction on $\mathcal{A}$.

(2) By induction on B. Note that any continuation type $\mathcal{A}$ is not a negation, so we have $\mathcal{A}^{\mathsf{c}} \equiv \neg \mathcal{S}$. □

Lemma 3 (Main Lemma). *For a CPS term P, $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} P : A$ implies $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} P : A^{\mathsf{c}}$.*

Proof. By induction on P. Note that any type of P is a negation, since any CPS term is a λ-abstraction. So we will show that $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} P : \neg \mathsf{A}$ implies $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} P : \neg \mathsf{A}^{\mathsf{d}}$.

Case $P \equiv \lambda k.Q\langle R, k\rangle$. Any derivation of $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} P : \neg \mathsf{A}$ has the following form.

$$\dfrac{\dfrac{\Gamma \vdash Q : \neg(\mathsf{B} \wedge \mathsf{A}) \quad \dfrac{\Gamma \vdash R : \mathsf{B} \quad \overline{k : \mathsf{A} \vdash k : \mathsf{A}}}{\Gamma, k : \mathsf{A} \vdash \langle R, k\rangle : \mathsf{B} \wedge \mathsf{A}}}{\Gamma, k : \mathsf{A} \vdash Q\langle R, k\rangle : \bot}}{\Gamma \vdash \lambda k.Q\langle R, k\rangle : \neg \mathsf{A}}$$

By the induction hypotheses, we have $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} Q : \neg(\mathsf{B}^{\mathsf{c}} \wedge \mathsf{A}^{\mathsf{d}})$ and $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} R : \mathsf{B}^{\mathsf{c}}$, so we have $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} P : \neg \mathsf{A}^{\mathsf{d}}$.

Case $P \equiv \lambda k.Q\langle \mathcal{B}, k\rangle$. Any derivation of $\Gamma \vdash_{\lambda^{\neg\wedge\exists}} P : \neg \mathsf{A}$ has the following form, where A must be $\mathsf{C}[X := \mathcal{B}]$.

$$\dfrac{\dfrac{\Gamma \vdash Q : \neg\exists X.\mathsf{C} \quad \dfrac{\overline{k : \mathsf{A} \vdash k : \mathsf{A}}}{k : \mathsf{A} \vdash \langle \mathcal{B}, k\rangle : \exists X.\mathsf{C}}}{\Gamma, k : \mathsf{A} \vdash Q\langle \mathcal{B}, k\rangle : \bot}}{\Gamma \vdash \lambda k.Q\langle \mathcal{B}, k\rangle : \neg \mathsf{A}}$$

By the induction hypothesis, $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} Q : \neg\exists X.\mathsf{C}^{\mathsf{d}}$ holds, so we have $\Gamma^{\mathsf{c}} \vdash_{\mathsf{cps}} P : \neg \mathsf{C}^{\mathsf{d}}[X := \mathcal{B}]$ by letting $k : \mathsf{C}^{\mathsf{d}}[X := \mathcal{B}]$, where $\mathsf{C}^{\mathsf{d}}[X := \mathcal{B}]$ is identical to $(\mathsf{C}[X := \mathcal{B}])^{\mathsf{d}}$ by Lemma 2 (2).

Other cases are similarly proved. □

4.4 Proof of Undecidability

By the main lemma, we can reduce TC and TI of DF-$\lambda 2$ to those of DF-$\lambda^{\neg\wedge\exists}$, and then conclude undecidability of TC and TI in DF-$\lambda^{\neg\wedge\exists}$.

Proposition 3. *(1) $\Gamma \vdash_{\lambda 2} M : A$ holds if and only if $\neg\Gamma^\bullet \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!] : \neg A^\bullet$ holds.*

(2) For any DF-$\lambda 2$-term M, $\Gamma \vdash_{\lambda 2} M : A$ holds for some Γ and A if and only if $\Gamma' \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!] : \mathsf{A}'$ holds for some Γ' and A'.

Proof. (1) The only-if part is Proposition 1, so we will show the if part. If $\neg\Gamma^\bullet \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!] : \neg A^\bullet$ holds, by Lemma 3, we have $(\neg\Gamma^\bullet)^{\mathsf{c}} \vdash_{\mathsf{cps}} [\![M]\!] : (\neg A^\bullet)^{\mathsf{c}}$, from which $\neg\Gamma^\bullet \vdash_{\mathsf{cps}} [\![M]\!] : \neg A^\bullet$ follows by Lemma 2 (1). By Proposition 2 (2), $\Gamma \vdash_{\lambda 2} M : A$ holds.

(2) The only-if part follows from the only-if part of (1). The if part follows from Lemma 3 and Proposition 2 (2). □

Proof of Theorem 1. Undecidability of TC and TI in DF-$\lambda^{\neg\wedge\exists}$ are proved by Proposition 3 and Theorem 2. □

4.5 TC and TI of DF-$\lambda_g^{\neg\wedge\exists}$ Are Undecidable

The discussion for DF-$\lambda^{\neg\wedge\exists}$ in the previous subsections can be applied to a variant DF-$\lambda_g^{\neg\wedge\exists}$ with general elimination rules by defining a suitable CPS translation from DF-$\lambda 2$ to DF-$\lambda_g^{\neg\wedge\exists}$.

Definition 7 (DF-$\lambda_g^{\neg\wedge\exists}$)**.** *The terms of DF-$\lambda_g^{\neg\wedge\exists}$ are defined by*
$M ::= x \mid \lambda x.M \mid \langle M, M\rangle \mid \langle \mathsf{A}, M\rangle \mid MM \mid M[xx.M] \mid M[Xx.M]$. *The typing rules of DF-$\lambda_g^{\neg\wedge\exists}$ are the same as DF-$\lambda^{\neg\wedge\exists}$ except for replacing* ($\wedge$E1) *and* ($\wedge$E2) *by the following rule.*

$$\frac{\Gamma_1 \vdash M : \mathsf{A} \wedge \mathsf{B} \quad \Gamma_2, x : \mathsf{A}, y : \mathsf{B} \vdash N : \mathsf{C}}{\Gamma_1, \Gamma_2 \vdash M[xy.N] : \mathsf{C}}\ (\wedge\mathsf{E})$$

$\Gamma \vdash_{\lambda_g^{\neg\wedge\exists}} M : \mathsf{A}$ is defined similarly to that in DF-$\lambda^{\neg\wedge\exists}$.

Definition 8. *The CPS translation $[\![\cdot]\!]$ of DF-$\lambda_g^{\neg\wedge\exists}$ and its inverse $(\cdot)^\#$ are the same as those of DF-$\lambda^{\neg\wedge\exists}$ except for the cases of λ-abstractions, which are defined by $[\![\lambda x.M]\!] \equiv \lambda k.k[xk'.[\![M]\!]k']$, and $(\lambda k.k[xk'.Pk'])^\# \equiv \lambda x.P^\#$, where the definition of CPS terms is also changed by*
$P ::= \lambda k.xk \mid \lambda k.k[xk'.Pk'] \mid \lambda k.k[Xk'.Pk'] \mid \lambda k.P\langle P, k\rangle \mid \lambda k.P\langle \mathcal{A}, k\rangle$.
$\neg\Gamma \vdash_{\mathsf{g\text{-}cps}} P : \neg\mathcal{A}$ is defined similarly to that in DF-$\lambda^{\neg\wedge\exists}$.

Lemma 4 (Main Lemma). *If P is a CPS term, $\Gamma \vdash_{\lambda_g^{\neg\wedge\exists}} P : \mathsf{A}$ implies $\Gamma^{\mathsf{c}} \vdash_{\mathsf{g\text{-}cps}} P : \mathsf{A}^{\mathsf{c}}$.*

Proposition 4. *(1) $\Gamma \vdash_{\lambda 2} M : A$ holds if and only if $\neg\Gamma^\bullet \vdash_{\lambda_g^{\neg\wedge\exists}} [\![M]\!] : \neg A^\bullet$ holds.*

(2) For any DF-λ2*-term M, $\Gamma \vdash_{\lambda 2} M : A$ holds for some Γ and A if and only if $\Gamma' \vdash_{\lambda_g^{\neg\wedge\exists}} [\![M]\!] : \mathsf{A}'$ holds for some Γ' and A'.*

Theorem 3. *Type-checking and type-inference of* DF-$\lambda_g^{\neg\wedge\exists}$ *are undecidable.*

Proof. By Proposition 4 and Theorem 2. □

5 A Target of CPS Translations

In this section, we discuss that DF-$\lambda^{\neg\wedge\exists}$ is an essence of a target of CPS translations by showing it works well as a CPS target for the call-by-value computational λ-calculus, the call-by-value $\lambda\mu$-calculus, and delimited continuations. At first sight, $\lambda^{\neg\wedge\exists}$ may look weak as a computational system, but it suffices as a target calculus of several CPS translations [4,5]. Moreover, the domain-free style calculus with existence works also as a CPS target of the domain-free call-by-value $\lambda\mu$-calculus $\lambda_V\mu$ [3].

First, we define the reduction relation in DF-$\lambda^{\neg\wedge\exists}$. We omit η-rules, but the results in this section can be extended straightforwardly to η-rules.

Definition 9. *The reduction rules of* DF-$\lambda^{\neg\wedge\exists}$ *are the following.*

$(\beta_\to)$ $\quad (\lambda x.M)N \to M[x := N]$
$(\beta_\wedge)$ $\quad \langle M_1, M_2\rangle\pi_i \to M_i \qquad (i = 1 \text{ or } 2)$
$(\beta_\exists)$ $\quad \langle \mathsf{A}, M\rangle[Xx.N] \to N[X := \mathsf{A}, x := M]$

The relation $\to_{\lambda^{\neg\wedge\exists}}$ is the compatible closure of the above rules, and the relation $\to^*_{\lambda^{\neg\wedge\exists}}$ is its reflexive transitive closure.

5.1 Call-by-Value Second-Order Computational λ-Calculus

In [11], Sabry and Wadler gave a call-by-value CPS translation from the computational λ-calculus λ_c [8] to a CPS calculus λ_{cps}, which is a subsystem of the ordinary λ-calculus. Furthermore, they gave an inverse translation from λ_{cps} to λ_c, and showed that those translations form a reflection of λ_{cps} in λ_c.

DF-$\lambda^{\neg\wedge\exists}$ can be a target of a CPS translation for λ_c with polymorphic types. In this subsection, we define DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ as a subsystem of DF-$\lambda^{\neg\wedge\exists}$, and show that we have a reflection of DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ in λ_c with polymorphic types.

Definition 10 (DF-$\lambda_c^\forall$). *The system* DF-$\lambda_c^\forall$ *is an extension of* DF-λ2 *by adding* `let`*-expressions with the typing rule for them as follows.*

$$\frac{\Gamma_1 \vdash M : A \quad \Gamma_2, x : A \vdash N : B}{\Gamma_1, \Gamma_2 \vdash \texttt{let } x = M \texttt{ in } N : B}\ (\mathsf{let})$$

The values are defined by $V ::= x \mid \lambda x.M \mid \lambda X.M$. We use P, Q,... to denote terms that are not values. The call-by-value reduction is defined by the following rules.

$(\beta.v)$ $(\lambda x.M)V \to M[x := V]$
$(\beta.t)$ $(\lambda X.M)A \to M[X := A]$
$(\beta.\mathtt{let})$ $\mathtt{let}\ x = V\ \mathtt{in}\ M \to M[x := V]$
(ass) $\mathtt{let}\ y = (\mathtt{let}\ x = L\ \mathtt{in}\ M)\ \mathtt{in}\ N \to \mathtt{let}\ x = L\ \mathtt{in}\ (\mathtt{let}\ y = M\ \mathtt{in}\ N)$
$(\mathtt{let}.1)$ $PM \to \mathtt{let}\ x = P\ \mathtt{in}\ xM$
$(\mathtt{let}.2)$ $VP \to \mathtt{let}\ x = P\ \mathtt{in}\ Vx$
$(\mathtt{let}.3)$ $PA \to \mathtt{let}\ x = P\ \mathtt{in}\ xA$

In (ass), N *must not contain* x *freely.*

Definition 11 (DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$)**.** *(1) Let* k *be a fixed term variable. The value types (denoted by* $\mathcal{A}$, $\mathcal{B}$,*...), the terms (denoted by* M, N,*...), the values (denoted by* V, W,*...), and the continuations (denoted by* K,*...) of* DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ *are defined by*

$\mathcal{A} ::= X \mid \neg(\mathcal{A} \wedge \neg\mathcal{A}) \mid \neg\exists X.\neg\mathcal{A}$,
$M ::= KV \mid V\langle V, K\rangle \mid V\langle\mathcal{A}, K\rangle$,
$V ::= x \mid \lambda c.(\lambda x.(\lambda k.M)(c\pi_2))(c\pi_1) \mid \lambda c.c[Xk.M]$,
$K ::= k \mid \lambda x.M$,

where c *is a fresh variable, and occurrences of* c *denote those of the same variable. We write* $\lambda\langle x, k\rangle.M$ *for* $\lambda c.(\lambda x.(\lambda k.M)(c\pi_2))(c\pi_1)$*, and* $\lambda\langle X, k\rangle.M$ *for* $\lambda c.c[Xk.M]$.

(2) The reduction rules of DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ *are the following.*

$(\beta.v)$ $(\lambda\langle x, k\rangle.M)\langle V, K\rangle \to M[x := V, k := K]$
$(\beta.t)$ $(\lambda\langle X, k\rangle.M)\langle\mathcal{A}, K\rangle \to M[x := \mathcal{A}, k := K]$
$(\beta.\mathtt{let})$ $(\lambda x.M)V \to M[x := V]$

DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ is a subsystem of DF-$\lambda^{\neg\wedge\exists}$, and closed under the reduction. The first-order fragment of DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ is isomorphic to λ_{cps} in [11].

Definition 12. *(1) The negative translation* $(\cdot)^{\triangle}$ *from* $\to\forall$*-types to value types and its inverse* $(\cdot)^{\triangledown}$ *are defined by*

$X^{\triangle} \equiv X$, $\quad X^{\triangledown} \equiv X$,
$(A{\to}B)^{\triangle} \equiv \neg(A^{\triangle} \wedge \neg B^{\triangle})$, $\quad (\neg(\mathcal{A} \wedge \neg\mathcal{B}))^{\triangledown} \equiv \mathcal{A}^{\triangledown}{\to}\mathcal{B}^{\triangledown}$,
$(\forall X.A)^{\triangle} \equiv \neg\exists X.\neg A^{\triangle}$, $\quad (\neg\exists X.\neg\mathcal{A})^{\triangledown} \equiv \forall X.\mathcal{A}^{\triangledown}$.

