Lecture Notes in Computer Science 1584

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Georg Gottlob Etienne Grandjean
Katrin Seyr (Eds.)

Computer Science Logic

12th International Workshop, CSL'98
Annual Conference of the EACSL
Brno, Czech Republic, August 24-28, 1998
Proceedings

Springer

Volume Editors

Georg Gottlob
Katrin Seyr
TU Wien, Institut für Informationssysteme
Paniglgasse 16, A-1040 Wien, Austria
E-mail: {gottlob,seyr}@dbai.tuwien.ac.at

Etienne Grandjean
Université de Caen, Campus 2
F-14032 Caen Cedex, France
E-mail: Etienne.Grandjean@info.unicaen.fr

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Computer science logic : 12th International Workshop ; proceedings / CSL'98, Brno, Czech Republic, August 24 - 28, 1998. Georg Gottlob ... (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Annual Conference of the EACSL ... ; 1998)
(Lecture notes in computer science ; Vol. 1584)
ISBN 3-540-65922-6

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

ISSN 0302-9743
ISBN 3-540-65922-6 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10703163 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The 1998 Annual Conference of the European Association for Computer Science Logic, CSL'98, was held in Brno, Czech Republic, during August 24-28, 1998. CSL'98 was the 12th in a series of workshops and the 7th to be held as the Annual Conference of the EACSL.

The conference was organized at Masaryk University in Brno by the Faculty of Informatics in cooperation with universities in Aaachen, Caen, Haagen, Linz, Metz, Pisa, Szeged, Vienna, and other institutions. CSL'98 formed one part of a federated conferences event, the other part being MFCS'98, the 23rd International Symposium on the Mathematical Foundations of Computer Science. This federated conferences event consisted of common plenary sessions, invited talks, several parallel technical programme tracks, a dozen satellite workshops organized in parallel, and tutorials.

The Federated CSL/MFCS'98 Conferences event included 19 invited talks, four of them joint CSL/MFCS'98 talks (D. Harel, W. Maass, Y. Matiyasevic, and M. Yannakakis), four for CSL (P. Hajek, J. Mitchell, Th. Schwentick, and J. Tiuryn), and eleven for MFCS. Last but not least, two tutorials were organized by CSL on the day preceding the symposium on "Inference Rules in Fragments of Arithmetic" by Lev Beklemishev and on "Proofs, Types, and Safe Mobile Code" by Greg Morrisett.

A total of 345 persons attended the Federated CSL/MFCS'98 Conference which was a great success.

The program committee of CSL'98 selected 27 of 74 papers submitted for the conference. From the 27 papers selected for presentation, 25 have been accepted, following the standard refereeeing procedure, for publication in the present proceedings. Three invited speakers submitted papers, that were likewise refereeed and accepted.

Katrin Seyr as co-editor has performed the principal editing work needed in connection with collecting the successive versions of the papers and tidying things up for the final appearance of the CSL'98 proceedings as a Springer LNCS volume using LNCS LATEX style.

We are most grateful to the numerous referees for their work. Finally, we express special thanks to the MFCS/CSL'98 organizing committee chaired by Jan Staudek and to Jozef Gruska and Jiri Zlatuska (co-chairs of the MFCS'98 program committee) for the perfect organization of the federated conference.

February 1999 Georg Gottlob and Etienne Grandjean

Program Committee

K. R. Apt (The Netherlands)
F. Baader (Germany)
A. Carbone(France)
M. Fitting (USA)
A. Goerdt (Germany)
G. Gottlob (co-chair) (Austria)
M. Kanovich (Russia)
E. Grandjean (co-chair) (France)
C. Lautemann (Germany)
A. Leitsch (Austria)
D. Leivant (USA)
G. Longo (France)
J. Paredaens (Belgium)
A. A. Razborov (Russia)
A. Scedrov (USA)
K. Stroetmann (Germany)
A. Voronkov (Sweden)

Organizing Committee

M. Brandejs
L. Brim
T. Dudaško
J. Foukalová
I. Hollandová
D. Janoušková
A. Kučera
L. Motyčková
J. Obdržálek
M. Povolný
P. Smrž
P. Sojka
J. Srba
J. Staudek (chair)
P. Starý
Z. Walletzká

Referees

P. Aczel
A. Avron
L. Bachmair
D. Beauquier
U. Berger
S. Brass
H. Comon
N. Creignou
E. Dahlhaus
D. de Jongh
J. Despeyroux
J. Dix
U. Egly
Ch. Fermüller
E. Gerstorfer
C. Ghidini
E. Grädel
R. Hähnle
B. Heinemann
D. Hofbauer
H. Kirchner
P. Koiran
Y. Lafont
M. Maidl
D. Miller
T. Mossakowski
P. Narendran
R. Nieuwenhuis
E.-R. Olderog
L. Pacholski
M. Pentus
P. Pudlak
E. Ritter
P. Rozière
A.A. Sapozhenko
M. Schmidt-Schauss
A. Setzer
G. Smolka
I. Stewart
G. Struth
P. Ageron
P.A. Abdulla
K. Barthelmann
L. Beklemishev
H. Blair
R. Caferra
T. Coquand
J. Cuellar
V. Danos
M. de Rougemont
M. Dezani
G. Dowek
Th. Eiter
M. Fiore
H. Geuvers
E. Gimenez
B. Gramlich
P. Hajek
L. Hella
J.-B. Joinet
J. Köbler
J.-L. Krivine
H. Leiss
R. Matzinger
A. Montanari
M. Müller
J. Niehren
R. Nossum
V. Orevkov
V. Padovani
J.-E. Pin
G. Renardel
P.H. Rodenburg
P. Ruet
U. Sattler
Th. Schwentick
W. Slany
S. Soloviev
K. Stokkermans
V.S. Subrahmanian
M. Alekhnovich
M. Baaz
P. Baumgartner
V. Benzaken
T. Borzyszkowski
L. Colson
J. Courant
G. D'Agostino
A. Dawar
A. Degtyarev
A. Dikovsky
M. Droste
P. Enjalbert
M. Fitting
N. Ghani
F. Gire
M. Grohe
P. Hancock
H. Herbelin
M. Kaltenbach
R. König
M. Kummer
L. Levin
K. Meer
M. More
N. Murray
I. Niemelä
P.W. O'Hearn
M. Otto
Ch. Paulin
A. Podelski
B. Reus
E. Rosen
G. Salzer
V. Sazonov
L. Serafini
A. Slissenko
R. Stärk
T. Streicher
M. Takeyama

T. Tammet
H. Tompits
J. Tromp
P. Urzyczyn
H. Veith
P. Vitanyi
B. Werner
C. Zaniolo
A. Tarlecki
L. Tortora de Falco
K.U. Schulz
J. van Oosten
K. Verchinine
U. Waldmann
Th. Wilke
M. Zimand
M. Thielscher
Ch. Tresp
A. Urquhart
M. van Lambalgen
N. Vereshagin
H. Wansing
F. Wolter

Table of Contents

Trakhtenbrot Theorem and Fuzzy Logic

Petr Hájek

Institute of Computer Science, Academy of Sciences
182 07 Prague, Czech Republic
hajek@uivt.cz
Dedicated to Professor Boris Abramovich Trakhtenbrot.

Abstract. Trakhtenbrot theorem is shown to be valid for the three main fuzzy logics - Łukasiewicz, Gödel and product logic.

1 Introduction

It follows from Gödel's completeness theorem that the set of all tautologies of the classical (Boolean) predicate logic (denote this set by $TAUT^{Bool\forall}$) is recursively enumerable, i. e. Σ_1 and it is known that it is Σ_1-complete. Tautologies are formulas true in all models and it is crucial that both finite and infinite models are considered. Finite model theory (flourishing due to its obvious relevance for databases) uses the language of classical logic but admits only finite models. See [3]. Let $fTAUT^{Bool\forall}$ be the set of all formulas true in all finite models. Clearly, $TAUT^{Bool\forall} \subseteq fTAUT^{Bool\forall}$. Trakhtenbrot proved as early as in 1950 [7] that the set $fTAUT^{Bool\forall}$ is *not* recursively enumerable (hence not recursively axiomatizable); moreover, the set is Π_1-complete. Due to the properties of classical negation it follows that the set $fSAT^{Bool\forall}$ of all formulas ϕ true at least in one finite model is Σ_1-complete. The fact that there is no recursive axiomatic system complete for tautologies of finite model theory means that deductive methods have only limited importance for database theory.

Fuzzy logic generalizes Boolean logic by introducing more than two truth values; typically the real unit interval $[0,1]$ serves as the ordered set of truth values (truth degrees). Let us stress that fuzzy logic can be developed rather far in the style of mathematical logic (see [2,1]). On the other hand, there is a research in fuzzy databases [6]. Thus whether and in which form Trakhtenbrot theorem generalizes to fuzzy logic appears to be very natural. To answer this question is the main purpose of this paper. We shall investigate three important fuzzy predicate calculi having $[0,1]$ for their truth set - Łukasiewicz predicate logic Ł$\forall$, Gödel predicate logic $G\forall$ and product predicate logic $\Pi\forall$. Let $\mathcal{C}$ vary over $\text{Ł}, G, \Pi$, let $fTAUT_1^{\mathcal{C}\forall}$ be the set of all formulas true in the sense of $\mathcal{C}\forall$ in all finite models and $fSAT_1^{\mathcal{C}\forall}$ be the set of all formulas true in the sense of $\mathcal{C}\forall$ in at least one finite model. Our main result is as follows:

Theorem. For $\mathcal{C}$ being $\text{Ł}, G, \Pi$, the set $fTAUT_1^{\mathcal{C}\forall}$ is Π_1-complete and the set $fSAT_1^{\mathcal{C}}\forall$ is Σ_1-complete.

This – and much more – will be proved in Sect. 3. Section 2 contains preliminaries on arithmetical hierarchy and fuzzy logic.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 1–8, 1999.

Acknowledgements This research has been partially supported by the COST action 15 (Many-valued logics for computer science applications.)

2 Preliminaries

The reader is assumed to be familiar with basic properties of *recursive sets* (of natural numbers, words etc.), recursive relations and recursive functions. A set A is Σ_1 (or *recursively enumerable*) if there is a binary recursive relation R such that

$$A = \{n|(\exists m)R(m,n)\}.$$

A is Π_1 if there is a binary recursive relation R such that

$$A = \{n|(\forall m)R(m,n)\}.$$

Similarly, A is Σ_2 if for some ternary recursive relation R,

$$A = \{n|(\exists m)(\forall b)R(m,k,n)\}$$

etc. A is Σ_1-*complete* if it is Σ_1 and each Σ_1 - set B is recursively reducible to A, i. e. for some recursive function f,

$$B = \{n|f(n) \in A\}.$$

Similarly for Π_1-complete etc. A set is Δ_1 if it is both Σ_1 and Π_1. Recall that Δ_1 sets are exactly all recursive sets. See [5] for more information. We also assume that the reader knows basic notion of the theory of computational complexity, i. e. what it means that a set is in P (recognized by a deterministic Turing machine running in polynomial time) or in NP ($\cdots$ nondeterministic Turing machine $\cdots$). Here we deal with polynomial reducibility and NP-completeness as well as co-NP-completeness. See [4].

Now we recall some basic facts on fuzzy logics. A logic with the truth set $[0,1]$ is given by the choice of *truth functions* determining the truth value of a compound formula from the truth values of its components. In [1] the reader may find some theory of continuous t-norms as possible truth functions for the *conjunction*, their residua as truth functions of *implication* and the corresponding truth functions of *negation*. We shall not need this; we shall only need three particular choices. (They are extremely outstanding choices.) Here they are:

Łukasiewicz (Ł):

$$x * y = \max(0, x + y - 1); \tag{1}$$

$$x \Rightarrow y = 1 \text{ for } x \leq y, \tag{2}$$

$$x \Rightarrow y = 1 - x + y \text{ for } x \geq y; \tag{3}$$

$$(-)x = 1 - x \tag{4}$$

Gödel (G):

$$x * y = \min(x, y) \tag{5}$$
$$x \Rightarrow y = 1 \text{ for } x \leq y, \tag{6}$$
$$x \Rightarrow y = y \text{ for } x > y; \tag{7}$$
$$(-)0 = 1, \tag{8}$$
$$(-)x = 0 \text{ for } x > 0. \tag{9}$$

Product (Π):

$$x * y = x \cdot y \text{ (usual multiplication)} \tag{10}$$
$$x \Rightarrow y = 1 \text{ for } x \leq y, \tag{11}$$
$$x \Rightarrow y = y/x \text{ for } x > y; \tag{12}$$
$$(-x) \text{ as in Gödel .} \tag{13}$$

The corresponding propositional logic has formulas built from propositional variable, the constant $\bar{0}$ and connectives $\&, \rightarrow$. Negation, the min-conjunction and the max-disjunction are defined as follows:

$$\neg\varphi \text{ is } \varphi \rightarrow \bar{0},$$

$$\varphi \wedge \psi \text{ is } \varphi\&(\varphi \rightarrow \psi),$$
$$\varphi \vee \psi \text{ is } ((\varphi \rightarrow \psi) \rightarrow \psi) \wedge ((\psi \rightarrow \varphi) \rightarrow \varphi).$$

Each evaluation e of propositional variables by elements of $[0, 1]$ extends uniquely to the evaluation $e_{\mathcal{C}}(\varphi)$ of each formula φ using the truth function of $\mathcal{C}$ (e being Ł, G, Π). We are lead to the following

Definition. Let $\mathcal{C}$ stand for Ł, G, or Π.
$SAT_1^{\mathcal{C}} = \{\varphi |$ for some $[0,1]$-evaluation e, $e_{\mathcal{C}}(\varphi) = 1\}$,
$SAT_{pos}^{\mathcal{C}} = \{\varphi |$ for some $[0,1]$-evaluation e, $e_{\mathcal{C}}(\varphi) > 0\}$,
$TAUT_1^{\mathcal{C}} = \{\varphi |$ for each $[0,1]$-evaluation e, $e_{\mathcal{C}}(\varphi) = 1\}$,
$TAUT_{pos}^{\mathcal{C}} = \{\varphi |$ for each $[0,1]$-evaluation e, $e_{\mathcal{C}}(\varphi) > 0\}$.

Clearly, SAT_1 stands for 1-satisfiable, SAT_{pos} for positively satisfiable and similarly $TAUT$ for tautologies. The following summarizes the results on complexity:

Theorem. ([1] 6.2.17)
(1) $SAT_1^G = SAT_{pos}^G = SAT_1^{\Pi} = SAT_{pos}^{\Pi} = SAT^{Bool}$ is NP-complete.
(2) $TAUT_{pos}^G = TAUT_{pos}^{\Pi} = TAUT^{Bool}$, is co-NP-complete.
(3) $TAUT_1^G$, $TAUT_1^{\Pi}$, $TAUT^{Bool}$ are pairwise distinct and all co-NP complete.
(4) $SAT_{pos}^{\text{Ł}} \supset SAT_1^{\text{Ł}} \supset SAT^{Bool}$, all NP-complete.
(5) $TAUT_1^{\text{Ł}} \subset TAUT_{pos}^{\text{Ł}} \subset TAUT^{Bool}$, all co-NP-complete.

*

The *predicate calculus* $\mathcal{C}\forall$ has a language consisting of predicates (each having a positive natural arity). (Here we disregard object constants.)

Atomic formulas have the form $P(x_1, \ldots, x_n)$ where P is an n-ary predicate and $x_1, \ldots, x_n$ are *object variables*. If φ, ψ are formulas then $\varphi\&\psi, \varphi \to \psi, (\forall x)\varphi, (\exists x)\varphi$ are formulas; $\bar{0}$ is a formula.

A *model* has the form $\mathbf{M} = (M, (r_P)_{P\, predicate})$ where $M \neq \emptyset$ is a set and $r_P : M^{ar(P)} \to [0,1]$ is a fuzzy relation M of arity equal to the arity of P. An *evaluation of variables* is a mapping $v : Var \to M$ (Var being the set of object variables. The truth value of a formula φ (over $\mathcal{C}$) given by $\mathbf{M}, v$ is defined inductively in Tarski's style, i.e.

$$\|P(x_1, \ldots, x_n)\|^{\mathcal{C}}_{\mathbf{M},v} = r_P(v(x_1), \ldots, v(x_n)),$$

$$\|\varphi\&\psi\|^{\mathcal{C}}_{\mathbf{M},v} = \|\varphi\|^{\mathcal{C}}_{\mathbf{M},v} * \|\psi\|^{\mathcal{C}}_{\mathbf{M},v}, \text{analogous for } \to, \Rightarrow;$$

$$\|(\forall x)\varphi\|^{\mathcal{C}}_{\mathbf{M},v} = \inf\{\|\varphi\|^{\mathcal{C}}_{\mathbf{M},w} | v \equiv_x w\},$$

Analogously for $\exists, \sup$ (Note that $v \equiv_x w$ mean that v coincides with w for all arguments except possibly x.) $TAUT_1^{\mathcal{C}\forall}$ is the set of all formulas φ such that $\|\varphi\|^{\mathcal{C}}_{\mathbf{M},v} = 1$ for all $\mathbf{M}, v$; $SAT_1^{\mathcal{C}\forall}$ is the set of all φ such that $\|\varphi\|^{\mathcal{C}}_{\mathbf{M},v} = 1$ for some $\mathbf{M}, v$. Similarly for $TAUT^{\mathcal{C}\forall}_{pos}$ and $SAT^{\mathcal{C}\forall}_{pos}$.

Fact. $TAUT_1^{G\forall}$ is Σ_1 complete; $TAUT_1^{\mathbf{L}\forall}$ is Π_2 complete; $TAUT_1^{\Pi\forall}$ is Π_2-hard (i. e. each Π_2 set is reducible to $TAUT_1^{\Pi\forall}$; whether the latter set is itself Π_2 is unknown.) See [1]. For results on positive tautologicity/satisfiability see Appendix here.

3 The results

Recall the definition of $fTAUT_1^{\mathcal{C}\forall}$ and of $fSAT_1^{\mathcal{C}\forall}$. Similarly we define $fTAUT^{\mathcal{C}\forall}_{pos}$ to be the set of all formulas having a positive value in the sense of $\mathcal{C}\forall$ in all finite models and $fSAT^{\mathcal{C}\forall}_{pos}$ the set of all formulas having a positive value in the sense of $\mathcal{C}\forall$ in at least one finite model.

Surprizingly, the results are rather analogous to those on computational complexity summarized above.

Theorem.
(1) $fSAT_1^G = fSAT^G_{pos} = fSAT_1^{\Pi} = fSAT^{\Pi}_{pos} = fSAT^{Bool}$ is Σ_1-complete.
(2) $fTAUT^G_{pos} = fTAUT^{\Pi}_{pos} = fTAUT^{Bool}$, Π_1-complete.
(3) $fTAUT_1^G$, $fTAUT_1^{\Pi}$, $fTAUT^{Bool}$ are pairwise distinct and all Π_1-complete.
(4) $fSAT^{\mathbf{L}}_{pos} \supset fSAT_1^{\mathbf{L}} \supset fSAT^{Bool}$, all Σ_1-complete.
(5) $fTAUT_1^{\mathbf{L}} \subset fTAUT^{\mathbf{L}}_{pos} \subset fTAUT^{Bool}$, all Π_1-complete.
The rest of the section contains a proof.

Lemma Let $i(0) = 0, i(x) = 1$ for $x > 0$ (double Gödel negation). For $\mathbf{M} = (M, (r_P)_P)$ let $\mathbf{M}^* = (M, (r_P^*)_P)$ where $r_P^*(a, \ldots, b) = i(r_P(a, \ldots, b))$ (i.e. $\mathbf{M}^*$ is crisp). If M is finite and $\mathcal{C} =$G or Π then for each φ, v

$$\|\varphi\|^{\mathcal{C}}_{M^*}[v] = i(\|\varphi\|_M[v]).$$

Thus

$$\|\varphi\|^{\mathcal{C}}_M[v] > 0 \text{ implies } \|\varphi\|^{\mathcal{C}}_{M^*}[v] = 1, \tag{14}$$

$$\|\varphi\|^{\mathcal{C}}_M[v] = 0 \text{ implies } \|\varphi\|^{\mathcal{C}}_{M^*}[v] = 0. \tag{15}$$

Proof easy by induction.

Corollary. For $\mathcal{C}$ being G or Π,

(i) $fSAT_1^{\mathcal{C}\forall} = fSAT_{pos}^{\mathcal{C}\forall} = fSAT^{Bool\forall}$,

(ii) $fTAUT_{pos}^{\mathcal{C}\forall} = fTAUT^{Bool\forall}$.

Proof. $fSAT^{Bool\forall} \subseteq fSAT_1^{\mathcal{C}\forall} \subseteq fSAT_{pos}^{\mathcal{C}\forall}$ is obvious; $fSAT_{pos}^{\mathcal{C}\forall} \subseteq fSAT^{Bool\forall}$ follows by (1) of the preceding lemma. $fTAUT_{pos}^{\mathcal{C}\forall} \subseteq fTAUT^{Bool\forall}$ is obvious; the converse inclusion follows by (2) of the lemma.

This proves (1) and (2) of our Theorem.

Now we shall elaborate a technique of coding formulas of predicate logic by some formulas of propositional logic and finite models by some evaluation of propositional variables.

Definition. Let $\mathbf{M} = (M, (r_{P_i})_{i=1}^k)$ be a finite model, let M have n elements. For each predicate P_i of arity s we introduce n^s propositional variables $p_{ij_1 \ldots j_s}$ where $j_1, \ldots j_s \in \{1, \ldots, n\}$ (assume $M = \{1, \ldots, n\}$). Define an evaluation $e_{\mathbf{M}}$ of these propositional variables by setting $e_{\mathbf{M}}(p_{ij_1,\ldots j_s}) = r_{P_i}(j_1, \ldots j_s)$ (i. e. the truth value of $p_{ij_1 \ldots j_s}$ is the degree in which $(j_1, \ldots j_s)$ is in the relation r_{P_i}).

Investigate formulas of predicate logic with free variable substituted by elements of M. For each such object φ we define its translation $\varphi^{*,n}$ as follows:
$(P_i(j_1, \ldots, j_s))^* = p_{ij_1,\ldots j_s}$; $(\varphi \& \psi)^* = \varphi^* \& \psi^*$; analogously $\to$;
$(\bar{0})^* = \bar{0}$; $((\forall x)\varphi(x))^* = \bigwedge_{j=1}^n \varphi^*(j), ((\exists x)\varphi(x))^* = \bigvee_{j=1}^n \varphi^*(j)$.

Note that if φ is as assumed (free variables replaced by elements of M) then $\|\varphi\|_{\mathbf{M}}$ has the obvious meaning $\|\varphi\|_{\mathbf{M},v}$ where v just assigns to each free variable the corresponding element of M (and otherwise arbitrary).

Lemma. For each finite $\mathbf{M}$ of cardinality n and φ as above,

$$\|\varphi\|_{\mathbf{M}} = e_{\mathbf{M}}(\varphi^{*,n}).$$

Proof. obvious by induction on φ observing that on a finite domain $\forall$ reduces to a finite $\wedge$-conjunction and analogously $\exists, v$. Note that $\varphi^{*,n}$ is a recursive function of φ and n, the language $P_1, \ldots, P_k$ being given.

Some instances of the following lemma are redundant (since they follow from the preceding), but we prefer a uniform formulation and proof.

Lemma. For $\mathcal{C}$ being Ł, G, Π, and # being 1 or *pos*, the set $fSAT^{\mathcal{C}\forall}_{\#}$ is Σ_1 and the set $fTAUT^{\mathcal{C}\forall}_{\#}$ is Π_1.

Proof. Let φ vary over closed formulas. Recall that the sets $SAT^{\mathcal{C}}_{\#}, TAUT^{\mathcal{C}}_{\#}$ of propositional formulas are of low computational complexity (co-NP, NP) and hence recursive. Now

$$\varphi \in fSAT^{\mathcal{C}\forall}_{\#} \text{ iff } (\exists n)(\varphi^{*,n} \in SAT^{\mathcal{C}}_{\#}),$$

$$\varphi \in fTAUT^{\mathcal{C}\forall}_{\#} \text{ iff } (\forall n)(\varphi^{*,n} \in TAUT^{\mathcal{C}}_{\#}).$$

This proves the lemma.

Lemma For $\mathcal{C} =$ G or Π, $fTAUT^{\mathcal{C}\forall}_1$ is Π_1-complete.

Proof. The set in question is in Π_1 by the previous lemma. To reduce $fTAUT^{Bool\forall}$ to $fTAUT^{\mathcal{C}\forall}_1$ we use the double negation interpretation:

Let $\varphi^{\neg\neg}$ result from φ by attaching double negation to each atomic formula. Then (cf. [1] 6.2.8, 6.3.1)

$$\varphi \in fTAUT^{Bool\forall} \text{ iff } \varphi^{\neg\neg} \in fTAUT^{\mathcal{C}\forall}.$$

Lemma The sets $fSAT^{\text{Ł}\forall}_1$ and $fSAT^{\text{Ł}\forall}_{pos}$ are Σ_1-complete.

Proof. Let $Crisp(P_i)$ be the formula $(\forall \mathbf{x})(P(\mathbf{x}) \vee \neg P(\mathbf{x}))$. It is easy to show that $\varphi \in fSAT^{Bool\forall}$ iff $\bigwedge_i Crisp(P_i) \wedge \varphi$ is in $fSAT^{\text{Ł}\forall}_1$. (This argument also works for G, Π as an alternative to the above proof.) Thus again we have a recursive reduction.

For $SAT^{\text{Ł}\forall}_{pos}$ we proceed as follows: (using a method of Ragaz, cf. [1] 6.3.6 - 6.3.9).: for each closed ψ, $\psi \in fSAT^{Bool\forall}$ iff $\bigwedge Crisp^2(P_i) \wedge \psi^2$ is positively finitely satisfiable, i. e. iff there is a finite model $\mathbf{M}$ such that $\|Crisp^2(D_i) \wedge \psi^2\|_{\mathbf{M}} > 0$. (Cf. [1] 6.2.13 and 6.3.10.) Here one has to assume that ψ is *classical* in the sense that the only connectives used are $\wedge, \vee, \neg$.

This reduces $fSAT^{Bool\forall}$ to $SAT^{\text{Ł}\forall}_{pos}$.

Corollary The sets $fTAUT^{\text{Ł}\forall}_1$ and $fTAUT^{\text{Ł}\forall}_{pos}$ are Π_1-complete.

Proof. This is because, thanks to the properties of Łukasiewicz negation, for each sentence φ, $\varphi \in SAT^{\text{Ł}\forall}_1$ iff $(\neg\varphi) \notin TAUT^{\text{Ł}\forall}_{pos}$, and similarly for 1 and *pos* interchanged. Thus the complement of $TAUT^{\text{Ł}\forall}_{pos}$ is Σ_1-complete and so is the complement of $TAUT^{\text{Ł}\forall}_1$. This completes the proof of our theorem.

4 Appendix

It is of some interest to observe that $\mathcal{C}\forall$ has the *rational model property*:

Claim. For $\mathcal{C}$ being Ł, G, Π, the following holds:

(1) If there is a finite $\mathbf{M}$ with $\|\varphi\|^{\mathcal{C}}_{\mathbf{M}} = 1$ then there is a rational-valued model $\mathbf{M}'$ (of the same cardinality) with $\|\varphi\|^{\mathcal{C}}_{M'} = 1$.

(2) the same with < 1 instead of $= 1$.
(Note that this gives an alternative proof of $fSAT^{\mathcal{C}} \in \Sigma_1, fTAUT^{\mathcal{C}} \in \Pi_1$. Cf. [1] 6.3.6)

Proof. Due to our representation (Sect. 2), it is enough to show for each *propositional* formula φ that if $e_{\mathcal{C}}(\varphi) = 1$ then for some rational-valued e', $e'_{\mathcal{C}}(\varphi) = 1$ and the same for < 1. First, this is easy for G since if $0 < z_1 < \ldots < z_n < 1$ contain all the values $e(p_i)$ involved and $0 < r_1 < \ldots < r_n < 1$ are rationals then one easily gets an *isomorphism* of $[0, 1]$ with respect to Gödel connectives moving z_i to r_i. For Ł and < 1 the claim follows immediately from the continuity of truth functions; for Ł and $= 1$ we use [1] 3.3.17. Finally, investigate Π and recall the transformation φ^I [1] 6.2.2 such that, for each I, φ^I does not contain $\bar{0}$ or φ^I is $\bar{0}$; and for each e such that $e(p_i) = 0$ iff $i \in I, e_{\Pi}(\varphi) = e_{\Pi}(\varphi^I)$. For "$< 1$" observe that for positive values $x_i = e(p_i)\,(i \notin I)$, the value $e_{\Pi}(\varphi_I)$ is continuous (and positive) in $x_i\,(i \notin I)$. (If φ_I is $\bar{0}$ then there is nothing to prove.)

Finally for "$= 1$" observe that if e is such that $e_{\Pi}(\varphi) = 1$ then for some *boolean* e', $e'_{\Pi}(\varphi) = 1$ ($e'(p_i) = 0$ if $e(p_i) = 0$, $e'(p_i) = 1$ otherwise). This completes the proof.

Finally we present some easy facts on arithmetical complexity not contained in [1].