(2) The CPS translation $[\![\cdot]\!]$ *from* DF-$\lambda^{\forall}_{c}$ *to* DF-$\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ *is defined by*

$[\![M]\!] \equiv M : k$,
$\Phi(x) \equiv x$,
$\Phi(\lambda x.M) \equiv \lambda\langle x, k\rangle.[\![M]\!]$,
$\Phi(\lambda X.M) \equiv \lambda\langle X, k\rangle.[\![M]\!]$,

$V : K \equiv K\Phi(V)$,
$VW : K \equiv \Phi(V)\langle\Phi(W), K\rangle$,
$PW : K \equiv P : \lambda m.m\langle\Phi(W), K\rangle$,
$VQ : K \equiv Q : \lambda n.\Phi(V)\langle n, K\rangle$,
$PQ : K \equiv P : \lambda m.(Q : \lambda n.m\langle n, K\rangle)$,
$VA : K \equiv \Phi(V)\langle A^{\triangle}, K\rangle$,
$PA : K \equiv P : \lambda m.m\langle A^{\triangle}, K\rangle$,
$\mathtt{let}\ x = M\ \mathtt{in}\ N : K \equiv M : \lambda x.(N : K)$,

where m *and* n *are fresh variables.*

(3) The inverse translation $(\cdot)^{\#}$ *from* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ *to* $\mathsf{DF}\text{-}\lambda^{\forall}_{c}$ *is defined by*

$$
\begin{array}{ll}
(KV)^{\#} \equiv K^{\flat}[V^{\natural}], & x^{\natural} \equiv x, \\
(V\langle W, K\rangle)^{\#} \equiv K^{\flat}[V^{\natural}W^{\natural}], & (\lambda\langle x, k\rangle.M)^{\natural} \equiv \lambda x.M^{\#}, \\
(V\langle \mathcal{A}, K\rangle)^{\#} \equiv K^{\flat}[\mathcal{A}^{\triangledown}W^{\natural}], & (\lambda\langle X, k\rangle.M)^{\natural} \equiv \lambda X.M^{\#}, \\
k^{\flat} \equiv [], & \\
(\lambda x.M)^{\flat} \equiv \mathtt{let}\ x = [\,]\ \mathtt{in}\ M^{\#}. &
\end{array}
$$

Proposition 5. *(1)* $\Gamma \vdash_{\lambda^{\forall}_{c}} M : A$ *implies* $\Gamma^{\triangle}, k : \neg A^{\triangle} \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!] : \bot$.

(2) $[\![\cdot]\!]$ *and* $(\cdot)^{\#}$ *form a reflection of* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$ *in* $\mathsf{DF}\text{-}\lambda^{\forall}_{c}$*, that is, (a)* $[\![\cdot]\!]$ *and* $(\cdot)^{\#}$ *preserve reduction relation* $\rightarrow^{*}$*, (b)* $[\![M^{\#}]\!] \equiv M$ *holds for any term* M *of* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}_{\mathsf{cps/v}}$*, and (c)* $M \rightarrow^{*} [\![M]\!]^{\#}$ *holds for any term* M *of* $\mathsf{DF}\text{-}\lambda^{\forall}_{c}$.

5.2 Call-by-Value $\lambda\mu$-Calculus

The $\lambda\mu$-calculus was introduced by Parigot in [9] as an extension of λ-calculus, and it corresponds to the classical natural deduction for second-order propositional logic by the Curry-Howard isomorphism. In [3], Fujita pointed out that the Curry-style call-by-value $\lambda\mu$-calculus does not enjoy the subject reduction property, so he introduced a domain-free call-by-value $\lambda\mu$-calculus $\lambda_{V}\mu$ to avoid the problem. In this subsection, we show that $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$ works as a target calculus of a CPS translation for $\lambda_{V}\mu$.

Definition 13 ($\lambda_{V}\mu$)**.** *(1) The system* $\lambda_{V}\mu$ *has a set of another sort of variables called* μ*-variables (denoted by* α*,* β*,...). The types of* $\lambda_{V}\mu$ *are the* $\rightarrow\forall$*-types. The terms (denoted by* M*,* N*,...), and the values (denoted by* V*,* W*,...) of* $\lambda_{V}\mu$ *are defined by*

$M ::= V \mid MM \mid MA \mid \mu\alpha.[\alpha]M$,
$V ::= x \mid \lambda x.M \mid \lambda X.M$.

(2) The typing rules of $\lambda_{V}\mu$ *are the following.*

$$\frac{}{\Gamma, x : A \vdash x : A; \Delta}\ (\mathsf{Ax}) \qquad \frac{\Gamma \vdash M : B; \Delta}{\Gamma \vdash \mu\alpha.[\beta]M : A; (\Delta, \beta : B) - \{\alpha : A\}}\ (\mu)$$

$$\frac{\Gamma, x : A \vdash M : B; \Delta}{\Gamma \vdash \lambda x.M : A{\rightarrow}B; \Delta}\ (\rightarrow\mathsf{I}) \qquad \frac{\Gamma_1 \vdash M : A{\rightarrow}B; \Delta_1 \quad \Gamma_2 \vdash N : A; \Delta_2}{\Gamma_1, \Gamma_2 \vdash MN : B; \Delta_1, \Delta_2}\ (\rightarrow\mathsf{E})$$

$$\frac{\Gamma \vdash M : A; \Delta}{\Gamma \vdash \lambda X.M : \forall X.A; \Delta}\ (\forall\mathsf{I}) \qquad \frac{\Gamma \vdash M : \forall X.A; \Delta}{\Gamma \vdash MB : A[X := B]; \Delta}\ (\forall\mathsf{E})$$

Γ *denotes a context similarly to* $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$*.* Δ *denotes a* μ*-context, which is a finite set of type assignments for* μ*-variables in the form of* $(\alpha : A)$*. In the rule* $(\forall\mathsf{I})$*, the lower sequent must not contain* X *freely.*

(3) The singular contexts are defined by $\mathcal{C} ::= []M \mid V[] \mid []A$*. The term* $\mathcal{C}[M]$ *is obtained from* $\mathcal{C}$ *by replacing* $[]$ *by* M*. The structural substitution* $M[\alpha{\Leftarrow}\mathcal{C}]$ *is obtained from* M *by replacing each subterm* $[\alpha]L$ *by* $[\alpha]\mathcal{C}[L[\alpha{\Leftarrow}\mathcal{C}]]$*. The reduction rules of* $\lambda_{V}\mu$ *are the following.*

$(\beta_{\mathsf{tm}})\quad (\lambda x.M)N \to M[x := N] \qquad (\mu)\quad \mathcal{C}[\mu\alpha.M] \to \mu\alpha.M[\alpha \Leftarrow \mathcal{C}]$
$(\beta_{\mathsf{tp}})\quad (\lambda X.M)A \to M[X := A]$

Definition 14. *The negative translation* $(\cdot)^{\triangle}$ *and the CPS translation* $[\![\cdot]\!]$ *from* $\lambda_V\mu$ *to* DF-$\lambda^{\neg\wedge\exists}$ *are the same as Definition 12, except for replacing the definition for* `let` *by* $\mu\alpha.[\beta]M : K \equiv (M : x_\beta)[x_\alpha := K]$*, where we suppose that* DF-$\lambda^{\neg\wedge\exists}$ *contains a term variable* x_α *for each* μ*-variable* α*.*

Proposition 6. *(1)* $\Gamma \vdash_{\lambda_V\mu} M : A; \Delta$ *implies* $\Gamma^{\triangle}, \neg\Delta^{\triangle}, k\colon \neg A^{\triangle} \vdash_{\lambda^{\neg\wedge\exists}} [\![M]\!]\colon \bot$*.*
(2) $M \to^*_{\lambda_V\mu} N$ *implies* $[\![M]\!] \to^*_{\lambda^{\neg\wedge\exists}} [\![N]\!]$*.*

5.3 Delimited Continuations

The $\neg\wedge\exists$-fragments are also useful as a target of a CPS translation for delimited continuations such as `shift` and `reset` [2]. For calculi with delimited continuations, we consider multi-staged CPS translations, and we need call-by-value calculi as intermediate CPS calculi. However, in order to have a sound CPS translation to an $\to\exists$-fragment, the calculus has to have not only the call-by-value η-reduction, but also the full η-reduction. On the other hand, as it was shown in [6], we can define a sound CPS translation from a calculus with `shift` and `reset` to a call-by-value $\neg\wedge\exists$-fragment without full η-reduction.

6 Undecidability in Implicational Fragment

Our method by means of CPS translations can be used for the domain-free typed λ-calculus DF-$\lambda^{\to\exists}$ with implication and existence. In this section, we define DF-$\lambda^{\to\exists}$ and a CPS translation from DF-$\lambda 2$ to DF-$\lambda^{\to\exists}$, by which TC and TI of DF-$\lambda 2$ are reduced to those of DF-$\lambda^{\to\exists}$.

Definition 15 (DF-$\lambda^{\to\exists}$). *The types (called* $\to\exists$*-types) and the terms of* DF-$\lambda^{\to\exists}$ *are defined by*
$\mathsf{A} ::= X \mid \bot \mid \mathsf{A}\to\mathsf{A} \mid \exists X.\mathsf{A}$,
$M ::= x \mid \lambda x.M \mid \langle \mathsf{A}, M\rangle \mid MM \mid M[Xx.M]$,
We write $\neg\mathsf{A}$ *for* $\mathsf{A}\to\bot$*. The typing rules of* DF-$\lambda^{\to\exists}$ *are* (Ax), $(\exists\mathsf{I})$, $(\exists\mathsf{E})$ *of* DF-$\lambda^{\neg\wedge\exists}$ *and*

$$\frac{\Gamma, x : \mathsf{A} \vdash M : \mathsf{B}}{\Gamma \vdash \lambda x.M : \mathsf{A}\to\mathsf{B}}\ (\to\mathsf{I}), \qquad \frac{\Gamma_1 \vdash M : \mathsf{A}\to\mathsf{B} \quad \Gamma_2 \vdash N : \mathsf{A}}{\Gamma_1, \Gamma_2 \vdash MN : \mathsf{B}}\ (\to\mathsf{E}).$$

Definition 16 (CPS translation). *(1) The negative translation* $(\cdot)^\bullet$ *from* $\to\forall$*-types to* $\to\exists$*-types and its inverse* $(\cdot)^\circ$ *from continuation types to* $\to\forall$*-types are defined by*

$X^\bullet \equiv X$, $\qquad X^\circ \equiv X$,
$(A\to B)^\bullet \equiv \neg(\neg A^\bullet \to \neg B^\bullet)$, $\qquad (\neg(\neg\mathcal{A}\to\neg\mathcal{B}))^\circ \equiv \mathcal{A}^\circ\to\mathcal{B}^\circ$,
$(\forall X.A)^\bullet \equiv \exists X.A^\bullet$, $\qquad (\exists X.\mathcal{A})^\circ \equiv \forall X.\mathcal{A}^\circ$,

where the continuation types are defined by $\mathcal{A} ::= X \mid \neg(\neg\mathcal{A}\to\neg\mathcal{A}) \mid \exists X.\mathcal{A}$. *The CPS types are defined as types of the form* $\neg\mathcal{A}$.

(2) The CPS translation from terms in DF-$\lambda 2$ *to terms in* DF-$\lambda^{\to\exists}$ *and its inverse from CPS terms to terms of* DF-$\lambda 2$ *are defined by*

$$
\begin{array}{ll}
[\![x]\!] \equiv \lambda k.xk, & (\lambda k.xk)^{\#} \equiv x,\\
[\![\lambda x.M]\!] \equiv \lambda k.k(\lambda x.[\![M]\!]), & (\lambda k.k(\lambda x.P))^{\#} \equiv \lambda x.P^{\#},\\
[\![\lambda X.M]\!] \equiv \lambda k.k[Xk'.[\![M]\!]k'], & (\lambda k.k[Xk'.Pk'])^{\#} \equiv \lambda X.P^{\#},\\
[\![MN]\!] \equiv \lambda k.[\![M]\!](\lambda m.m[\![N]\!]k), & (\lambda k.P(\lambda m.mQk))^{\#} \equiv P^{\#}Q^{\#},\\
[\![MA]\!] \equiv \lambda k.[\![M]\!]\langle A^{\bullet}, k\rangle, & (\lambda k.P\langle\mathcal{A}, k\rangle)^{\#} \equiv P^{\#}A^{\circ},
\end{array}
$$

where the CPS terms are defined by

$P ::= \lambda k.xk \mid \lambda k.k(\lambda x.P) \mid \lambda k.k[Xk'.Pk'] \mid \lambda k.P(\lambda m.mPk) \mid \lambda k.P\langle\mathcal{A}, k\rangle$, *where occurrences of* k *and* k' *denote those of the same variable.*

(3) The system $\mathsf{DF}\text{-}\lambda^{\to\exists}_{\mathsf{cps}}$ *is defined as a subsystem of* DF-$\lambda^{\to\exists}$ *by restricting terms and types to CPS terms and CPS types, respectively. We write* $\neg\Gamma \vdash_{\to\exists\mathsf{cps}} P : \neg\mathcal{A}$ *to denote that the judgment is derivable in* $\mathsf{DF}\text{-}\lambda^{\to\exists}_{\mathsf{cps}}$.

Lemma 5. *(1) For any* $\to\forall$*-type* A, $A^{\bullet}$ *is a continuation type, and* $A^{\bullet\circ} \equiv A$ *holds.*

(2) For any DF-$\lambda 2$*-term* M, $[\![M]\!]$ *is a CPS term, and* $[\![M]\!]^{\#} \equiv M$ *holds.*

Proposition 7. *(1)* $\Gamma \vdash_{\lambda 2} M : A$ *implies* $\neg\Gamma^{\bullet} \vdash_{\lambda\to\exists} [\![M]\!] : \neg A^{\bullet}$.

(2) $\neg\Gamma \vdash_{\to\exists\mathsf{cps}} P : \neg\mathcal{A}$ *implies* $\Gamma^{\circ} \vdash_{\lambda 2} P^{\#} : \mathcal{A}^{\circ}$.

(3) $\neg\Gamma^{\bullet} \vdash_{\to\exists\mathsf{cps}} [\![M]\!] : \neg A^{\bullet}$ *implies* $\Gamma \vdash_{\lambda 2} M : A$.

Definition 17 (Contraction Translation). *Let* $\mathcal{S}$ *be a fixed closed continuation type. The contraction translation* $(\cdot)^{\mathsf{c}}$ *from* $\to\exists$*-types to CPS types is defined by*

$(\mathsf{A}\to\mathsf{B})^{\mathsf{c}} \equiv \neg\mathsf{A}^{\mathsf{d}}$,
$\mathsf{A}^{\mathsf{c}} \equiv \neg\mathcal{S}$ *(*A *is not an implication),*
$X^{\mathsf{d}} \equiv X$,
$\bot^{\mathsf{d}} \equiv \mathcal{S}$,
$((\mathsf{A}\to\mathsf{B}\to\mathsf{C})\to\mathsf{D})^{\mathsf{d}} \equiv \neg(\mathsf{A}^{\mathsf{c}}\to\neg\mathsf{B}^{\mathsf{d}})$,
$((\mathsf{A}\to\mathsf{B})\to\mathsf{D})^{\mathsf{d}} \equiv \neg(\mathsf{A}^{\mathsf{c}}\to\neg\mathcal{S})$, *(*$\mathsf{B}$ *is neither an implication nor* $\bot$*),*
$(\mathsf{A}\to\mathsf{D})^{\mathsf{d}} \equiv \mathcal{S}$ *(otherwise),*
$(\exists X.\mathsf{A})^{\mathsf{d}} \equiv \exists X.\mathsf{A}^{\mathsf{d}}$.

Lemma 6. *(1) For any continuation type* $\mathcal{A}$, $(\neg\mathcal{A})^{\mathsf{c}} \equiv \neg\mathcal{A}$ *and* $\mathcal{A}^{\mathsf{d}} \equiv \mathcal{A}$ *hold.*

(2) For any continuation type $\mathcal{A}$ *and any* $\to\exists$*-type* B, $(\mathsf{B}[X := \mathcal{A}])^{\mathsf{c}} \equiv \mathsf{B}^{\mathsf{c}}[X := \mathcal{A}]$ *and* $(\mathsf{B}[X := \mathcal{A}])^{\mathsf{d}} \equiv \mathsf{B}^{\mathsf{d}}[X := \mathcal{A}]$ *hold.*

Proof. (1) is straightforwardly proved by induction. For (2), we use the fact $\mathcal{A}^{\mathsf{c}} \equiv \mathcal{S}$ and $(\mathcal{A}\to\mathsf{B})^{\mathsf{d}} \equiv \mathcal{S}$. □

Lemma 7 (Main Lemma). *If* P *is a CPS term,* $\Gamma \vdash_{\lambda\to\exists} P : \mathsf{A}$ *implies* $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} P : \mathsf{A}^{\mathsf{c}}$.

Proof. By induction on P. Note that any type of a CPS term is an implication, so we will show that $\Gamma \vdash_{\lambda\to\exists} P : \mathsf{A}_1{\to}\mathsf{A}_2$ implies $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} P : \neg\mathsf{A}_1^{\mathsf{d}}$. We will show only non-trivial cases, and other cases are proved similarly to $\mathsf{DF}\text{-}\lambda^{\neg\wedge\exists}$.

Case $P \equiv \lambda k.k(\lambda x.Q)$. Any derivation of $\Gamma \vdash_{\lambda\to\exists} P : \mathsf{A}_1{\to}\mathsf{A}_2$ has the following form, where A_1 must be $(\mathsf{B}_1{\to}\mathsf{B}_2){\to}\mathsf{A}_2$.

$$\dfrac{\dfrac{\dfrac{}{k : \mathsf{A}_1 \vdash k : \mathsf{A}_1} \quad \dfrac{\Gamma, x : \mathsf{B}_1 \vdash Q : \mathsf{B}_2}{\Gamma \vdash \lambda x.Q : \mathsf{B}_1{\to}\mathsf{B}_2}}{\Gamma, k : \mathsf{A}_1 \vdash k(\lambda x.Q) : \mathsf{A}_2}}{\Gamma \vdash \lambda k.k(\lambda x.Q) : \mathsf{A}_1{\to}\mathsf{A}_2}$$

Note that B_2 is an implication since it is a type of a CPS term Q, so we have $\mathsf{A}_1^{\mathsf{d}} \equiv ((\mathsf{B}_1{\to}\mathsf{B}_2){\to}\mathsf{A}_2)^{\mathsf{d}} \equiv \neg(\mathsf{B}_1^{\mathsf{c}}{\to}\mathsf{B}_2^{\mathsf{c}})$ by Definition 17. By the induction hypothesis, we have $\Gamma^{\mathsf{c}}, x : \mathsf{B}_1^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} Q : \mathsf{B}_2^{\mathsf{c}}$, so $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} P : \neg\neg(\mathsf{B}_1^{\mathsf{c}}{\to}\mathsf{B}_2^{\mathsf{c}})$.

Case $P \equiv \lambda k.Q(\lambda m.mRk)$. Any derivation of $\Gamma \vdash_{\lambda\to\exists} P : \mathsf{A}_1{\to}\mathsf{A}_2$ has the following form, where B must be $(\mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D}){\to}\mathsf{D}$.

$$\dfrac{\Gamma \vdash Q : \mathsf{B}{\to}\mathsf{A}_2 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{m : \mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D} \vdash m : \mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D}} \quad \Gamma \vdash R : C}{\Gamma, m : \mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D} \vdash mR : \mathsf{A}_1{\to}\mathsf{D}} \quad \dfrac{}{k : \mathsf{A}_1 \vdash k : \mathsf{A}_1}}{\Gamma, k : \mathsf{A}_1, m : \mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D} \vdash mRk : \mathsf{D}}}{\Gamma, k : \mathsf{A}_1 \vdash \lambda m.mRk : (\mathsf{C}{\to}\mathsf{A}_1{\to}\mathsf{D}){\to}\mathsf{D}}}{\Gamma, k : \mathsf{A}_1 \vdash Q(\lambda m.mRk) : \mathsf{A}_2}}{\Gamma \vdash \lambda k.Q(\lambda m.mRk) : \mathsf{A}_1{\to}\mathsf{A}_2}$$

By the induction hypotheses, we have $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} Q : \neg\mathsf{B}^{\mathsf{d}}$ and $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} R : \mathsf{C}^{\mathsf{c}}$, where B^{d} is identical to $\neg(\mathsf{C}^{\mathsf{c}}{\to}\neg\mathsf{A}_1^{\mathsf{d}})$. So we have $\Gamma^{\mathsf{c}} \vdash_{\to\exists\mathsf{cps}} P : \neg\mathsf{A}_1^{\mathsf{d}}$ by letting $k : \mathsf{A}_1^{\mathsf{d}}$ and $m : \mathsf{C}^{\mathsf{c}}{\to}\neg\mathsf{A}_1^{\mathsf{d}}$. □

Proposition 8. *(1) $\Gamma \vdash_{\lambda 2} M : A$ holds if and only if $\neg\Gamma^{\bullet} \vdash_{\lambda\to\exists} [\![M]\!] : \neg A^{\bullet}$ holds.*

(2) For any $\mathsf{DF}\text{-}\lambda 2$-term M, $\Gamma \vdash_{\lambda 2} M : A$ holds for some Γ and A if and only if $\Gamma' \vdash_{\lambda\to\exists} [\![M]\!] : \mathsf{A}'$ holds for some Γ' and A'.

Theorem 4. *Type-checking and type-inference of $\mathsf{DF}\text{-}\lambda^{\to\exists}$ are undecidable.*

7 Concluding Remarks

We can consider the Curry-style system with negation, conjunction, and existence, where the inference rules for $\exists$ are

$$\dfrac{\Gamma \vdash N : \mathsf{A}[X := \mathsf{B}]}{\Gamma \vdash \langle \exists, N\rangle : \exists X.\mathsf{A}}\ (\exists\mathsf{I}), \qquad \dfrac{\Gamma_1 \vdash M : \exists X.\mathsf{A} \quad \Gamma_2, x : \mathsf{A} \vdash N : \mathsf{C}}{\Gamma_1, \Gamma_2 \vdash M[x.N] : \mathsf{C}}\ (\exists\mathsf{E}),$$

where terms do not contain any type information [12]. We could not directly apply our approach to this system. Proving undecidability of TC and TI in this system would be future work.

Acknowledgments. The authors would like to thank Professor Ken-etsu Fujita for helpful comments, and Professor Masahito Hasegawa for a copy of his draft [6]. The first author was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B) 18700008.

References

1. Barthe, G., Sørensen, M.H.: Domain-free pure type systems. J. Functional Programming 10, 412–452 (2000)
2. Danvy, O., Fillinski, A.: Representing Control: a Study of the CPS Translation. Mathematical Structures in Computer Science 2(4), 361–391 (1992)
3. Fujita, K.: Explicitly typed $\lambda\mu$-calculus for polymorphism and call-by-value. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 162–177. Springer, Heidelberg (1999)
4. Fujita, K.: Galois embedding from polymorphic types in to existential types. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 194–208. Springer, Heidelberg (2005)
5. Hasegawa, M.: Relational parametricity and control. Logical Methods in Computer Science 2(3:3), 1–22 (2006)
6. Hasegawa, M.: (unpublished manuscript, 2007)
7. Mitchell, J.C., Plotkin, G.D.: Abstract types have existential type. ACM Transactions on Programming Languages and Systems 10(3), 470–502 (1988)
8. Moggi, E.: Computational lambda-calculus and monads. In: Proceedings of 4th Annual Symposium on Logic in Computer Science (LICS 1989), pp. 14–23 (1989)
9. Parigot, M.: $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In: Voronkov, A. (ed.) LPAR 1992. LNCS, vol. 624, pp. 190–201. Springer, Heidelberg (1992)
10. Plotkin, G.: Call-by-name, call-by-value, and the λ-calculus. Theoretical Computer Science 1, 125–159 (1975)
11. Sabry, A., Wadler, P.: A reflection on call-by-value. ACM Transactions on Programming Languages and Systems 19(6), 916–941 (1997)
12. Tatsuta, M.: Simple saturated sets for disjunction and second-order existential quantification. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 366–380. Springer, Heidelberg (2007)
13. Tatsuta, M., Fujita, K., Hasegawa, R., Nakano, H.: Inhabitance of Existential Types is Decidable in Negation-Product Fragment. In: Proceedings of 2nd International Workshop on Classical Logic and Computation (CLC 2008) (2008)
14. Thielecke, H.: Categorical Structure of Continuation Passing Style. Ph.D. Thesis, University of Edinburgh (1997)
15. van Benthem Jutting, L.S.: Typing in pure type systems. Information and Computation 105, 30–41 (1993)
16. Wells, J.B.: Typability and type checking in the second-order λ-calculus are equivalent and undecidable. In: Proceedings of 9th Symposium on Logic in Computer Science (LICS 1994), pp. 176–185 (1994)

Type-Based Termination with Sized Products

Gilles Barthe[1,⋆], Benjamin Grégoire[2], and Colin Riba[2]

[1] IMDEA Software, Madrid, Spain
gilles.barthe@imdea.org
[2] INRIA Sophia-Antipolis, France
{Benjamin.Gregoire,Colin.Riba}@sophia.inria.fr

Abstract. Type-based termination is a semantically intuitive method that ensures termination of recursive definitions by tracking the size of datatype elements, and by checking that recursive calls operate on smaller arguments. However, many systems using type-based termination rely on a semantical anomaly to guarantee strong normalization; namely, they impose that non-recursive elements of a datatype, e.g. the empty list, have size 1 instead of 0. This semantical anomaly also prevents functions such as quicksort to be given a precise typing.

The main contribution of this paper is a type system that remedies this anomaly, and still ensures termination. In addition, our type system features prenex stage polymorphism, a weakening of existential quantification over stages, and is precise enough to type quicksort as a non-size increasing function. Moreover, our system accomodate stage addition with all positive inductive types.

1 Introduction

Type-based termination is a method to guarantee termination of recursive definitions by a non-standard type system in which datatype elements are assigned a size, which is used by the typing rule for letrec to ensure that recursive calls are made on smaller elements, i.e. elements with a smaller size. The semantical intuition behind size-based termination is embedded in the (simplified) typing rule for recursive definitions, which states that the definition of a function on elements of size $\imath$ can only make recursive calls on elements of smaller size:

$$\frac{\Gamma, f : \mathsf{List}^{\imath}\tau \to \sigma \vdash e : \mathsf{List}^{\widehat{\imath}}\tau \to \sigma}{\Gamma \vdash \mathsf{letrec}\ f = e : \mathsf{List}^{\infty}\tau \to \sigma} \tag{1}$$

where $\imath$ is a size variable, $\mathsf{List}^{\imath}$ denotes the type of lists of size less or equal to $\imath$, and $\widehat{\ }$ is the successor function on stages, $\mathsf{List}^{\widehat{\imath}}$ denotes the type of lists of size less or equal to $\widehat{\imath}$ and List^{∞} denotes the usual type of lists.