Fact. (1) $\mathrm{TAUT}^{\mathrm{Ł}\forall}_{pos}$ is Σ_1-complete.
(2) $\mathrm{TAUT}^{G\forall}_{pos}$ is Σ_1-complete.

Proof. (1) $\mathrm{TAUT}^{\mathrm{Ł}\forall}_{pos}$ is Σ_1-complete by [1] 6.1.13 (1). To prove (2) observe $\varphi \in \mathrm{TAUT}^{\mathrm{G}\forall}_{pos}$ iff $\neg\neg\varphi \in \mathrm{TAUT}^{G\forall}_{1}$; hence the former set is in Σ_1. Moreover, we shall show that for each classical φ, $\varphi \in \mathrm{TAUT}^{Bool\forall}$ iff $(Crisp \to \varphi) \in \mathrm{TAUT}^{\mathrm{Ł}\forall}_{pos}$, where $Crisp$ is the formula defined above and expressing crispness of all predicates occuring in φ; thus $\mathrm{TAUT}^{\mathrm{Ł}\forall}_{pos}$ is Σ_1-complete.

Indeed iff $(Crisp \to \varphi) \in TAUT^{\mathrm{G}\forall}_{pos}$ then obviously $\varphi \in TAUT^{Bool\forall}$. Conversely, if $\|Crisp \to \varphi\|_{\mathbf{M}} = 0$ for some $\mathbf{M}$ then $\|\varphi\|_M = 0$ and $\|Crisp\|_{\mathbf{M}} = t$ for some $t > 0$. Then we can construct the crisp model $\mathbf{M}^*$ as above (but now possibly infinite) and show that for each sentence φ, $\|\varphi\|^*_{\mathbf{M}} = i(\|\varphi\|_{\mathbf{M}})$. Hence $\mathbf{M}^*$ is a crisp model and φ is false in $\mathbf{M}^*$; hence $\varphi \notin TAUT^{Bool\forall}$.

Remark. Concerning $\mathrm{TAUT}^{\Pi\forall}_{pos}$, the only obvious thing is that it is reducible to $\mathrm{TAUT}^{\Pi\forall}_{1}$ (by mapping φ to $\neg\neg\varphi$) and that $\mathrm{TAUT}^{Bool\forall}$ reduces to $\mathrm{TAUT}^{\Pi\forall}_{pos}$.

Fact.
(1) $\mathrm{SAT}^{\mathrm{Ł}\forall}_{1}$ is Π_1-complete.
(2) $\mathrm{SAT}^{\mathrm{Ł}\forall}_{pos}$ is Σ_2-complete.
(3) $\mathrm{SAT}^{G\forall}_{1}$ is Π_1-complete.
(4) $\mathrm{SAT}^{G\forall}_{pos}$ is Π_1-complete.

Proof. (1) $\varphi \in \mathrm{SAT}_1^{\mathbf{L}\forall}$ iff $(\neg\varphi) \notin \mathrm{TAUT}_{pos}^{\mathbf{L}\forall}$ and $\varphi \in \mathrm{TAUT}_{pos}^{\mathbf{L}\forall}$ iff $(\neg\varphi) \notin \mathrm{SAT}_1^{\mathbf{L}\forall}pos$.
(2) Similarly, with "1" and "pos" exchanged.
(3) $\varphi \in \mathrm{SAT}_1^{G\forall}$ iff $\{\varphi\}$ is consistent over G$\forall$; thus $\mathrm{SAT}_1^{G\forall}$ is Π_1. Moreover, $\varphi \in \mathrm{SAT}^{Bool\forall}$ iff (Crisp & φ) $\in \mathrm{SAT}_1^{G\forall}$, thus we have Π_1-completeness.

Remarks. (1) Little is known about $\mathrm{SAT}_{pos}^{\Pi\forall}$, $\mathrm{SAT}_1^{\Pi\forall}$.
(2) It would be interesting to investigate possible generalizations of our results to other many-valued logics.

References

1. Hjek, P. Metamathematics of fuzzy logic, Kluwer 1998 1, 2, 3, 4, 6, 6, 6, 7, 7, 7, 7, 7
2. Hjek P. Fuzzy logic from the logical point of view, in: Proc. SOFSEM95, Lect. Notes Comp. Sci. 1012, Springer 1995, 31-49 1
3. Ebbinghaus H. D. and Flum J. Finite model theory. Springer-Verlag 1995 1
4. Papadimitriu C. H. Computational Complexity. Addison-Wesley 1994. 2
5. Rogers H. Jr. Theory of recursive functions and effective computability. McGraw-Hill 1987. 2
6. Bosc P., Kacprzyk J.(ed.) Fuzines in Database Management Systems. Physica-Verlag Wien 1995 1
7. Trakhtenbrot B. A. Impossibility of an algorithm for the decision problem on finite classes. Doklady Akademii Nauk SSSR 70 (1950) 509-572 (in Russian) 1

Descriptive Complexity, Lower Bounds and Linear Time

Thomas Schwentick

Johannes Gutenberg-Universität Mainz
tick@informatik.uni-mainz.de
http://www.Informatik.Uni-Mainz.DE/ tick/

Abstract. This paper surveys two related lines of research:
- Logical characterizations of (non-deterministic) linear time complexity classes, and
- non-expressibility results concerning sublogics of existential second-order logic.

Starting from Fagin's fundamental work there has been steady progress in both fields with the effect that the weakest logics that are used in characterizations of linear time complexity classes are closely related to the strongest logics for which inexpressibility proofs for concrete problems have been obtained. The paper sketches these developments and highlights their connections as well as the obstacles that prevent us from closing the remaining gap between both kinds of logics.

1 Introduction

The theory of computational complexity is quite successful in classifying computational problems with respect to their intrinsic consumption of resources. Unfortunately it is until now much less successful in proving that the complexity classes that are used for these classifications are different. Examples of separations of complexity classes are the hierarchy theorems [30,53,8,52] which show that more of the same kind of resource enables a Turing machine to solve more complicated problems. E.g., the class **PSPACE** of problems that can be solved with polynomial space is strictly larger than the class **LOGSPACE** of problems that can be solved with logarithmic space. In combination with the concept of hardness one can conclude lower bounds from these results. E.g., because the evaluation of quantified Boolean formulas is complete for **PSPACE**, such formulas can, in general not be evaluated with logarithmic space. Other separation results show that in nondeterministic time T, Turing machines can solve more problems than in deterministic time T [41] and that in space S a Turing machine can compute more than in time S [31]. The proofs of all the mentioned results combine simulation and diagonalization methods adopted from recursion theory. In particular, they give no general methods for proving precise lower bounds for *concrete* computational problems. Furthermore, it has been noticed early by Baker et al. [5], proved by so-called relativization, that these methods

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 9–28, 1999.

are not suitable to separate, e.g., **P** from **NP**. This insight has directed the attempts to prove lower bounds to concrete computational problems on restricted computational models. Such models include decision trees, branching programs, and Boolean circuits.
A different approach is taken by *descriptive complexity*. The idea of descriptive complexity is to measure the *syntactic complexity* of formulas that express a certain property instead of its computational complexity. This allows a "machine-independent" view at the complexity of problems and it makes the adaption of proof methods of mathematical logic to complexity questions possible. A fundamental result in that direction is *Fagin's Theorem* [17] which states that a property is in the class **NP** if and only if it can be expressed by an existential second-order formula.[1] Hence, a proof that the class of problems which can be expressed by such formulas is not closed under complementation would yield $\mathbf{NP} \neq \mathbf{coNP}$ and therefore $\mathbf{P} \neq \mathbf{NP}$. As a first step in that direction Fagin showed that the class of problems that can be expressed by a *monadic* existential second-order formula is not closed under complementation [18].
These two results of Fagin are the starting points for the two lines of research which we are going to sketch in this paper.

(1) The first line characterizes subclasses of **NP** by fragments of existential second order logic. In our context, the aim is to find characterizations with as simple as possible fragments of ESO to facilitate non-expressibility proofs. We are going to concentrate here on non-deterministic linear time classes.
(2) The second line tries to improve our ability to prove non-expressibility results to extensions of monadic existential second-order logic. The objective is to prove non-expressibility results for a logic that is strong enough to capture real computation. Here we focus on extensions of monadic existential second-order logic by *built-in relations* on the one hand and on fragments of binary existential second-order logic on the other hand.

For a general introduction to descriptive complexity we refer to the textbooks of Immerman and Ebbinghaus, Flum [32,14].
The paper is organized as follows. In Section 2 we give basic definitions and introduce some notation. In Section 3 we review the mentioned results of Fagin in more detail. In section 4 we give a survey of logical characterizations of linear time classes and in Section 5 we describe some related non-expressibility results. We give a short conclusion in Section 6.
I would like to thank Nicole Schweikardt for helping me to prepare this manuscript and an anonymous referee for many useful suggestions.

2 Definitions and Notations

2.1 Finite Structures

Whereas computational complexity usually talks about sets of strings, descriptive complexity uses *finite structures*. One benefit of this approach is that finite

[1] Precise definitions are given in Section 2.

structures allow in many cases a more natural representation of problems. For example it allows to represent a graph G by the set V of its vertices together with a binary relation E over V, which mirrors its adjacency matrix. In general, a finite structure consists of

- a *universe* U, i.e., a set of basic objects,
- some constants, i.e., elements from U,
- some relations over U, and
- some functions over U.

We will usually assume that U is of the form $\{1, \ldots, n\}$ for some natural number n. We denote with $|A|$ the size of the universe of finite structure A. Usually we are only interested in sets of structures of the same kind, i.e. with the same number and arity of relations, functions and constants. This is formalized by the notion of a *signature*. A signature (or *vocabulary*) $\tau = (R_1, \ldots, R_k, f_1, \ldots, f_l, c_1, \ldots, c_m)$ is a tuple of relation symbols, function symbols and constant symbols, where for each relation symbol and function symbol there is an associated arity $a(R_i)$ ($a(f_i)$ resp.). For example, the signature of the before mentioned representation of graphs is (E), where E is a binary relation symbol.
We can also represent strings as finite structures, as follows.[2] The universe of a string $w = w_1 \cdots w_n$ over an alphabet Σ consist of the set $\{1, \ldots, n\}$ of *positions* in w. The letters in w are represented by unary relations Q_σ, one for each $\sigma \in \Sigma$, such that $i \in Q_\sigma$ iff $w_i = \sigma$. As we will see, a logical formula is, in general, only allowed to refer to w via the relations, functions and constants in its representation. In particular, unless otherwise stated, they are not allowed to use any "arithmetical knowledge" about the natural numbers (like their order or how they are added). Therefore we need an explicit information about the order of the elements of the universe as part of the finite structure representation of w. There are various possibilities, e.g. by a successor relation, a successor function or a linear order relation. As an example, a signature for strings over alphabet $\{a, b, c\}$ would be $(\text{succ}, \min, \max, Q_a, Q_b, Q_c)$, where succ is a binary relation symbol, min and max are constant symbols and Q_a, Q_b and Q_c are unary relation symbols. The string *abcab* would then be represented by the finite structure $\langle \{1, \ldots, 5\}, \text{succ} = \{(1,2),(2,3),(3,4),(4,5)\}, \min = 1, \max = 5, Q_a = \{1,4\}, Q_b = \{2,5\}, Q_c = \{3\} \rangle$.

2.2 Formulas

We presume a basic knowledge about syntax and semantics of first-order formulas. An example of a first-order formula φ is $\forall x \, \forall y \, [\text{succ}(x,y) \wedge Q_a(x)] \rightarrow Q_b(y)$ which holds in a string if and only if in this string every a is followed by a b. In particular, it holds in the example string *abcab*. We write $w \models \varphi$ to express the fact that a formula φ holds in a string w.[3]

[2] In Section 4 we will see another way to represent strings.

[3] This notation is a bit sloppy. w is meant here as an abbreviation for the finite structure exhibited above. We are going to freely use this kind of identification of a string and its finite structure

Second-order logic allows the (existential or universal) quantification over relations. *Existential second-order logic (ESO)* consists of all second-order formulas in prenex form[4] in which second-order quantification is only existential. As an example, the formula

$$\exists X\ (\exists x, y\ X(x) \wedge \neg X(y)) \wedge (\forall x, y\ [X(x) \wedge \neg X(y)] \rightarrow \neg E(x, y))$$

expresses that a graph is *not connected*. Intuitively, the formula says that the vertices of the graph can be coloured with two colours, say blue and red, such that there are no edges between a blue and a red vertex, but there exists at least one vertex of each colour. As in this formula, we use uppercase letters to denote relational variables and lowercase letters to denote individual variables. This formula has a particular form, as X is a unary relation symbol. ESO formulas that only quantify over unary relations are called *monadic* ESO (MESO) formulas. We say that a formula φ characterizes a set L of finite structures if, for every structure A (of the appropriate signature) it holds that $A \models \varphi \iff A \in L$.

3 Descriptive Complexity

In this section we are going to have a closer look at the two theorems of Fagin that were already mentioned in the introduction. We start with the characterization of **NP**.

Theorem 1 (Fagin 1974). *A set of strings L is in* **NP** *if and only if L can be characterized by an existential second-order formula.*

Proof (Sketch). We only give the main ideas of the proof. Let us first assume that L is characterized by an ESO formula $\Phi = \exists R_1, \ldots, R_k \varphi$, where φ is first-order. Let $w = w_1 \cdots w_n$ be a string. We can identify l-tuples t over the universe $\{1, \ldots, n\}$ with natural numbers in $\{1, \ldots, n^l\}$ in a natural way. E.g., we can define $t(j)$ to be the j-th tuple in the lexicographic order of all l-tuples over $\{1, \ldots, n\}$. In particular $t(1) = (1, \ldots, 1)$, $t(2) = (1, \ldots, 2)$ and $t(n^l) = (n, \ldots, n)$. We can represent an l-ary relation R over $\{1, \ldots, n\}$ by a binary string v of length n^l by setting $v_j = 1$ iff $t(j) \in R$. Now it is easy to see that a non-deterministic Turing machine can evaluate Φ by first guessing (strings that represent) relations $R_1, \ldots, R_k$ and then evaluating the first-order formula φ on the structure $\langle w, R_1, \ldots, R_k \rangle$. The former involves guessing $O(kn^l)$ bits, if l is the maximum arity of a relation R_i and the evaluation of a first-order formula with quantifier-depth d can be done in time $O(n^d)$ (and d is fixed for L), so we can conclude that $L \in$ **NP**.
Now let $L \in$ **NP** and M be a non-deterministic Turing machine which accepts L in time $< n^k$. We assume w.l.o.g. that M is a one-tape Turing machine that is sufficiently normalized. We think of the start configuration of M as a tuple $(q_0, 1, v_0)$, where v_0 is a string of length n^k that consists of the input string w

[4] In particular, all second-order quantifiers are in front of everything else.

padded by blank symbols, hence $v_0 = w\sqcup^{n^k-n}$. q_0 is the initial state of M and the 1 indicates that the head of M is at the first position of v_0. In general a configuration C of M consists of a tuple (q, p, v), where q is a state of M, v is a string of length n^k and p is a position in v, i.e., a number in $\{1, \ldots, n^k\}$. We say that configuration C' is a successor of configuration C, if C' describes a situation which can be obtained from C by one step of M. Note that, as M is non-deterministic, there might exist more than one successor configurations of a configuration. As the behaviour of Turing machines is very local, the strings v, v' of two successive configurations can only differ in one position. Furthermore it is easy to test, given M, whether a configuration C' is a successor of a configuration C'.
A whole computation of M can now be viewed as a sequence of n^k configurations. The main idea of the proof is to encode such sequences into relations over $\{1, \ldots, n\}$. Let Q be the set of states of M and let Γ be the tape alphabet of M. We use the following relations.

- For every $\sigma \in \Gamma$, a $2k$-ary relation R_σ with the intension $(\overline{i}, \overline{j}) \in R_\sigma$ if in the i-th configuration of the computation there is a σ at position j. Here we write $\overline{i}$ for the k-tuple representation of i.
- For every $q \in Q$, a k-ary relation R_q with the intension $(\overline{i}) \in R_q$ if in the i-th configuration of the computation M is in state q.
- A $2k$-ary relation R_h with the intension $(\overline{i}, \overline{j}) \in R_h$ if in the i-th configuration of the computation the head of M is at position j.

It remains to show that there is a first-order formula φ which holds on $\langle w, (R_\sigma)_{\sigma\in\Sigma}, (R_q)_{q\in Q}, R_h\rangle$ if and only if the relations represent a correct and accepting computation of M on input w. This is the more technical part of the proof which we omit. In the end we get a formula $\exists(R_\sigma)_{\sigma\in\Sigma}, (R_q)_{q\in Q}, R_h\varphi$ which characterizes L.

To be able to discuss the proof of the second theorem of Fagin we need some preparation. *Ehrenfeucht games* [21,15] are two person games on pairs of (in our context finite) structures. The two players are often named as the *spoiler* and the *duplicator*. Intuitively, in the game on structures A and B, the spoiler tries to show that A and B are different, whereas the duplicator tries to make them look alike. The game consists of a number k of rounds. In each round i the spoiler selects first an element a_i of A or an element b_i of B. The duplicator then answers by correspondingly selecting an element b_i of B or a_i of A. In the end, the duplicator has won the game if the substructures of A and B that are induced by the selected vertices are isomorphic under the mapping which maps a_i to b_i, for every $i \leq k$, otherwise the spoiler has won. The usefulness of such games is connected with the following result.

Theorem 2 (Ehrenfeucht 61, Fraïssé 54). *A set L of structures can be characterized by a first-order formula of quantifier-depth k if and only if, for every $A \in L$ and every $B \notin L$, the spoiler has a winning strategy in the k-round Ehrenfeucht game on A and B.*

There is not only a correspondence between the number of rounds in the game and the quantifier depth of the respective formula. In a precise way, rounds in which the spoiler chooses a vertex in A correspond to existential quantifiers and rounds in which he chooses a vertex in B correspond to universal quantifiers. For details, see [14]. As we want to prove non-expressibility results, we are especially interested in the following easy consequence of this theorem. If, for every k, there are structures $A \in L$ and $B \notin L$ such that the duplicator has a winning strategy in the k-round Ehrenfeucht game on A and B then L cannot be characterized by a first-order formula.

There are a lot of variants of Ehrenfeucht games for all kinds of logics. We are going to consider Ehrenfeucht games for fragments of existential second-order logics. It is straightforward to extend the (first-order) Ehrenfeucht game to such logics. E.g., to get a game for monadic ESO formulas that quantify over l relations, we would simply extend the game by an additional round in the beginning, in which the spoiler selects l unary relations of A and the duplicator selects l unary relations of B.[5] Unfortunately this game is hard to win for the duplicator. In fact, the original proof [18] of Theorem 4 is quite involved. It was a major breakthrough when Ajtai and Fagin [2] invented a game that is much easier to play for the duplicator, as she is allowed to choose B *after* the duplicator has coloured A. It came as a surprise, but is actually not hard to prove, that this game still characterizes monadic ESO. The rules of the (l, k)-*Ajtai-Fagin game* on a set L of structures are as follows.

- The duplicator first selects a structure $A \in L$.
- The spoiler colours it with l colours.
- The duplicator selects a structure $B \notin L$ and colours it with l colours.
- Finally they play a k round (first-order) Ehrenfeucht game on the two coloured structures.

Theorem 3 (Ajtai, Fagin 90). *A set L of structures can be characterized by a monadic existential second-order formula if and only if, for some l and k, the spoiler has a winning strategy in the (l, k)-Ajtai-Fagin game on L.*

The invention of the Ajtai-Fagin game did not only lead to proofs of extensions of Fagin's non-expressibility theorem but actually simplified the proof of the theorem itself significantly. We are now prepared to state it and give a sketch of its proof.

Theorem 4. *The set of undirected, connected graphs cannot be characterized by a monadic existential second-order formula.*

[5] These l unary relations associate an l-tuple with every vertex v. There are 2^l such tuples. It is often more convenient to think of these tuples as 2^l colours. In this view, each element has one and only one colour. We adopt this view in the following considerations. Henceforth, l denotes the number of colours, which represent $\log l$ unary relations.

Proof (Sketch). The presentation follows essentially [20]. It is sufficient to show that, for every l and k, the duplicator has a winning strategy in the (l, k)-Ajtai-Fagin game on the class L of connected graphs. Let l and k be given. We write p for $2^k + 2$. We choose n large enough such that in every sequence of n elements that are coloured with l different colours there must exist 2 disjoint subsequences of length $2p$ that are coloured identically. A straightforward calculation shows that $n = 2p(l^{2p} + 1)$ is sufficient for this purpose.
Let G be the graph with vertex set $\{1, \ldots, n\}$ and edges between i and $i + 1$, for each $i \in \{1, \ldots, n - 1\}$, and between n and 1. G is a cycle, in particular it is connected. Let G be coloured by the spoiler with l colours. As n was chosen appropriately, there are 2 disjoint intervals starting at vertices, say $r + 1$ and $s + 1$, such that, w.l.o.g. $r + 2p < s$, $s + 2p \leq n$ and, for every i, $i \in \{1, \ldots, 2p\}$ the vertices $r + i$ and $s + i$ have the same colour. The duplicator chooses a graph G' which is almost the same as G, in particular, all vertices are coloured in the same way, but instead of the edges $(r + p, r + p + 1)$ and $(s + p, s + p + 1)$ it has edges $(s + p, r + p + 1)$ and $(r + p, s + p + 1)$. As Figure 5 illustrates, G' is not connected.

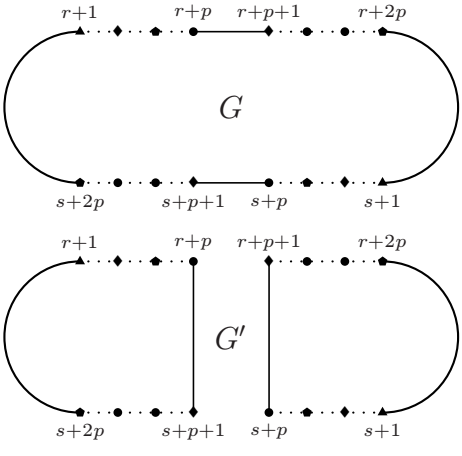


Fig. 1. The graphs G and G'. Colours are indicated by the different shapes of the vertices.

It is now straightforward, by using either the Hanf method [20] or the gap method [48], to prove that the duplicator has a winning strategy for the k-round first-order game on the two coloured structures.

– The Hanf method can be applied because in both graphs the same p-neighbourhoods occur with the same frequency. Here, a p-neighbourhood is a subgraph which is induced by all vertices that have distance $\leq p$ from some vertex v.

– The gap method can be applied as follows. Let us call the subgraphs of G and G' that are induced by the vertices $r+p, r+p+1, s+p, s+p+1$ the *inner subgraphs* and the subgraphs that are induced by all vertices except $\{s{+}1, \ldots, s{+}2p\}$ and $\{r{+}1, \ldots, r{+}2p\}$ the *outer subgraphs*. In the beginning of the game the inner subgraphs as well as the outer subgraphs are isomorphic and their distance is $p - 1 > 2^k$. During the game the inner and the outer graph may be extended by the vertices that are chosen by the spoiler and the duplicator. The proof now relies on the fact that the duplicator is able to maintain a gap (of size $> 2^i$, when there are still i rounds to play) between the inner and the outer subgraph, while still keeping them isomorphic.

The two theorems of Fagin give an upper and a lower bound for a logic with the two desired properties, namely a logic which is strong enough to capture a meaningful complexity class and which is weak enough to allow non-expressibility results. The rest of the paper deals with the process of making the gap between these two bounds, ESO logic and monadic ESO logic, smaller.

4 Linear Time

In the definition of the main complexity classes like (deterministic or non-deterministic) polynomial time or polynomial space, the choice of the underlying machine model is not very crucial. The definition does not depend on whether one chooses (1-tape or multi-tape or multi-dimensional) Turing machines or RAMs. As long as the cost associated with a computation is reasonable, all definitions define the same classes (with the notable exception of the non-reasonable unit-cost model for RAMs that are allowed to multiply numbers). Linear time, on the other hand, is a much more delicate notion. For instance, as in algorithm design, it is very sensitive to the representation of inputs and to changes of the computational model. It is hard to define a robust notion of linear time. Some authors tried to circumvent this problem by considering so-called *quasi-linear time*, i.e., time $O(n\text{polylog}(n))$ [44,29,22]. But there is even no general agreement whether linear time on Turing machines is too weak or too powerful [42].
In a series of articles [25,24,26] Grandjean invented a very reasonable formalization of (deterministic as well as non-deterministic) linear time. Before we get into the details of Grandjean's definitions let us first have a closer look at why linear time on Turing machines ($\mathbf{DTIME}(n)$) has little to do with linear time algorithms on, say, graphs. First of all, such algorithms rely usually on more sophisticated representations of the input graphs, like adjacency lists. Such lists are basically pointer structures, and the algorithms usually make excessive use of "pointer jumping". As the movements of the heads of a Turing machine are only local, it is hard to see how such "pointer jumping" algorithms could be simulated on a Turing machine in linear time. On the other hand, Random Access Machines are obviously very well suited to perform such algorithms. The definition of Grandjean's linear time classes is based on the idea of representing strings by pointer structures in analogy to efficient graph representations. In a

first step a string $w = w_1 \cdots w_n$ of length n is partitioned into $m := \lceil 2n/\log n \rceil$ pieces of length $\frac{1}{2}\log n$. Then such a partitioned string is encoded as a unary function $f : \{1, \ldots, m\} \to \{0, \ldots, \lceil \sqrt{m} \rceil\}$ by defining $f(i)$ to be the number which is encoded[6] by the i-th piece of the (partitioned) string w. A computation is linear time if it needs only a linear number of steps in m. The second characteristic feature of Grandjean's model is that during a computation only numbers of value $O(m)$ are allowed.[7] The model is quite robust with respect to arithmetical operations. We can choose addition and subtraction as the basic arithmetical operations. Allowing multiplication would not change the computational power (in our context). For more details of Grandjean's RAM model we refer to [26]. With **DLIN** and **NLIN** we denote the class of problems that can be computed on such a deterministic (non-deterministic, resp.) RAM in linear time. Both classes are quite robust and seem to be very reasonable formalizations of the intuitive notion of linear time.
Figure 7 shows the known inclusions between **DLIN**, **NLIN** and the linear time classes for (multi-tape) Turing machines, **DTIME**(n) and **NTIME**(n).

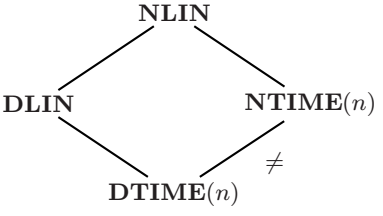


Fig. 2. The known inclusions between the considered linear time classes.

It comes as no surprise that we need a non-standard representation of strings as finite structures in order to get logical characterizations of **NLIN** and **DLIN**. A binary string $w = w_1 \cdots w_n$ is represented as a finite structure with universe $\{0, \ldots, m\}$, where $m = \lceil 2n/\log n \rceil$, the natural successor function succ and a unary function g_w, where $g_w(i)$ is the number that is encoded by the i-th piece of w, as before. With this representation in mind, **NLIN** can be characterized as follows.

Theorem 5. *A set of strings is in* **NLIN** *if and only if it is characterized by a formula*

$$\exists f_1, \ldots, f_k \forall x\, \varphi,$$

where the f_i are unary function symbols, x is one single individual variable and φ is a quantifier-free formula.

[6] via the dyadic encoding $0 \cdots 0 \to 0$, $0 \cdots 1 \to 1$, and so on.

[7] Although this is not so essential in the non-deterministic case. Here, allowing polynomial values makes no difference.

Proof (Sketch). Before we sketch the proof, we have a closer look at the underlying RAM model. For our purposes, a RAM has two accumulators A and B and a sequence $(R_i)_{i\geq 0}$ of registers. The registers R_i can be accessed via the accumulators by instructions $R_A := B$ and $B := R_A$ which write the value of B into the register whose number is in A and reads the value of the register whose number is in A into B, respectively. We assume for simplicity that during a computation only numbers $\leq m$ occur, hence only registers in $\{R_0, \ldots, R_m\}$ are ever used. It can be shown that the defined classes do not change, if we allow a fixed number of sequences of registers instead of only one such sequence. Nondeterminism is introduced to this model by allowing to guess numbers (of size $\leq m$).

The if-part of the proof is relatively straightforward. The RAM guesses unary functions $f_1, \ldots, f_k$ by using k sequences of registers and verifies for every $i \in \{0, \ldots, m\}$ that φ holds. As, for each i, the evaluation of φ can be done in constant time this is a (non-deterministic) linear time computation. It should be noted that this part of the proof is quite analogous to the respective part of the proof of Theorem 1.

The only-if-part needs some special care. Fagin's proof made use of guessing a quadratic amount of information (namely $2k$-ary relations for n^k-time computations). Here, we have to be a bit more economical. Let M be a non-deterministic RAM that decides a language L in linear time. A computation of M is encoded into the following functions with the indicated intended meaning.[8]

- $f_A(i)$ is the value of accumulator A after step i.
- $f_B(i)$ is the value of accumulator B after step i.
- $f_I(i)$ is the number of the instruction that is performed in step i.
- $f_R^<(i)$ is the value of register $R_{f_A(i)}$ *before* step i.
- $f_R^>(i)$ is the value of register $R_{f_A(i)}$ *after* step i.