One distinguishing feature of type-based termination is its expressiveness. Indeed, even the simplest systems of type-based termination are sufficiently expressive to allow to give precise typings for some structurally recursive functions:

$$\mathsf{map} : (X \to Y) \to \mathsf{List}^{\imath}X \to \mathsf{List}^{\imath}Y$$

⋆ Most of this work was performed while working at INRIA Sophia-Antipolis.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 493–507, 2008.
© Springer-Verlag Berlin Heidelberg 2008

and to type functions that are not structurally recursive such as the quicksort function:

$$\begin{array}{l}\mathsf{letrec}\ qs = \lambda l.\ \mathsf{case}\ l\ \mathsf{of}\\ \qquad\qquad \mid \mathsf{nil}\ \Rightarrow\ \mathsf{nil}\\ \qquad\qquad \mid \mathsf{cons}\ x\ xs\ \Rightarrow\ \mathsf{let}\ \langle z_1, z_2\rangle = (\mathsf{filter}\ x\ xs)\\ \qquad\qquad\qquad\qquad\ \ \mathsf{in}\ \mathsf{app}\ (qs\ z_1)\ (\mathsf{cons}\ x\ (qs\ z_2))\end{array} \tag{2}$$

Many type-based termination systems [1, 2, 3, 7] allow the typing:

$$\mathsf{quicksort} : \mathsf{List}^{\infty} X \to \mathsf{List}^{\infty} X \tag{3}$$

but cannot yield the more precise typing:

$$\mathsf{quicksort} : \mathsf{List}^{\imath} X \to \mathsf{List}^{\imath} X \tag{4}$$

Achieving a precise typing for quicksort requires extending the type system so that it yields precise typings for app and filter: first, app must be given a precise typing by means of *stage addition*:

$$\mathsf{app} : \mathsf{List}^{\imath} X \to \mathsf{List}^{\jmath} X \to \mathsf{List}^{\imath+\jmath} X \tag{5}$$

Second, we have to express that filter divides a list of size $\imath$ into two lists whose respective sizes $\jmath_1$ and $\jmath_2$ sum up to $\imath$ which could be expressed using constrained existential types, as in [4, 10]:

$$\mathsf{filter} : X \to \mathsf{List}^{\imath} X \to \exists \jmath_1, \jmath_2\ (\jmath_1 + \jmath_2 = \imath).\ \mathsf{List}^{\jmath_1} X \times \mathsf{List}^{\jmath_2} X \tag{6}$$

Unfortunately, adding constrained existential quantification over stages may break subject reduction (see Section 2) and leads to complex type systems, where type checking requires solving constraints in Presburger arithmetic.

Furthermore, having nil of size at least $\widehat{0}$ (as in [1, 2, 3, 7]), we cannot type filter as in (6), and this prevents the typing quicksort as in (4). Thus, we must give the size 0 to nil. Alas, using the typing rule for fixpoints of [1, 2, 3, 7], and letting $\mathsf{nil} : \mathsf{List}^{0} X$, leads to typable non-terminating terms: using the typing rule for fixpoints of [1, 2, 3, 7], $(\mathsf{letrec}\ f\, x = f\, \mathsf{nil})\, \mathsf{nil}$ is typable (using the subtyping rule $\mathsf{List}^{0} X \leq \mathsf{List}^{\imath} X$) but not terminating.

Thus, defining a simple yet precise type system that enjoys good meta-theoretical properties is a challenge. The main contribution of this article is the definition of a type system $F_{\mathbb{X}}^{\widehat{}}$ that features a monoidal structure on stages (with zero and addition), that simulates existential quantification over stages, and still enjoys subject reduction and strong normalization for first-order and higher-order inductive types. Technically, we achieve subject reduction for existentials by attaching existential quantification to a container structure: this way, introduction and elimination of existential quantification is linked with introduction and elimination of the corresponding type constructor. This leads to a system which features subject reduction and where eliminations of existential quantification are easier to write for the user. The resulting system provides a well-behaved

intermediate step between basic type-based termination criterion [1, 2, 3, 7], and more powerful but less tractable constraint based approaches [4, 10]. For simplicity, in this paper we focus on a binary product, which we call *sized product*.

To achieve strong normalization, we resort to constraining the form of recursive definitions, requiring that the body of the function immediately performs a case analysis on its recursive argument; this syntactic restriction forces a style of definitions close to rewriting. However, in contrast with rewriting, the body of recursive definitions are still part of the terms. This allows for a more powerful intensional equality between functions than with rewriting. Another feature of $F^{\widehat{}}_{\times}$ is the associativity and commutativity of the addition on the stages of higher-order inductive datatypes. This is possible because, in the model construction for strong normalization, stage addition is interpreted by the natural addition on the ordinals which interpret the stages of higher-order inductive datatypes.

The paper is organized as follows. In Sect. 2 we discuss related works and presents informally the main characteristics of $F^{\widehat{}}_{\times}$. Sect. 3 is devoted to the formal definition of the system, while Sect. 4 and Sect. 5 outline respectively the proofs of subject reduction and strong normalization proofs.

2 Overview and Related Work

The purpose of this section is to present the main characteristics of $F^{\widehat{}}_{\times}$ and its relation with other works on type-based termination. We begin with a brief overview of the works that support precise typings for quicksort, and then explain the main specificities of our work.

There has been a lot of interest in using type systems to guarantee termination or to characterize the complexity of recursive functions, see e.g. [1] for an overview. Most systems share the semantical anomaly of $F^{\widehat{}}$ and we are only aware of three systems in which quicksort can be given its exact typing.

The first system is that of Chin and Khoo [5], which annotates every type with size annotations and infers a formula of Presburger arithmetic that guarantees termination. We believe that their system, while expressive, generates constraints which are too complex to be used in practice. The second system is that of Xi [10], which uses restricted dependent types to ensure termination. The third system is that of Blanqui and Riba [4]. Recursive functions are defined by rewrite rules, and as in $F^{\widehat{}}_{\times}$, having non-recursive constructors of size 0 is not problematic.

We now discuss the two main characteristics of $F^{\widehat{}}_{\times}$: the sized product and the typing of fixpoints.

2.1 The Sized Product

Advanced systems of type-based termination, such as Xi and Blanqui, feature constrained existential and universal stage quantification, respectively written $\exists \imath P.\tau$ and $\forall \imath P.\tau$, where P is a constraint and τ is a type. These systems deal with judgments of the form $K\,;\,\Gamma \vdash e : \tau$ where K is a conjunction of constraints, and their type checking algorithm generate constraints in Presburger arithmetic.

Apart from the inherent complexity of type checking, there are some known difficulties with existential types.

Fully explicit existential types, which are used by Xi [10], enjoy subject reduction but rely on a complex elaboration mechanism for type checking. In contrast, implicit existential types do not satisfy subject reduction. Blanqui and Riba [4] use the elimination rule:

$$(\exists\text{-elim})\ \frac{K\,;\,\Gamma \vdash e : \exists \imath P.\tau \qquad K, P\,;\,\Gamma, x{:}\tau \vdash e' : \sigma}{K\,;\,\Gamma \vdash \mathsf{let}\ x = e\ \mathsf{in}\ e' : \sigma}\ \imath \notin K, \Gamma, \sigma$$

together with the let-reduction rule:

$$\mathsf{let}\ x = e\ \mathsf{in}\ e' \mapsto e'[x := e]$$

Subject reduction fails in this system (Tatsuta's example [9] is easily adapted). Because let-reduction is performed even if the typing of e does not end with an $\exists \imath P.\tau$-introduction, the sharing information given by the second premise of ($\exists$-elim), which is lost by let-reduction, is not regained by the witness information given by $\exists \imath P.\tau$-introductions. Note that the failure of subject reduction is actually not a so big problem in [4] since the constrained type system is used to analyze rewrite rules, while ensuring their termination in a standard type system which features subject reduction.

The above discussion illustrates the difficulties with existential quantification over stages and justifies our choice to focus on a simpler system that partially simulates, but does not have, existential quantification. Indeed, our type system $F^{\widehat{}}_{\times}$ achieves a similar effect using prenex stage quantification and using instead of explicitly existential quantification an embedding of existential quantification inside some specific type constructors. This way, introduction and elimination of existential quantification is linked with introduction and elimination of the corresponding type constructor. This leads to a system which features subject reduction and where eliminations of existential quantification are easier to write for the user. For simplicity, in this paper we focus on a binary product, which we call *sized product*. We explain it with an example. To express that $\mathsf{filter}\ x\ l$ computes a pair of lists whose sizes sum up to the size of l, we write

$$\mathsf{filter} : X \to \mathsf{List}^{\imath} X \to \mathsf{List}\, X \times^{\imath} \mathsf{List}\, X$$

The existential information on $\jmath_1$ and $\jmath_2$ in (6) is expressed using the rule (let), which is inspired by the usual elimination of existential quantification

$$(\mathrm{let})\ \frac{\begin{array}{l} K\,;\,\Gamma \vdash \mathsf{filter}\ x\ l : \mathsf{List}\ X \times^{\imath} \mathsf{List}\ X \\ K,\ \jmath_1 + \jmath_2 \leq \imath\ \ ;\ \ \Gamma,\ z_1{:}\mathsf{List}^{\jmath_1} X,\ z_2{:}\mathsf{List}^{\jmath_2} X \vdash e : \mathsf{List}^{\imath}\ X \end{array}}{K\,;\,\Gamma \vdash \mathsf{let}\ \langle z_1, z_2\rangle = \mathsf{filter}\, x\, l\ \mathsf{in}\ e : \mathsf{List}^{\imath}\ X}\ \jmath_1, \jmath_2 \notin K, \Gamma$$

The sized product allows to combine pair opening and let-reduction, which corresponds to the elimination of the existential information in the rule (let). This leads to the following rewrite rule, which is the key-point for subject reduction in $F^{\widehat{}}_{\times}$:

$$\mathsf{let}\ \langle x_1, x_2\rangle = \langle e_1, e_2\rangle\ \mathsf{in}\ e \quad \mapsto_{\theta} \quad e[x_1 := e_1, x_2 := e_2]$$

2.2 Typing of Fixpoints

In order to reject the non-terminating term in the introduction, we constrain the form of recursive definitions, requiring that the body of the function immediately performs a case analysis on its recursive argument, and distinguishing in the typing rule between recursive and non-recursive constructors. This approach is connected to definitions by rewriting [4], where fixpoints and case analysis are performed in a single definition. To avoid the non-normalizing term $(\mathsf{letrec}\ f\, x = f\, \mathsf{nil})\, \mathsf{nil}$ while having nil of size 0, we ensure that no evaluation strategy of a typable expression of the form $(\mathsf{letrec}\ f = e)\, \mathsf{nil}$ can make recursive calls to $\mathsf{letrec}\ f = e$. The simplest syntactic way to achieve this is to stick fixpoints definitions $\mathsf{letrec}\ f = e$ to case analysis, and to make a distinction between the *recursive* and the *non-recursive* constructors of a datatype d. Intuitively, d can not occur in the type of the arguments of a non-recursive constructors. Then, our fixpoints have the shape

$$\mathsf{letrec}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^{r}} \Rightarrow \boldsymbol{e^{r}}\}$$

where $\boldsymbol{c^{nr}}$ are non-recursive constructors and $\boldsymbol{c^{r}}$ are recursive constructors, and where $\boldsymbol{e^{nr}}$ can not depend on f. Thus, the following term is strongly normalizing:

$$(\mathsf{letrec}\ f\ \mathsf{case}\ \{0 \Rightarrow 1 \mid \mathsf{s} \Rightarrow \lambda x.(f\ 0) + (f\ x)\})\ 0$$

It would be desirable to separate fixpoints from case analysis, as in $F^{\widehat{}}$, but we have been unable to find a strongly normalizing system for the usual syntax.

In contrast, Xi [10] can have $\mathsf{nil} : \mathsf{List}^0\tau$ without disturbing normalization. Instead of (1), fixpoints can be typed as follows:

$$\frac{K\,;\, \Gamma, f : \forall \imath < \jmath.\mathsf{List}^{\imath}\tau \to \sigma \vdash e : \mathsf{List}^{\jmath}\tau \to \sigma}{\Gamma \vdash \mathsf{letrec}\ f = e : \forall \imath.\mathsf{List}^{\imath}\tau \to \sigma}$$

It is possible for Xi to rely on a strict ordering on stages, because he relies on existential quantification to encode $\mathsf{List}^{\infty}\tau$ as $\exists \imath.\mathsf{List}^{\imath}\tau$. In our case, we cannot use such a strict ordering, because $\imath \leq \infty$ but *not* $\imath < \infty$.

3 System $F^{\widehat{}}_{\times}$

In this section, we present the syntax of system $F^{\widehat{}}_{\times}$. We begin by the stages, then define types, terms, reductions and present the typing rules of the system.

3.1 Stages

Stages expression are built from a set $\mathcal{V}_{\mathrm{S}} = \{\imath, \jmath, \kappa, \dots\}$ of stage variables. They use a binary stage addition $+$, a successor operation $\widehat{\cdot}$ and the constants 0, ∞, denoting respectively the least and the greatest stage.

Definition 3.1 (Stages). *The set* $\mathcal{S}$ *of* stage expressions *is given by the abstract syntax:*

$$s, r \ ::= \ \mathcal{V}_{\mathcal{S}} \ \mid \ 0 \ \mid \ \infty \ \mid \ \widehat{s} \ \mid \ s + r$$

The substitution $s[\imath := r]$ *of the stage variable* $\imath$ *for* r *in* s *is defined in the obvious way.*

The system uses inequalities $s \leq r$ on stage expressions. They are derived by judgments of the form $K \vdash s \leq r$, where K is a conjunction of stages inequalities called *stage constraint*. These judgments are defined by the *substage relation*.

Definition 3.2. *A* stage constraint *is a finite set* $K \in \mathcal{K}$ *of* stage inequalities $s \leq r$ *with* $s, r \in \mathcal{S}$. *The* substage relation *is the smallest relation* $K \vdash s \leq r$, *where* $K \in \mathcal{K}$ *and* $s, r \in \mathcal{S}$, *such that*

$$(ax)\frac{}{K, s \leq r \vdash s \leq r} \qquad (refl)\frac{}{K \vdash s \leq s} \qquad (inf)\frac{}{K \vdash 0 \leq s} \qquad (sup)\frac{}{K \vdash s \leq \infty}$$

$$(trans)\frac{K \vdash s \leq r \quad K \vdash r \leq p}{K \vdash s \leq p} \qquad (mon)\frac{K \vdash s \leq r}{K \vdash s + p \leq r + p} \qquad (inj)\frac{K \vdash \widehat{s} \leq \widehat{r}}{K \vdash s \leq r}$$

$$(com)\frac{}{K \vdash s + r \leq r + s} \qquad (assoc)\frac{}{K \vdash s + (r + p) \leq (s + r) + p}$$

$$(succ)\frac{}{K \vdash s + \widehat{r} = \widehat{s + r}} \qquad (zero)\frac{}{K \vdash 0 + s = s}$$

where $\overline{K \vdash s = r}$ *abbreviates the conjunction of* $\overline{K \vdash s \leq r}$ *and* $\overline{K \vdash r \leq s}$.

Note that the rule *(sup)* implies $\widehat{\infty} \leq \infty$. Moreover, we can derive $K \vdash s \leq \widehat{s}$, $K \vdash (s + r) + p \leq s + (r + p)$, $K \vdash s \leq s + r$; and $K \vdash \widehat{s} \leq \widehat{r}$ from $K \vdash s \leq r$.

3.2 Types and Datatypes Declarations

The system $F_{\widehat{\times}}$ is an extension of Church's style System F. In addition to the function space and the second order type quantification $\Pi X.\tau$, sized types are built using the sized product $\_ \times^s \_$ and the *bounded* universal stage quantification $\forall \imath \leq s.\tau$, which binds $\imath$ in τ but *not* in s (so that, by Barendregt convention, we may always assume $\imath \notin s$). The bound s in universal stage quantification is essential for the typing of fixpoints (typing rule (rec)).

We consider three sets of types: erased types $|\tau| \in |\mathcal{T}|$, that do not carry size annotations, sized types $\tau \in \mathcal{T}$, in which stage variables are free, and constrained types $\underline{\tau} \in \underline{\mathcal{T}}$, which are built from sized types using *prenex* universal stage quantification. Erased types are needed because Church's typing imposes types to appear at the term level, while we do not want size annotations to appear in terms because it makes fail subject reduction (see Sect. 2.4 of [3]).

We assume given a set $\mathcal{V}_{\mathcal{T}} = \{X, Y, \dots\}$ of type variables and a set $\mathcal{D}$ of datatype identifiers. Each datatype identifier comes equipped with an arity $\mathsf{ar}(d)$.

Definition 3.3 (Types). *The sets $|\mathcal{T}|$, $\mathcal{T}$ and $\underline{\mathcal{T}}$ of erased types, sized types and constrained types are given by the following abstract syntaxes:*

$$\begin{array}{rcl} |\mathcal{T}| & ::= & \mathcal{V}_{\mathcal{T}} \mid |\mathcal{T}| \to |\mathcal{T}| \mid \Pi\mathcal{V}_{\mathcal{T}}.|\mathcal{T}| \mid \mathcal{D}\,|\boldsymbol{\mathcal{T}}| \mid |\mathcal{T}| \times |\mathcal{T}| \\ \mathcal{T} & ::= & \mathcal{V}_{\mathcal{T}} \mid \mathcal{T} \to \mathcal{T} \mid \Pi\mathcal{V}_{\mathcal{T}}.\mathcal{T} \mid \mathcal{D}^{\mathcal{S}}\,\boldsymbol{\mathcal{T}} \mid \mathcal{D}\,\boldsymbol{\mathcal{T}} \times^{\mathcal{S}} \mathcal{D}\,\boldsymbol{\mathcal{T}} \\ \underline{\mathcal{T}} & ::= & \mathcal{T} \mid \forall\imath \le s.\underline{\mathcal{T}} \end{array}$$

where $\imath \notin s$ and in the clause for datatypes, it is assumed that the length of the vectors $\boldsymbol{\mathcal{T}}$ and $|\boldsymbol{\mathcal{T}}|$ is exactly the arity of the datatype.

Let $|\tau|, |\theta|, |\sigma|, \ldots$ range over erased types, $\tau, \theta, \sigma, \ldots$ range over sized types and $\underline{\tau}, \underline{\theta}, \underline{\sigma}, \ldots$ range over constrained types. We write $\forall\imath.\underline{\tau}$ for $\forall\imath \le \infty.\underline{\tau}$ and $\forall \boldsymbol{\imath} \le \boldsymbol{s}.\underline{\tau}$ for $\forall\imath_1 \le s_1. \cdots \forall\imath_n \le s_n.\underline{\tau}$. Note that every $\underline{\tau} \in \underline{\mathcal{T}}$ can be written $\forall \boldsymbol{\imath} \le \boldsymbol{s}.\tau$ with $\tau \in \mathcal{T}$. Moreover, we denote by $|.|$ the obvious erasure map from $\underline{\mathcal{T}}$ to $|\mathcal{T}|$. A type is *closed* if it contains no free stage variables and no free type variables.

The subtyping relation is inherited from the substage relation. The rules are syntax-directed.

Definition 3.4 (Subtyping). *The* subtyping relation *is the smallest relation $K \vdash \underline{\tau} \sqsubseteq \underline{\sigma}$, where $K \in \mathcal{K}$ and $\underline{\tau}, \underline{\sigma} \in \underline{\mathcal{T}}$, such that*

$$(var)\frac{}{K \vdash X \sqsubseteq X} \qquad (cst)\frac{K \vdash r \le s \quad K, \imath \le r \vdash \underline{\tau} \sqsubseteq \underline{\sigma}}{K \vdash \forall\imath \le s.\underline{\tau} \sqsubseteq \forall\imath \le r.\underline{\sigma}}\ \imath \notin K$$

$$(func)\frac{K \vdash \tau' \sqsubseteq \tau \quad K \vdash \sigma \sqsubseteq \sigma'}{K \vdash \tau \to \sigma \sqsubseteq \tau' \to \sigma'} \qquad (prod)\frac{K \vdash \sigma \sqsubseteq \sigma'}{K \vdash \Pi X.\sigma \sqsubseteq \Pi X.\sigma'}$$

$$(data)\frac{K \vdash s \le r \quad K \vdash \boldsymbol{\tau} \sqsubseteq \boldsymbol{\tau'}}{K \vdash d^s\boldsymbol{\tau} \sqsubseteq d^r\boldsymbol{\tau'}} \qquad (pair)\frac{K \vdash s \le r \quad K \vdash \boldsymbol{\tau} \sqsubseteq \boldsymbol{\sigma} \quad K \vdash \boldsymbol{\tau'} \sqsubseteq \boldsymbol{\sigma'}}{K \vdash d\boldsymbol{\tau} \times^s d'\boldsymbol{\tau'} \sqsubseteq d\boldsymbol{\sigma} \times^r d'\boldsymbol{\sigma'}}$$

Note that the rule (cst) is contravariant wrt. stages inequalities.

Lemma 3.5. *The relation $K \vdash \_ \sqsubseteq \_$ is reflexive and transitive.*

We now turn to datatype declarations. We assume given a fixed set $\mathcal{C}$ of *constructors*, and a function $\mathsf{C} : \mathcal{D} \mapsto \wp(\mathcal{C})$ such that $\mathsf{C}(d) \cap \mathsf{C}(d') = \emptyset$ for every distinct $d, d' \in \mathcal{D}$. Each constructor $c \in \mathsf{C}(d)$ is given an erased type of the form $\Pi\boldsymbol{X}.|\boldsymbol{\theta}| \to d\,\boldsymbol{X}$, where $|\boldsymbol{\theta}|$ is an erased type in which d and $\boldsymbol{X}$ occur only positively [2]. Note that the arity condition on d imposes that $\boldsymbol{X}$ has the same length for all $c \in \mathsf{C}(d)$. We let $\mathsf{C} =_{\text{def}} \bigcup\{\mathsf{C}(d) \mid d \in \mathcal{D}\}$.

Moreover, we distinguish between recursive and non-recursive constructors. This is essential to annotate constructor types and to the reduction and typing rules of fixpoints. Formally, we assume that $\mathsf{C}(d) = \mathsf{C}_{nr}(d) \uplus \mathsf{C}_r(d)$. Then, $c \in \mathsf{C}(d)$ is *recursive* if $c \in \mathsf{C}_r(d)$ and *non-recursive* otherwise. Intuitively, c is non-recursive iff d does not occur in $|\boldsymbol{\theta}|$. For instance, the constructor $\mathsf{nil} : \Pi X.\mathsf{List}\,X$ is non-recursive, while $\mathsf{cons} : \Pi X.X \to \mathsf{List}\,X \to \mathsf{List}\,X$ is recursive. We write c^r (resp. c^{nr}) to denote a recursive (resp. non-recursive) constructor.

Constructor types are annotated as follows: each occurrence of $d' \neq d$ in $|\boldsymbol{\theta}|$ is annotated with ∞, and each occurrence of d in $|\boldsymbol{\theta}|$ is annotated with a stage

variable $\imath$. Then, the constrained type of c is of the form $\forall \imath.\Pi \boldsymbol{X}.\boldsymbol{\theta} \to d^{\widehat{\imath}}\boldsymbol{X}$ if c is recursive and of the form $\forall \imath.\Pi \boldsymbol{X}.\boldsymbol{\theta} \to d^{\imath}\boldsymbol{X}$ otherwise. In particular, we get $\mathsf{nil}\,|\tau| : \mathsf{List}^{\widehat{0}}\tau$ and $\mathsf{cons}\,|\tau|\,a\,l : \mathsf{List}^{\widehat{s}}\tau$ whenever $l : \mathsf{List}^{s}\tau$ and $a : \tau$.

Definition 3.6 (Inductive datatypes)

(i) A signature *is a map $\Sigma : \mathsf{C} \mapsto \underline{\mathcal{T}}$ such that for all $c \in \mathsf{C}(d)$, $\Sigma(c)$ is a closed type of the form $\forall \imath.\Pi \boldsymbol{X}.\boldsymbol{\theta} \to d^{\overline{\imath}}\boldsymbol{X}$ where*

– $\boldsymbol{\theta}$ are sized types on the abstract syntax T^+ (where $d' \neq d$):

$$\begin{array}{rcl} T^+ ::= & \mathcal{V}_{\mathcal{T}} \mid T^- \to T^+ \mid \Pi\mathcal{V}_{\mathcal{T}}.T^+ \mid d'^{\infty}\,\boldsymbol{T}^+ \mid d^{\imath}\,\boldsymbol{X} \\ T^- ::= & \mathcal{V}_{\mathcal{T}} \setminus \boldsymbol{X} \mid T^+ \to T^- \mid \Pi\mathcal{V}_{\mathcal{T}}.T^- \mid d'^{\infty}\,\boldsymbol{T}^- \end{array}$$

– if d occurs in $\boldsymbol{\theta}$ then $c \in \mathsf{C}_r(d)$ and $\overline{\imath} = \widehat{\imath}$; otherwise $c \in \mathsf{C}_{nr}(d)$ and $\overline{\imath} = \imath$. Moreover, let $\mathsf{Inst}(c, s, \boldsymbol{\tau}, \sigma) =_{def} \boldsymbol{\theta}[\boldsymbol{X} := \boldsymbol{\tau}, \imath := s] \to \sigma$.

(ii) Let $d \leq_\Sigma d'$ iff d occurs in $\Sigma(c)$ for some $c \in \mathsf{C}(d')$. Σ is well-formed *if $\leq_\Sigma$ is a partial order whose strict part $<_\Sigma$ is well-founded.*

Note that if $c \in \mathsf{C}(d)$ has type $\forall \imath.\Pi \boldsymbol{X}.\boldsymbol{\theta} \to d^{\overline{\imath}}\boldsymbol{X}$, then $\imath$ is the sole stage variable occurring in $\boldsymbol{\theta}$. Hence, if c is non-recursive, $\mathsf{Inst}(c, s, \boldsymbol{\tau}, \sigma)$ is of the form $\boldsymbol{\theta}[\boldsymbol{X} := \boldsymbol{\tau}] \to \sigma$, and we write it $\mathsf{Inst}(c, \_, \boldsymbol{\tau}, \sigma)$.

Note also that well-formed signatures rule out heterogeneous datatypes, and for simplicity, mutually inductive datatypes also. Besides, the positivity requirement for $d^{\imath}\boldsymbol{X}$ is standard to guarantee strong normalization. Also, the positivity requirement for $\boldsymbol{X}$ is added to guarantee the soundness of the subtyping rule (data) for datatypes, and to avoid considering polarity, as in e.g. [8].

In the remaining of the paper we assume given a well-formed signature Σ.

3.3 Terms and Reductions

Terms are built from variables, abstractions, applications, constructors, case-expressions, pairs, let-expressions and recursive definitions letrec. Recursive definitions come with case analysis for pattern matching, and let-expressions bind *pairs* of variables. We assume given a set $\mathcal{V}_{\mathcal{E}} = \{f, x, y, z, \ldots\}$ of *term variables*.

Definition 3.7. *The set $\mathcal{E}$ of* terms *is given by the syntax (where $|\tau| \in |\mathcal{T}|$):*

$$\begin{array}{rcl} e, e' & ::= & \mathcal{V}_{\mathcal{E}} \mid \lambda x : |\tau|.e \mid e\,e' \mid \Lambda X.e \mid e\,|\tau| \mid c \\ & \mid & \mathsf{case}_{|\tau|}\ e\ \mathsf{of}\ \{\boldsymbol{c} \Rightarrow \boldsymbol{e}\} \\ & \mid & \langle e, e' \rangle \mid \mathsf{let}\ \langle x, x' \rangle = e\ \mathsf{in}\ e' \\ & \mid & \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^{r}} \Rightarrow \boldsymbol{e^{r}}\} \end{array}$$

Free and bound variables are defined as usual with the following proviso: in letrec expressions, the (fixpoint) variable f is bound in the branches $\boldsymbol{e^r}$ for the recursive constructors, but *not* in the branches $\boldsymbol{e^{nr}}$ for the non-recursive ones. Hence, by Barendregt convention, we may assume that $f \notin \boldsymbol{e^{nr}}$. This is important for the typing and reduction rules of letrec. Note that no stage variable occurs in a term $e \in \mathcal{E}$.

Substitutions are maps $\rho : (\mathcal{V}_{\mathcal{E}} \mapsto \mathcal{E}) \uplus (\mathcal{V}_{\mathcal{T}} \mapsto |\mathcal{T}|)$ of finite domain. The capture-avoiding application of ρ to e is denoted $e\rho$, but we may also write $e[\boldsymbol{x} := \rho(\boldsymbol{x}), \boldsymbol{X} := \rho(\boldsymbol{X})]$ when $\mathsf{dom}(\rho) = \boldsymbol{x} \uplus \boldsymbol{X}$. The reductions are as follows.