To verify that such functions encode a valid computation, the quantifier-free formula φ has to test, among other things, that $f_R^<$, $f_R^>$ and f_A are consistent. If $i < j$, $f_A(i) = f_A(j)$ and $f_A(s) \neq f_A(i)$, for every s, $i < s < j$, then it must hold $f_R^>(i) = f_R^<(j)$. Intuitively, the formula must be able to find the matching i for a given j, i.e., the last time step i at which A had the same value as in step j (unless j is the first such time step, which can easily be detected, and which means that $f_R^<(j)$ has to be $g_w(f_A(j))$, the respective input value). A quantifier-free formula surely cannot do this. But it can be done by guessing another function f_l which encodes the lexicographic order on all pairs $(f_A(j), j)$. More precisely, $f_l(0)$ is the j for which $(f_A(j), j)$ is maximal in the lexicographic order of such pairs, $f_l(1)$ is the second pair, and so on. Let f_l' denote the inverse of f_l. Now, for each j, $f_l(\mathrm{succ}(f_l'(j)))$ is the matching i. By guessing some more unary functions the arithmetic operations can be verified, too. For the details of the proof we refer to [27].

[8] Of course this description works only if the computation lasts less than m steps. The general case is a straightforward extension.

This result does not only give a surprisingly clean characterization of the **NLIN**, thereby contributing to the robustness of this class, it also paves the way for a concrete lower bound result: We consider pairs (A, k), where A is the encoding of a finite automaton, in which the specification of the transition function δ_A is possibly incomplete. Let RISA[9] consists of all pairs (A, k) such that δ_A can be extended in such a way that the resulting automaton has an equivalent automaton with at most k states.

Theorem 6 (Grandjean 88).

$$RISA \notin \mathbf{DTIME}(n)$$

Proof (Sketch). The theorem is implied by the two following facts.[10]

- RISA is complete for **NLIN** under **DTIME**(n) reductions [25].
- **DTIME**$(n) \subsetneq$ **NTIME**$(n) \subseteq$ **NLIN** [41,25].

There are several refinements of this result. Durand and Ranaivoson [13] show that instead of unary functions one can quantify over one single binary relation of bounded outdegree. Olive [40] shows that the quantifier-free formula φ can be normalized strongly. In fact, it is enough to consider formulas of the form $\bigwedge_i t_i(x) = t'_i(x)$, where the t_i and t'_i are terms over $\{\text{succ}, f_1, \ldots, f_k\}$. Grandjean and Olive [27] prove that every set of strings in **NLIN** can be characterized by a monadic ESO formula in the presence of a built-in addition. For the definition of expressibility in the presence of built-in relations we refer to Section 5. Schwentick [49] gives algebraic and logical characterizations for **DLIN**, which are more complicated than the characterization of **NLIN**. In particular, it is not clear how they could be used to prove lower bound results. Grandjean and Schwentick [28] give simpler characterizations of **DLIN** and a natural problem which is complete for this class under **DTIME**(n) reductions.

We now turn to logical characterizations of linear time on Turing machines. As will become apparent in the next section, the techniques that are available for proving non-expressibility results seem not to be suitable for the formulas that are used in the characterization of **NLIN**. This is because the unrestricted choice of unary functions allows the spoiler to change the topology of the given structure strongly. As we want to close the gap between logics that capture complexity classes and logics that allow non-expressibility results, we go one step further and consider linear time on Turing machines.
It was already shown by Lynch in [36,38] that **NTIME**(n^k) is captured by ESO logic, in which quantification is restricted to k-ary relations (k-ary ESO, for short) and that **NTIME**(n) is captured by monadic ESO logic with addition.

[9] RISA stands for *Reduction of incompletely specified automata.*
[10] In the original proof it was shown that RISA is hard for **NTIME**(n) under **DTIME**(n) reductions [23].

That a class is *captured* by a logic shall mean that every set in the class is characterized by a formula in the logic but not necessarily vice versa.
In [35] Lautemann et al. show that $\mathbf{NTIME}(n)$ can be characterized in a similar way as **NLIN**, by restricting the shape of the quantified functions. In this characterization strings are represented by finite structures in the straightforward way that was described in Section 2. Let us call a unary function f on $\{1, \ldots, n\}$ *decreasing*, if $f(1) = 1$ and $f(i) < i$, for every $i > 1$, and *non-crossing*, if $f(i) \leq j \leq i$ implies $f(i) \leq f(j) \leq i$.

Theorem 7. *A set L of strings is in $\mathbf{NTIME}(n)$ if and only if it can be characterized by a formula of the form*

$$\exists f_1, \ldots, f_k \forall x \, \varphi,$$

where

- *the f_i are unary function symbols,*
- *φ is quantifier-free,*
- *the quantification of functions is restricted to non-crossing, decreasing functions, and*
- *in φ it is not allowed that two different function symbols f_i, f_j occur in the same equation.*

Proof (Sketch). The only-if part makes use of a characterization of Turing machine computations by Book and Greibach [6]. They show that a set of strings is in $\mathbf{NTIME}(n)$ if and only if it can be recognized by a Turing machine M with the following properties.

- M has a read-only input-tape. Furthermore, in step i the head of the input tape is at position i.[11]
- M has three additional pushdown tapes.
- On inputs of length n M makes exactly n steps.

The movements of M on a pushdown tape can be encoded by a unary function f. For each $j \in \{1, \ldots, n\}$, $f(j)$ is the time step at which the top entry of the pushdown at step j, was pushed. Consequently, $f(f(j))$ is the time the second entry was pushed and so forth. It follows that every set in $\mathbf{NTIME}(n)$ can be characterized by a formula in which only 3 functions (plus some unary relations) are quantified. Furthermore, it is possible to restrict the f_i such that at most 2 elements of the universe have the same function value.
For the if-part the crucial observation is that, on the other hand, a Turing machine can guess a non-crossing, decreasing function with the help of a pushdown tape which has, at step j, from top to bottom, the entries $f(j)$, $f(f(j))$ and so forth. For details we refer to [35].

[11] Such a device is called an *online tape*.

The proof shows in particular that linear time on non-deterministic Turing machines with an on-line input tape and l pushdown tapes can be characterized by formulas of the type

$$\exists f_1, \ldots, f_l \, R_1, \ldots, R_m \forall x \, \varphi,$$

where the R_i are unary relation symbols. Maass et al. [39] have shown that with 3 pushdown tapes such machines are more powerful than with 2 pushdown tapes. Furthermore, with 2 pushdown tapes they have more power than with 1 pushdown tape (i.e., pushdown automata). Hence, by restricting the number of quantified functions, we get a strict three-level hierarchy whose levels are induced by one function, two functions and at least three functions, respectively.
There is a natural candidate for the separation between **NTIME**(n) and **NLIN**. Let $U = (u_1, \ldots, u_n)$ and $V = (v_1, \ldots, v_m)$ denote lists of 0-1-strings. Let CHECKSORT consist of all pairs (U, V) in which V contains exactly the same strings as U, but where V is sorted with respect to the lexigraphical order. It can be shown that CHECKSORT is in **NTIME**(n) if and only if **NTIME**$(n) =$ **NLIN**. This follows from the facts that CHECKSORT is in **NLIN** (even in **DLIN**) and that every set in **NLIN** can be recognized in linear time by a non-deterministic Turing-machine that is allowed to perform a constant number of sorting steps during its computation [25]. In a sorting step the strings that are written on a special sorting tape, separated by commas, are sorted in one step.
In a recent paper Eiter et al. [16] considered fragments of ESO that are defined by restricting the *first-order* quantifier prefix of formulas. For a string $Q \in \{\exists, \forall\}^*$ let ESO(Q) denote the set of ESO formulas in which the first-order prefix is of the form Q. They obtain the following classification.

- If Q is in $\exists^*\forall$ then ESO(Q) contains only star-free regular languages.
- If Q contains $\forall\exists$ or $\forall\forall$ and is contained in $\exists^*\forall\exists^*$ or $\exists^*\forall\forall$ then ESO(Q) can characterize exactly all regular languages.
- If Q contains $\forall\forall\forall$, $\forall\exists\forall$ or $\forall\forall\exists$ then ESO(Q) can characterize a certain **NP**-complete language. As one can show that in each of these logics it can be tested whether a given binary relation is the graph of a unary function, it turns out that these logics capture **NTIME**(n) (even **NLIN**).

5 Nonexpressibility

In this section we are going to survey non-expressibility results for two kinds of sublogics of ESO: monadic ESO with built-in relations and sublogics of ESO that allow the quantification of unary functions.

5.1 Monadic ESO

First of all we have to define the notion of built-in relations. In this subsection, the universe of a structure will always be of the form $\{1, \ldots, n\}$. Let $(R_n)_{n \geq 1}$ be a sequence of relations, such that R_n is a relation over $\{1, \ldots, n\}$ (and all relations

are of the same arity). We say that a set L of finite structures is characterized by a formula φ *in the presence of built-in relations* $(R_n)_{n\geq 1}$, if for every structure A (of the appropriate signature) it holds $\langle A, R_{|A|}\rangle \models \varphi \iff A \in L$.
Built-in relations, especially linear orders or arithmetic relations, play a crucial role in many characterizations of complexity classes. As an example, consider the formula

$$\exists X\, X(\min) \wedge X(\max) \wedge \forall x, y[\mathrm{succ}(x, y) \wedge X(x)] \rightarrow \neg X(y),$$

which expresses that a graph has an odd number of vertices. Intuitively, the formula says that there is a colouring of the vertices with two colours, say blue (X) and red ($\neg X$), that is alternating with respect to the successor relation and which colours the minimal and the maximal vertex blue. Here, the built-in relation is a successor relation.[12]
For strings it follows from Büchi's Theorem [7] that monadic ESO with built-in linear orders can characterize exactly all regular languages. On the other hand, we have mentioned before that MESO with built-in addition captures **NLIN** [27].
There is a whole series of papers that show non-expressibility of graph connectivity for extensions of MESO logic by various kinds of built-in relations. The following table summarizes some of these results.

successor relations	de Rougemont [10]
relations of degree $(\log n)^{o(1)}$	Fagin, Stockmeyer, Vardi [20]
linear order	Schwentick [45]
relations of degree $n^{o(1)}$	Schwentick [47]
trees	Kreidler, Seese [33]
$n + n^{o(1)}$ edges	Kreidler, Seese [33]
planar graphs	Kreidler Seese [34]
K_l-free	Kreidler, Seese [34]

Table 1. A list of built-in relations that do not enable MESO logic to express graph connectivity.

Most of these results can be essentially proved by adapting Fagin's idea of "one cycle vs. two cycles". As a first step, let us consider built-in relations $(R_n)_{n\geq 1}$ in which one can find many points that are far from each other. More precisely, let us call built-in relations *separated*, if they have the following property.

> For each m and d there exists an n such that there is a subset $V \subseteq \{1, \ldots, n\}$ of size at least m such that two different elements of V have

[12] In the context of well-structured built-in relations, like successor, linear order or addition, a different point of view is convenient. Instead of fixing one built-in relation for each structure size and varying the graphs, one can think of fixing a graph and varying all possible built-in relations, e.g., all linear orders.

at least distance d in the finite structure with universe $\{1, \ldots, n\}$ and the relation R_n.[13]

If built-in relations $(R_n)_{n\geq 1}$ are separated they do not enable MESO to characterize Graph Connectivity. The recent results of Kreidler and Seese make use of the fact that this still holds, if $(R_n)_{n\geq 1}$ become separated after deleting a constant number of elements. More precisely, call built-in relations $(R_n)_{n\geq 1}$ *k-separable*, if they have the following property.

For each m and d there exists an n such that there are sets $V, S \subseteq \{1, \ldots, n\}$ with $|V| = m$ and $|S| = k$, such that two different elements of V have distance at least d in the finite structure with universe $\{1, \ldots, n\} - S$ and the relation that is induced by R_n on this universe.

We call built-in relations *separable*, if they are k-separable, for some k.

Theorem 8. *Graph connectivity can not be expressed by monadic existential second-order formulas even in the presence of separable built-in relations.*

It is still open whether graph connectivity can be expressed by a MESO formula in the presence of addition. However, as this logic is at least as powerful as the unary function fragment of ESO [27], it can be shown that such formulas can express whether a graph has a connected component of size $\geq n/\log n$.
Ajtai and Fagin show that directed reachability cannot be expressed by a monadic ESO formula, even in the presence of several kinds of built-in relations [2]. Cosmadakis showed how non-expressibility results can be transfered to other problems via suitable reductions [9]. For a generalization of these reductions we refer also to [46].
With the exception of linear orders all the built-in relations that are listed in Table 1 are in fact separable. Hence separable built-in relations are at the limit of our ability to prove non-expressibility results concerning monadic ESO. Let us call built-in relations *strong*, if they enable monadic ESO to capture **NLIN** and *weak* if they do not enable monadic ESO to express graph connectivity. It should be noted that strongness is concerned with strings whereas weakness is concerned with graphs. In [50] it was shown that the gap between weak and strong built-in relations is not very large. For instance,

- for every $\epsilon > 0$, there are strong built-in relations of degree n^ϵ.
- for every $\epsilon > 0$, there are strong built-in relations with $n + n^\epsilon$ edges.

At the moment, very little is known about how to show the existence of a winning strategy for the duplicator on structures that are very dense. A notable exception is an article of Lynch in which he uses Ehrenfeucht games to prove non-expressibility results for first-order logic in the presence of a built-in addition

[13] Even if R_n is not binary the notion of distance can be defined by setting $d(a, b) = 1$, if $a \neq b$ and a and b occur in the same tuple of R_n and extending this definition in the obvious way.

[37]. By using similar methods as Lynch Ruhl showed very recently that first-order logic with addition and unary counting quantifiers is not able to express connectivity [43].
The effort to prove non-expressibility results for stronger and stronger extensions of monadic ESO has led to the development of many useful tools for dealing with Ehrenfeucht games. Besides the already mentioned Ajtai-Fagin game, there have been invented several ways to simplify the proof of the existence of a winning strategy for the duplicator in the first-order Ehrenfeucht game. We refer the interested reader to [20,4,45,51] and, for a survey to [19].
A new development in the area of monadic ESO was initiated by Ajtai et al. [3]. They consider various *closures* of monadic ESO, e.g., formulas that allow first-order quantification in front of existential monadic second-order quantifiers and they prove very nice separation results for some of the resulting logics.

5.2 Unary Functions

In the Ehrenfeucht game that is associated with monadic ESO logic the spoiler is not allowed to define new connections in the two structures. As we have seen in the last subsection, built-in relations that tie all parts of the structures close together make the Ehrenfeucht game difficult for the duplicator. This statement holds even more if the spoiler is allowed to choose such (non-unary) relations. Therefore it is no surprise that playing the Ehrenfeucht game for, say, binary ESO logic is hard and that there are only few non-expressibility results. The most important result here was given Ajtai [1] who showed that for each k there is a property that can be expressed by $k+1$-ary ESO formulas but not by k-ary ESO formulas. The separating set is, for every k, the set of all $k+1$-ary relations that contain an odd number of tuples. It is still open for which $k > 1$ this separation holds for graphs, too. Ajtai's result has also been used as a tool to separate subclasses of binary ESO. By restricting the relations that are quantified in binary ESO formulas Durand et al. investigated the fine structure of binary ESO [12]. They established, with respect to graphs, the following strict four level hierarchy.

- binary relations, partial orders
- unary functions, equivalence relations, linear orders, graphs of bounded out-degree
- permutations, successor relations, graphs of bounded in- and out-degree
- unary relations

The separations between the first two levels as well as between the second and the third level used the mentioned results of Ajtai.
Durand et al. showed that, by quantifying over two unary functions one can express more properties than with only one function [11].

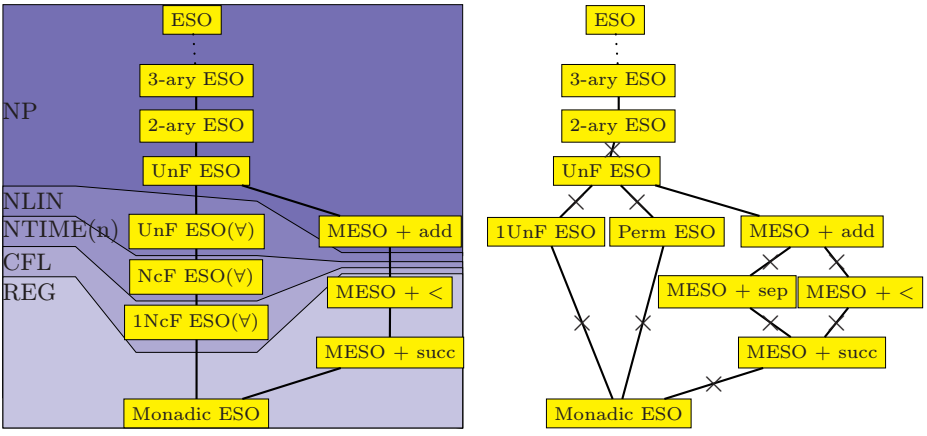


Fig. 3. Logical characterizations of complexity classes by fragments of ESO logic (left diagram) and separations between sublogics of ESO (right diagram. The underlying structures are graphs (left) and strings (right). (UnF ≡ unary Functions, NcF ≡ non-crossing functions, (∀) ≡ 1 universal fo-quantifier)

6 Conclusion

Figure 3 summarizes some of the mentioned logical characterizations of complexity classes and non-expressibility results.
We tried to demonstrate that descriptive complexity is able to characterize even fine-grained differences between complexity classes. Furthermore we wanted to point out that the logics that are used to characterize linear time complexity classes are not very different from several logics for which there have been non-expressibility proofs for concrete problems. Nevertheless, there is still a gap, originating in the different underlying representations of structures (strings for characterizations, graphs for non-expressibility results) on one hand, and in the fact that there seems to be a border between dense and non-dense structures that is hard to overcome.
We conclude with two concrete open problems.

- Is CHECKSORT in **NTIME**(n)? (**DTIME**(n)?)
- Is graph connectivity expressible by MESO formulas in the presence of addition?

References

1. M. Ajtai. Σ^1_1 formulae on finite structures. *Ann. of Pure and Applied Logic*, 24:1–48, 1983. 24
2. M. Ajtai and R. Fagin. Reachability is harder for directed than for undirected finite graphs. *Journal of Symbolic Logic*, 55(1):113–150, 1990. 14, 23

3. M. Ajtai, R. Fagin, and L. Stockmeyer. The closure of monadic NP. IBM Research Report RJ 10092, 1997. 24
4. Sanjeev Arora and Ronald Fagin. On winning strategies in Ehrenfeucht–Fraïssé games. *Theoretical Computer Science*, 174(1–2):97–121, 1997. 24
5. Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM Journal on Computing*, 4(4):431–442, December 1975. 9
6. Ronald V. Book and Sheila A. Greibach. Quasi-realtime languages. *Mathematical System Theory*, 4(2):97–111, 1970. 20
7. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960. 22
8. Stephen A. Cook. A hierarchy for nondeterministic time complexity. *Journal of Computer and System Sciences*, 7(4):343–353, August 1973. 9
9. S. Cosmadakis. Logical reducibility and monadic NP. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 52–61, 1993. 23
10. M. de Rougemont. Second-order and inductive definability on finite structures. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 33:47–63, 1987. 22
11. Arnaud Durand, Ron Fagin, and Bernd Loescher. Spectra with only unary function symbols. In *11th Annual Conference of the EACSL, CSL '97*, pages 189–202, 1997. 24
12. Arnaud Durand, Clemens Lautemann, and Thomas Schwentick. Subclasses of binary NP. *Journal of Logic and Computation*, 8(2):189–207, April 1998. 24
13. Arnaud Durand and Solomampionona Ranaivoson. First-order spectra with one binary predicate. *Theoretical Computer Science*, 160(1–2):305–320, 10 June 1996. 19
14. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995. 10, 14
15. A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49:129–141, 1961. 13
16. Thomas Eiter, Georg Gottlob, and Yuri Gurevich. Existential second-order logic over strings. LICS '98, 1998. 21
17. R. Fagin. Generalized first–order spectra and polynomial–time recognizable sets. In R. M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings, Vol. 7*, pages 43–73, 1974. 10
18. R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975. 10, 14
19. R. Fagin. Easier ways to win logical games. In *Proceedings of the DIMACS Workshop on Finite Models and Descriptive Complexity*. American Mathematical Society, 1997. 24
20. R. Fagin, L. Stockmeyer, and M. Vardi. On monadic NP vs. monadic co-NP. *Information and Computation*, 120:78–92, 1995. Preliminary version appeared in 1993 IEEE Conference on Structure in Complexity Theory, pp. 19-30. 15, 15, 22, 24
21. R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publ. Sci. Univ. Alger. Sér. A*, 1:35–182, 1954. 13
22. E. Grädel. On the notion of linear time computability. *Int. J. Found. Comput. Sci.*, 1:295–307, 1990. 16
23. E. Grandjean. A natural NP-complete problem with a nontrivial lower bound. *SIAM Journal of Computing*, 17:786–809, 1988. 19
24. E. Grandjean. Invariance properties of RAMs and linear time. *Computational Complexity*, 4:62–106, 1994. 16

25. E. Grandjean. Linear time algorithms and NP-complete problems. *SIAM Journal of Computing*, 23:573–597, 1994. 16, 19, 19, 21
26. E. Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16:183–236, 1996. 16, 17
27. E. Grandjean and F. Olive. Monadic logical definability of NP–complete problems. *In Proc. 1994 of the Annual Conference of the EACSL*, pages 190–204, 1994. Extended version submitted. 18, 19, 22, 23
28. Etienne Grandjean and Thomas Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. In preparation, 1998. 19
29. Y. Gurevich and S. Shelah. Nearly linear time. In A. Meyer and M. Taitslin, editors, *Logic at Botik '89, Lecture Notes in Computer Science 363*, pages 108–118. Springer Verlag, 1989. 16
30. J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Transactions of the AMS*, 117:285–306, 1965. 9
31. John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM*, 24(2):332–337, April 1977. 9
32. Neil Immerman. *Descriptive Complexity*. Graduate Texts in Computer Science. Springer-verlag, New York, 1998. 10
33. M. Kreidler and D. Seese. Monadic NP and built-in trees. In *Proceedings CSL 96*, volume 1258 of *LNCS*, pages 260–274. Springer-Verlag, 1997. 22, 22
34. Martin Kreidler and Detlef Seese. Monadic NP and graph minors. CSL '98, 1998. 22, 22
35. Clemens Lautemann, Nicole Schweikardt, and Thomas Schwentick. A logical characterization of nondeterministic linear time on Turing machines. to be presented at STACS 99, 1998. 20, 20
36. J. F. Lynch. Complexity classes and theories of finite models. *Mathematical System Theory*, 15:127–144, 1982. 19
37. J. F. Lynch. On sets of relations definable by addition. *Journal of Symbolic Logic*, 47:659–668, 1982. 24
38. J. F. Lynch. The quantifier structure of sentences that characterize nondeterministic time complexity. *Computational Complexity*, 2:40–66, 1992. 19
39. Wolfgang Maass, Georg Schnitger, Endre Szemerédi, and György Turán. Two tapes versus one for off-line turing machines. *Computational Complexity*, 3(4):392–401, 1993. 21
40. F. Olive. A conjunctive logical characterization of nondeterministic linear time. In *11th Annual Conference of the EACSL, CSL '97*, pages 360–372, 1997. 19
41. Wolfgang J. Paul, Nicholas Pippenger, Endre Szemerédi, and William T. Trotter. On determinism versus non-determinism and related problems (preliminary version). In *24th Annual Symposium on Foundations of Computer Science*, pages 429–438, Tucson, Arizona, 7–9 November 1983. IEEE. 9, 19
42. K. Regan. Machine models and linear time complexity. *SIGACT News*, 24:4, Fall 1993. 16
43. Matthias Ruhl. Counting and addition cannot express deterministic transitive closure. 1998. 24
44. C. P. Schnorr. Satisfiability is quasilinear complete in NQL. *Journal of the ACM*, 25:136–145, 1978. 16
45. T. Schwentick. Graph connectivity and monadic NP. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 614–622, 1994. 22, 24
46. T. Schwentick. On winning Ehrenfeucht games and monadic NP. Doktorarbeit, Universität Mainz, 1994. 23

47. T. Schwentick. Graph connectivity, monadic NP and built-in relations of moderate degree. In *Proc. 22nd International Colloq. on Automata, Languages, and Programming*, pages 405–416, 1995. 22
48. T. Schwentick. On winning Ehrenfeucht games and monadic NP. *Annals of Pure and Applied Logic*, 79:61–92, 1996. 15
49. T. Schwentick. Algebraic and logical characterizations of deterministic linear time classes. In *Proc. 14th Symposium on Theoretical Aspects of Computer Science STACS 97*, pages 463–474, 1997. 19
50. T. Schwentick. Padding and the expressive power of existential second-order logics. In *11th Annual Conference of the EACSL, CSL '97*, pages 461–477, 1997. 23
51. T. Schwentick and K. Barthelmann. Local normal forms for first-order logic with applications to games and automata. In *Proc. 15th Symposium on Theoretical Aspects of Computer Science STACS 98*, pages 444–454, 1998. 24
52. J. I. Seiferas, M. J. Fischer, and A. R. Meyer. Refinements of the nondeterministic time and space hierarchies. In *14th Annual Symposium on Switching and Automata Theory*, pages 130–137, The University of Iowa, 15–17 October 1973. IEEE. 9
53. R. E. Stearns, J. Hartmanis, and P. M. Lewis II. Hierarchies of memory limited computations. In *Proceedings of the Sixth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 179–190. IEEE, 1965. 9

Testing of Finite State Systems

Mihalis Yannakakis and *David Lee*

Bell Laboratories, Lucent Technologies
Murray Hill, New Jersey
USA

ABSTRACT: Finite state machines have been used to model a wide variety of systems, including sequential circuits, and more recently, communication protocols. In testing problems we are given a system M, which we may test by providing inputs and observing the outputs produced. The goal is to design test sequences so that we can deduce desired information, such as the state of M, or whether M implements correctly a given specification machine S. In this paper we will discuss algorithmic work on basic testing problems for systems modeled by different types of finite state machines.

1. Introduction

Testing is an essential component of the software (and hardware) development cycle. Many control intensive systems are typically modeled by finite state machines. Examples include communication protocols in networks, switching features in telephony, and many others. Testing such systems is an important area that has attracted, and continues to attract, a lot of research and development activity both in industry and academia.

In testing problems, we are given a reactive system M (the "Implementation Under Test"), i.e. a system which takes inputs and produces outputs in response. The problem is to generate appropriate tests to apply to M in order to infer some desired unknown information about the system. such as the structure or the state of M. For example, a fundamental problem is the *conformance testing* or *fault detection* problem: Given a specification finite state machine S, test the implementation M (a "black box" observed through its input-output behavior) to check that it conforms to its specification. For example, S could be a protocol standard to which the system M is required to conform. Or, S could be the model of a feature (either a high level requirement model or a more detailed design model), and we wish to check that the feature has been implemented correctly.

The area of testing finite state systems has quite an extensive literature, starting from the mid 50's with Moore's seminal paper on "gedanken experiments" [Mo56]. In this paper Moore set up the basic testing framework, posed a number of fundamental problems (including for example the conformance testing and the machine and the

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 29-44, 1999.

state identification problems), and defined corresponding test sequences that address these problems, such as homing, distinguishing, and checking sequences. The following years saw the active development of automata theory including a number of papers on these problems (see [Ko78] for a survey). Precise bounds and characterizations were obtained for some of these sequences; yet others were not resolved until more recently, and some questions still remain open.

The early work was motivated mainly by automata theory and circuits. After a period of reduced activity during the 70's and early 80's, the problems resurfaced again and were taken up by the protocols community, where a large number of papers has been published motivated mainly by conformance testing of communications protocols.

In this paper we will summarize some of the basic theory with emphasis on the algorithmic work. The literature in this area is quite extensive, so we will only touch on some of the results here. For a more detailed survey of much of the material we refer to [LY96a].

The rest of this paper is organized as follows. After introducing briefly in the next section some of the underlying definitions, we discuss in Section 3 a number of testing problems for deterministic finite state machines: we define different types of test sequences for various purposes, and summarize the results on their existence, length, and associated algorithms and complexity issues. In Section 4 we discuss some optimization problems related to conformance testing and coverage of FSM's. Section 5 concerns Extended Finite State Machines (EFSM's). Finite state machines are a useful model of the control portions of protocols at a high level of abstraction. At a more detailed level, it is useful in practice to augment FSM's with variables, leading to an Extended Finite State Machine model, where transitions may depend on the values of variables and may modify the variables. In Section 6 we discuss some results on nondeterministic and probabilistic FSM's, and in Section 7 we conclude.

2. Preliminaries

Finite state systems can usually be modeled by *Mealy* machines that produce outputs on their state transitions after receiving inputs. There is a variant (the Moore model) in which outputs are produced at the states instead of the transitions; the theory is essentially the same for the two models.

Formally, a deterministic (Mealy) finite state machine (FSM) consists of a finite set S of states, a (finite) input alphabet I, output alphabet O, a state transition function δ: $S \times I \rightarrow S$, and an output function λ: $S \times I \rightarrow O$. When the machine is in a current state s in S and receives an input a from I it moves to the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$. Graphically, a FSM can be represented by its *state transition diagram*, a labeled directed graph whose nodes correspond to the states, the edges correspond to the state transitions, and each edge is labeled with a pair a/b, the input a and output b associated with the transition. (see

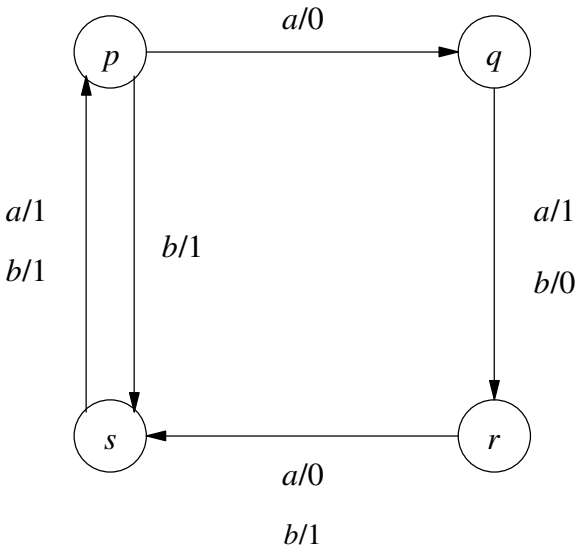


Figure 1

Fig. 1). We denote the number of states, inputs, and outputs by $n = |S|$, $p = |I|$, and $q = |O|$, respectively.

The transition function δ and output function λ can be extended as usual to input strings and sets of states. We assume familiarity with the basic theory of FSM's. In particular, recall that two states s, t are *equivalent* if for every input sequence x they produce the same output sequence, $\lambda(s, x) = \lambda(t, x)$. The equivalence relation among states can be computed efficiently. Two machines M and M' are *equivalent* iff for every state in M there is a corresponding equivalent state in M', and vice versa. Two machines are *isomorphic* if they are identical except for a renaming of states. Given a machine, we can "merge" equivalent states and construct a *minimized (reduced)* machine which is equivalent to the given machine and no two states are equivalent. The minimized machine is unique up to isomorphism.

In a testing problem we have a FSM M, about which we have only partial information, and wish to infer other missing information by providing inputs and observing the produced outputs. The problem is to design a test that allows us to deduce the desired information. There are two kinds of tests: *preset*, in which the input test sequence is determined ahead of time before the experiment starts, and *adaptive*, in which the input is determined adaptively online, i.e. the observed output symbols influence subsequent input symbols. A preset test is simply an input string. An adaptive test is formally a decision tree: a rooted tree whose internal nodes are labeled by input symbols, and the edges branching out of each node are labeled by distinct output symbols (see Fig. 2). Starting from the root of the tree, at each step of the test we

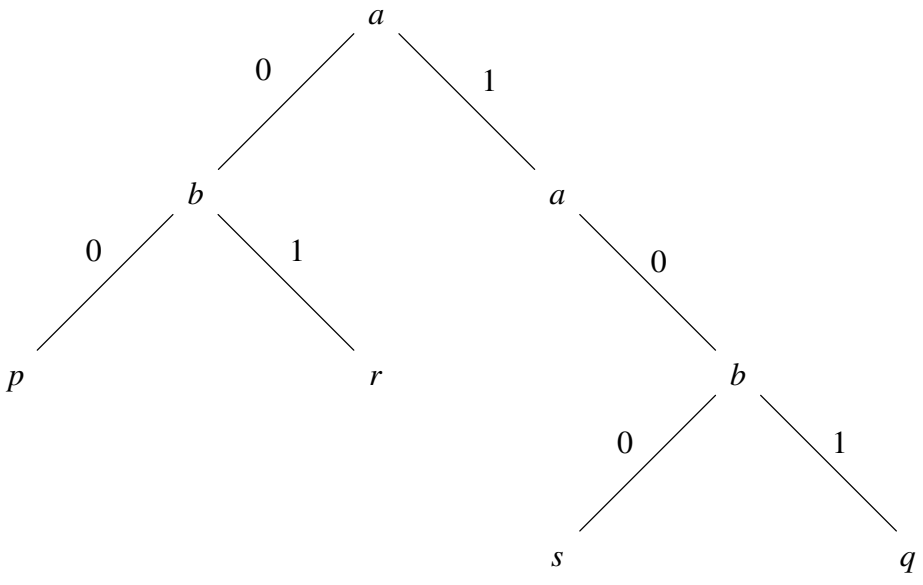


Figure 2

input the symbol that labels the current node, and depending on the output produced we move to the appropriate child. At a leaf a decision is made concerning the missing information.

In each testing problem the basic questions of interest are:
Existence: Is it possible to design a suitable test that achieves the desired goal?
Length: How long does it need to be?
Algorithms and Complexity: How hard is it to determine whether the desired tests exist and to generate them?

The above notion of testing is called sometimes *active* testing to indicate the fact that the tester probes actively the machine M under test. There is also a notion of *passive* testing, where the tester is simply a passive observer following passively both the inputs to the system M, generated independently by some other entity (i.e. by another component interacting with M), and the produced outputs. We will discuss mainly active testing, and touch briefly on passive testing at the end.

3. Deterministic Machines

In this section we summarize results on five basic problems for deterministic FSMs. In the first three problems we have a complete description of the machine M under test (i.e. its state transition diagram), but we do not know in which state it is in, i.e. its initial state. The objective of the test is to identify or verify the state before or

after the test. In the other two problems, we do not know the structure of M; the objective of the test is to identify or verify the state transition diagram of M.

3.1. State Identification and Verification Problems

It is assumed that the machine M is minimized; otherwise the state can be determined only up to equivalence.

Problem 1. Determine the final state of M after the test.

An input sequence that solves this problem is called a *homing* sequence. Thus, a (preset) homing sequence is an input sequence x such that for any two states s, t, if $\lambda(s, x) = \lambda(t, x)$ then also $\delta(s, x) = \delta(t, x)$. For example, a homing sequence for the FSM of Fig. 1 is ab: the output is either 11 or 00 or 01, and the corresponding final state is respectively s, r, p. The homing sequence problem was completely solved early on (see eg [Ko78]): Every minimized machine has a homing sequence of length at most $n(n-1)/2$, which can be constructed efficiently. Furthermore, this bound is optimal among both preset and adaptive sequences, i.e. there is a FSM whose shortest (preset or adaptive) homing sequence has length $n(n-1)/2$.

In a homing sequence we do not know the final state until we perform the test and we observe the output. There is a related type of sequence called a *synchronizing* sequence (sometimes called also a *reset* sequence) where the final state is independent of the output. That is, x is a synchronizing sequence if $\delta(s, x) = \delta(t, x)$ for all states s, t. Not every minimized machine has such a sequence; for example, the FSM of Fig. 1 does not have one. Given a FSM, it can be efficiently determined (in $O(pn^2)$ time) whether M has a synchronizing sequence; if it does, then one can always find efficiently such a sequence of length $O(n^3)$. On the other hand the best lower bound is $\Omega(n^2)$, i.e. there are machines that have a synchronizing sequence and the shortest one has length $\Omega(n^2)$. Closing this gap between the quadratic lower bound and the cubic upper bound is still an open problem.

Problem 2. Determine the initial state of the machine before the test starts (State Identification Problem).

An input sequence that solves this problem is called a *distinguishing* sequence. Thus, a preset distinguishing sequence is an input sequence x such that any two states s, t have $\lambda(s, x) \neq \lambda(t, x)$. Not every minimized FSM has a preset or adaptive distinguishing sequence. For example, the FSM of Fig. 1 does not have a preset distinguishing sequence. To see this, note that such a sequence cannot start with a b because then we will never be able to tell whether the machine started at state p or r. Furthermore, after any sequence of a's, the machine can be in one of the two states p, r (recall in a preset test we pick the input sequence without observing the output), thus we can never apply b for the same reason. However, a sequence of a's cannot distinguish between the initial states p and r. Although distinguishing sequences were defined early on by Moore, and used extensively since then, their complexity was not addressed until [LY94]. It is shown there that determining whether a FSM has

a preset distinguishing sequence is a PSPACE-complete problem. Furthermore, there are machines that have such a sequence but the shortest preset distinguishing sequence is of exponential length.

A FSM may not have a preset distinguishing sequence, but may have an adaptive one. The FSM of Fig. 1 is such an example, with the adaptive distinguishing sequence shown in Fig. 2. The leaves of the tree are labelled by the initial states of the machine. There are however minimized machines that don't have even an adaptive distinguishing sequence.

Sokolovskii showed that if there is an adaptive distinguishing sequence then there is one of quadratic length [So71]. His argument was nonconstructive (i.e. did not suggest an efficient algorithm). The complexity of the existence question was resolved in [LY94] which gave a polynomial time algorithm (more precisely, $O(pn \log n)$) that determines whether a given FSM has an adaptive distinguishing sequence. In this case one can construct such a sequence of length $n(n-1)/2$ (which is best possible in general) in time $O(pn^2)$.

Problem 3. The machine is supposed to start at a particular initial state s_0; verify that it is initially indeed in that state (State Verification Problem).

An input sequence that solves this problem is called a *UIO (Unique Input Output)* sequence for s_0. It is a sequence x that has the property that $\lambda(s_0, x) \neq \lambda(t, x)$ for any other state t. For a minimized FSM it is possible that all, some or none of the states have a UIO sequence. If the FSM has an adaptive distinguishing sequence then all its states have a UIO sequence; for example, a UIO sequence for the state s of the FSM of Fig. 1 is the sequence aab that labels the nodes on the path from the root to the leaf that decides that the initial state is s. Adaptiveness does not make a difference for the state verification problem.

UIO sequences have been studied more recently in the protocols community, in particular there is a large number of papers following [SD88] that design conformance tests using UIO sequences. In general, for a given FSM it is a PSPACE-complete problem to determine whether a state has a UIO sequence; furthermore, even if there is one, it is possible that the shortest such sequence is of exponential length [LY94].

The sequences defined in this subsection are useful also for the following problems.

3.2. Machine Identification and Verification Problems

In this setting we do not know the state diagram of the machine M that is being tested.

Problem 4. Determine the state diagram of M (Machine Identification Problem).

Some assumptions need to be made on M to permit a solution to the problem. It is typically assumed that we know (an upper bound on) the number of states n and of course the input alphabet, and that M is minimized (otherwise we can only determine it up to equivalence) and is strongly connected.

The machine identification problem was addressed by Moore in his original paper [Mo56]. He showed that under the above assumptions the machine M can be always identified, and he gave an exponential algorithm for it. Furthermore, he showed that an exponential length test is in general required both for preset and adaptive tests.

Work in the machine learning community has obtained positive results on the machine identification problem in a more relaxed model, where one has recourse to an oracle (the "teacher") that answers equivalence queries, i.e. one can ask the oracle whether M is equivalent to a conjectured machine S, and receive either an affirmative answer or a counterexample input string x that distinguished the two machines. In this model, Angluin gave a deterministic polynomial time identification (learning algorithm) in the case of machines with a reset capability, i.e., an input symbol r that takes every state to the same initial state [An87]. Rivest and Schapire devised a randomized polynomial time algorithm in the absence of a reset [RS89].

Problem 5. We are given the complete description of a "specification" machine S; determine whether M is equivalent to S (Machine Verification Problem).

This is the basic testing problem, usually called the Conformance Testing or Fault Detection Problem. An input test sequence that solves it is called a *checking* sequence. Again certain assumptions need to be made to ensure a solution. Typical assumptions are that the specification machine S is strongly connected and minimized, and the input alphabet of M is the same as S. Note that if S has a reset symbol, then its transitions are included in establishing the strong connectivity of S. Furthermore, most of the studies assume that M has the same number of states as S; in this case, machine equivalence means isomorphism. This last assumption is equivalent to a fault model where there are two types of faults, *output faults*, i.e. transitions may produce the wrong output, and *next state* (or transfer) faults, i.e., transitions may go to the wrong next state. In general there may be an arbitrary number of such faults which can mask each other and make testing harder.

Suppose we do not know the initial state. A checking experiment usually starts by applying a homing sequence (of S), after which we know the state s_0 of M (if it is correct). The rest of the test starts from this initial state s_0. The implementation machine M passes the test if it produces the same output sequence as the specification machine S starting from s_0.

Output Faults. If the only possible faults are output faults, then we only need follow a transition tour of S starting from s_0, i.e. a path that traverses all the

transitions of S. Such a tour has polynomial length and can be easily constructed, in fact the shortest tour can be constructed in polynomial time using a Chinese Postman algorithm [EJ73].

Output and Next State Faults. This is a harder problem due to the controllability and observability problems. If the specification machine S has a reset r which is "reliable", i.e. works correctly in the implementation machine M (maps every state to the same reset state), then there is a deterministic polynomial time algorithm that constructs a checking sequence of length $O(pn^3)$ [Ch78, CVI89, Va73]. This bound is best possible, in the sense that there are specification machines S which require this length up to a constant factor.

If there is no reset or there is a reset but it is not reliable, then the problem is harder. A number of algorithms have been proposed over the years, starting with Hennie's work [He64]. They all have the same basic structure, and check the transitions of S one by one. Checking a transition from s to t with label a/b involves (1) applying an input sequence that takes the machine to s, (2) applying input a (checking that M outputs b), and (3) applying a sequence to verify the state t. This segment may have to be repeated several times with different verification sequences for step (3). The various methods use different types of sequences to verify the endstate t of a transition. Hennie used distinguishing sequences and proved that if S has a distinguishing sequence x *, then it has a checking sequence of length polynomial in x *. Although preset distinguishing sequences can be exponentially long, one may use just as well adaptive distinguishing sequences which can be assumed to be $O(n^2)$ if they exist, yielding in this case checking sequences of length $O(pn^3)$. Several other papers starting with [SD88] use UIO sequences to verify the endstate t of transitions. As observed in [CVI89] however, the resulting test sequence does not necessarily guarantee that a machine M that passes this test is equivalent to S.

In the general case, a randomized polynomial time algorithm is given in [YL95]. The algorithm constructs from the given FSM S a test sequence of polynomial length (namely, $O(pn^3 + p'n^4 \log n)$ where $p' = min(p, n)$) which is a checking sequence with high probability. The probability is with respect to the random choices of the algorithm; the specification S is worst-case (not probabilistic). Random FSM's are much better behaved and have in most cases checking sequence of length within polylog factors of the number pn of transitions.

It remains an open problem to give a deterministic polynomial length construction in general. An interesting combinatorial universal traversal problem related to this question is the following. Consider all directed graphs with n nodes and outdegree d, i.e. every node has d outgoing arcs labeled $1, \ldots, d$. Let us say that a sequence x over $\{1, \ldots, d\}$ is a *blocking* sequence if for every degree-d graph G and starting node v, the path of G traversed by following the sequence x starting from v traverses all the arcs out of at least one node of G (one node is enough). A simple argument shows that a random sequence of polynomial length (in n and d) is a blocking sequence with high probability. Is there a deterministic construction? Such a

construction would yield a deterministic polynomial construction of checking sequences for all FSM's S.

The above results extend to the case of implementation machines that may have extra states beyond those of S; the length is multiplied by an exponential factor in the number of additional states and this factor is inherent [Va73]. The methods extend also to partially specified specification FSM's (see [YL95] for the details).

4. Optimizations

The number and size of tests that one can run is determined by the type of testing and the test execution environment. For example, system tests run in a test lab typically involve setting up the equipment and take actual lab time, so one can only run a limited number of them. In any case, it is important to optimize the use of resources as far as possible and choose carefully tests to minimize their number and length while meeting a desired level of fault coverage.

Checking sequences guarantee complete fault coverage for output and next state faults, but they are often too long for many practical applications and thus one has to lower the sights and use heuristic or less complete procedures. For example, in circuit testing, test sequences are generated based on specific fault models that significantly limit the possible faults [AS88]. An objective that is often used in both circuit testing and protocol testing is to generate a test sequence that exercise each transition of the specification machine at least once. This criterion corresponds to the output fault model. As we mentioned earlier, a shortest covering path (a Postman Tour) for a given (strongly connected) specification machine S can be computed in polynomial time [EJ73, NT81, UD86].

If one can afford longer sequences, then one would like to apply also some testing of the endstates of the transitions. Suppose that we have a "verification" sequence x_t for each state t (for example a UIO sequence if it exists). An objective then might be to construct a path through the machine S which contains for each transition (s,t) of S, a segment consisting of the transition followed by the verification sequence x_t for t. For instance, one may seek an ordering of the transitions of S and then construct a path by taking each transition (s,t) in turn, transferring by a shortest path from the current state to the head state s of the transition, going to the tail t and then applying the verification sequence x_t. Choosing the best ordering can be translated to the Rural Postman Problem. It is in general an NP-hard problem, but under some constraints a polynomial time solution can be obtained for a class of communication protocols [ADLU91]. In this method (that combines segments of the transitions in some order) the segments of the different transitions are disjoint. It is possible that a shorter test sequence can be obtained by overlapping the segments. There are several papers in the literature that propose heuristics for taking advantage of overlaps in order to reduce the total length of tests [CCK90, SL89, YU90].

In the above discussion it is assumed that the specification machine S is strongly connected, thus it can be covered by one sequence. In many applications, S has a distinguished initial state s_0 and it can be considered strongly connected only by virtue of a reset (reinitialization). In these cases, when testers talk about a test sequence (or scenario) they refer to an execution that starts at the initial state. For example, in telephony, the initial state is the idle state and a scenario corresponds to the processing of a call (which many involve several call participants and features) from beginning to end.

The primary concern in this setting is to minimize the number of tests, and the secondary concern is the length of the tests. Given a FSM S, we can compute in polynomial time a set of test sequences that (1) minimizes the number of tests, and (2) minimizes their total length (among all sets that minimize the number). The algorithm is useful in various contexts. It is being incorporated for example in *uBET* the L*u*cent *B*ehavior *E*ngineering *T*oolset for requirement capture and analysis.

Other coverage criteria are also of interest sometimes, where one may not be able to optimize in polynomial time both objectives, the number of tests and their length. We mention for example the case of node coverage. In this case, one can compute a set of test that minimizes the number, but minimizing the length is in general NP-hard. The hard case is that of strongly connected graphs; for acyclic graphs one can minimize both objectives, but in the strongly connected case the problem amounts to the directed TSP.

5. Extended Finite State Machines

In many applications it is convenient to use variables to model protocols and system designs at a more detailed level; the pure finite state machine model is not powerful enough to model in a succinct way systems at this level. Extended finite state machines, which are finite state machines extended with variables, are commonly used either directly or more commonly through various related design specification languages such as SDL and VFSM. For instance, IEEE 802.2 LLC [ANSI89] is specified by 14 control states, a number of variables, and a set of transitions (pp. 75-117). For example, a typical transition is (p. 96):

current_state SETUP
input ACK_TIMER_EXPIRED
predicate S_FLAG = 1
output CONNECT_CONFIRM
action P_FLAG := 0; REMOTE_BUSY := 0
next_state NORMAL

In state SETUP and upon input ACK_TIMER_EXPIRED, if variable S_FLAG has value 1, then the machine outputs CONNECT_CONFIRM, sets variables P_FLAG and REMOTE_BUSY to 0, and moves to state NORMAL.

An extended finite state machine (EFSM) is a FSM augmented with a (finite) set of variables (such as Boolean variables and counters). Every transition, say $s \rightarrow t$, has an associated predicate P on the set of variables and an action A (transformation on the variable values), in addition to the input and output. The transition can take place if the system is at state s, the variable values satisfy the predicate P and the appropriate input is received. Then the appropriate output is produced, the variables are updated according to A and the system moves to state t.

Given an EFSM M, each combination of a state and variable assignment represents the global state of the system and is called a *configuration*. If the variables have bounded domains (for example Boolean variables), then there is a finite number of configurations and thus an EFSM M is really a compact representation of an equivalent ordinary FSM M'. In principle one could test an EFSM M by expanding it to M' and applying one of the FSM methods. However, in practice the expanded FSM is often quite large due to the state explosion problem and hence the methods for ordinary FSM's yield large tests and are not applicable.

Thus, in this case we have to look for tests that achieve more modest goals. The most common goal is to ensure that each transition of the EFSM is executed at least once. Typically, there is a specified initial configuration u_0 of the EFSM (i.e. initial state and variable assignment) and the objective is to generate a minimum set of test sequences that is "complete" in the sense that it covers all the transitions of the EFSM. Unlike the case of ordinary FSM's, this is not a computationally easy (polynomial) problem anymore.

>From the given EFSM M and initial configuration u_0, one can construct the *reachability* graph G_r consisting of all configurations and transitions reachable from u_0 (a subgraph of the expanded FSM M'). This graph is often large. One can use in place of the reachability graph an equivalent minimized graph $G_{\min}$ which collapses all configurations of the reachability graph that are equivalent in terms of the transitions that they can perform. Such a minimized graph can be constructed efficiently directly from the EFSM M in an online fashion, provided one can use suitable symbolic representations for sets of configurations [LY92]. Let G in the following be the reachability graph G_r or its minimized version $G_{\min}$. Every transition of the EFSM gives rise to a number of transitions (edges) of G. Associate a unique "color" with each transition of the EFSM M, and color correspondingly every edge of G. The problem of generating a minimum complete test set for the EFSM M translates to the following colored graph covering problem: generate a minimum set of paths in G that start at the initial node u_0 and cover all the colors [LY96b]. We could in addition color the nodes of G. The covering problem can be reduced to the case of DAGs, by shrinking the strong components of G and associating with the resulting node the set of colors that occur within the component.

The above colored graph covering problem is NP-hard, and furthermore it cannot be approximated in polynomial time to a factor better than $\log n$ (unless of course P=NP). A greedy heuristic is the following: seek a path that covers the maximum

number of colors, delete the colors and iterate. This would achieve a $\log n$ approximation factor if only we could find such a maximum color path. Unfortunately, this is an NP-hard problem as well, and it is also MAX SNP-hard (i.e. cannot be approximated arbitrarily close to 1). One can apply a greedy heuristic to look for a good path in the DAG of strong components that covers many colors. There are several greedy variants. Most can be in the worst case quite bad: it is possible that the optimal path covers c colors and the greedy heuristics construct a path with $O(1)$ colors; the best greedy variant constructs a path with $O(\sqrt{c})$ colors. It is an interesting open problem to find better approximation algorithms for these problems.

In spite of the negative results in the worst case, the greedy heuristics were applied to real systems with very good results (i.e. coming very close to easy lower bounds). In these cases it turned out that a few tests covered a large number of colors, and the remaining tests (which formed the bulk of the whole set) covered a few additional colors each. In the latter case one can actually find efficiently the best path: if the maximum number c of colors in a path is bounded (i.e. is a constant) then one can compute the maximum color path in essentially linear time in the size of the graph (more precisely, randomized time $2^{O(c)}m$ or deterministic time $2^{O(c)}m \log k$ where m is the size of the graph and k the number of colors).

Some of these algorithms are included in Pithia, an internal Lucent test generation tool for FSM and EFSM models [LY99b]. There is also a commercial test generation tool based on EFSMs by Teradyne called TestMaster (see eg. [Ap95] and the teradyne.com web site for more information).

6. Nondeterministic and Probabilistic Machines

In a nondeterministic machine a state may have several transitions corresponding to the same input, which may go to different states or produce different outputs. In a probabilistic machine, every transition has an associated probability, with the restriction that for each state s and input a, the sum of the probabilities of all the transitions out of state s on input a is equal to 1. Thus a probabilistic FSM is essentially a Markov Decision Process, where the inputs correspond to the actions of the process.

Nondeterminism may model different aspects in a specification or implementation FSM. One case is that of a specification S (for example a standard) that offers different alternative choices that are modeled by the nondeterminism. Different ways of resolving the nondeterminism yield different deterministic machines derived from S, all of which are considered as acceptable implementations (i.e. conforming to S). A simple example is the case of partially specified (deterministic) FSMs: the partial specification prescribes only a subset of the transitions, while the rest of the transitions can be chosen independently.

A different kind of nondeterminism in the specification or implementation machine arises when there are aspects that have been abstracted away from the FSM state, or there are actions that are not under the control of the tester that provides the

inputs; hence the machine may behave differently at different times — either in an arbitrary fashion (nondeterministic FSMs) or following some probabilistic rule (probabilistic FSMs). There has been less algorithmic work on the testing of nondeterministic and probabilistic machines. The problems are generally harder.

Consider the first interpretation of nondeterminism. That is, we are given a nondeterministic specification FSM S, and a deterministic FSM M, and we wish to determine whether M conforms to S, i.e. can be derived from S by choosing one transition for each state and input. Even the white-box testing problem, where we have complete knowledge of the implementation FSM M and its state, is in general NP-hard [LY99a]. Under some circumstances (basically if the specification FSM S has a substantial core of deterministic transitions that can distinguish the states) it is possible to perform efficiently black-box (as well as white box) testing.

Consider the second interpretation of nondeterminism, and suppose we are given a black-box NFSM M. Can we design a test sequence that tells apart a specification NFSM S from a given possible "faulty" machine S', i.e. an input sequence x which we can apply to M, observe the output and either conclude "Pass" if $M = S$, "Fail" if $M = S'$ (and the conclusion can be arbitrary if M is not equal to either S or S')? In the deterministic case, this amounts to machine (or state) equivalence, which of course can be done efficiently. For nondeterministic machines the problem is much harder. In the preset case the problem is PSPACE-complete, and in the adaptive case it is EXPTIME-complete [ACY95].

Similar questions can be posed for probabilistic FSMs. In this case we want to distinguish between different machines (or between different states) with probability tending to 1. The complexity of the problems turns out the same, i.e. PSPACE-complete in the preset case, and EXPTIME-complete in the adaptive case although the algorithms are different [ACY95].

These distinguishing problems for nondeterministic and probabilistic FSM can be viewed as two person games with incomplete information. The nodeterministic FSM case corresponds to a game between a purposeful player (the tester) and a malicious adversary; in the probabilistic case the adversary plays at random. Preset testing corresponds to blindfold (no information) games, and adaptive testing corresponds to partial information games.

7. Conclusions

There has been extensive work over the years on problems related to the testing of finite state systems. We summarized here some of the algorithmic results on these problems for different types of finite state machines. We discussed mainly active testing. There is some interesting work also on passive testing, for example work on inference of Markov chains from their outputs (eg. in the information theory and CS theory communities), and inference of finite automata (in learning theory and robotics).

Developing effective methods for automatic test generation of finite state machines is important from the practical side given the wide use of FSM models in many types of applications. At the same time it poses several interesting theoretical questions. Much work remains to be done in particular for extended, for nondeterministic, and for communicating finite state machines.

References

[AS88] V. D. Agrawal and S. C. Seth, *Test Generation for VLSI Chips,* Computer Society Press, 1988.

[ADLU91] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE Trans. on Communication*, vol. 39, no. 11, pp. 1604-15, 1991.

[ACY95] R. Alur, C. Courcoubetis, and M. Yannakakis, "Distinguishing tests for nondeterministic and probabilistic machines," *Proc. 27th Ann. ACM Symp. on Theory of Computing*, pp. 363-372, 1995.

[An87] D. Angluin, "Learning regular sets from queries and counterexamples," *Inform. and Comp., 75, pp. 87-106, 1987.*

[ANSI89] International standard ISO 8802-2, ANSI/IEEE std 802.2, 1989.

[Ap95] L. Apfelbaum. "Automated functional test generation," *Proc. IEEE Autotestcon Conference*, 1995.

[CVI89] W. Y. L. Chan, S. T. Vuong, and M. R. Ito, "An improved protocol test generation procedure based on UIOs," *Proc. SIGCOM*, pp. 283-294, 1989.

[CZ93] S. T. Chanson and J. Zhu, "A unified approach to protocol test sequence generation", *Proc. INFOCOM,* pp. 106-14, 1993.

[CCK90] M.-S. Chen Y. Choi, and A. Kershenbaum, "Approaches utilizing segment overlap to minimize test sequences," *Proc. IFIP WG6.1 10th Intl. Symp. on Protocol Specification, Testing, and Verification*, North-Holland, L. Logrippo, R. L. Probert, and H. Ural Ed. pp. 85-98, 1990.

[Ch78] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. on Software Engineering*, vol. SE-4, no. 3, pp. 178-87, 1978.

[EJ73] J. Edmonds and E.L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical Programming*, vol. 5, pp. 88-124, 1973.

[FM71] A. D. Friedman and P. R. Menon, *Fault Detection in Digital Circuits*, Prentice-Hall, 1971.

[He64] F. C. Hennie, "Fault detecting experiments for sequential circuits," *Proc. 5th Ann. Symp. Switching Circuit Theory and Logical Design,* pp. 95-110, 1964.

[Ho91] G. J. Holzmann, *Design and Validation of Computer Protocols,* Prentice-Hall, 1991.

[Kn93] K. G. Knightson, *OSI Protocol Conformance Testing*, McGraw Hill, 1993.

[Ko78] Z. Kohavi, *Switching and Finite Automata Theory,* 2nd Ed., McGraw-Hill, 1978.

[Ku95] R. P. Kurshan, *Computer-aided Verification of Coordinating Processes,* Princeton University Press, Princeton, New Jersey, 1995.

[LSKP96] D. Lee, K. K. Sabnani, D. M. Kristol, and S. Paul, "Conformance testing of protocols specified as communicating finite state machines - a guided random walk based approach," *IEEE Trans. on Communications,* vol. 44, no. 5, pp. 631-640, 1996.