Definition 3.8 (Reductions). *The relation of $\beta\iota\mu\theta$-reduction $\rightarrow$ is the smallest rewrite relation containing $\mapsto_{\beta\iota\mu\theta}$, where*

$$(\lambda x : |\tau|.e)\ e' \ \mapsto_\beta\ e[x := e'] \qquad (\Lambda X.e)\ |\tau| \ \mapsto_\beta\ e[X := |\tau|]$$

$$\begin{array}{rcl} \mathsf{case}_{|\tau|}\ (c_i\ |\boldsymbol{\sigma}|\ \boldsymbol{a})\ \mathsf{of}\ \{\boldsymbol{c} \Rightarrow \boldsymbol{e}\} & \mapsto_\iota & e_i\ \boldsymbol{a} \\ \mathsf{let}\ \langle x_1, x_2\rangle = \langle e_1, e_2\rangle\ \mathsf{in}\ e & \mapsto_\theta & e[x_1 := e_1, x_2 := e_2] \end{array}$$

$$\begin{array}{rcl} \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^r} \Rightarrow \boldsymbol{e^r}\}\ (c_i^{nr}\ |\boldsymbol{\sigma}|\ \boldsymbol{a}) & \mapsto_\mu & e_i^{nr}\ \boldsymbol{a} \\ \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^r} \Rightarrow \boldsymbol{e^r}\}\ (c_i^{r}\ |\boldsymbol{\sigma}|\ \boldsymbol{a}) & \mapsto_\mu & e_i^{r}[f := e_f]\ \boldsymbol{a} \end{array}$$

with $e_f = \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^r} \Rightarrow \boldsymbol{e^r}\}$.

The rewrite system $\mapsto_{\beta\iota\mu\theta}$ is orthogonal and thus confluent.

3.4 Typing Rules

The typing system is an extension of [3] with sized products and prenex bounded universal stage quantification. Recall that $\underline{\tau} \in \underline{\mathcal{T}}$ denotes a constrained type, ie. a type of the form $\forall \boldsymbol{\imath} \leq \boldsymbol{s}.\tau$ where $\tau \in \mathcal{T}$ is a sized type. The capture-avoiding substitution of τ for X in σ is written $\sigma[X := \tau]$.

Definition 3.9 (Typing). *A* context *is a map $\Gamma : \mathcal{V}_\mathcal{E} \mapsto \underline{\mathcal{T}}$ of finite domain. The* typing relation *is the smallest relation $K\,;\,\Gamma \vdash e : \underline{\tau}$ which is closed by the rules of Fig. 1.*

The positivity condition $\imath\ \mathsf{pos}\ \sigma$ in the rule (rec) is defined in the usual way [2]. Note that the expression $\lambda x : \mathrm{Nat}.x$ has type $\forall \imath.\mathrm{Nat}^\imath \rightarrow \mathrm{Nat}^\imath$.

The rule (rec) combines the usual rule of fixpoints (1) with the rule (case). The first premise line is the typing of the branches corresponding to non-recursive constructors. They can not depend on the fixpoint variable f. The others premises are the branches for the recursive constructors, which can depend on the fixpoint variable f. The intuition of the termination argument is the following. Assume that we typecheck the approximation of f at type $d^{\widehat{\jmath}}\boldsymbol{\tau} \rightarrow \theta[\imath := \widehat{\jmath}]$ and let $c_k^r : \forall \imath.\Pi \boldsymbol{X}.\boldsymbol{\theta} \rightarrow d^{\widehat{\imath}}\boldsymbol{X}$. The branch e_k^r corresponding to c_k^r must be of type $\boldsymbol{\theta}[\boldsymbol{X} := \boldsymbol{\tau}, \imath := \jmath] \rightarrow \theta[\imath := \widehat{\jmath}]$, provided that f is used with type $\forall \imath \leq \jmath.d^\imath\boldsymbol{\tau} \rightarrow \theta$. That is, only strictly less defined approximations of f can be used to type e_k^r.

The μ-reduction of fixpoints takes into account the difference between recursive and non-recursive constructors: the fixpoint variable is only substituted in the recursive branches.

Finally, a crucial point with constrained-based approaches is that the satisfiability of the constraints K in judgments $K\,;\,\Gamma \vdash e : \tau$ must be preserved by typing rules read bottom up. With general constraints systems like [4, 10], satisfiability tests of K during type-checking generate existential constraints. This is manageable when stages are interpreted by natural numbers, but this may not be the case when constraints have to be interpreted by countable ordinals. By restricting to *bounded universal quantifications* $\forall \imath \leq s.\tau$, type checking generates constraints of the form $\imath \leq s$, which are always satisfiable by $[\imath := s]$. As a

(var) $$\frac{}{K\,;\,\Gamma, x{:}\underline{\sigma} \vdash x : \underline{\sigma}}$$

(abs) $$\frac{K\,;\,\Gamma, x{:}\tau \vdash e : \sigma}{K\,;\,\Gamma \vdash \lambda x : |\tau|.e : \tau \to \sigma}$$

(app) $$\frac{K\,;\,\Gamma \vdash e : \tau \to \sigma \qquad K\,;\,\Gamma \vdash e' : \tau}{K\,;\,\Gamma \vdash e\ e' : \sigma}$$

(T-abs) $$\frac{K\,;\,\Gamma \vdash e : \sigma}{K\,;\,\Gamma \vdash \Lambda X.e : \Pi X.\sigma} \quad \text{if } X \notin \Gamma$$

(T-app) $$\frac{K\,;\,\Gamma \vdash e : \Pi X.\sigma}{K\,;\,\Gamma \vdash e\ |\tau| : \sigma[X := \tau]}$$

(S-gen) $$\frac{K, \imath \le s\,;\,\Gamma \vdash e : \underline{\tau}}{K\,;\,\Gamma \vdash e : \forall \imath \le s.\underline{\tau}} \quad \text{if } \imath \notin \Gamma, K, s$$

(S-inst) $$\frac{K\,;\,\Gamma \vdash e : \forall \imath \le s.\underline{\tau} \qquad K \vdash r \le s}{K\,;\,\Gamma \vdash e : \underline{\tau}[\imath := r]}$$

(cons) $$\frac{c \in \mathsf{C}(d) \text{ for some } d}{K\,;\,\Gamma \vdash c : \Sigma(c)}$$

(pair) $$\frac{K\,;\,\Gamma \vdash e_1 : d_1^{s_1}\ \boldsymbol{\tau_1} \qquad K\,;\,\Gamma \vdash e_2 : d_2^{s_2}\ \boldsymbol{\tau_2}}{K\,;\,\Gamma \vdash \langle e_1, e_2\rangle : d_1\ \boldsymbol{\tau_1} \times^{s_1+s_2} d_2\ \boldsymbol{\tau_2}}$$

(let) $$\frac{\begin{array}{l}K\,;\,\Gamma \vdash e : d_1\ \boldsymbol{\tau_1} \times^{s} d_2\ \boldsymbol{\tau_2} \\ K, \imath_1 + \imath_2 \le s\,;\,\Gamma, x_1{:}d_1^{\imath_1}\ \boldsymbol{\tau_1}, x_2{:}d_2^{\imath_2}\ \boldsymbol{\tau_2} \vdash e' : \sigma\end{array}}{K\,;\,\Gamma \vdash \mathsf{let}\ \langle x_1, x_2\rangle = e\ \mathsf{in}\ e' : \sigma}$$
where $\imath_1, \imath_2 \notin \Gamma, K, \sigma, s, \boldsymbol{\tau_1}, \boldsymbol{\tau_2}$

(case) $$\frac{\begin{array}{c}K\,;\,\Gamma \vdash e : \widehat{d^{s}}\boldsymbol{\tau} \\ \mathsf{C}(d) = \{c_1, \ldots, c_n\} \quad K\,;\,\Gamma \vdash e_k : \mathsf{Inst}(c_k, s, \boldsymbol{\tau}, \theta) \quad (1 \le k \le n)\end{array}}{K\,;\,\Gamma \vdash \mathsf{case}_{|\theta|}\ e\ \mathsf{of}\ \{\boldsymbol{c} \Rightarrow \boldsymbol{e}\} : \theta}$$

(rec) $$\frac{\begin{array}{l}\mathsf{C}_{nr}(d) = \{c_1^{nr}, \ldots, c_n^{nr}\} \qquad K\,;\,\Gamma \vdash e_k^{nr} : \mathsf{Inst}(c_k^{nr}, \_, \boldsymbol{\tau}, \theta) \quad (1 \le k \le n) \\ \mathsf{C}_r(d) = \{c_1^r, \ldots, c_m^r\} \\ K\,;\,\Gamma, f{:}\forall \imath \le \jmath.d^{\imath}\boldsymbol{\tau} \to \theta \vdash e_k^r : \mathsf{Inst}(c_k^r, \jmath, \boldsymbol{\tau}, \theta[\imath := \widehat{\jmath}]) \quad (1 \le k \le m)\end{array}}{K\,;\,\Gamma \vdash \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^{r}} \Rightarrow \boldsymbol{e^{r}}\} : \forall \imath.d^{\imath}\boldsymbol{\tau} \to \theta}$$
where $\imath \notin K, \Gamma, \boldsymbol{\tau} \qquad \imath\ \mathsf{pos}\ \theta \qquad \jmath \notin \imath, K, \Gamma, \boldsymbol{\tau}, \theta \qquad |\tau| = d|\boldsymbol{\tau}| \to |\theta|$

(sub) $$\frac{K\,;\,\Gamma \vdash e : \underline{\sigma} \qquad K \vdash \underline{\sigma} \sqsubseteq \underline{\tau}}{K\,;\,\Gamma \vdash e : \underline{\tau}}$$

Fig. 1. Typing rules for $F_{\times}^{\widehat{}}$

consequence, the satisfiability of K in $K\,;\,\Gamma \vdash e : \tau$ is preserved by typing rules, and there is no need of satisfiability tests during type checking. That is why we have a tractable system with stage addition and all positive inductive types.

4 Subject Reduction

The proof of the subject reduction property relies on the usual intermediate properties, namely inversion and substitution.

For inversion of typing, it is convenient to work in an equivalent type system, in which stage quantification is independent from the introduction and elimination rules of term constructs.

Definition 4.1. *Let $F_\times^{\curlywedge\prime}$ be the type system identical to $F_\times^{\curlywedge}$, except for the rules (cons) and (rec) which are replaced with*

$$\textit{(cons')} \qquad \frac{c \in \mathsf{C}(d) \text{ for some } d \qquad \Sigma(c) = \forall \imath.\sigma}{K\,;\,\Gamma \vdash c : \sigma[\imath := s]}$$

$$\textit{(rec')} \qquad \frac{\begin{array}{l}\mathsf{C}_{nr}(d) = \{c_1^{nr}, \ldots, c_n^{nr}\} \qquad K\,;\,\Gamma \vdash e_k^{nr} : \mathsf{Inst}(c_k^{nr}, \_, \boldsymbol{\tau}, \theta) \quad (1 \le k \le n)\\ \mathsf{C}_r(d) = \{c_1^r, \ldots, c_m^r\}\\ K\,;\,\Gamma, f : \forall \imath \le \jmath.d^s\boldsymbol{\tau} \to \theta \vdash e_k^r : \mathsf{Inst}(c_k^r, \jmath, \boldsymbol{\tau}, \theta[\imath := \widehat{\jmath}]) \quad (1 \le k \le m)\end{array}}{\begin{array}{c}K\,;\,\Gamma \vdash \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c^{nr}} \Rightarrow \boldsymbol{e^{nr}} \mid \boldsymbol{c^r} \Rightarrow \boldsymbol{e^r}\} : d^s\boldsymbol{\tau} \to \theta[\imath := s]\\ \textit{where} \quad \imath \notin K, \Gamma, \boldsymbol{\tau} \qquad \imath\ \mathsf{pos}\ \theta \qquad \jmath \notin \imath, K, \Gamma, \boldsymbol{\tau}, \theta \qquad |\tau| = d|\boldsymbol{\tau}| \to |\theta|\end{array}}$$

Lemma 4.2. *$K\,;\,\Gamma \vdash e : \underline{\tau}$ is derivable in $F_\times^{\curlywedge}$ iff it is derivable in $F_\times^{\curlywedge\prime}$.*

Proposition 4.3 (Inversion of stage quantification). *Let $e \notin \mathcal{V}_\mathcal{X}$. In $F_\times^{\curlywedge\prime}$, if $K\,;\,\Gamma \vdash e : \forall \boldsymbol{\imath} \le \boldsymbol{p}.\tau$ then there is a sized type σ such that $K, \boldsymbol{\imath} \le \boldsymbol{p} \vdash \sigma \sqsubseteq \tau$ and $K, \boldsymbol{\imath} \le \boldsymbol{p}\,;\,\Gamma \vdash e : \sigma$ is derivable in a derivation whose last rule is neither (S-gen), (S-inst) nor (sub).*

The main point in using $F_\times^{\curlywedge\prime}$ instead of $F_\times^{\curlywedge}$ in Prop. 4.3 is to ensure that the last rule of the derivation is the rule corresponding to the top symbol of e, when e is not a variable. This leads to the usual inversion properties for typing in $F_\times^{\curlywedge\prime}$, and thus in $F_\times^{\curlywedge}$ using Lem. 4.2.

Theorem 4.4 (Subject reduction). *In $F_\times^{\curlywedge}$,*

$$(K\,;\,\Gamma \vdash e_1 : \underline{\tau} \quad \wedge \quad e_1 \to e_2) \quad \Longrightarrow \quad K\,;\,\Gamma \vdash e_2 : \underline{\tau}$$

5 Strong Normalization

We outline a realizability proof that typable terms are strongly normalizing. We begin by the interpretation of stages, and then turn to the strong normalization proof itself, which relies on Tait's saturated sets.

Stages are interpreted by the ordinals used to build the interpretation of inductive types. While first-order inductive types can be interpreted by induction

on $\mathbb{N}$, higher-order inductive types may require an induction on countable ordinals. Existing systems with stage addition [4, 10] are restricted to first-order inductive types, and stages constraints are formulas of Presburger arithmetic.

We go one step further by allowing at the same time all positive inductive types and stage addition. Stage addition is interpreted by the *natural addition* on countable ordinals. This operation is associative and commutative, in contrast with the usual ordinal addition which is in general *not* commutative.

In the whole section, if f is a map from A to B, $a \in A$ and $b \in B$, then $f(a := b) : A \mapsto B$ maps a to b and is equal to f everywhere else.

5.1 The Stage Model

Stages are interpreted by ordinals below the first uncountable cardinal.