[LY92] D. Lee and M. Yannakakis, "On-line minimization of transition systems," *Proc. 24th Ann. ACM Symp. on Theory of Computing,* pp. 264-274, 1992.

[LY94] D. Lee and M. Yannakakis, "Testing finite state machines: state identification and verification," *IEEE Trans. on Computers*, vol. 43, no. 3, pp. 306-320, 1994.

[LY96a] D. Lee and M. Yannakakis, "Principles and Methods of Testing Finite State Machines - a Survey," *Proceedings of IEEE*, Vol. 84, No. 8, pp. 1089-1123, August 1996.

[LY96b] D. Lee and M. Yannakakis, "Optimization Problems from Feature Testing of Communication Protocols", *Proc. of Intl. Conf. on Network Protocols*, pp. 66-75,1996.

[LY99a] D. Lee and M. Yannakakis, in preparation.

[LY99b] D. Lee and M. Yannakakis, "Pithia - An automatic test generation software tool for communication systems," in preparation.

[MP93] R. E. Miller and S. Paul, "On the generation of minimal length test sequences for conformance testing of communication protocols," *IEEE/ACM Trans. on Networking,*, Vol. 1, No. 1, pp. 116-129, 1993.

[Mo56] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies,* Annals of Mathematics Studies, Princeton University Press, no. 34, pp. 129-153, 1956.

[NT81] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transitions tours," *Proc. IEEE Fault Tolerant Comput. Symp.*, IEEE Computer Society Press, pp. 238-43, 1981.

[RS89] R. L. Rivest and R. E. Schapire, "Inference of finite automata using homing sequences," *Proc. 21st Ann. Symp. on Theory of Computing,* pp.411-420, 1989.

[SD88] K. K. Sabnani and A. T. Dahbura, ‘‘A protocol test generation procedure,’’ *Computer Networks and ISDN Systems,* vol. 15, no. 4, pp. 285-97, 1988.

[SB84] B. Sarikaya and G.v. Bochmann, ‘‘Synchronization and specification issues in protocol testing,’’ *IEEE Trans. on Commun.,* vol. COM-32, no. 4, pp. 389-395, 1984.

[SL89] D. P. Sidhu and T.-K. Leung, ‘‘Formal methods for protocol testing: a detailed study,’’ *IEEE Trans. Soft. Eng.*, vol. 15, no. 4, pp. 413-26, April 1989.

[So71] M. N. Sokolovskii, ‘‘Diagnostic experiments with automata,’’ *Kibernetika*, 6, pp. 44-49, 1971.

[UD86] M.U. Uyar and A.T. Dahbura, ‘‘Optimal test sequence generation for protocols: the Chinese postman algorithm applied to Q.931,’’ *Proc. IEEE Global Telecommunications Conference*, 1986.

[Va73] M. P. Vasilevskii, ‘‘Failure diagnosis of automata,’’ *Kibernetika*, no. 4, pp. 98-108, 1973.

[YU90] B. Yang and H. Ural, ‘‘Protocol conformance test generation using multiple UIO sequences with overlapping,’’ *Proc. SIGCOM*, pp. 118-125, 1990.

[YL95] M. Yannakakis and D. Lee, ‘‘Testing finite state machines: fault detection,’’ *J. of Computer and System Sciences*, *Vol. 50, No. 2, pp. 209-227, 1995.*

On the Power of Quantifiers in First-Order Algebraic Specification*

David Kempe[1] and Arno Schönegge[2]

[1] Department of Computer Science
Cornell University, Ithaca, NY 14853
kempe@cs.cornell.edu
[2] Institut für Logik, Komplexität und Deduktionssysteme
Universität Karlsruhe, 76128 Karlsruhe, Germany
schoenegge@ira.uka.de

Abstract. The well-known completeness theorem of Bergstra & Tucker [BT82,BT87] states that all computable data types can be specified without quantifiers, i.e., quantifiers can be dispensed with—at least if the introduction of auxiliary (hidden) functions is allowed.
However, the situation concerning the specification *without* hidden functions is quite different. Our main result is that, in this case, quantifiers *do* contribute to expressiveness. More precisely, we give an example of a computable data type that has a monomorphic first-order specification (without hidden functions) and prove that it fails to possess a monomorphic quantifier-free specification (without hidden functions).

1 Introduction

The expressive power of first-order algebraic specification methods has been extensively studied over several years, e.g., by Majster [Maj79], Bergstra & Tucker [BT82,BT87], the ADJ-group [TWW82], and others [Ore79,BBTW81,Wir90].

One main result is that, if equipped with hiding mechanisms, all common algebraic specification methods are adequate for computable[1] data types. Here, hiding means that local definitions of auxiliary functions can be added for reasons of specification only; these functions are hidden from the user of the specified data type. Bergstra & Tucker [BT82,BT87] proved that any computable algebra A possesses a quantifier-free specification involving at most $2n+3$ hidden functions (where n is the number of sorts in A) that defines A under both its initial and final algebra semantics. Hence, by using hiding mechanisms, a monomorphic quantifier-free specification can be constructed for any computable data type.[2]

* This research has partly been supported by the "Deutsche Forschungsgemeinschaft" within the "Schwerpunktprogramm Deduktion". The results were obtained in the course of [Kem98].

[1] We define the notions of (computable) data types, specifiability, etc. in section 2.

[2] On the other hand, any data type defined by a monomorphic quantifier-free specification is computable (cf. [Wir90, Theorem 4.2.1]). Hence, the computability of a data type is characterized by specifiability without quantifiers.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 45–57, 1999.

Consequently, if one is interested in specifying computable data types only (which is reasonable in most practical applications), the use of quantifiers does not enhance expressiveness.[3]

However, the situation changes substantially if we do not allow hidden functions. In particular, the completeness for computable data types is lost: there are examples of data types that are computable but fail to possess monomorphic first-order specifications without hidden functions [Ber93,Sch97a,Sch97b]. The loss of expressiveness caused by disabling hiding therefore cannot be compensated completely, even if arbitrary first-order quantification is used. Naturally, the question arises whether quantification makes up for this loss at least to some extent. In other words: if a quantifier-free specification without hidden functions does not exist, can we hope to find at least a first-order specification (using quantifiers) without hidden functions?

This paper gives a positive answer by proving

Theorem 1. *There is a computable data type that possesses a monomorphic first-order specification without hidden functions but fails to have a monomorphic quantifier-free specification without hidden functions.*

While this theorem states what one would expect, it is difficult to identify examples of the required class of data types. Especially, in order to prove that a data type cannot be specified without quantifiers, non-standard models have to be constructed for any potential quantifier-free specification.

The paper is organized as follows: Section 2 recalls some basic definitions. In section 3, we demonstrate that following a naive approach does not result in examples of data types that point out the necessity of quantifiers. Section 4 presents a solution, i.e. a data type A that is claimed to have a monomorphic specification with quantifiers but none without (both not using hidden functions). The first property is established in section 5, the second in section 6. Finally, in the last section, we draw conclusions and discuss related work.

2 Preliminaries

We assume the reader to be familiar with the very basic notions of algebraic specification (cf. e.g. [Wir90] or [LEW96]) like those of (many-sorted) *signature* $\Sigma = (S, F)$, (total) Σ*-algebra* $A = ((s^A)_{s\in S}, (f^A)_{f\in F})$, and Σ*-homomorphism*.

The set of *terms* $T(\Sigma, X)$ with variables taken from X is defined as usual. Terms without variables are called *ground terms*. A Σ-algebra A is called *term-generated* if there is a denotation, i.e. a ground term $t \in T(\Sigma, \emptyset)$ with $t^A = a$, for each of its carrier elements $a \in s^A$ $(s \in S)$. The class of all term-generated Σ-algebras is denoted by $Gen(\Sigma)$.

[3] Of course, there are *non-computable* data types which can be specified using quantifiers but—as argued in footnote 2—not without. More precisely, even all *arithmetical* data types possess a monomorphic first-order specification (cf. the proof of [Wir90, Lemma 5.3.8]).

Two Σ-algebras A, B are called *isomorphic* if there is a bijective Σ-homomorphism $h : A \to B$. The isomorphism class of a Σ-algebra A, also called the "abstract" *data type*, is denoted by $[A]$. Sometimes, we simply speak of the *data type* A.

A Σ-algebra A is called *computable* if it is isomorphic to a *computable number algebra*, an algebra in which all carrier sets are decidable subsets of $\mathbb{N}$ and all functions are computable.

Atomic Σ-formulas are Σ-equations $t_1 = t_2$ (where $t_1, t_2 \in T(\Sigma, X)$ are terms of the same sort) and the boolean constant **false**. *First-order Σ-formulas* are built from the atomic ones using the logical connectives $\neg$ and $\wedge$, and the quantifier $\exists$. Further logical operators such as **true**, $\neq$, $\vee$, $\to$, $\leftrightarrow$, and $\forall$ are regarded as abbreviations.

The usual *satisfaction* of a Σ-formula φ by a Σ-algebra A w.r.t. a valuation $v : X \to A$ is denoted by $A, v \models \varphi$. We write $A \models \varphi$, and say that φ is *valid in* A, if $A, v \models \varphi$ holds for all valuations v. For a set Φ of Σ-formulas, we write $A \models \Phi$ if $A \models \varphi$ for all $\varphi \in \Phi$.

A *(first-order algebraic) specification* $SP = (\Sigma, \Phi)$ consists of a signature Σ and a finite set Φ of Σ-formulas, called *axioms*. If the formulas in Φ do not contain quantifiers, we speak of *quantifier-free specification*. We adopt loose semantics and define $Mod(SP) := \{A \in Gen(\Sigma) \mid A \models \Phi\}$ to be the set of *models* of SP. If $A \models \varphi$ for all $A \in Mod(SP)$, we simply write $SP \models \varphi$.

A specification SP is called *monomorphic* if any two of its models are isomorphic, i.e. if there is—up to isomorphism—at most one element in $Mod(SP)$. A specification $SP = (\Sigma, \Phi)$ is said to *specify* a Σ-algebra A (and also the corresponding abstract data type $[A]$) if $Mod(SP) = [A]$. Hence, a specification SP specifies a Σ-algebra A if and only if SP is monomorphic and $A \in Mod(SP)$.

A Σ-algebra A with $\Sigma = (S, F)$ can be specified *with hidden functions* if there is a super-signature $\Sigma' = (S, F \cup HF)$ (where HF denotes the additional hidden function symbols) and a Σ'-algebra A' with A as its Σ-reduct, i.e. $A'|_\Sigma = A$, such that A' possesses a specification.

3 A first approach: primality of numbers

When looking for a suitable example to prove theorem 1, one that might come to mind immediately is that of the primality predicate: its customary definition in first-order logic looks like the following:

$$\text{prime}(x) \quad \leftrightarrow \quad (x > 1) \wedge \neg\exists y, z.(y > 1 \wedge z > 1 \wedge x = yz)$$

and thus involves a quantifier. Although one might suspect that this quantifier is indispensable, it turns out that a quantifier-free specification of the primality predicate without further auxiliary functions actually exists. This result follows from a more general fact, stating the quantifier-free specifiability of every decid-

able predicate using the signature

signature

sorts	Nat, Bool		
functions	true, false :		→ Bool
	zero :		→ Nat
	succ :	Nat	→ Nat
	add, mult :	Nat × Nat	→ Nat
	char :	Nat	→ Bool

end signature

For every $M \subseteq \mathbb{N}$, A_M denotes the algebra that interprets Nat, Bool and their basic operations in the usual way and furthermore defines

$$\texttt{char}^{A_M}(x) \quad = \quad \begin{cases} \mathit{true} & \text{if } x \in M \\ \mathit{false} & \text{otherwise} \end{cases}$$

to be the characteristic function of M.

Fact 2. *Let M be a* decidable *subset of the natural numbers.*[4] *Then, there exists a quantifier-free specification SP without hidden functions such that*

$$\mathit{Mod}(SP) \quad = \quad [A_M].$$

Proof. The proof is a consequence of the characterization of enumerable sets by diophantine equations. A good introduction to the subject is [Mat93].

Seeing that the basic operations on Nat and Bool can be easily specified without quantifiers, we focus on the specifiability of the function $\texttt{char}^{A_M}$.

Since M is decidable, both M and its complement are recursively enumerable. Matiyasevich's Theorem states that all recursively enumerable sets are diophantine. Hence, M and $\mathbb{N} \setminus M$ are diophantine, meaning that there are polynomials p_1, q_1, p_2, q_2 with positive integer coefficients such that

$$\begin{array}{lll} x \in M & \iff & \text{there exist } y_1, \ldots, y_m \text{ with } p_1(x, y_1, \ldots, y_m) = q_1(x, y_1, \ldots, y_m), \\ x \notin M & \iff & \text{there exist } y_1, \ldots, y_n \text{ with } p_2(x, y_1, \ldots, y_n) = q_2(x, y_1, \ldots, y_n). \end{array}$$

Since the polynomials can be written as terms using zero, succ, add, and mult only, we obtain two quantifier-free axioms specifying $\texttt{char}^{A_M}$:

$$\begin{array}{rcl} p_1(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_m) = q_1(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_m) & \rightarrow & \mathtt{char}(\mathtt{x}) = \mathtt{true} \\ p_2(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_n) = q_2(\mathtt{x}, \mathtt{y}_1, \ldots, \mathtt{y}_n) & \rightarrow & \mathtt{char}(\mathtt{x}) = \mathtt{false}. \end{array}$$

Note that we are not introducing any new signature symbols: p_1, q_1, p_2, q_2 are merely denotations for the terms representing the polynomials. ■

Choosing M to be the set of all prime numbers, the above fact 2 yields the desired specification. In this special case, polynomials p_2, q_2 can easily be found, whereas determining p_1, q_1 requires a great amount of number theory. In [Mat93, P. 55], a polynomial representation of the set of prime numbers is given.

[4] The statement can easily be generalized to decidable subsets of $\mathbb{N}^k$ for any k and also to computable functions from $\mathbb{N}^k$ to $\mathbb{N}$.

4 The example: completeness of graphs

Looking once more at the reason why the previous section's example did indeed have a quantifier-free specification, we realize that the powerful arithmetical operations allowed us to code all decidable properties of natural numbers. Therefore, we turn to properties of structures possessing fewer operations, namely the completeness of graphs (for the same reason, graphs and their properties are often considered in finite model theory, cf. section 7).

More specifically, we consider directed graphs with a finite number of edges, and whose vertices are exactly the natural numbers. We can therefore identify a graph with the set of its edges, i.e., a *graph* G is a *finite* subset $G \subset \mathbb{N} \times \mathbb{N}$. It is called *complete* if $(n, m) \in G$ for every $n, m \in G$. Here, $n \in G$ means that there exists an $n' \in \mathbb{N}$ such that $(n, n') \in G$ or $(n', n) \in G$.

To define the data type A of graphs equipped with a completeness predicate, we provide the signature

signature Σ

sorts	Bool, Nat, Graph		
functions	true, false :		→ Bool
	zero :		→ Nat
	succ	: Nat	→ Nat
	empty :		→ Graph
	connect	: Nat × Nat × Graph	→ Graph
	complete	: Graph	→ Bool

end signature

and interpret the symbols in the obvious way:

$$
\begin{array}{lcl}
\texttt{Bool}^A & := & \{\mathit{true}, \mathit{false}\} \\
\texttt{Nat}^A & := & \mathbb{N} \\
\texttt{Graph}^A & := & \left\{ G \subset \mathbb{N} \times \mathbb{N} \;\middle|\; G \text{ finite} \right\} \\
\texttt{true}^A & := & \mathit{true} \\
\texttt{false}^A & := & \mathit{false} \\
\texttt{zero}^A & := & 0 \\
\texttt{succ}^A(n) & := & n + 1 \\
\texttt{empty}^A & := & \emptyset \\
\texttt{connect}^A(n, m, G) & := & G \cup \{(n, m)\} \\
\texttt{complete}^A(G) & := & \begin{cases} \mathit{true} & \text{if } G \text{ is complete} \\ \mathit{false} & \text{otherwise.} \end{cases}
\end{array}
$$

Notice that the data type A is obviously term-generated and computable. The following two sections will establish the existence of a monomorphic specification for A with quantifiers and the non-existence of one without (both not using hidden functions).

5 Specifiability using quantifiers

In order to show that, by using quantifiers, the data type A can be specified without introducing hidden functions, we subsequently give a suitable specification. For convenience, we agree on two abbreviations (but note that we are not adding any new symbols to the signature):

$$\begin{aligned} \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}) &:\equiv (\mathtt{connect}(\mathtt{x},\mathtt{y},\mathtt{g}) = \mathtt{g}) \\ \mathit{in}(\mathtt{x},\mathtt{g}) &:\equiv \exists\, \mathtt{z}.\, (\mathit{connected}(\mathtt{x},\mathtt{z},\mathtt{g}) \vee \mathit{connected}(\mathtt{z},\mathtt{x},\mathtt{g}))\,. \end{aligned}$$

specification COMPL

signature Σ

variables $\mathtt{x},\mathtt{y},\mathtt{u},\mathtt{v} : \mathtt{Nat}$
$\mathtt{g},\mathtt{h} : \mathtt{Graph}$

axioms

(1) $\mathtt{true} \neq \mathtt{false}$

(2) $\mathtt{succ}(\mathtt{x}) \neq \mathtt{zero}$

(3) $\mathtt{succ}(\mathtt{x}) = \mathtt{succ}(\mathtt{y}) \rightarrow \mathtt{x} = \mathtt{y}$

(4) $\neg\, \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{empty})$

(5) $\mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{connect}(\mathtt{u},\mathtt{v},\mathtt{g}))$
$\quad \leftrightarrow (\mathtt{x} = \mathtt{u} \wedge \mathtt{y} = \mathtt{v}) \vee \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g})$

(6) $\mathtt{g} = \mathtt{h} \leftrightarrow \forall\, \mathtt{x},\mathtt{y}.\, (\mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}) \leftrightarrow \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{h}))$

(7) $\mathtt{complete}(\mathtt{g}) = \mathtt{true} \vee \mathtt{complete}(\mathtt{g}) = \mathtt{false}$

(8) $\mathtt{complete}(\mathtt{g}) = \mathtt{true}$
$\quad \leftrightarrow \forall\, \mathtt{x},\mathtt{y}.\, (\mathit{in}(\mathtt{x},\mathtt{g}) \wedge \mathit{in}(\mathtt{y},\mathtt{g}) \rightarrow \mathit{connected}(\mathtt{x},\mathtt{y},\mathtt{g}))$

end specification

Theorem 3. *The first-order specification* COMPL *specifies the data type* A*, i.e.* $\mathit{Mod}(\mathrm{COMPL}) = [A]$.

Proof. We can easily verify that all axioms are valid in A. Since A is term-generated, A is a model of COMPL and it remains to prove the monomorphicity of COMPL.

For that purpose, we use the fact that a specification SP is monomorphic iff SP fixes the ground equations, i.e., iff for all ground equations $t_1 = t_2$, either $SP \models t_1 = t_2$ or $SP \models t_1 \neq t_2$ (cf. [Wir90, Fact 2.3.2]). We check each kind of ground equation separately:

(a) $t_1 = t_2$ for ground terms t_1, t_2 of the sort `Nat`:
Due to axioms (2) and (3), we get $SP \models t_1 \neq t_2$ whenever t_1 and t_2 differ syntactically; otherwise, $SP \models t_1 = t_2$ holds trivially.

(b) $t_1 = t_2$ for ground terms t_1, t_2 of the sort `Graph`:
We first use induction on the structure of t and axioms (4) and (5) to show that for all ground terms t_1', t_2' of the sort `Nat` and t of the sort `Graph`, either COMPL $\models \mathit{connected}(t_1', t_2', t)$ or COMPL $\models \neg\, \mathit{connected}(t_1', t_2', t)$.

Then, it immediately follows from axiom (6) that COMPL fixes arbitrary ground equations of the sort `Graph`.

(c) $t_1 = t_2$ for ground terms t_1, t_2 of the sort `Bool`:
Since the atoms on the right hand side of axiom (8) are equations of the sort `Graph`, it follows from (b) that for all ground terms t of the sort `Bool`, either COMPL $\models$ `complete`$(t) =$ `true` or COMPL $\models$ `complete`$(t) \neq$ `true`.
Using axiom (7), we find that either COMPL $\models$ `complete`$(t) =$ `true` or COMPL $\models$ `complete`$(t) =$ `false`. Ground terms `complete`(t) can thus be "reduced" to either `true` or `false`, i.e. all ground equations $t_1 = t_2$ of the sort `Bool` can be "reduced" to equations $t_1' = t_2'$ with $t_1', t_2' \in \{$`true`, `false`$\}$. The latter are fixed due to axiom (1). ■

6 Non-specifiability without quantifiers

The previous section shows that the abstract data type A can be specified without hidden functions, but using quantifiers. We now turn to the question whether the use of quantifiers is indeed necessary.[5] We will answer this question positively by proving the following

Theorem 4. *There is no quantifier-free specification SP without hidden functions such that Mod*$(SP) = [A]$.

Before presenting the actual proof, we give a short outline. While a "local property"—namely that of a missing edge between two non-isolated vertices (vertices that are part of at least one of the graph's edges)—is sufficient for *non-completeness* of a graph, a graph's *completeness* can only be guaranteed by looking at *all* pairs of its vertices and checking whether an edge exists between them.

However, a finite formula without quantifiers can only make statements about a relation between a bounded number of edges. If the size of a graph exceeds this bound, the specification fails to recognize this graph's completeness. Or, if the graph's completeness is actually recognized, then it is upon conditions that are not sufficient in the general case, and the specification will falsely classify some other graph as complete.

While the basic idea can be described fairly easily, the proof itself turns out to be quite cumbersome. We start with some conventions and definitions.

1. m, n are always natural numbers.
2. $t, t_i, t_j, t_0, t_1, t_2$ are terms of the sort `Graph`.
3. x, x_i are variables of the sort `Graph`.
4. v always denotes a valuation and $(t)^{A,v}$ denotes the evaluation of term t in the algebra A under v.
5. φ always denotes a disjunction of literals (i.e. of equations or inequations between terms), while ϕ stands for an arbitrary formula.

[5] For instance, the equality of graphs (in axiom (6)) could also have been specified *without* quantifiers—at the cost of a more difficult proof of theorem 3.

Definition 5. 1. For every m, G_m denotes the complete graph

$$G_m \;:=\; \bigcup_{i,j=0}^{m} \{(i,j)\},$$

and the algebra A_m is defined to be identical to A except for

$$\texttt{complete}^{A_m}(G_m) \;=\; \mathit{false}.$$

2. The *distance* $d\,(G_1, G_2)$ between two graphs G_1, G_2 is the cardinality of their symmetric set difference, i.e. $d\,(G_1, G_2) := |(G_1 \setminus G_2) \cup (G_2 \setminus G_1)|$.
3. connects(ϕ) is the number of occurrences of the function symbol `connect` in the formula ϕ; connects(t) is defined likewise.
4. The *maximum* $\max(\phi, A, v)$ of a formula ϕ under a valuation v in an algebra A is the greatest natural number occurring in the evaluation of `Nat` and `Graph` terms of ϕ under v and A.
5. Two terms t_1, t_2 are *adjacent in* φ if both appear in φ (possibly as subterms), and either one term is a subterm of the other or a literal $t_1 \neq t_2$ exists in φ. $\sim_\varphi$ denotes the transitive, symmetric closure of adjacency and is obviously an equivalence relation. If $t_1 \sim_\varphi t_2$ holds, then t_1 and t_2 are called *connected in* φ.
6. $T_{\varphi,v,m}$ denotes the set of all terms t such that there exists a term t_0 in φ with $t \sim_\varphi t_0$ and $(t_0)^{A,v} = G_m$.
   The set $T_{\varphi,v,m}$ is a kind of "influence sphere" around the terms t_0 evaluated to G_m. Since we later want to modify the evaluation of those terms t_0 (which might help to distinguish between the two algebras A and A_m) without affecting the validity of other parts of the formula, the evaluations of the terms in $T_{\varphi,v,m}$ have to be changed simultaneously as well.
   Note that $t \in T_{\varphi,v,m}$ implies that t appears in φ. Also, $T_{\varphi,v,m}$ is closed under subterms by definition of $\sim_\varphi$. In particular, if a term t contains a variable x, the term t is in $T_{\varphi,v,m}$ iff $x \in T_{\varphi,v,m}$, a fact that we will be using regularly without explicitly mentioning it.
7. For a valuation v, the valuation $v_{\varphi,m,n}$ is defined to be identical to v except that we set $v_{\varphi,m,n}(x) := v(x) \cup \{(n+1, n+2)\}$ for variables $x \in T_{\varphi,v,m}$.

Lemma 6. *Let v be an arbitrary valuation:*

(a) *If t does not contain a variable of the sort* `Graph`*, then*

$$\left|(t)^{A,v}\right| \;\leq\; \text{connects}(t).$$

(b) *If $t \in T_{\varphi,v,m}$ and none of the inequations of the sort* `Graph` *in φ is valid under v, then*

$$d\left(G_m, (t)^{A,v}\right) \;\leq\; \text{connects}(\varphi).$$

Proof. (a) can be proven by a simple induction on the structure of t.

To prove (b), let t_0 be a term with $t \sim_\varphi t_0$ that is evaluated to G_m under v and A. Such a term t_0 must exist due to the definition of $T_{\varphi,v,m}$.

Let $(t_1, t_2, \ldots, t_{j-1}, t_j)$ be a path of minimal length j connecting t and t_0 in the adjacency relation, i.e. $t_1 = t$ and $t_j = t_0$. Such a path must exist because of $t \sim_\varphi t_0$. For any term t' in φ, there is at most one pair (t_i, t_{i+1}) that is adjacent through being subterms of t'. Otherwise, if $i_1 < i_2$ were indices of two such pairs, we could obtain a path of length $j - (i_2 - i_1)$ by leaving out the terms $t_{i_1+1}, \ldots, t_{i_2}$.

The above statement also implies that for any two pairs (t_{i_1}, t_{i_1+1}), (t_{i_2}, t_{i_2+1}), that are adjacent through being subterms of t'_{i_1} or t'_{i_2}, t'_{i_1} cannot be a subterm of t'_{i_2}, and neither can t'_{i_2} be a subterm of t'_{i_1}. Thus, all t'_i can be chosen to be maximal, i.e. not to be subterms of any other term t' in φ.

To prove the desired inequation, we look at every step i in the path connecting t and t_0, and add up their contributions to the total set difference of the terms' evaluations. If t_i and t_{i+1} are connected through an inequation in φ, then the precondition that none of the `Graph` inequalities hold yields that both terms are evaluated to the same graph, so that this step's contribution is 0.

Otherwise, t_i and t_{i+1} are both subterms of some term t'_i in φ. This implies that their evaluation can differ by no more than $\mathrm{connects}(t'_i)$ edges (which can be proven by induction on the structure of t'_i). Using the above observation about the choice of the terms t'_i, we obtain:

$$\begin{aligned} & d\left((t)^{A,v}, (t_0)^{A,v}\right) \\ \leq\ & \sum_{i=1}^{j-1} d\left((t_i)^{A,v}, (t_{i+1})^{A,v}\right) \\ \leq\ & \sum_{t' \in \varphi \text{ maximal}} \mathrm{connects}(t') \\ \leq\ & \mathrm{connects}(\varphi). \end{aligned}$$

■

Lemma 7. *Let $m \geq \mathrm{connects}(\varphi)$, and $n \geq \max(\varphi, A, v)$. For all literals λ in φ that do not contain a term* `complete`(t) *with $t \in T_{\varphi,v,m}$, we assume $A, v \not\models \lambda$. Then, for all of these literals, it holds that $A, v_{\varphi,m,n} \not\models \lambda$.*

Proof. We check the different forms that the literals λ can have. The claim is obvious for equations and inequations between terms of the sorts `Nat` or `Bool` (if a literal includes a term `complete`(t), the precondition yields $t \notin T_{\varphi,v,m}$), as well as equations and inequations between terms of the sort `Graph` that do not contain any variables $x \in T_{\varphi,v,m}$. In all of these cases, every such term is evaluated in the same way under $v_{\varphi,m,n}$ and v.

The remaining cases are literals of the form $t_1 = t_2$ or $t_1 \neq t_2$, where at least one of the terms contains a variable $x \in T_{\varphi,v,m}$. If this applies to both terms, it can be easily checked that the valuation $v_{\varphi,m,n}$ adds the same edge $(n+1, n+2)$ to the evaluations of both terms, and that this edge was not previously contained in any of them. Thus, the equation holds under the valuation $v_{\varphi,m,n}$ iff it holds under v.

If both terms contain variables of the sort `Graph`, but only one of them is in $T_{\varphi,v,m}$, the literal has to be an equation (otherwise, the other variable would be in $T_{\varphi,v,m}$ by definition). Since the edge $(n+1, n+2)$ is added to the valuation of the term in $T_{\varphi,v,m}$ under $v_{\varphi,m,n}$, but not to that of the other term, it is clear that the equation cannot hold.