Definition 5.1 (Countable ordinals). *We denote by $(\Omega, \leq_\Omega)$ the well-ordered set of countable ordinals and by $+_\Omega$ the usual ordinal addition on Ω.*

Recall that $\mathbb{N}$ can be seen as a proper subset of Ω and that $+_\Omega$ coincide with the usual addition on $\mathbb{N}$. We want an associative and commutative addition on Ω, but $+_\Omega$ is in general not commutative on Ω. Instead, we use the *natural addition* on ordinals. To define it, we use the well-known fact that every $\alpha \in \Omega$ can be written in *Cantor normal form*,

$$\alpha \quad = \quad c_n.\omega^{\alpha_n} +_\Omega \cdots +_\Omega c_1.\omega^{\alpha_1}$$

where $\alpha_1 <_\Omega \ldots <_\Omega \alpha_n \in \Omega$ and $c_1, \ldots, c_n \in \mathbb{N}$. The *natural addition* $\oplus$ on Ω is then defined as

$$\begin{array}{ll} & (c_n.\omega^{\alpha_n} +_\Omega \cdots +_\Omega c_1.\omega^{\alpha_1}) \ \oplus \ (d_n.\omega^{\alpha_n} +_\Omega \cdots +_\Omega d_1.\omega^{\alpha_1}) \\ =_{\text{def}} & (c_n +_\Omega d_n).\omega^{\alpha_n} +_\Omega \cdots +_\Omega (c_1 +_\Omega d_1).\omega^{\alpha_1} \end{array}$$

Proposition 5.2. *The* natural addition *is associative, commutative and with neutral element* 0*. Moreover, for all $\alpha \in \Omega$, the successor ordinal of α is $\alpha \oplus 1$.*

We are now ready to define our stage model. Each inductive type can be interpreted using an induction up to a countable ordinal (see Prop. 5.9). We can thus interpret ∞ by Ω. This motivates the following definition.

Definition 5.3 (Stage model). *Let $\widehat{\Omega} =_{def} \Omega \cup \{\Omega\}$. For all $\alpha, \beta \in \widehat{\Omega}$, let*

$$\alpha < \beta \ \ \textit{iff} \ \ (\beta = \Omega \ \vee \ \alpha <_\Omega \beta) \qquad \textit{and} \qquad \alpha + \beta =_{def} \begin{cases} \alpha \oplus \beta & \textit{if } \alpha, \beta \in \Omega \\ \Omega & \textit{otherwise} \end{cases}$$

So we have an addition $+$ on stages which is monotone, associative and commutative. Moreover we have $\alpha + \beta < \Omega$ for all $\alpha, \beta < \Omega$.

Definition 5.4 (Interpretation of stages). *A* stage valuation *is a map π from $\mathcal{V}_\mathcal{S}$ to $\widehat{\Omega}$, and is extended to a* stage interpretation *$(\!|.|\!)_\pi : \mathcal{S} \mapsto \widehat{\Omega}$ as follows:*

$$(\!|0|\!)_\pi =_{def} 0 \qquad (\!|\infty|\!)_\pi =_{def} \Omega \qquad (\!|\widehat{s}|\!)_\pi =_{def} (\!|s|\!)_\pi + 1 \qquad (\!|s + r|\!)_\pi =_{def} (\!|s|\!)_\pi + (\!|r|\!)_\pi$$

We let $\pi \models K$ if $(\!|s|\!)_\pi \leq (\!|p|\!)_\pi$ for all $s \leq p \in K$, and let $K \models s \leq r$ if $\pi \models s \leq r$ for all π such that $\pi \models K$. K is satisfiable *if there is π such that $\pi \models K$.*

5.2 Type Interpretation

Let SN be the set of strongly normalizing terms. We interpret types by saturated sets. It is convenient to define them by means of *elimination contexts*:

$$\begin{array}{rcl} E[\,] & ::= & [\,] \mid E[\,]\, e \mid E[\,]\,|\tau| \mid \mathsf{case}_{|\tau|}\ E[\,]\ \mathsf{of}\ \{\boldsymbol{c} \Rightarrow \boldsymbol{e}\} \\ & \mid & \mathsf{let}\ \langle x, x'\rangle = E[\,]\ \mathsf{in}\ e' \mid \mathsf{letrec}_{|\tau|}\ f\ \mathsf{case}\ \{\boldsymbol{c} \Rightarrow \boldsymbol{e}\}\ E[\,] \end{array}$$

Note that the hole $[\,]$ of $E[\,]$ never occurs under a binder. Thus $E[\,]$ can be seen as a term with one occurrence of a special variable $[\,]$.

Let $E[e] \to_{\mathrm{wh}} E[e']$ if $e \mapsto_{\beta\iota\mu\theta} e'$.

Definition 5.5 (Saturated sets)
A set $S \subseteq \mathsf{SN}$ is saturated *($S \in \mathsf{SAT}$) if*

(SAT1) *$E[x] \in S$ for all $E[\,] \in \mathsf{SN}$ and all $x \in \mathcal{V}_{\mathcal{X}}$,*
(SAT2) *if $e \in \mathsf{SN}$ and $e \to_{wh} e'$ for some $e' \in S$ then $e \in S$.*

It is well-known that $\mathsf{SN} \in \mathsf{SAT}$ and that $\bigcap \mathcal{Y}, \bigcup \mathcal{Y} \in \mathsf{SAT}$ for all non-empty $\mathcal{Y} \subseteq \mathsf{SAT}$. Hence, for each $X \subseteq \mathsf{SN}$ there is a smallest saturated set containing X, written $\overline{X}$.

As usual, the *function space* on SAT is given for $X, Y \in \mathsf{SAT}$ by:

$$X \to Y \ =_{\mathrm{def}}\ \{e \mid \forall e'.\ e' \in X \implies e\, e' \in Y\}$$

The interpretation of types is defined in two steps. We first define the interpretation scheme of types, given an interpretation of datatypes. We then define the interpretation of datatypes.

Definition 5.6. *An* interpretation of datatypes *is a family of functions $(I_d)_{d\in\mathcal{D}}$ where $I_d : \mathsf{SAT}^{\mathsf{ar}(d)} \times \widehat{\Omega} \mapsto \mathsf{SAT}$ for each $d \in \mathcal{D}$. Given an interpretation of datatypes I, a stage valuation π and a* type valuation *$\xi : \mathcal{V}_{\mathcal{T}} \mapsto \mathsf{SAT}$, the* type interpretation *$[\![.]\!]^I_{\pi,\xi} : \underline{\mathcal{T}} \mapsto \mathsf{SAT}$ is defined by induction on types as follows*

$$\begin{array}{rcl} [\![\forall \iota \le s.\underline{\tau}]\!]^I_{\pi,\xi} & = & \bigcap\{[\![\underline{\tau}]\!]^I_{\pi(\iota:=\alpha),\xi} \mid \alpha \le (\!|s|\!)_\pi\} \\ [\![X]\!]^I_{\pi,\xi} & = & \xi(X) \\ [\![\tau \to \sigma]\!]^I_{\pi,\xi} & = & [\![\tau]\!]^I_{\pi,\xi} \to [\![\sigma]\!]^I_{\pi,\xi} \\ [\![\Pi X.\tau]\!]^I_{\pi,\xi} & = & \{e \mid \forall |\sigma| \in |\mathcal{T}|,\ \forall S \in \mathsf{SAT},\quad e\ |\sigma| \in [\![\tau]\!]^I_{\pi,\xi(X:=S)}\} \\ [\![d\boldsymbol{\tau} \times^s d'\boldsymbol{\tau'}]\!]^I_{\pi,\xi} & = & \bigcup\{\langle I_d([\![\boldsymbol{\tau}]\!]^I_{\pi,\xi}, \alpha), I_{d'}([\![\boldsymbol{\tau'}]\!]^I_{\pi,\xi}, \alpha')\rangle \mid \alpha + \alpha' \le (\!|s|\!)_\pi\} \\ [\![d^s\boldsymbol{\tau}]\!]^I_{\pi,\xi} & = & I_d([\![\boldsymbol{\tau}]\!]^I_{\pi,\xi}, (\!|s|\!)_\pi) \end{array}$$

where $\langle S_1, S_2\rangle =_{def} \overline{\{\langle e_1, e_2\rangle \mid e_1 \in S_1 \ \wedge\ e_2 \in S_2\}}$ for all $S_1, S_2 \in \mathsf{SAT}$.

Note that unions and intersections are always taken over non-empty sets of saturated sets.

We now define the interpretation of inductive datatypes. Recall that the relation $<_\Sigma$ is assumed to be a well-founded strict partial order (see Def. 3.6). The interpretation $(I_d)_{d\in\mathcal{D}}$ is defined by induction on $<_\Sigma$, and for each $d \in \mathcal{D}$, the map $I_d : \mathsf{SAT}^{\mathsf{ar}(d)} \times \widehat{\Omega} \mapsto \mathsf{SAT}$ is defined by induction on $\widehat{\Omega}$.

Substitution	$(\!\|p[\imath := s]\|\!)_\pi$	$=$	$(\!\|p\|\!)_{\pi(\imath:=(\!\|s\|\!)_\pi)}$
	$[\![\underline{\tau}[\imath := s]]\!]_{\pi,\xi}$	$=$	$[\![\underline{\tau}]\!]_{\pi(\imath:=(\!\|s\|\!)_\pi),\xi}$
	$[\![\underline{\tau}[X := \sigma]]\!]_{\pi,\xi}$	$=$	$[\![\underline{\tau}]\!]_{\pi,\xi(X:=[\![\sigma]\!]_{\pi,\xi})}$
Stage monotony	$\alpha \leq \beta$	$\Rightarrow$	$I_d(\boldsymbol{S},\alpha) \subseteq I_d(\boldsymbol{S},\beta)$
	$\alpha \leq \beta \ \wedge\ \imath \ \mathsf{pos}\ \theta$	$\Rightarrow$	$[\![\theta]\!]_{\pi(\imath:=\alpha),\xi} \subseteq [\![\theta]\!]_{\pi(\imath:=\beta),\xi}$
	$\alpha \leq \beta \ \wedge\ \imath \ \mathsf{neg}\ \theta$	$\Rightarrow$	$[\![\theta]\!]_{\pi(\imath:=\beta),\xi} \subseteq [\![\theta]\!]_{\pi(\imath:=\alpha),\xi}$
Substage soundness	$K \vdash s \leq p$	$\Rightarrow$	$K \models s \leq p$
Subtyping soundness	$K \vdash \underline{\tau} \sqsubseteq \underline{\sigma} \ \wedge\ \pi \models K$	$\Rightarrow$	$[\![\underline{\tau}]\!]_{\pi,\xi} \subseteq [\![\underline{\sigma}]\!]_{\pi,\xi}$

Fig. 2. Properties of the type interpretation

Definition 5.7. *For all $d \in \mathcal{D}$, all $\boldsymbol{S} \in \mathsf{SAT}^{\mathsf{ar}(d)}$ and all $\alpha \in \widehat{\Omega}$, we define $I_d(\boldsymbol{S},\alpha)$, by induction on pairs (d,α) ordered by $(<_\Sigma, <)_{lex}$, as follows:*

$$I_d(\boldsymbol{S},0) = \bigcup\{c^{nr}\,[\![\boldsymbol{\theta}]\!]^I_{\emptyset,\boldsymbol{X}:=\boldsymbol{S}} \mid c^{nr} \in \mathsf{C}_{nr}(d) \wedge \Sigma(c^{nr}) = \forall\imath.\Pi\boldsymbol{X}.\boldsymbol{\theta} \to d^\imath\boldsymbol{X}\}$$

$$I_d(\boldsymbol{S},\alpha\oplus 1) = \bigcup\{c\,[\![\boldsymbol{\theta}]\!]^I_{\imath:=\alpha,\boldsymbol{X}:=\boldsymbol{S}} \mid c \in \mathsf{C}(d) \wedge \Sigma(c) = \forall\imath.\Pi\boldsymbol{X}.\boldsymbol{\theta} \to d^{\overline{\imath}}\boldsymbol{X}\}$$

$$I_d(\boldsymbol{S},\lambda) = \bigcup\{I_d(\boldsymbol{S},\alpha) \mid \alpha < \lambda\} \qquad \text{if } \lambda \text{ is a limit ordinal}$$

where $c\,\boldsymbol{S} =_{def} \overline{\{c\,|\boldsymbol{\tau}|\,\boldsymbol{a} \mid \boldsymbol{a} \in \boldsymbol{S} \ \wedge\ |\boldsymbol{\tau}| \in |\mathcal{T}|\}}$ *for all* $\boldsymbol{S} \in \mathsf{SAT}$.

Note that $I_d(\boldsymbol{S},\alpha\oplus 1)$ only uses $c[\![\boldsymbol{\theta}]\!]^I_{\imath:=\alpha,\boldsymbol{X}:=\boldsymbol{S}}$ with $c \in \mathsf{C}(d)$, which in turn only uses $I_{d'}(\boldsymbol{U},\beta)$ with $(d',\beta)\,(<_\Sigma,<)_{\mathrm{lex}}\,(d,\alpha\oplus 1)$.

Definition 5.8. *Let* $[\![.]\!]_{\pi,\xi} =_{def} [\![.]\!]^I_{\pi,\xi}$.

Fig. 2 collects some essential properties of $(\!|.|\!)_\pi$ and $[\![.]\!]_{\pi,\xi}$. The following proposition states that each inductive datatype can be interpreted by a countable ordinal. This is crucial in order to deal with the rule (cons) in the proof of Thm. 5.10. The key-point is that for every *countable* $S \subseteq \Omega$, there is $\beta \in \Omega$ such that $\alpha < \beta$ for all $\alpha \in S$ [6].

Proposition 5.9. *For all $d \in \mathcal{D}$ and all $\boldsymbol{S} \in \mathsf{SAT}^{\mathsf{ar}(d)}$, there is an ordinal $\alpha < \Omega$ such that $I_d(\boldsymbol{S},\alpha) = I_d(\boldsymbol{S},\beta)$ for all β such that $\alpha \leq \beta \leq \Omega$.*

As usual, soundness is shown by induction on typing derivations. Note that if K is satisfied by π, then every K' occurring in the derivation of $K\,;\,\Gamma \vdash e : \underline{\tau}$ is satisfied by an extension of π. Thus, in contrast to [4], there is no need of tests of the form $K \vdash \exists\imath.P$ in typing derivations.