If λ is an equation such that one of the terms does not contain a variable of the sort `Graph`, an argument similar to the one just presented shows the claim. The case that λ is an inequation such that one of the terms does not contain a variable cannot occur, because this would imply $\texttt{empty} \in T_{\varphi,v,m}$, and lemma 6(b) would yield $d(G_m, \emptyset) \leq \text{connects}(\varphi) \leq m$, clearly a contradiction to $|G_m| = (m+1)^2$. ■

Lemma 8. *Let $\varphi = \bigvee_{j=1}^{J} \lambda_j$ such that all literals λ_j of the sort* `Bool` *are either of the form* $\texttt{complete}(t) = \texttt{true}$ *or of the form* $\texttt{complete}(t) = \texttt{false}$.[6] *Then, for all $m \geq \text{connects}(\varphi)$, we have*

$$A \models \varphi \implies A_m \models \varphi.$$

Proof. We fix an arbitrary $m \geq \text{connects}(\varphi)$. For any valuation v, we fix $n \geq \max(\varphi, A, v)$ and show that if both $A, v \models \varphi$ and $A, v_{\varphi,m,n} \models \varphi$ hold, then $A_m, v \models \varphi$. To show that this implies the lemma, let v be an arbitrary valuation. By assumption, $A \models \varphi$, so $A, v \models \varphi$, and $A, v_{\varphi,m,n} \models \varphi$. We thus get $A_m, v \models \varphi$, and because v was arbitrary, $A_m \models \varphi$.

If there is a $j \in \{1, \ldots, J\}$ with $A, v \models \lambda_j$ such that λ_j is not of the form $\texttt{complete}(t_j) = \texttt{true}$ with $(t_j)^{A,v} = G_m$, it is obvious that $A_m, v \models \lambda_j$, since the only difference between A and A_m is the evaluation of `complete` at G_m. This implies $A_m, v \models \varphi$ and hence our claim.

Otherwise, $A, v \models \lambda_j$ can only hold for literals $\texttt{complete}(t_j) = \texttt{true}$, where $(t_j)^{A,v} = G_m$ (especially implying $t_j \in T_{\varphi,v,m}$). For all other literals, we have $A, v \not\models \lambda_j$. Therefore, we can use lemma 7 to conclude that the only literals with $A, v_{\varphi,m,n} \models \lambda_i$ can be those including a term $\texttt{complete}(t_i)$ with $t_i \in T_{\varphi,v,m}$, and because of $A, v_{\varphi,m,n} \models \varphi$, there must be at least one such $i \in \{1, \ldots, J\}$.

t_i must contain a variable of the sort `Graph`, because otherwise, we could use lemma 6(a) to conclude that the evaluation of t_i has at most m edges, in contradiction to t_i being evaluated to G_m (which has $(m+1)^2$ edges). We can therefore use the definition of $v_{\varphi,m,n}$ and induction on the structure of t_i to show that the evaluation of t_i under $v_{\varphi,m,n}$ includes the edge $(n+1, n+2)$, but not the edge $(n+1, n+1)$—thus, it is not a complete graph. Hence, $A, v_{\varphi,m,n} \models \lambda_i$ implies that λ_i is of the form $\texttt{complete}(t_i) = \texttt{false}$ (here, we use the restriction placed on literals of the sort `Bool`).

We now look at the evaluation of t_i under v, writing $G := (t_i)^{A,v}$. If $G = G_m$, we immediately have $A_m, v \models \lambda_i$ because of A_m's definition. Otherwise, lemma 6(b) states that G and G_m differ by at most $\text{connects}(\varphi) \leq m$ edges. Since any complete graph other than G_m differs from G_m by at least $2m+1$ edges

[6] This assumption is only used to make the proof less technical. The lemma also holds without the assumption.

(which can be proven quite easily), G cannot be complete, and hence $A_m, v \models \lambda_i$, completing our proof. ■

Proof of theorem 4. Using lemma 8, the proof of theorem 4 is quite straightforward. If SP is any quantifier-free specification without auxiliary functions, and Φ is the set of its axioms, we form the conjunction of all axioms in Φ. We then replace every occurrence of any equation $t_1^B = t_2^B$ (with terms t_1^B, t_2^B of the sort `Bool`) by

$$t_1^B = \texttt{true} \quad \leftrightarrow \quad t_2^B = \texttt{true},$$

and similarly any inequation $t_1^B \neq t_2^B$ by

$$t_1^B = \texttt{true} \quad \leftrightarrow \quad t_2^B = \texttt{false},$$

Transforming the resulting formula into conjunctive normal form and replacing inequations $t^B \neq \texttt{true}$ (resp. $t^B \neq \texttt{false}$) by $t^B = \texttt{false}$ (resp. $t^B = \texttt{true}$) yields a formula $\phi = \bigwedge_{k=1}^{K} \varphi_k$ such that each φ_k is a disjunction of literals satisfying the requirements of lemma 8.

Clearly, $A \in Mod(SP)$ implies $A \models \phi$. We choose some $m \geq \text{connects}(\varphi_k)$ for all $k \in \{1, \ldots, K\}$. The lemma then implies $A_m \models \varphi_k$ for all k, and therefore we have $A_m \models \phi$. This implies $A_m \models \Phi$, and hence $A_m \in Mod(SP)$. Thus $Mod(SP) \neq [A]$. ■

7 Conclusions

It is well known that there is no need for introducing quantifiers in order to specify computable data types if hiding facilities are used. Matters are quite different if hiding facilities are disabled: in this paper, we provided an example of a (natural) computable data type that has a monomorphic first-order specification but no monomorphic quantifier-free specification (both without hidden symbols).

Even if the use of quantifiers does not completely compensate the loss of expressiveness caused by disabling hiding (cf. [Ber93,Sch97a,Sch97b]), theorem 1 indicates that quantifiers make up for this loss at least to some extent. In other words, we have shown that the use of quantifiers can help to avoid introducing hidden functions. Since hidden functions often make specifications less readable and may cause technical inconvenience (e.g. the handling of clashes between hidden symbols when combining specifications), this paper can be seen as support for the provision of quantifiers in specification languages.[7]

While we believe that our result is proven for the first time, the problem was already mentioned in [BDP+79]: Broy et. al. give several examples for specifications that demonstrate the usefulness of quantifiers, and they even claim:

[7] Notice that our arguments are also valid for the case that initial algebra semantics is adopted: Since there are no homomorphisms from A to the non-standard models A_m considered in section 6, the restriction to initial models does not eliminate the necessity of either quantifiers or hidden functions. (Unfortunately, quantifiers would spoil the guarantee of the existence of initial models and executability.)

> "A formulation without existential quantifiers requires the introduction of additional (hidden) functions, which makes the presentation inconvenient ..."

However, they refer to a non-monomorphic data type[8] and do not provide a proof for their statement.

Furthermore, our work is somewhat related to the work done in finite model theory and descriptive complexity, where especially the (first-order) definability of properties of finite graphs is studied (cf. e.g. [EF95] and [Imm82]). A typical result is, for instance, that there is no first-order formula φ such that

$$G \models \varphi \iff G \text{ is connected}$$

for all finite graphs G (where the variables in φ are required to range over the nodes of G) (cf. [Gai82]). Unfortunately, such results do not carry over to our setting because we can use variables ranging over graphs (which allows, for instance, a first-order specification of connectedness) and even inductive definitions (since the property to be specified is named by a symbol, in our case `complete`).

Moreover, instead of merely providing an example, one could be interested in a *characterization* of the data types that can or cannot be specified without hiding and without quantifiers. It is well known that for any first-order formula φ there is an equivalent quantifier-free formula φ' iff the class of models of φ is closed under subalgebras [Rob74, Theorem 3.3.4]. However, this result does not apply in our context, because even if the class of *all* models of φ is not closed under subalgebras, there can be a quantifier-free formula having the same *term-generated* models.

Another approach is pursued in [BT87], [BBTW81], and others: the authors characterize specifiability (using hidden functions) in terms of complexity (cf. e.g. [EM81,AT82]). However, fact 2 indicates that specifiability without hidden functions generally does not imply any bound on the data type's complexity. Conversely, it might be interesting to study whether (certain) complexity bounds guarantee specifiability without hidden functions.

Acknowledgments. We thank the anonymous referees for their helpful comments.

References

AT82. P.R.J. Asveld and J.V. Tucker. Complexity theory and the operational structure of algebraic programming systems. *Acta Inform.*, 17:451–476, 1982. 56

BBTW81. J.A. Bergstra, M. Broy, J.V. Tucker, and M. Wirsing. On the power of algebraic specifications. In J. Gruska and M. Chytil, editors, *Proc. of the 10th Symp. on Math. Found. of Comp. Sci.*, volume 118 of *LNCS*, pages 193–204. Springer, 1981. 45, 56

[8] In the case of non-monomorphic data types, it is much easier to construct an example that demonstrates the necessity of quantifiers (e.g. [BDP+79, p. 84]).

BDP+79. M. Broy, W. Dosch, H. Partsch, P. Pepper, and M. Wirsing. Existential quantifiers in abstract data types. In H.A. Maurer, editor, *Automata, Languages and Programming, 6th Colloquium*, volume 71 of *LNCS*, pages 73–87. Springer, 1979. 55, 56

Ber93. R. Berghammer. On the characterization of the integers: the hidden function problem revisited. *Acta Cybernetica*, 11(1-2):85–96, 1993. 46, 55

BT82. J.A. Bergstra and J.V. Tucker. The completeness of the algebraic specification methods for computable data types. *Information and Control*, 54:186–200, 1982. 45, 45, 45

BT87. J.A. Bergstra and J.V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoret. Comput. Sci.*, 50:137–181, 1987. 45, 45, 45, 56

EF95. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995. 56

EM81. H. Ehrig and B. Mahr. Complexity of algebraic implementations for abstract data types. *Computer and System Sciences*, 23(2):223–253, 1981. 56

Gai82. H. Gaifman. On local and non-local properties. In J. Stern, editor, *Proc. of the Herbrand Symp., Logic Colloq.'81*, pages 105–135. North-Holland, 1982. 56

Imm82. N. Immerman. Upper and lower bounds for first order expressibility. *Computer and System Sciences*, 25:76–98, 1982. 56

Kem98. D. Kempe. Ausdrucksmächtigkeit von Quantoren in algebraischen Spezifikationen. Master's thesis, Universität Karlsruhe, August 1998. 45

LEW96. J. Loeckx, H.-D. Ehrich, and M. Wolf. *Specification of abstract data types*. Verlag Wiley-Teubner, 1996. 46

Maj79. M.E. Majster. Data types, abstract data types and their specification problem. *Theoret. Comput. Sci.*, 8:89–127, 1979. 45

Mat93. Y. Matiyasevich. *Hilbert's Tenth Problem*. MIT Press, 1993. 48, 48

Ore79. F. Orejas. On the power of conditional specifications. *ACM SIGPLAN Notices*, 14(7):78–81, 1979. 45

Rob74. A. Robinson. *Introduction to model theory and to the metamathematics of algebra*. Studies in Logic and the Foundations of Mathematics. North Holland Publishers, 1974. 56

Sch97a. A. Schönegge. An answer to the hidden function question for algebraic specification methods (abstract), 4th Workshop on Logic, Language, Information and Computation. *Logic Journal of the IGPL*, 5(6), 1997. 46, 55

Sch97b. A. Schönegge. The hidden function question revisited. In *Proc. of the 6th Int. Conf. on Algebraic Methodology and Software Technology (AMAST '97)*, volume 1349 of *LNCS*, pages 451–464. Springer, 1997. 46, 55

TWW82. J.W. Thatcher, E.G. Wagner, and J.B. Wright. Data type specification: Parameterization and the power of specification techniques. In *ACM Trans. on Programming Languages and Systems*, volume 4, pages 711–732, 1982. 45

Wir90. M. Wirsing. *Algebraic Specification*, volume B of *Handbook of Theoretical Computer Science*, pages 675–788. Elsevier Science Publishers B. V., 1990. 45, 45, 46, 46, 50

On the Effective Semantics of Nondeterministic, Nonmonotonic, Temporal Logic Databases

Fosca Giannotti[1], Giuseppe Manco[1], Mirco Nanni[2], and Dino Pedreschi[2]

[1] CNUCE Institute of CNR, Via S. Maria 36, 56125 Pisa, Italy
F.Giannotti@cnuce.cnr.it, G.Manco@guest.cnuce.cnr.it
[2] Dipartimento di Informatica, Università di Pisa, C.so Italia 40, 56125 Pisa, Italy
{nnanni,pedre}@DI.Unipi.IT

Abstract. We consider in this paper an extension of Datalog with mechanisms for temporal, nonmonotonic and nondeterministic reasoning, which we refer to as Datalog++. We study its semantics, and show how iterated fixpoint and stable model semantics can be combined to the purpose of clarifying the interpretation of Datalog++ programs, and supporting their efficient execution. On this basis, the design of appropriate optimization techniques for Datalog++ is also discussed.

1 Introduction

Motivations. The name **Datalog++** is used in this paper to refer to Datalog extended with mechanisms supporting:

- *temporal* reasoning, by means of temporal, or *stage*, arguments of relations, ranging over a discrete temporal domain, in the style of Datalog$_{1S}$ [5];
- *nonmonotonic* reasoning, by means of a form of stratified negation w.r.t. the stage arguments, called *XY-stratification* [19];
- *nondeterministic* reasoning, by means of the nondeterministic `choice` construct [8].

Datalog++, which is essentially a fragment of $\mathcal{LDL}$++ [2], and is advocated in [20, Chap. 10], revealed a highly expressive language, with applications in diverse areas such as AI planning [4], active rules [18], object databases [7], semistructured information management and Web restructuring [7]. However, we are still missing a thorough study of the semantics of Datalog++, which will provide a basis to sound and efficient implementations and optimization techniques. A preliminary study of the semantics for a generalization of Datalog++ is sketched in [4], but their approach presents some problems that are fixed in this paper.

Objective. The goal of this paper is to provide a declarative semantics for Datalog++, which accommodates and integrates the temporal, nonmonotonic and nondeterministic mechanisms, and which justifies the adoption of an iterated fixpoint semantics for the language, that leads to its efficient implementation.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 58–72, 1999.

We proceed as follows:

1. a natural, noneffective, semantics for Datalog++ is assigned using the notion of a *stable model*;
2. an effective semantics is then assigned using an iterative procedure which exploits the stratification induced by the progression of the temporal argument;
3. in the main result of this paper, we show that 1. and 2. are equivalent, provided that a natural syntactic restriction is fulfilled, which imposes a disciplined use of the temporal argument within the *choice* construct.

On the basis of this result, we finally discuss a repertoire of optimization techniques, especially tailored for Datalog++. In particular, we discuss how it is possible to support efficient history-insensitive temporal reasoning by means of real side-effects during the iterated computation [13].

Related Work. Nondeterminism is introduced in deductive databases by means of the `choice` construct. The original proposal in [11] was later revised in [17], and refined in [8]. These studies exposed the close relationship connecting nonmonotonic reasoning with nondeterministic constructs, leading to the definition of a stable model semantics for `choice`. While the declarative semantics of `choice` is based on *stable model* semantics which is computationally intractable in general, `choice` is amenable to efficient implementations, and it is actually supported in the logic database language $\mathcal{LDL}$ [14] and its evolution $\mathcal{LDL}$++ [2].

On the other side, stratification has been a crucial notion for the introduction of nonmonotonic reasoning in deductive databases. From the original idea in [1] of a static stratification based on predicate dependencies, stratified negation has been refined to deal with dynamic notions, as in the case of locally stratified programs [15] and modularly stratified programs [16]. Dynamic, or local, stratification has a close connection with temporal reasoning, as the progression of time points yields an obvious stratification of programs—consider for instance Datalog$_{1S}$ [5]. It is therefore natural that non monotonic and temporal reasoning are combined in several deductive database languages, such as those in [12], [10], [7], [20, Chap. 10].

However, a striking mismatch is apparent between the above two lines of research: nondeterminism leads to a multiplicity of (stable) models, whereas stratification leads to a unique (perfect) model. So far, no comprehensive study has addressed the combination of the two lines, which occurs in Datalog++, and which requires the development of a *non deterministic iterated fixpoint* procedure. We notice however the mentioned exception of [4], where an approach to this problem is sketched with reference to locally stratified programs augmented with `choice`. In the present paper, we present instead a thorough treatment of Datalog++ programs, and repair a problem of the approach in [4] concerning the incompleteness of the iterated fixpoint procedure.

2 Background on NonDeterminism and XY-stratification

Nondeterministic choice. The `choice` construct is used to nondeterministically select subsets of answers to queries, which obey a specified FD constraint. For instance, the rule

$$\texttt{st\_ad}(\texttt{St}, \texttt{Ad}) \leftarrow \texttt{major}(\texttt{St}, \texttt{Area}), \texttt{faculty}(\texttt{Ad}, \texttt{Area}), \texttt{choice}((\texttt{St}), (\texttt{Ad})).$$

assigns to each student a unique, arbitrary advisor from the same area, since the `choice` goal constrains the `st_ad` relation to obey the FD ($\texttt{St} \rightarrow \texttt{Ad}$).

The semantics of `choice` is assigned using the so-called *stable model* semantics of Datalog¬ programs, a concept originating from autoepistemic logic, which was applied to the study of negation in Horn clause languages by Gelfond and Lifschitz [6]. To define the notion of a stable model we need to introduce a transformation H which, given an interpretation I, maps a Datalog¬ program P into a positive Datalog program $H(P, I)$:

$$H(P, I) = \{A \leftarrow B_1, \ldots, B_n \mid A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m \in \mathit{ground}(P) \wedge \{C_1, \ldots, C_m\} \cap I = \emptyset\}$$

Next, we define:

$$S_P(I) = T_{H(P,I)} \uparrow \omega$$

Then, M is said to be a *stable model* of P if $S_P(M) = M$. In general, Datalog¬ programs may have zero, one or many stable models. The multiplicity of stable models can be exploited to give a declarative account of nondeterminism.

We can in fact define the *stable version* of a program P, $SV(P)$, to be the program transformation where all the references to the `choice` atom in a rule $r : H \leftarrow B, \texttt{choice}(\texttt{X}, \texttt{Y})$ are replaced by the atom $\texttt{chosen}_\texttt{r}(\texttt{X}, \texttt{Y})$, and define the $\texttt{chosen}_\texttt{r}$ predicate with the following rules:

$$\texttt{chosen}_\texttt{r}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{B}, \neg\texttt{diffchoice}_\texttt{r}(\texttt{X}, \texttt{Y}).$$
$$\texttt{diffchoice}_\texttt{r}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{chosen}_\texttt{r}(\texttt{X}, \texttt{Y}'), \texttt{Y} \neq \texttt{Y}'.$$

where, for any fixed value of `X`, each choice for `Y` inhibits all the other possible ones via $\texttt{diffchoice}_\texttt{r}$, so that in the stable models of $SV(P)$ there is (only) one of them. Notice that, by construction, each occurrence of a `choice` atom has its own pair of `chosen` and `diffchoice` atoms, thus bounding the scope of the atom to the rule it appears in. The various stable models of the transformed program $SV(P)$ thus correspond to the choice models of the original program.

XY-programs. Another notion used in this paper is that of XY-programs originally introduced in [19]. The language of such programs is $Datalog^{\neg}_{1S}$, which admits negation on body atoms and a unary constructor symbol, used to represent a temporal argument usually called the *stage argument*. A general definition of XY-programs is the following. A set P of rules defining mutually recursive predicates, is an XY-program if it satisfies the following conditions:

1. each recursive predicate has a distinguished stage argument;
2. every recursive rule r is either an X-rule or a Y-rule, where:
   - r is an X-rule when the stage argument in every recursive predicates in r is the same variable,
   - r is a Y-rule when (i) the head of r has a stage argument $s(J)$, where J is a variable, (ii) some goal of r has J as its stage argument, and (iii) the remaining recursive goals have either J or $s(J)$ as their stage argument.

Intuitively, in the rules of XY-programs, an atom $\mathtt{p(J,\_)}$ denotes the extension of relation `p` at the current stage (present time) `J`, whereas an atom $\mathtt{p(s(J),\_)}$ denotes the extension of relation `p` at the next stage (future time) $\mathtt{s(J)}$. By using a different *primed* predicate symbol $\mathtt{p'}$ in the $\mathtt{p(s(J),\_)}$ atoms, we obtain the so-called *primed version* of an XY-program. We say that an XY-program is *XY-stratified* if its primed version is a stratified program. Intuitively, if the dependency graph of the primed version has no cycles through negated edges, then it is possible to obtain an ordering on the original rules modulo the stage arguments. As a consequence, an XY-stratified program is also locally stratified, and has therefore a unique stable model that coincides with its perfect model [15].

Let P be an XY-stratified program. Then, for each $i > 0$, define P_i as

$$P_i = \{r[s^i(nil)/I] \mid r \in P, I \text{ is the stage argument of the head of } r\}$$

i.e., P_i is the set of rule instances of P that define the predicates with stage argument $s^i(nil) = i$ (here $r[x/I]$ stands for "replacing I with x in r"). Then the iterated fixpoint procedure for computing the (unique) minimal model of P can be defined as follows:

1. compute M_0 as the minimal model of P_0;
2. for each $j > 0$ compute M_j as the minimal model of $P_j \cup M_{j-1}$.

Notice that for each $j \geq 0$, P_j is stratified by the definition, and hence its perfect model M_j is computable via an iterated fixpoint procedure.

In this paper, we use the name Datalog++ to refer to the language of XY-programs augmented with `choice` goals.

3 A Semantics for Datalog++

When `choice` constructs are allowed in XY-programs, a multiplicity of stable models exists for any given program, and therefore it is needed to clarify how this phenomenon combines with the iterated fixpoint semantics of `choice`-free XY-programs. This task is accomplished in three steps.

1. First, we present a general result stating that, whenever a Datalog¬ program P is stratifiable into a hierarchy of recursive cliques $Q_1, Q_2, \ldots$, then any stable model of the entire program P can be reconstructed by iterating the construction of approximating stable models, each associated to a clique.

2. Second, we observe that, under a syntactic restriction on the use of the `choice` construct that does not compromise expressiveness, Datalog++ programs can be naturally stratified into a hierarchy of recursive cliques Q_1, $Q_2, \ldots$, by using the temporal arguments of recursive predicates.
3. Third, by the observation in 2., we can apply the general result in 1. to Datalog++ programs, thus obtaining that the stable models of the entire program can be computed by an iterative fixpoint procedure which follows the stratification induced by the temporal arguments.

Given a (possibly infinite) program P, consider a (possibly infinite) topological sort of its distinct recursive cliques $Q_1 \prec Q_2 \ldots \prec Q_i \prec \ldots$, induced by the dependency relation over the predicates of P. Given an interpretation I, we use the notation I_i to denote the subset of atoms of I whose predicate symbols are predicates defined in clique Q_i.

The following observations are straightforward:

- $\bigcup_{i>0} I_i = I$, and analogously $\bigcup_{i>0} Q_i = P$;
- the predicates defined in Q_{i+1} depend only on the definitions in $Q_1 \cup \ldots \cup Q_i$; as a consequence, the interpretation of Q_{i+1} is $I_1 \cup \ldots \cup I_i \cup I_{i+1}$ (i.e., we can ignore $\bigcup_{j>i+1} I_j$).

The next definition shows how to transform each clique, within the given topological ordering, in a self-contained program which takes into account the information deduced by the previous cliques. Such transformation resembles the Gelfond-Lifschitz transformation reported in Sect. 2.

Definition 1. *Consider a program P, a topological sort of its cliques $Q_1 \prec Q_2 \ldots \prec Q_i \ldots$, and an interpretation $I = \bigcup_{i>0} I_i$. Now define*

$$\begin{aligned} Q_i^{red(I)} = \{ & H \leftarrow B_1, \ldots, B_n \mid H \leftarrow B_1, \ldots, B_n, C_1, \ldots, C_m \in ground(Q_i) \\ & \wedge\ B_1, \ldots, B_n \text{ are defined in } Q_i \\ & \wedge\ C_1, \ldots, C_m \text{ are defined in } (Q_1 \cup \ldots \cup Q_{i-1}) \\ & \wedge\ I_1 \cup \ldots \cup I_{i-1} \models C_1, \ldots, C_m \} \end{aligned}$$ □

The idea underlying the transformation is to remove from each clique Q_i all the dependencies induced by the predicates which are defined in lower cliques. We abbreviate $Q_i^{red(I)}$ by Q_i^{red}, when the interpretation I is clear by the context.

Example 1. Consider the program $P = \{\mathtt{p} \leftarrow \mathtt{q}, \mathtt{r}.\ \ \mathtt{q} \leftarrow \mathtt{r}, \mathtt{t}.\ \ \mathtt{r} \leftarrow \mathtt{q}, \mathtt{s}.\}$ and the cliques $Q_1 = \{\mathtt{q} \leftarrow \mathtt{r}, \mathtt{t}.\ \ \mathtt{r} \leftarrow \mathtt{q}, \mathtt{s}.\}$ and $Q_2 = \{\mathtt{p} \leftarrow \mathtt{q}, \mathtt{r}.\}$. Now, consider the interpretation $I = \{\mathtt{s}, \mathtt{q}, \mathtt{r}\}$. Then $Q_1^{red} = \{\mathtt{q} \leftarrow \mathtt{r}, \mathtt{t}.\ \ \mathtt{r} \leftarrow \mathtt{q}, \mathtt{s}.\}$ and $Q_2^{red} = \{\mathtt{p} \leftarrow .\}$. □

The following Lemma 1 states the relation between the models of the transformed cliques and the models of the program. We abbreviate $I_1 \cup \ldots \cup I_i$ with $I^{(i)}$, and analogously for $Q^{(i)}$.

Lemma 1. *Given a (possibly infinite) Datalog¬ program P and an interpretation I, let $Q_1 \prec Q_2 \ldots \prec Q_i \ldots$ and $I_1 \prec I_2 \ldots \prec I_i \ldots$ be the topological sorts on P and I induced by the dependency relation of P. Then the following statements are equivalent:*

1. $S_P(I) = I$
2. $\forall i > 0.\ S_{Q_i^{red}}(I_i) = I_i$
3. $\forall i > 0.\ S_{Q^{(i)}}(I^{(i)}) = I^{(i)}$

Proof. See Appendix. □

This result states that an arbitrary Datalog¬ program has a stable model if and only if each its approximating clique, according to the given topological sort, has a *local* stable model. This result gives us an intuitive idea for computing the stable models of an approximable program by means of the computation the stable models of its approximating cliques.

Notice that Lemma 1 holds for arbitrary programs, provided that a stratification into a hierarchy of cliques is given. In this sense, this result is more widely applicable than the various notions of stratified programs, such as that of *modularly stratified programs* [16], in which it is required that each clique Q_i^{red} is locally stratified. On the contrary, we do not require here that each clique is, in any sense, stratified. This is motivated by the objective of dealing with non determinism, and justifies why we adopt the (nondeterministic) stable model semantics, rather than other deterministic semantics for (stratified) Datalog¬ programs, such as, for instance, perfect model semantics [15].

We turn now our attention to XY-programs. The result of instantiating the clauses of an XY-program P with all possible values (natural numbers) of the stage argument, yields a new program $SG(P)$ (for *stage ground*). More precisely, $SG(P) = \bigcup_{i \geq 0} P_i$, where

$$P_i = \{r[i/I] \mid r \text{ is a rule of } P,\ I \text{ is the stage argument of } r\}.$$

The stable models of P and $SG(P)$ are closely related:

Lemma 2. *Let P be an XY-program. Then, for each interpretation I:*

$$S_P(I) = I \iff S_{SG(P)}(I) = I$$

Proof. See Appendix. □

However, the dependency graph of $SG(P)$ (which is obviously the same as P) does not induce necessarily a topological sort, because in general XY-programs are not stratified, and therefore Lemma 1 is not directly applicable. To tackle this problem, we distinguish the predicate symbol p in the program fragment P_i from the same predicate symbol in all other fragments P_j with $j \neq i$, by differentiating the predicate symbols using the temporal argument. Therefore, if $\mathtt{p(i, x)}$ is an atom involved in some rule of P_i, its modified version is $\mathtt{p_i(x)}$. More precisely, we introduce, for any XY-program P, its modified version $SO(P)$ (for *stage-out*), defined by $SO(P) = \bigcup_{i \geq 0} SO(P)_i$ where $SO(P)_i$ is obtained from the program fragment P_i of $SG(P)$ by extracting the stage arguments from any atom, and adding it to the predicate symbol of the atom. Similarly, the modified version $SO(I)$ of an interpretation I is defined. Therefore, the atom $\mathtt{p(i, x)}$ is

in I iff the atom $\mathtt{p_i(x)}$ is in $SO(I)$, where $\mathtt{i}$ is the value in the stage argument position of relation $\mathtt{p}$.

Unsurprisingly, the stable models of $SG(P)$ and $SO(P)$ are closely related:

Lemma 3. *Let P be an XY-program. Then, for each interpretation I:*

$$S_{SG(P)}(I) = I \Longleftrightarrow S_{SO(P)}(SO(I)) = SO(I).$$

Proof. See Appendix. □

Our aim is now to conclude that, for a given Datalog++ program P:

(a) $SO(P)_0 \prec SO(P)_1 \prec \cdots$ is the topological sort over $SO(P)$ in the hypothesis of Lemma 1 [1]; recall that, for $i \geq 0$, $SO(P)_i$ consists of the rules from $SO(P)$ with stage argument i in their heads;

(b) by Lemmas 1, 2 and 3, an interpretation I is a stable model of P iff I can be constructed as $\bigcup_{i\geq 0} I_i$, where, for $i \geq 0$, I_i is a stable model of $SO(P)_i^{red(I^{(i)})}$, i.e. the clique $SO(P)_i$ reduced by substituting the atoms deduced at stages earlier than i.

On the basis of (b) above, it is possible to define an iterative procedure to construct an arbitrary stable model M of P as the union of the interpretations $M_0, M_1, \ldots$ defined as follows:

Iterated stable model procedure.

Base case. M_0 is a stable model of the bottom clique $SO(P)_0$.

Induction case. For $i > 0$, M_i is a stable model of $SO(P)_i^{red(M^{(i)})}$, i.e. the clique $SO(P)_i$ reduced with respect to $M_0 \cup \cdots \cup M_{i-1}$. □

The interpretation $M = \bigcup_{i\geq 0} M_i$ is called an *iterated stable model* of P.

It should be observed that this construction is close to the procedure called *iterated choice fixpoint* in [4]. Also, following the approach of [9], each local stable model M_i can in turn be efficiently constructed by a nondeterministic fixpoint computation, in polynomial time.

Unfortunately, the desired result that the notions of stable model and iterated stable model coincide does not hold in full generality, in the sense that the iterative procedure in is not complete for arbitrary Datalog++ programs. In fact, as demonstrated by the example below, an undisciplined use of `choice` in Datalog++ programs may cause the presence of stable models that cannot be computed incrementally over the hierarchy of cliques.

Example 2. Consider the following simple Datalog++ program P:

[1] In general, $SO(P)_i$ can be composed by more than one clique, so that in the above expression it should be replaced by $SO(P)_i^1 \prec \cdots \prec SO(P)_i^{n_i}$. However, for ease of presentation we ignore it, since such general case is trivially deduceable from what follows.

$$\mathtt{q}(\mathtt{0},\mathtt{a}).$$
$$\mathtt{q}(\mathtt{s}(\mathtt{I}),\mathtt{b}) \leftarrow \mathtt{q}(\mathtt{I},\mathtt{a}).$$
$$\mathtt{p}(\mathtt{I},\mathtt{X}) \leftarrow \mathtt{q}(\mathtt{I},\mathtt{X}), \mathtt{choice}((),\mathtt{X}).$$

In the stable version $SV(P)$ of P, the rule defining predicate p is replaced by:

$$\mathtt{p}(\mathtt{I},\mathtt{X}) \leftarrow \mathtt{q}(\mathtt{I},\mathtt{X}), \mathtt{chosen}(\mathtt{X}).$$
$$\mathtt{chosen}(\mathtt{X}) \leftarrow \mathtt{q}(\mathtt{I},\mathtt{X}), \neg\mathtt{diffchoice}(\mathtt{X}).$$
$$\mathtt{diffchoice}(\mathtt{X}) \leftarrow \mathtt{chosen}(\mathtt{Y}), \mathtt{Y} \neq \mathtt{X}.$$

It is readily checked that $SV(P)$ admits two stable models, namely $\{\mathtt{q}(\mathtt{0},\mathtt{a}), \mathtt{q}(\mathtt{s}(\mathtt{0}),\mathtt{b}), \mathtt{p}(\mathtt{0},\mathtt{a})\}$ and $\{\mathtt{q}(\mathtt{0},\mathtt{a}), \mathtt{q}(\mathtt{s}(\mathtt{0}),\mathtt{b}), \mathtt{p}(\mathtt{s}(\mathtt{0}),\mathtt{b})\}$, but only the first model is an iterated stable models, and therefore the second model cannot be computed using the *iterated choice fixpoint* of [4]. □

The technical reason for this problem is that the free use of the `choice` construct inhibits the possibility of defining a topological sort on $SO(P)$ based on the value of the stage argument. In the example 2, the predicate dependency relation of $SO(SV(P))$ induces a dependency among stage i and the stages $j > i$, because of the dependency of the `chosen` predicate from the predicates $\mathtt{q}_\mathtt{i}$ for all stages $\mathtt{i} \geq \mathtt{0}$.

To prevent this problem, it is suffices to require that `choice` goals refer the stage argument I in the domain of the associated functional dependency. The Datalog++ programs which comply with this constraint are called *choice-safe*. The following is a way to turn the program of example 2 into a choice-safe program (with a different semantics):

$$\mathtt{p}(\mathtt{I},\mathtt{X}) \leftarrow \mathtt{q}(\mathtt{I},\mathtt{X}), \mathtt{choice}(\mathtt{I},\mathtt{X}).$$

This syntactic restriction, moreover, does not greatly compromise the expressiveness of the query language, in that it is possible to simulate within this restriction most of the general use of `choice` (see [13]).

The above considerations are summarized in the following main result of the paper, which, under the mentioned restriction of choice-safety, is a direct consequence of Lemmas 1, 2 and 3.

Theorem 1 (Correctness and completeness of the iterated stable model procedure).
Let P be a choice-safe Datalog++ program and I an interpretation. Then I is a stable model of $SV(P)$ iff it is an iterated stable model of P. □

The following example shows a computation with the iterated stable model procedure.

Example 3. Consider the following Datalog++ version of the *seminaive* program, discussed in [19], which non-deterministically computes a maximal path from node a over a graph `g`:

$\texttt{delta}(\texttt{0},\texttt{a})$.
$\texttt{delta}(\texttt{s}(\texttt{I}),\texttt{Y}) \leftarrow \texttt{delta}(\texttt{I},\texttt{X}), \texttt{g}(\texttt{X},\texttt{Y}), \neg\texttt{all}(\texttt{I},\texttt{Y}), \texttt{choice}((\texttt{I},\texttt{X}),\texttt{Y})$.
$\texttt{all}(\texttt{I},\texttt{X}) \leftarrow \texttt{delta}(\texttt{I},\texttt{X})$.
$\texttt{all}(\texttt{s}(\texttt{I}),\texttt{X}) \leftarrow \texttt{all}(\texttt{I},\texttt{X}), \texttt{delta}(\texttt{s}(\texttt{I}),\_)$.

Assume that the graph is given by $\texttt{g} = \{\langle \texttt{a},\texttt{b}\rangle, \langle \texttt{b},\texttt{c}\rangle, \langle \texttt{b},\texttt{d}\rangle, \langle \texttt{d},\texttt{e}\rangle\}$. The following interpretations are carried out at each stage of the iterated stable model procedure:

1. $I_0 = \{\texttt{delta}_0(\texttt{a}), \texttt{all}_0(\texttt{a})\}$.
2. $I_1 = \{\texttt{all}_1(\texttt{a}), \texttt{all}_1(\texttt{b}), \texttt{delta}_1(\texttt{b})\}$.
3. $I_2^1 = \{\texttt{all}_2(\texttt{a}), \texttt{all}_2(\texttt{b}), \texttt{delta}_2(\texttt{c}), \texttt{all}_2(\texttt{c})\}$,
   $I_2^2 = \{\texttt{all}_2(\texttt{a}), \texttt{all}_2(\texttt{b}), \texttt{delta}_2(\texttt{d}), \texttt{all}_2(\texttt{d})\}$
4. $I_3^1 = \emptyset$, $I_3^2 = \{\texttt{all}_3(\texttt{a}), \texttt{all}_3(\texttt{b}), \texttt{all}_3(\texttt{d}), \texttt{delta}_3(\texttt{e}), \texttt{all}_3(\texttt{e})\}$
5. $I_j = \emptyset$ for $j > 3$.

By Theorem 1, we conclude that there are two stable models for the program: $I^1 = I_0 \cup I_1 \cup I_2^1$ and $I^2 = I_0 \cup I_1 \cup I_2^2 \cup I_3^2$. Clearly, any realistic implementation, such as that provided in $\mathcal{LDL}$++, computes non deterministically only *one* of the possible stable models. □

4 Optimization of Datalog++ queries

A systematic study of query optimization techniques is needed to achieve efficient implementations of the iterated stable model procedure. In this section we sketch the direction along with our research is developing. A first line is concerned with defining ad hoc optimizations for Datalog++, by exploiting the particular syntactic structure due to the temporal arguments, which represents a sistematization and an extension of ideas first presented in [19]. A second line of research investigates how to gear classic optimizations, such as magic sets, to Datalog++.

Forgetful-fixpoint computations. In many applications (e.g., modeling updates and active rules [18,7]) queries are issued with reference to the final stage only (which represents the commit state of the database). Such queries often exhibit the form $\texttt{p}(\texttt{I},\texttt{X}), \neg\texttt{p}(\texttt{s}(\texttt{I}),\_)$, with the intended meaning "find the value X of p in the final state of p". This implies that (i) when computing the next stage, we can forget all the preceding states but the last one (see [19]), and (ii) if there exists a stage I such that $\texttt{p}(\texttt{I},\texttt{X}), \neg\texttt{p}(\texttt{s}(\texttt{I}),\_)$ is unique, we can stop the computation process once the above query is satisfied. For instance, the program in Example 3 with the query $\texttt{delta}(\texttt{I},\texttt{X}), \neg\texttt{delta}(\texttt{s}(\texttt{I}),\_)$ computes the last node visited in a (nondeterministically chosen) maximal path starting from a. To answer this query, it suffices to consider either the partial model I_2^1 or the partial model I_3^2, and hence we can discard the previous partial models during the computation.

Another interesting case occurs when the answer to the query is distributed along the stages, e.g., when we are interested in the answer to a query such as $\texttt{delta}(\_, \texttt{X})$, which ignores the stage argument. In this case, we can collect the partial answers via a gathering predicate defined with a copy-rule. For instance, the `all` predicate in example 3 collects all the nodes reachable from a in the selected path. Then the query $\texttt{all}(\texttt{I}, \texttt{X}), \neg\texttt{all}(\texttt{s}(\texttt{I}), \_)$, which is amenable for the described optimization, is equivalent to the query $\texttt{delta}(\_, \texttt{X})$, which on the contrary does not allow it. Therefore, by (possibly) modifying the program with copy-rules for the `all` predicate, we can apply systematically the space optimized forgetful-fixpoint.

Delta-fixpoint computations. We already mentioned the presence of a *copy-rule* in example 3:

$$\texttt{all}(\texttt{s}(\texttt{I}), \texttt{X}) \leftarrow \texttt{all}(\texttt{I}, \texttt{X}), \texttt{delta}(\texttt{s}(\texttt{I}), \_).$$

Its effect is that of copying all the tuples from the stage `I` to the next one, if any. We can avoid such useless space occupation, by mantaining for each stage only the modifications which are to be applied to the original relation in order to obtain the actual version. For example, the above rule represents no modification at all, and hence it should not have any effect; indeed, it suffices to keep track of the additions to the original database dictated by the other rule:

$$\texttt{all}(\texttt{I}, \texttt{X}) \leftarrow \texttt{delta}(\texttt{I}, \texttt{X}).$$

which can be realised by a supplementary relation $\texttt{all}^+$ containing, at each stage, the new tuples produced. If we replace the *copy-rule* with a *delete-rule* of the form:

$$\texttt{all}(\texttt{s}(\texttt{I}), \texttt{X}) \leftarrow \texttt{all}(\texttt{I}, \texttt{X}), \texttt{delta}(\texttt{s}(\texttt{I}), \_), \neg\texttt{q}(\texttt{X}).$$

we need simply to keep track of the negative contribution due to literal $\neg\texttt{q}(\texttt{X})$, which can be stored in a relation $\texttt{all}^-$. Each $\texttt{all}(\texttt{I}, \ldots)$ can then be obtained by integrating $\texttt{all}(0, \ldots)$ with all the $\texttt{all}^+(\texttt{J}, \ldots)$ and $\texttt{all}^-(\texttt{J}, \ldots)$ atoms, with $J \leq I$.

Side-effect computations. A direct combination of the previous two techniques gives rise to a form of *side-effect* computation. Let us consider, as an example, the nondeterministic ordering of an array performed by swapping at each step any two elements which violate ordering. Here, the array $\texttt{a} =< \texttt{a}_1, \cdots, \texttt{a}_\texttt{n} >$ is represented by the relation `a` with extension $\texttt{a}(1, \texttt{a}_1), \cdots, \texttt{a}(\texttt{n}, \texttt{a}_\texttt{n})$.

$$\begin{aligned}
&\texttt{ar}(0, \texttt{P}, \texttt{Y}) && \leftarrow \texttt{a}(\texttt{P}, \texttt{Y}).\\
&\texttt{swp}(\texttt{I}, \texttt{P1}, \texttt{P2}) && \leftarrow \texttt{ar}(\texttt{I}, \texttt{P1}, \texttt{X}), \texttt{ar}(\texttt{I}, \texttt{P2}, \texttt{Y}),\ \texttt{X} > \texttt{Y},\ \texttt{P1} < \texttt{P2},\\
& && \quad \texttt{choice}((\texttt{I}), (\texttt{P1}, \texttt{P2})).\\
&\texttt{ar}(\texttt{s}(\texttt{I}), \texttt{P}, \texttt{X}) && \leftarrow \texttt{ar}(\texttt{I}, \texttt{P}, \texttt{X}),\ \neg\texttt{swp}(\texttt{I}, \texttt{P}, _)\ \neg\texttt{swp}(\texttt{I}, _, \texttt{P}).\\
&\texttt{ar}(\texttt{s}(\texttt{I}), \texttt{P}, \texttt{X}) && \leftarrow \texttt{ar}(\texttt{I}, \texttt{P1}, \texttt{X}),\ \texttt{swp}(\texttt{I}, \texttt{P1}, \texttt{P}).\\
&\texttt{ar}(\texttt{s}(\texttt{I}), \texttt{P}, \texttt{X}) && \leftarrow \texttt{ar}(\texttt{I}, \texttt{P1}, \texttt{X}),\ \texttt{swp}(\texttt{I}, \texttt{P}, \texttt{P1}).\\
&? \ \texttt{ar}(\texttt{I}, \texttt{X}, \texttt{Y}), \neg\texttt{ar}(\texttt{s}(\texttt{I}), _, _)
\end{aligned}$$

At each stage i we nondeterministically select an unordered pair x, y of elements, delete the array atoms ar(i, p1, x) and ar(i, p2, y) where they appear, and add the new atoms ar(s(i), p1, y) and ar(s(i), p2, x) representing the swapped pair. The query allows a forgetful-fixpoint computation (in particular, the stage selected by the query is unique), and the definition of predicate ar is composed by delete-rules and an add-rule. This means that at each step we can (i) forget the previously computed stages (but the last), and (ii) avoid copying most of relation ar, keeping track only of the deletions and additions to be performed. If the requested update are immediately performed, the execution of the proposed program, then, boils down to the efficient iterative computation of the following (nondeterministic) Pascal-like program:

$$\textbf{while } \exists I \ a[I] > a[I+1] \textbf{ do } swap(a[I], a[I+1]) \textbf{ od}$$

Magic Sets. Consider choice-free XY-programs. It is well known that the standard magic-sets technique cannot be directly applied to programs with negation, hence neither to XY-programs. choice-free XY-programs are locally stratified and then also modularly locally stratified, so we can apply, for example, Ross' extension [16]. One disadvantage of this approach is that the XY-structure is destroyed, and therefore an evaluation method is required which is not compatible with the one presented in this paper, so the optimizations discussed above are not applicable. In order to preserve opportunities for optimization, we apply the magic set transformation in such a way that the XY-structure is not corrupted. To obtain this, it is necessary to avoid the constraints propagation backward along the stages, and hence to make the optimization local to each stage. This point is illustrated by a simple example:

Example 4. Let consider the following stratified program:

$$\left.\begin{array}{l} \mathtt{p(X,Y)} \leftarrow \mathtt{b_1(X,Y)}. \\ \mathtt{p(X,Y)} \leftarrow \mathtt{p(X,Z)}, \mathtt{b_1(Z,Y)}. \end{array}\right\} S_1$$

$$\left.\begin{array}{l} \mathtt{q(X)} \leftarrow \mathtt{b_1(d,X)}, \neg \mathtt{p(a,X)}. \\ \mathtt{q(Y)} \leftarrow \mathtt{q(e)}, \mathtt{p(e,Y)}. \end{array}\right\} S_2$$

$$?\mathtt{q(c)}. \qquad (\mathtt{b_1} \in \mathtt{EDB})$$

Here we can apply the standard transformation *locally* to each stratum. This means that each relation used in a stratum but defined in a previous one is handled as an EDB relation, not involving it in the transformation (i.e., we don't compute its corresponding magic rules). In our example, this avoids the creation of the dependency q → m_p, thus preserving the stratification. Obviously, now each rule body goal referring a predicate defined in a previous stratum plays the role of a query for it, and then we have to produce a corresponding *seed*. In our example, we obtain the supplementary seeds m_p(a) and m_p(e), in addition to the standard m_q(c). □

We can then extend the method to a XY-program P by simply applying the transformation locally to each stage instance of P. Each such instance is a stratified subprogram, and hence we can apply to it any suitable version of magic

sets (e.g., that in [3], or the above mentioned in [16]), noticing that now the XY-structure is preserved.

Example 5. Consider the following XY-program:

```
queue(nil, X) ← first(X).                          % A queue of people.
queue(s(I), Y) ← next(X, Y), queue(I, X).
candidate(I, X, N) ← queue(I, X), level(X, N).     % Each invited one can
candidate(I, Y, N) ← join(X, Y), candidate(I, X, _), % invite somebody.
                     level(Y, N).
accept(I, X) ← candidate(I, X, N), N < 6.          % Only N < 6 enter.
?accept(I, mary).                                  % Can Mary enter ?
```

It is a positive program, and then locally to each stage we can apply the simplest standard magic sets. Denoting the *local magic* predicates `p` by `lm_p`, we obtain:

```
% Modified rules
queue(nil, X) ← lm_queue(nil, X), first(X).
queue(s(I), Y) ← lm_queue(s(I), Y), next(X, Y), queue(I, X).
candidate(I, X, N) ← lm_candidate(I, X), queue(I, X), level(X, N).
candidate(I, Y, N) ← lm_candidate(I, Y), join(X, Y), candidate(I, X, _),
                     level(Y, N).
accept(I, X). ← lm_accept(I, X), candidate(I, X, N), N < 6.
% Magic rules
lm_queue(I, X) ← lm_candidate(I, X).
lm_candidate(I, X) ← lm_candidate(I, Y), join(X, Y).
lm_candidate(I, X) ← lm_accept(I, X).
% Seeds
lm_queue(I, X) ← .
lm_accept(I, mary) ← .
%
?accept(I, mary).
```

Our restriction causes the absence of any magic rule corresponding to the second original rule, which computes a inter-stage transitive closure. Such magic rule, instead, is replaced by the seed `lm_queue(I, X)` ← , which inhibits any optimization regarding the predicate `queue`. Notice that we could also eliminate every occurrence of the magic predicate `lm_queue(I, X)` from the magic version of the program, since the corresponding seed forces to true value all instances of the predicate. Moreover, the elimination of the rule `lm_queue(I, X)` ← makes the program safe. □

Next, we observe that the magic sets transformation applied to programs with `choice` does not preserve completeness with the respect of their stable models. In other words, the propagation of constraints may cut off some stable models of the program. We can see this on the following simple example, where `a` is an EDB relation with extension `a(1), a(2)`:

```
b(X) ← a(X), choice((), X).
?b(1)
```

with magic version:

```
b(X) ← m_b(X), a(X), choice((), X).
m_b(1).
?b(1)
```

The magic version always answers *yes*, while the original program non-deterministically answers *yes* or *no*.

The magic set optimizations for Datalog++ studied so far, although "minimal", represents a first step towards a more systematic study of the vast repertoire of this kind of optimizations, which have to be further investigated.

References

1. K. R. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Proc. Workshop on Found. of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufman, 1988. 59
2. N. Arni, K. Ong, S. Tsur, and C. Zaniolo. $\mathcal{LDL}$++: A Second Generation Deductive Databases Systems. Technical report, MCC Corporation, 1993. 58, 59
3. I. Balbin, G.S. Port, K. Ramamohanarao, and K. Meenakshi. Efficient Bottom-up Computation of Queries on Stratified Databases. *Journal of Logic Programming*, 11:295–344, 1991. 69
4. A. Brogi, V. S. Subrahmanian, and C. Zaniolo. The Logic of Totally and Partially Ordered Plans: a Deductive Database Approach. *Annals of Mathematics in Artificial Intelligence*, 19:59–96, 1997. 58, 58, 59, 59, 64, 65
5. J. Chomicki. Temporal deductive databases. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases: Theory, Design and Implementation*, pages 294–320. Benjamin Cummings, 1993. 58, 59
6. M Gelfond and V. Lifchitz. The Stable Model Semantics for logic programming. In *Proc. of the 5th Int. Conf. on Logic Programming*, pages 1070–1080, 1988. 60
7. F. Giannotti, G. Manco, and D. Pedreschi. A Deductive Data Model for Representing and Querying Semistructured Data. In *Proc. 5th Int. Conf. on Deductive and Object-Oriented Databases (DOOD97)*, December 1997. 58, 58, 59, 66
8. F. Giannotti, D. Pedreschi, D. Saccà, and C. Zaniolo. Non-Determinism in Deductive Databases. In *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases (DOOD91)*, volume 566 of *Lecture Notes in Computer Science*, pages 129–146, 1991. 58, 59
9. F. Giannotti, D. Pedreschi, and C. Zaniolo. Semantics and Expressive Power of Non Deterministic Constructs for Deductive Databases. Technical Report C96-04, The CNUCE Institute, 1996. Submitted. 64
10. D. Kemp, K. Ramamohanarao, and P. Stuckey. ELS programs and the efficient evaluation of non-stratified programs by transformation to ELS. In *Proc. 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD95)*, pages 91–108, 1995. 59
11. R. Krishnamurthy and S. Naqvi. Non-deterministic Choice in Datalog. In *Proc. 3rd Int. Conf. on Data and Knowledge Bases*, pages 416–424, 1988. 59
12. B. Lüdascher, U. Hamann, and G. Lausen. A logical framework for active rules. In *Proc. 7th COMAD Int. Conf. on Management of Data*. Tata McGraw-Hill, 1995. 59

13. M. Nanni. Nondeterminism and XY-Stratification in Deductive Databases (in Italian). Master's thesis, Department of Computer Science University of Pisa, 1997. 59, 65
14. S. Naqvi and S. Tsur. *A Logic Language for Data and Knowledge Bases*. Computer Science Press, 1989. 59
15. T. C. Przymusinski. Every logic program has a natural stratification and an iterated fix point model. In *Proc. 8th ACM PODS Symposium on Principles of Database Systems*, pages 11–21, 1989. 59, 61, 63
16. K. A. Ross. Modular Stratification and Magic Sets for Datalog Program with Negation. *Journal of ACM*, 41(6):1216–1266, November 1994. 59, 63, 68, 69
17. D. Saccà and C. Zaniolo. Stable Models and Non-determinism in Logic Programs with Negation. In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 205–217, 1990. 59
18. C. Zaniolo. Active Database Rules with Transaction Conscious Stable Model Semantics. In *Proc. 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD95)*, volume 1013 of *Lecture Notes in Computer Science*, pages 55–72, 1995. 58, 66
19. C. Zaniolo, N. Arni, and K. Ong. Negation and Aggregates in Recursive Rules: The $\mathcal{LDL}$++ Approach. In *Proc. 3rd Int. Conf. on Deductive and Object-Oriented Databases (DOOD93)*, volume 760 of *Lecture Notes in Computer Science*, 1993. 58, 60, 65, 66, 66
20. C. Zaniolo, S. Ceri, C. Faloutsos, R.T Snodgrass, V.S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufman, 1997. 58, 59

Appendix

Proof sketch of Lemma 1. The proof is structured as follows: (1) $\Longleftrightarrow$ (3) and (2) $\Longleftrightarrow$ (3).

(3) $\Longrightarrow$ (1) We next show that (a) $S_P(I) \subseteq I$, and (b) $I \subseteq S_P(I)$.
(a) Each rule in $H(P, I)$ comes from a rule r of P, which in turn appears in $Q^{(i)}$ for some i, and then $I^{(i)}$ is a model of r, by the hypothesis. No atom in $I \setminus I^{(i)}$ appears in r, so also I is model of r. I is then a model of $H(P, I)$, and hence $S_P(I) \subseteq I$.
(b) If $A \in I$, then $A \in I^{(i)}$ for some i, so (by the hypothesis and definition of S_P) for each I^* such that $I^* = T_{H(Q^{(i)}, I^{(i)})}(I^*)$, $A \in I^*$. Moreover, for each I' such that $I' = T_{H(P,I)}(I')$, it is readily checked that for each i $I'^{(i)} = T_{H(Q^{(i)}, I^{(i)})}(I'^{(i)})$, and then $I \subseteq S_P(I)$.

(1) $\Longrightarrow$ (3) We observe that $I = min\{I^* \mid I^* = T_{H(P,I)}(I^*)\}$, which implies: $I^{(i)} = min\{I^{*^{(i)}} \mid I^{*^{(i)}} = T_{H(Q^{(i)}, I^{(i)})}(I^{*^{(i)}})\}$.

(2) $\Longrightarrow$ (3) We proceed by induction on i. The base case is trivial. In the inductive case, we next show that (a) $S_{Q^{(i)}}(I^{(i)}) \subseteq I^{(i)}$, and (b) vice versa.
(a) Notice that from the induction hypothesis, $I^{(i)} \models Q^{(i-1)}$, and then it suffices to show that $I^{(i)} \models Q_i$ (by a simple case analysis).
(b) Exploiting the induction hypothesis, we have $I^{(i-1)} \subseteq S_{Q^{(i-1)}}(I^{(i-1)}) = S_{Q^{(i-1)}}(I^{(i)}) \subseteq S_{Q^{(i)}}(I^{(i)})$ (by definition of $H(P, I)$). We now show by induction on n that $\forall n \geq 0\ T^n_{H(Q_i^{red}, I_i)} \subseteq T^\omega_{H(Q^{(i)}, I^{(i)})}$. The base case

$n = 0$ is trivial. In the induction case ($n > 0$), if $A \in T^n_{H(Q_i^{red}, I_i)}$, then there exists a rule $A \leftarrow b_1, \ldots, b_h$ in $H(Q_i^{red}, I_i)$ such that $\{b_1, \ldots, b_h\} \subseteq T^{n-1}_{H(Q_i^{red}, I_i)}$. Now, by definition of H and Q_i^{red}, there exists a rule:
$A \leftarrow b_1, \ldots, b_h, \neg c_1, \ldots, \neg c_j, d_1, \ldots, d_k, \neg e_1, \ldots, \neg e_l$
in Q_i such that $\{c_1, \ldots, c_j\} \cap I_i = \emptyset$ and $I^{(i-1)} \models d_1 \wedge \ldots \wedge d_k \wedge \neg e_1 \wedge \ldots \wedge \neg e_l$. Observe now that by definition of H, $A \leftarrow b_1, \ldots, b_h, d_1, \ldots, d_k \in H(Q^{(i)}, I^{(i)})$. Furthermore, by the induction hypothesis and $I^{(i-1)} \subseteq S_{Q^{(i)}}(I^{(i)})$, we have the following: $\{b_1, \ldots, b_h, d_1, \ldots, d_k\} \subseteq T^{\omega}_{H(Q^{(i)}, I^{(i)})}$. Hence, by definition of T^{ω}, $A \in T^{\omega}_{H(Q^{(i)}, I^{(i)})}$, that is $A \in S_{Q^{(i)}}(I^{(i)})$. This completes the innermost induction, and we obtain that $I_i = S_{Q_i^{red}}(I_i) \subseteq S_{Q^{(i)}}(I^{(i)})$.

(3) $\Longrightarrow$ **(2)** We proceed is a way similar to the preceding case. To see that $\forall i\ I_i \subseteq S_{Q_i^{red}}(I_i)$, it suffices to verify that for each rule instance r with head A, the following property holds: $\forall n\ A \in T^n_{H(Q^{(i)}, I^{(i)})} \Rightarrow A \in T^n_{H(Q_i^{red}, I_i)}$. For the converse, we simply observe that I_i is a model of Q_i^{red}. □

Proof sketch of Lemma 2. We show by induction that $\forall n. T^n_{H(SG(P),I)}(\emptyset) = T^n_{H(P,I)}(\emptyset)$, which implies the thesis. The base case is trivial. For the inductive case, observe that since P is XY-stratified, if $A \in T^{n+1}_{H(P,I)}(\emptyset)$ then for each rule $A \leftarrow B_1, \ldots, B_n \in H(P, I)$ such that $\{B_1, \ldots, B_n\} \in T^n_{H(P,I)}(\emptyset) = T^n_{H(SG(P),I)}(\emptyset)$, we have $A \leftarrow B_1, \ldots, B_n \in H(SG(P), I)$. Vice versa, if $A \in T^{n+1}_{H(SG(P),I)}(\emptyset)$ then for each rule $A \leftarrow B_1, \ldots, B_n \in H(SG(P), I)$ such that $\{B_1, \ldots, B_n\} \in T^n_{H(SG(P),I)}(\emptyset) = T^n_{H(P,I)}(\emptyset)$, we have $A \leftarrow B_1, \ldots, B_n \in H(P, I)$. □

Proof sketch of Lemma 3. It is easy to see that $SO(SG(P)) = SO(P)$. Hence, the least Herbrand models of $SO(H(SG(P), I))$ and $H(SO(P), SO(I))$ coincide. □

Revision Programming = Logic Programming + Integrity Constraints

Victor Marek, Inna Pivkina, and Mirosław Truszczyński

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046
{marek,inna,mirek}@cs.engr.uky.edu

Abstract. We study *revision programming*, a logic-based mechanism for enforcing constraints on databases. The central concept of this approach is that of a *justified revision based on a revision program*. We show that for any program P and for any pair of initial databases $\mathcal{I}$ and $\mathcal{I}'$ we can transform (shift) the program P to a program P' so that the size of the resulting program does not increase and so that P-justified revisions of $\mathcal{I}$ are shifted to P'-justified revisions of $\mathcal{I}'$. Using this result we show that revision programming is closely related to a subsystem of general logic programming of Lifschitz and Woo. This, in turn, allows us to reduce revision programming to logic programming extended by the concept of a constraint with a suitably modified stable model semantics. Finally, we use the connection between revision programming and general logic programming to introduce a disjunctive version of our formalism.