Theorem 5.10 (Typing soundness). *Given $\pi : \mathcal{V}_\mathcal{S} \mapsto \widehat{\Omega}$, $\xi : \mathcal{V}_\mathcal{T} \mapsto \mathsf{SAT}$ and $\rho : (\mathcal{V}_\mathcal{E} \mapsto \mathcal{E}) \uplus (\mathcal{V}_\mathcal{T} \mapsto |\mathcal{T}|)$, we let $(\pi,\xi,\rho) \models K;\Gamma$ if and only if $\pi \models K$ and $\rho(x) \in [\![\Gamma(x)]\!]_{\pi,\xi}$ for all $x \in \mathsf{dom}(\Gamma)$.*

If $K\,;\,\Gamma \vdash e : \underline{\tau}$, then $e\rho \in [\![\underline{\tau}]\!]_{\pi,\xi}$ for all π,ξ,ρ such that $(\pi,\xi,\rho) \models K;\Gamma$.

We deduce the strong normalization of terms typable with satisfiable K.

Corollary 5.11. *If $K\,;\,\Gamma \vdash e : \underline{\tau}$ with K satisfiable, then $e \in \mathsf{SN}$.*

6 Conclusion

$F^{\widehat{}}_{\times}$ is a variant of $F^{\widehat{}}$ that supports simple yet precise typing by using sized products instead of existential quantification, for which subject reduction is problematic. We have proved strong normalization and subject reduction of $F^{\widehat{}}_{\times}$, and conjecture that type-checking is tractable. On the other hand, size inference seems more difficult than in [3], in particular for precise annotations with addition such as for the function append on lists.

Our main next objective is to extend our results to the Calculus of Inductive Constructions, and to implement type-based termination in Coq. It would also be interesting to study the expressivity of more general forms of sized products, both with general container types instead of cartesian products, and arbitrary binary operators instead of $+$. Moreover, it would be interesting to study the tractability of more liberal subtyping relations for universal stage quantifications. Finally, an outstanding issue is the design of a strongly normalizing type system in which non-recursive constructors can be given the size zero while keeping fixpoint definitions separated from case analysis.

References

1. Abel, A.: Type-Base Termination. A Polymorphic Lambda-Calculus with Sized Higher-Order Types. PhD thesis, LMU University, Munich (2006)
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. Barthe, G., Grégoire, B., Pastawski, F.: Practical Inference for Type-Based Termination in a Polymorphic Setting. In: Urzyczyn, P. (ed.) TLCA 2005. LNCS, vol. 3461, pp. 71–85. Springer, Heidelberg (2005)
4. 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)
5. Chin, W.-N., Khoo, S.-C.: Calculating Sized Types. Higher-Order and Symbolic Computation 14(2–3), 261–300 (2001)
6. Gallier, J.H.: What's So Special About Kruskal's Theorem and the Ordinal Γ_0? A Survey of Some Results in Proof Theory. Annals of Pure and Applied Logic 53(3), 199–260 (1991)
7. Hughes, J., Pareto, L., Sabry, A.: Proving the Correctness of Reactive Systems Using Sized Types. In: Proceedings of POPL 1996, pp. 410–423. ACM, New York (1996)
8. Steffen, M.: Polarized Higher-order Subtyping. PhD thesis, Department of Computer Science, University of Erlangen (1997)
9. Tatsuta, M.: Simple Saturated Sets for Disjunction and Second-Order Existential Quantification. In: Della Rocca, S.R. (ed.) TLCA 2007. LNCS, vol. 4583, pp. 366–380. Springer, Heidelberg (2007)
10. Xi, H.: Dependent Types for Program Termination Verification. Higher-Order and Symbolic Computation 15(1), 91–131 (2002)

The Ackermann Award 2008

J.A. Makowsky and D. Niwinski

Members of EACSL Jury for the Ackermann Award*

The fourth **Ackermann Award** is presented at this CSL'08. This is the second year in which the EACSL Ackermann Award is generously sponsored. Our sponsor for the remaining two years is the worlds leading provider of personal peripherals, Logitech S.A., situated in Romanel, Switzerland[1].

Eligible for the 2008 **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. 2006 and 31.12. 2007. The Jury received 13 nominations for the **Ackermann Award 2008**. The candidates came from 8 different nationalities from Europe, Russia and India, and received their PhDs in 8 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 2008 **only one** award. The 2008 **Ackermann Award** winner is

Krishnendu Chatterjee

for his thesis *Stochastic ω-Regular Games* issued by the University of California at Berkeley, supervised by Prof. Thomas A. Henzinger.

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.

Krishnendu Chatterjee

Citation. Krishnendu Chatterjee receives the *2008 Ackermann Award* of the European Association of Computer Science Logic (EACSL) for his thesis

Stochastic ω-Regular Games.

* We would like to thank T. Henzinger and L. de Alfaro for their help in preparing the citations.

[1] 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`.

M. Kaminski and S. Martini (Eds.): CSL 2008, LNCS 5213, pp. 508–512, 2008.
© Springer-Verlag Berlin Heidelberg 2008

The thesis greatly advances the algorithmics of repeated games, and indicates that the new degrees of freedom, such as stochastic and concurrent moves, need not increase the computational complexity. The results reveal the algorithmic aspect of determinacy of repeated games and enhance the scope of formal methods in verification of reactive systems, where repeated games form one of leading paradigms.

Background of the thesis. Since the pioneering work of Zermelo on determinacy of chess, determinacy of games has been recognized as an interesting mathematical problem. It cannot be taken for granted, even for perfect information games, if the players take liberty to play *ad infinitum* (as noted by Gale and Stewart in 1953). But the fundamental result of Martin (1975) establishes the determinacy of all turn-based perfect information games with Borel winning objectives.

While determinacy assures the existence of a winning strategy for one of the players, the algorithmic solution of a game consists in determining the winner effectively, and computing the strategy. The usefulness of determinacy of infinite games in the design of decision procedures was discovered by Büchi, and then by Gurevich and Harrington (and, independently, A.A.Muchnik) in their celebrated simplified proofs of the Rabin Tree Theorem (in 1982). The algorithmic approach also revealed the role of players' *memory* in the determinacy results, which can be actually finite for games with ω-regular objectives, and is not needed at all for the (essentially equivalent) parity games. The subsequent development of the mathematical theory of verification, synthesis, and control of reactive systems (especially by E.A.Emerson and his collaborators) showed that, in spite of a large variety of possible logical formalisms, the algorithmic content of the main verification problem – model checking – can be reduced to solving some conceptually simple (though infinite) repeated games on graphs, like parity games. In these games the players, in turns, choose which edge of the graph to follow, so that a play of the game forms an infinite path in the graph, which is winning for one of the players depending on the winning objective. Indeed, it is natural to formulate the verification problems directly within a game framework, e.g., as a game of a system versus environment, or a a controller versus a non-deterministic system under its control.

Now the game framework also opens paths to extensions of the model, less apparent in logical setting, but for long present in the main stream of the game theory. These are, in particular, games with quantitative rather than qualitative objectives (like mean-payoff games), games with simultaneous rather than turn-based moves (so-called concurrent games), or games where the players select probability distributions over successor states (so-called stochastic games). While the determinacy of the one-shot concurrent stochastic games is given by the classical von Neumann Minimax Theorem (of 1928), its powerful repeated game extension, the determinacy of Blackwell games, has been established by D.A.Martin only quite recently (1998). What is more, one can also consider a more general case, where the players' objectives are not antagonistic, and the

role of determinacy is taken by the central concept of the game theory – the Nash equilibrium.

All these features can make sense in a number of verification scenarios, which makes the aforementioned game models attractive for computer science. They are however inherently complex mathematically, in particular the very concept of winning has to be refined, according to if we search for (almost) sure winning, or for maximizing the winning probability (quantitative analysis). One could expect that the computational complexity of the decision problems of the new games is also prohibitive. By the time when Krishnendu Chatterjee started his research, only few results were known for some special cases (the work by Condon, Zwick and Paterson, Jurdziński on turn-based stochastic games, and Secchi and Sudderth on concurrent games with boolean safety objectives). His subsequent work had to change the picture essentially.

Chatterjee's thesis. Krishnendu Chatterjee has established a number of strong algorithmic results, proving the optimal or nearly optimal upper bounds for most of the games mentioned above, thus solving a long list of open problems. A general message of all these results is that enhancing the game model by concurrent and stochastic elements is much more feasible than it could appear at the first sight. This greatly improves our algorithmic understanding of games, and opens new perspectives for a variety of formal methods based on game scenarios. The main contributions are as follows.

- For turn-based stochastic games the author provides algorithms for both qualitative and quantitative analysis, which yield a PSPACE upper bound for games with Muller objective, and NP (resp. co-NP) upper bounds for games with Rabin and Streett objectives, respectively. This implies an NP $\cap$ co-NP upper bound for games with parity objective, which coincides with the best known upper bound in deterministic case. The previous upper bounds known for stochastic games, due to de Alfaro and Majumdar, were 2EXPTIME for parity games, and 3EXPTIME for the remaining games. The author also optimizes the amount of memory needed in optimal strategies, and designs a strategy improvement algorithm for turn-based Rabin and Streett games.
- For concurrent stochastic games, the author shows that the quantitative analysis can be achieved in PSPACE for games with parity objective, and in EXPSPACE for Muller, Rabin, and Streett objectives. The previous upper bounds here were 3EXPTIME and 4EXPTIME, respectively (again, due to de Alfaro and Majumdar). This impressive result, showing that a big advance in expressive power can be achieved with a relatively small complexity overhead, is obtained by a deep mathematical and game-theoretic insight.
- The author gives an elementary and combinatorial proof of existence of memoryless ε-optimal strategies in concurrent games with reachability objectives, for all $\varepsilon > 0$. The previous proof (in a monograph of Filek and Vrieze) used advanced results from analysis. The proof technique originally developed here also yields a strategy improvement algorithm for concurrent reachability games.

- The author makes a fundamental contribution to the theory of repeated games, by giving an EXPTIME algorithm for solving concurrent games with limit-average objective. This is a first algorithmic result in the analysis of these games, extensively studied in game theory for decades. The result also yields a PSPACE upper bound for solving concurrent games with discounted reward objectives.
- The author develops a game-theoretic model for an interaction between processes or components of a system, whose goals are not strictly conflicting, but rather conditionally competitive. This is a turn-based deterministic game with an ω-regular objective for each player. An essential conceptual contribution consists in an idea of a *secure* equilibrium, which is a special case of a Nash equilibrium, in which no player can lower the other player's payoff, without lowering her own payoff as well. The author establishes the existence and uniqueness of such equilibria, presents an algorithm to compute them, and demonstrates their applications in the modular verification of systems.

The results of the thesis were published in numerous papers, presented to conferences CSL, CONCUR, SODA, ICALP, QEST, LICS, FSTTCS, and TACAS, as well as in journals: *Theoretical Computer Science* and *International Journal of Game Theory*. Some of these papers were co-authored; the list of co-authors includes L. de Alfaro, T.A.Henzinger, M.Jurdziński, and R.Majumdar.

Biographic Sketch. Krishnendu Chatterjee was born in 1978 and received his B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, India, in 2001. He is the recipient of various excellent student awards, among them the *President of India Gold Medal* in 2001. He received his M.Sc. degree in Computer Science from the University of California, Berkeley, in 2004. He wrote his Ph.D. thesis under the supervision of professor Thomas A. Henzinger, and obtained the Ph.D. degree in Computer Science from the University of California in Berkeley in 2007. He is currently a post-doctoral researcher at the University of California in Santa Cruz.

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[2]. 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

[2] 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.dimi.uniud.it/eacsl/award.html`

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 J. van Benthem (Amsterdam), B. Courcelle (Bordeaux), M. Grohe (Berlin), M. Hyland (Cambridge), J.A. Makowsky (Haifa, President of EACSL), D. Niwinski (Warsaw, Vice President of EACSL), G. Plotkin (Edinburgh, LICS Organizing Committee) and A. Razborov (Moscow and Princeton).

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

Detailed reports on their work appeared in the CSL'05, CSL'06 and CSL'07 proceedings, and are also available via the EACSL homepage.

Author Index

Abel, Andreas 446

Barthe, Gilles 493
Berardi, Stefano 215
Beyersdorff, Olaf 199
Blanqui, Frédéric 1
Bonfante, Guillaume 49
Brochenin, Rémi 323

Cardelli, Luca 32
Charatonik, Witold 94
Chatterjee, Krishnendu 385
Colcombet, Thomas 416
Creignou, Nadia 109
Curien, Pierre-Louis 15

Dal Lago, Ugo 230
Dawar, Anuj 354
de'Liguoro, Ugo 215
Demri, Stéphane 323
Dezani-Ciancaglini, Mariangiola 461
Di Cosmo, Roberto 461
Doyen, Laurent 385

Eisinger, Jochen 431

Fontaine, Gaëlle 139

Giovannetti, Elio 461
Grädel, Erich 354
Grégoire, Benjamin 493

Hamano, Masahiro 262
Henzinger, Thomas A. 385
Hermant, Olivier 169
Hodkinson, Ian 308
Hofmann, Martin 79
Horbach, Matthias 293

Jouannaud, Jean-Pierre 1

Kahle, Reinhard 49
Kameyama, Yukiyoshi 478
Katsumata, Shin-ya 278
Kotek, T. 339
Kuncak, Viktor 124

La Torre, Salvatore 33
Laurent, Olivier 230
Lipton, James 169
Löding, Christof 416
Lozes, Etienne 323

Madhusudan, P. 33
Makowsky, J.A. 339, 508
Marion, Jean-Yves 49
McKenzie, Pierre 64
Montanari, Angelo 308
Müller, Sebastian 199

Nakano, Hiroshi 478
Nakazawa, Koji 478
Nguyen, Phuong 184
Niwinski, D. 508

Oitavem, Isabel 49

Parlato, Gennaro 33
Piskac, Ruzica 124
Place, Thomas 401

Riba, Colin 493
Rodriguez, Dulma 446
Rubio, Albert 1

Saurin, Alexis 154
Schewe, Sven 369
Schnoor, Henning 109
Schnoor, Ilka 109
Schöpp, Ulrich 79
Sciavicco, Guido 308

Takemura, Ryo 262
Tatsuta, Makoto 461, 478
Thomas, Michael 64
Thomas, Wolfgang 23
Tranquilli, Paolo 246

Vollmer, Heribert 64

Weidenbach, Christoph 293
Wrona, Michał 94

Zilber, B. 339