1 Introduction

Revision programming was introduced in [MT98] as a formalism to describe and study the process of database updates. In this formalism, the user specifies updates by means of *revision rules*, that is, expressions of the following two types:

$$\mathbf{in}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n) \tag{1}$$

or

$$\mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_m), \mathbf{out}(b_1), \ldots, \mathbf{out}(b_n), \tag{2}$$

where a, a_i and b_i are data items from some finite universe, say U. Rules of the first type are called *in-rules* and rules of the second type are called *out-rules*.

Revision rules have a declarative interpretation as constraints on databases. For instance, an in-rule (1) imposes on a database the following condition: a is *in* the database, or at least one a_i, $1 \leq i \leq m$, is *not* in the database, or at least one b_j, $1 \leq j \leq n$, is *in* the database.

Revision rules also have a computational interpretation that expresses a preferred way to enforce a constraint. Namely, assume that all data items a_i, $1 \leq i \leq m$, belong to the current database, say $\mathcal{I}$, and none of the data items b_j, $1 \leq j \leq n$, belongs to $\mathcal{I}$. Then, to enforce the constraint (1), the item a must be added to the database (removed from it, in the case of the constraint (2)), rather than some item a_i removed or some item b_j added.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 73–89, 1999.

In [MT98], a precise semantics for *revision programs* (collections of revision rules) was defined. Given a revision program P and a database $\mathcal{I}$, this semantics specifies a family of databases, each of which might be chosen as an update of $\mathcal{I}$ by means of the program P. These revised databases are called P-justified revisions of $\mathcal{I}$. In [MT98] (and in the earlier papers [MT94] and [MT95]), basic properties of justified revisions were established. Subsequently, revision programming was studied in the context of situation calculus [Bar97] and reasoning about actions [McCT95,Tur97].

Revision programming has also been investigated from the perspective of its close relationship with logic programming. In [MT98], it was argued that revision programming extends logic programming with stable semantics by showing that revision programs consisting of in-rules only can be identified with logic programs. A converse embedding — an encoding of revision programs as logic programs — was constructed in [PT97]. The techniques from this paper are now being exploited in the study of the problem of updating logic programs [AP97] and resulted in a new paradigm of dynamic logic programming [ALP+98]. Well-founded semantics for a formalism closely related to revision programming was discussed in [BM97].

The key property of revision programming is the duality of **in** and **out** literals. The duality theorem (Theorem 3.8 from [MT98]) demonstrated that every revision program P has a counterpart, a dual revision program P^D such that P-justified revisions of a database $\mathcal{I}$ are precisely the complements of the P^D-justified revisions of the complement of $\mathcal{I}$.

The key result of this paper, the shifting theorem (Theorem 4), is a generalization of the duality theorem from [MT98]. It states that P-justified revisions of a database $\mathcal{I}$ can be computed by revising an arbitrarily chosen database $\mathcal{I}'$ by means of a certain "shifted" revision program P'. This program P' is obtained from P by uniformly replacing some literals in P by their duals. The choice of literals to replace depends on $\mathcal{I}$ and $\mathcal{I}'$. In addition, $\mathcal{I}$ and $\mathcal{I}'$ determine also a method to reconstruct P-justified revisions of $\mathcal{I}$ from P'-justified revisions of $\mathcal{I}'$.

As a special case, the shifting theorem tells us that justified revisions of arbitrary databases are determined by revisions, via shifted programs, of the empty database. This result implies two quite surprising facts. First, it means that although a revision problem is defined as pair $(P, \mathcal{I})$ (revision program and a database), full information about any revision problem can be recovered from revision problems of very special type that deal with the empty database. Moreover, the reduction does not involve any growth in the size of the revision program. Second, the shifting theorem implies the existence of a natural equivalence relation between the revision problems: two revision problems are equivalent if one can be shifted onto another.

The first of these two observations (the possibility to project revision problems onto problems with the empty database) allows us to establish a direct correspondence between revision programming and a version of logic programming proposed by Lifschitz and Woo [LW92]. We will refer to this latter system as *general disjunctive logic programming* or, simply, *general logic programming*.

In general logic programming both disjunction and negation as failure operators are allowed in the heads of rules. In this paper we study the relationship between revision programming and general logic programming. First, in Section 3, we show that revision programming is equivalent to logic programming with stable model semantics extended by a concept of a constraint. Second, in Section 4, we extend revision programming to the disjunctive case.

2 Preliminaries

In this section we will review main concepts and results concerning revision programming that are relevant to the present paper. The reader is referred to [MT98] for more details.

Elements of some finite universe U are called *atoms*. Subsets of U are called *databases*. Expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$, where a is an atom, are called *literals*. Literals will be denoted by greek letters α, etc. For a literal $\mathbf{in}(a)$, its *dual* is the literal $\mathbf{out}(a)$. Similarly, the *dual* of $\mathbf{out}(a)$ is $\mathbf{in}(a)$. The dual of a literal α is denoted by α^D.

For a set of atoms $R \subseteq U$, we define

$$\mathcal{R}^c = \{\mathbf{in}(a) : a \in \mathcal{R}\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}\}.$$

A set of literals is *coherent* if it does not contain a pair of dual literals. Given a database $\mathcal{I}$ and a coherent set of literals L, we define

$$\mathcal{I} \oplus L = (\mathcal{I} \cup \{a\colon \mathbf{in}(a) \in L\}) \setminus \{a\colon \mathbf{out}(a) \in L\}.$$

Let P be a revision program. The *necessary change* of P, $NC(P)$, is the least model of P, when P is treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. The necessary change describes all insertions and deletions that are enforced by the program, independently of the initial database.

In the transition from a database $\mathcal{I}$ to a database $\mathcal{R}$, the status of some elements does not change. A basic principle of revision programming is the rule of *inertia* according to which, when specifying change by means of rules in a revision program, no explicit justification for *not* changing the status is required. Explicit justifications are needed only when an atom must be inserted or deleted. The collection of all literals describing the elements that do not change the status in the transition from a database $\mathcal{I}$ to a database $\mathcal{R}$ is called the *inertia set* for $\mathcal{I}$ and $\mathcal{R}$, and is defined as follows:

$$I(\mathcal{I}, \mathcal{R}) = \{\mathbf{in}(a)\colon a \in \mathcal{I} \cap \mathcal{R}\} \cup \{\mathbf{out}(a)\colon a \notin \mathcal{I} \cup \mathcal{R}\}.$$

By the *reduct* of P with respect to a pair of databases $(\mathcal{I}, \mathcal{R})$, denoted by $P_{\mathcal{I},\mathcal{R}}$, we mean the revision program obtained from P by eliminating from the body of each rule in P all literals in $I(\mathcal{I}, \mathcal{R})$.

The necessary change of the program $P_{\mathcal{I},\mathcal{R}}$ provides a justification for some insertions and deletions. These are exactly the changes that are (*a posteriori*)

justified by P in the context of the initial database $\mathcal{I}$ and a (putative) revised database $\mathcal{R}$. The database $\mathcal{R}$ is a *P-justified revision* of $\mathcal{I}$ if the necessary change of $P_{\mathcal{I},\mathcal{R}}$ is coherent and if $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$.

The following example illustrates the notion of justified revision.

Example 1. Assume that we need to form a committee. There are four people which can be on the committee: Ann, Bob, Tom and David. There are four conditions on the committee members which need to be satisfied.

First, Ann and Bob are experienced employees, and we want to see at least one of them on the committee. That is, if Ann is not on the committee, Bob must be there, and if Bob is not on the committee, Ann must be there. Second, Tom is an expert from another country and does not speak English well enough yet. So, if Tom is on the committee, David should be on the committee too, because David can serve as an interpreter. If David is not on the committee, Tom should not be there, too. Third, David asked not to be on the same committee with Ann. Fourth, Bob asked not to be on the same committee with David.

The initial proposal is to have Ann and Tom in the committee.

We want to form a committee which satisfies the four conditions and differs minimally from the initial proposal. This is a problem of computing justified revisions of initial database $\mathcal{I} = \{Ann, Tom\}$ with respect to revision program P:

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann)\\
\mathbf{in}(Ann) &\leftarrow \mathbf{out}(Bob)\\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Tom)\\
\mathbf{out}(Tom) &\leftarrow \mathbf{out}(David)\\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

Let us show that $\mathcal{R} = \{Ann\}$ is a P-justified revision of $\mathcal{I}$. Clearly, $U = \{Ann, Bob, Tom, David\}$. Thus,

$$I(\mathcal{I}, \mathcal{R}) = \{\mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(David)\}.$$

Therefore, $P_{\mathcal{I},\mathcal{R}}$ is the following.

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{out}(Ann)\\
\mathbf{in}(Ann) &\leftarrow\\
\mathbf{in}(David) &\leftarrow \mathbf{in}(Tom)\\
\mathbf{out}(Tom) &\leftarrow\\
\mathbf{out}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

Hence, $NC(P_{\mathcal{I},\mathcal{R}}) = \{\mathbf{in}(Ann), \mathbf{out}(Tom)\}$. It is coherent and $\mathcal{R} = \mathcal{I} \oplus NC(P_{\mathcal{I},\mathcal{R}})$. Consequently, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ (in fact, unique). □

In the paper we will use the following characterizations of justified revisions given in [MT98].

Theorem 1. **([MT98])** *The following conditions are equivalent:*
1. A database $\mathcal{R}$ is a P-justified revision of a database $\mathcal{I}$,
2. $NC(P \cup \{\alpha \leftarrow : \alpha \in I(\mathcal{I}, \mathcal{R})\}) = \mathcal{R}^c$,
3. $NC(P_{\mathcal{I},\mathcal{R}}) \cup I(\mathcal{I}, \mathcal{R}) = \mathcal{R}^c$.

Two results from [MT98] are especially pertinent to the results of this paper. Given a revision program P, let us define the *dual* of P (P^D in symbols) to be the revision program obtained from P by simultaneously replacing all occurrences of all literals by their duals. The first of the two results we will quote here, the duality theorem, states that revision programs P and P^D are, in a sense, equivalent. Our main result of this paper (Theorem 4) is a generalization of the duality theorem.

Theorem 2. (Duality Theorem [MT98]) *Let P be a revision program and let $\mathcal{I}$ be a database. Then, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $U \setminus \mathcal{R}$ is a P^D-justified revision of $U \setminus \mathcal{I}$.*

The second result demonstrates that there is a straightforward relationship between revision programs consisting of in-rules only and logic programs. Given a logic program clause c

$$p \leftarrow q_1, \ldots, q_m, not\ s_1, \ldots, not\ s_n$$

we define the revision rule $rp(c)$ as

$$\mathbf{in}(p) \leftarrow \mathbf{in}(q_1), \ldots, \mathbf{in}(q_m), \mathbf{out}(s_1), \ldots, \mathbf{out}(s_n).$$

For a logic program P, we define the corresponding revision program $rp(P)$ by: $rp(P) = \{rp(c) : c \in P\}$.

Theorem 3. **([MT98])** *A set of atoms M is a stable model of a logic program P if and only if M is an $rp(P)$-justified revision of $\emptyset$.*

It is also possible to represent revision programming in logic programming. This observation is implied by complexity considerations (both the existence of a justified revision and the existence of a stable model problems are NP-complete). An explicit representation was discovered in [PT97]. In addition to representing revision rules as logic program clauses, it encodes the initial database by means of new variables and encodes the inertia rule as logic program clauses. As a consequence to our main result (Theorem 4), we obtain an alternative (and in some respects, simpler) connection between revision programming and logic programming. Namely, we establish a direct correspondence between revision programs and general logic programs of [LW92].

3 Shifting initial databases and programs

In this section we will introduce a transformation of revision programs and databases that preserves justified revisions. Our results can be viewed as a generalization of the results from [MT98] on the duality between **in** and **out** in revision programming.

Let W be a subset of U. We define a *W-transformation* on the set of all literals as follows (below, $\alpha = \mathbf{in}(a)$ or $\alpha = \mathbf{out}(a)$):

$$T_W(\alpha) = \begin{cases} \alpha^D, & \text{when } a \in W \\ \alpha, & \text{when } a \notin W. \end{cases}$$

Thus, T_W replaces some literals by their duals and leaves other literals unchanged. Specifically, if a belongs to W then literals $\mathbf{in}(a)$ and $\mathbf{out}(a)$ are replaced by their duals.

The definition of T_W naturally extends to sets of literals and sets of atoms. Namely, for a set L of literals, we define $T_W(L) = \{T_W(\alpha)\colon \alpha \in L\}$. Similarly, for a set A of atoms, we define

$$T_W(A) = \{a : \mathbf{in}(a) \in T_W(A^c)\}.$$

The operator T_W has several useful properties. In particular, for a suitable set W, T_W allows us to transform any database $\mathcal{I}_1$ into another database $\mathcal{I}_2$. Specifically, we have:

$$T_{\mathcal{I}_1 \div \mathcal{I}_2}(\mathcal{I}_1) = \mathcal{I}_2,$$

where $\div$ denotes the symmetric difference operator. Thus, it also follows that

$$T_{\mathcal{I}}(\mathcal{I}) = \emptyset \quad \text{and} \quad T_U(\mathcal{I}) = U \setminus \mathcal{I}.$$

Some other properties of the operator T_W are gathered in the following lemma.

Lemma 1. *Let S_1 and S_2 be sets of literals. Then:*
1. $T_W(S_1 \cup S_2) = T_W(S_1) \cup T_W(S_2)$;
2. $T_W(S_1 \cap S_2) = T_W(S_1) \cap T_W(S_2)$;
3. $T_W(S_1 \setminus S_2) = T_W(S_1) \setminus T_W(S_2)$;
4. $T_W(S_1) = T_W(S_2)$ if and only if $S_1 = S_2$;
5. $T_W(T_W(S_1)) = S_1$.

In fact, Lemma 1 holds when S_1 and S_2 are sets of atoms as well.

The operator T_W can now be extended to revision rules and programs. For a revision rule $r = \alpha \leftarrow \alpha_1, \ldots, \alpha_m$, we define

$$T_W(r) = T_W(\alpha) \leftarrow T_W(\alpha_1), \ldots, T_W(\alpha_m).$$

Finally, for a revision program P, we define $T_W(P) = \{T_W(r)\colon r \in P\}$.

The main result of our paper, the shifting theorem, states that revision programs P and $T_W(P)$ are equivalent in the sense that they define essentially the same notion of change.

Theorem 4 (Shifting theorem). *Let P be a revision program. For every two databases $\mathcal{I}_1$ and $\mathcal{I}_2$, a database $\mathcal{R}_1$ is a P-justified revision of $\mathcal{I}_1$ if and only if $T_{\mathcal{I}_1 \div \mathcal{I}_2}(\mathcal{R}_1)$ is a $T_{\mathcal{I}_1 \div \mathcal{I}_2}(P)$-justified revision of $\mathcal{I}_2$.*

Proof. Let $W = \mathcal{I}_1 \div \mathcal{I}_2$. When calculating the necessary change, we treat literals as propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(b)$. Observe that W-transformation can be viewed as renaming these atoms. If we rename all atoms in the Horn program, find the least model of the obtained program, and then rename the atoms back, we will get the least model of the original program.

In other words,

$$T_W(NC(P_{\mathcal{I}_1,\mathcal{R}_1})) = NC(T_W(P_{\mathcal{I}_1,\mathcal{R}_1})).$$

Let $\mathcal{R}_2 = T_W(\mathcal{R}_1)$. Observe that by the definition of T_W, $I(\mathcal{I}_2, \mathcal{R}_2) = T_W(I(\mathcal{I}_1, \mathcal{R}_1))$. Hence, $T_W(P_{\mathcal{I}_1,\mathcal{R}_1}) = (T_W(P))_{\mathcal{I}_2,\mathcal{R}_2}$.

Theorem 1 and Lemma 1 imply the following sequence of equivalences.

- $\mathcal{R}_1$ is a P-justified revision of $\mathcal{I}_1$,
- $NC(P_{\mathcal{I}_1,\mathcal{R}_1}) \cup I(\mathcal{I}_1, \mathcal{R}_1) = \mathcal{R}_1^c$,
- $T_W(\ NC(P_{\mathcal{I}_1,\mathcal{R}_1}) \cup I(\mathcal{I}_1, \mathcal{R}_1)\ ) = T_W(\mathcal{R}_1^c)$,
- $T_W(NC(P_{\mathcal{I}_1,\mathcal{R}_1}))\ \cup\ T_W(I(\mathcal{I}_1, \mathcal{R}_1)) = T_W(\mathcal{R}_1^c)$,
- $NC(T_W(P_{\mathcal{I}_1,\mathcal{R}_1}))\ \cup\ I(\mathcal{I}_2, \mathcal{R}_2) = T_W(\{\mathbf{in}(a) : a \in \mathcal{R}_1\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}_1\})$,
- $NC((T_W(P))_{\mathcal{I}_2,\mathcal{R}_2}) \cup I(\mathcal{I}_2, \mathcal{R}_2) = \mathcal{R}_2^c$,
- $\mathcal{R}_2 = T_W(\mathcal{R}_1)$ is a $T_W(P)$-justified revision of $\mathcal{I}_2$. □

Theorem 2 (the duality theorem) is a special case of Theorem 4 when $\mathcal{I}_2 = U \setminus \mathcal{I}_1$.

At first glance, a revision problem seems to have two independent parameters: a revision program P that specifies constraints to satisfy, and an initial database $\mathcal{I}$ that needs to be revised by P. The shifting theorem shows that there is a natural equivalence relation between pairs $(P, \mathcal{I})$ specifying the revision problem. Namely, a revision problem $(P, \mathcal{I})$ is *equivalent* to a revision problem $(P', \mathcal{I}')$ if $P' = T_{\mathcal{I} \div \mathcal{I}'}(P)$. This is clearly an equivalence relation. Moreover, by the shifting theorem, it follows that if $(P, \mathcal{I})$ and $(P', \mathcal{I}')$ are equivalent then P-justified revisions of $\mathcal{I}$ are in one-to-one correspondence with P'-revisions of $\mathcal{I}'$. In particular, every revision problem $(P, \mathcal{I})$ can be "projected" onto an isomorphic revision problem $(T_{\mathcal{I}}(P), \emptyset)$. Thus, the domain of all revision problems can be fully described by the revision problems that involve the empty database. There is an important point to make here. When shifting a revision program, its size does not change (in other words, all revision programs associated with equivalent revision problems have the same size).

Example 2. Let us take the same problem about forming a committee which we considered in Example 1. Recall that $\mathcal{I} = \{Ann, Tom\}$. Let us apply transformation $T_{\mathcal{I}}$ (shift to the empty initial database). It is easy to see that $T_{\mathcal{I}}(P)$ consists

of the rules:

$$\begin{aligned}
\mathbf{in}(Bob) &\leftarrow \mathbf{in}(Ann)\\
\mathbf{out}(Ann) &\leftarrow \mathbf{out}(Bob)\\
\mathbf{in}(David) &\leftarrow \mathbf{out}(Tom)\\
\mathbf{in}(Tom) &\leftarrow \mathbf{out}(David)\\
\mathbf{in}(Ann) &\leftarrow \mathbf{in}(David)\\
\mathbf{out}(David) &\leftarrow \mathbf{in}(Bob)
\end{aligned}$$

This revision program has only one justified revision of $\emptyset$, $\{Tom\}$. Observe moreover that $\{Tom\} = T_{\mathcal{J}}(\{Ann\})$. This agrees with the assertion of Theorem 4. □

There is a striking similarity between the syntax of revision programs and nondisjunctive (unitary) general logic programs of Lifschitz and Woo [LW92]. The shifting theorem, which allows us to effectively eliminate an initial database from the revision problem, suggests that both formalisms may be intimately connected. In the next section we establish this relationship. This, in turn, allows us to extend the formalism of revision programming by allowing disjunctions in the heads.

4 General disjunctive logic programs and revision programming

Lifschitz and Woo [LW92] introduced a formalism called general logic programming (see also [Lif96] and [SI95]). General logic programming deals with clauses whose heads are disjunctions of atoms (we will restrict here to the case of atoms only, even though in the original paper more general syntax is studied) and atoms within the scope of the negation-as-failure operator. Specifically, Lifschitz and Woo consider *general program rules* of the form:

$$A_1|\ldots|A_k|not\ A_{k+1}|\ldots|not\ A_l \ \leftarrow\ A_{l+1},\ldots,A_m, not\ A_{m+1},\ldots,not\ A_n, \quad (3)$$

which can be also represented as

$$HPos \cup not(HNeg) \ \leftarrow\ BPos \cup not(BNeg),$$

where $A_1,\ldots,A_n$ are atoms, $HPos = \{A_1,\ldots,A_k\}$, $HNeg = \{A_{k+1},\ldots,A_l\}$, $BPos = \{A_{l+1},\ldots,A_m\}$, $BNeg = \{A_{m+1},\ldots,A_n\}$.

A *general logic program* is defined as a collection of general program rules.

Given a set of atoms M and a clause c of the form (3), M *satisfies* c if from the fact that every A_i, $l+1 \leq i \leq m$, belongs to M and no A_i, $m+1 \leq i \leq n$, belongs to M, it follows that one of A_i, $1 \leq i \leq k$, belongs to M or one of A_i, $k+1 \leq i \leq l$, does not belong to M.

Lifschitz and Woo introduced a semantics of general logic programs that is stronger than the semantics described above. It is the semantics of *answer sets*.

Answer sets are constructed in stages. First, one defines answer sets for programs that do not involve negation as failure, that is, consist of clauses

$$A_1|\ldots|A_k \leftarrow A_{k+1},\ldots,A_m \tag{4}$$

Given a program P consisting of clauses of type (4), a set of atoms M is an answer set for P if M is a minimal set of atoms satisfying all clauses in P.

Next, given a general logic program P (now possibly with negation as failure operator) and a set of atoms M, one defines the *reduct* of P with respect to M, denoted P^M, as the general logic program without negation as failure obtained from P by

- deleting each disjunctive rule such that $HNeg \not\subseteq M$ or $BNeg \cap M \neq \emptyset$, and
- replacing each remaining disjunctive rule by $HPos \leftarrow BPos$.

A set of atoms M is an *answer set* for P if M is an answer set for P^M.

4.1 Answer sets for general programs and justified revisions

We will now show that revision programming is closely connected with a special class of general logic programs, namely those for which all rules have a single atom in the head. We will call such rules and programs *unitary*.

The encoding of revision rules as general logic program clauses is straightforward. Given a revision program in-rule r:

$$\mathbf{in}(p) \leftarrow \mathbf{in}(q_1),\ldots,\mathbf{in}(q_m),\mathbf{out}(s_1),\ldots,\mathbf{out}(s_n)$$

we define the disjunctive rule $dj(r)$ as:

$$p \leftarrow q_1,\ldots,q_m,not\ s_1,\ldots,not\ s_n.$$

Similarly, given a revision program out-rule r:

$$\mathbf{out}(p) \leftarrow \mathbf{in}(q_1),\ldots,\mathbf{in}(q_m),\mathbf{out}(s_1),\ldots,\mathbf{out}(s_n)$$

we define the disjunctive rule $dj(r)$ as:

$$not\ p \leftarrow q_1,\ldots,q_m,not\ s_1,\ldots,not\ s_n.$$

Finally, for a revision program P, define $dj(P) = \{dj(r)\colon r \in P\}$.

The mapping $dj(\cdot)$ is a 1-1 correspondence between revision rules and unitary general logic program rules, and revision programs and unitary general logic programs.

The following result states that revision problems where the initial database is empty can be dealt with by means of general logic programs. This result can be viewed as a generalization of Theorem 3.

Theorem 5. *Let P be a revision program. Then, $\mathcal{R}$ is a P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is an answer set for $dj(P)$.*

Proof. Let $\mathcal{R}$ be a database. Let P' be a revision program obtained from P by deleting each out-rule that has $\mathbf{out}(a)$ in the head for some $a \notin \mathcal{R}$, and deleting each rule which has $\mathbf{out}(a)$ in the body for some $a \in \mathcal{R}$. Then, $dj(P')$ is the disjunctive program obtained from $dj(P)$ by deleting each disjunctive rule such that $HNeg \not\subseteq \mathcal{R}$ or $BNeg \cap \mathcal{R} \neq \emptyset$ (recall that this is the first step in constructing $dj(P)^{\mathcal{R}}$).

Observe that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is P'-justified revision of $\emptyset$. Indeed, inertia $I(\emptyset, \mathcal{R}) = \{\mathbf{out}(a) : a \notin \mathcal{R}\}$. From Theorem 1 we have that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) \;=\; NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}) \;=\; \mathcal{R}^c.$$

From the definition of P' and the fact that $NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\})$ is coherent (as it equals $\mathcal{R}^c$), it follows that

$$NC(P \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}) \;=\; NC(P' \cup \{\mathbf{out}(a) \leftarrow : a \notin \mathcal{R}\}).$$

Therefore, using Theorem 1 again, we get that $\mathcal{R}$ is P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is P'-justified revision of $\emptyset$.

Observe that if literal $\mathbf{out}(a)$, for some a, occurs in the body of a rule in P' then $\mathbf{out}(a) \in I(\emptyset, \mathcal{R})$. Also, inertia $I(\emptyset, \mathcal{R})$ consists only of literals of the form $\mathbf{out}(a)$. Therefore, $P'_{\emptyset,\mathcal{R}}$ is obtained from P' by eliminating each literal of the form $\mathbf{out}(a)$ from the bodies of the rules.

Let $P'_{\emptyset,\mathcal{R}} = P'' \cup P'''$, where P'' consists of all in-rules of $P'_{\emptyset,\mathcal{R}}$, $P''' = P'_{\emptyset,\mathcal{R}} \setminus P''$ consists of all out-rules of $P'_{\emptyset,\mathcal{R}}$. Note, that all rules from P'' and P''' have only literals of the form $\mathbf{in}(a)$ in their bodies. Observe that if $r \in P'''$, then its head, $head(r) = \mathbf{out}(a)$ for some $a \in \mathcal{R}$. By the definition, $dj(P)^{\mathcal{R}}$ is obtained from $dj(P')$ by replacing each disjunctive rule by $HPos \leftarrow BPos$. Therefore,

$$dj(P)^{\mathcal{R}} \;=\; dj(P'') \cup \{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

After this observations we are ready to prove the statement of the theorem.
($\Rightarrow$) Let $\mathcal{R}$ be a P-justified revision of $\emptyset$. It follows that $\mathcal{R}$ is a P'-justified revision of $\emptyset$. Thus, $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. Assume that there exists a literal $\mathbf{out}(a) \in NC(P'_{\emptyset,\mathcal{R}})$. Since $NC(P'_{\emptyset,\mathcal{R}})$ is a subset of heads of rules from $P'_{\emptyset,\mathcal{R}}$, it must be the case that $a \in \mathcal{R}$. This contradicts the fact that the necessary change is coherent and $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. Therefore, $NC(P'_{\emptyset,\mathcal{R}})$ consists only of literals of the form $\mathbf{in}(a)$. It implies that $NC(P'_{\emptyset,\mathcal{R}}) = \{\mathbf{in}(a) : a \in \mathcal{R}\}$ is the least model of P'', and for every rule $r \in P'''$ there exist b such that $\mathbf{in}(b) \in body(r)$ and $b \notin \mathcal{R}$. Hence, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in $dj(P)^{\mathcal{R}}$. Thus, $\mathcal{R}$ is an answer set for $dj(P)$.
($\Leftarrow$) Let $\mathcal{R}$ be an answer set for $dj(P)$. That is, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in

$$dj(P'') \cup \{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

Then, any subset of $\mathcal{R}$ satisfies all clauses in

$$\{ \leftarrow a_1, \ldots, a_k : \mathbf{out}(a) \leftarrow \mathbf{in}(a_1), \ldots, \mathbf{in}(a_k) \in P'''\}.$$

Therefore, $\mathcal{R}$ is the minimal set of atoms which satisfies all clauses in $dj(P'')$. Hence, $\{\mathbf{in}(a) : a \in \mathcal{R}\}$ is the least model of P'' and satisfies P'''. Consequently, $\{\mathbf{in}(a) : a \in \mathcal{R}\} = NC(P'_{\emptyset,\mathcal{R}})$, and $\mathcal{R} = \emptyset \oplus NC(P'_{\emptyset,\mathcal{R}})$. By the definition, $\mathcal{R}$ is P'-justified revision of $\emptyset$. Therefore, $\mathcal{R}$ is P-justified revision of $\emptyset$. □

It might appear that the scope of Theorem 5 is restricted to the special case of revision programs that update the empty database. However, the shifting theorem allows us to extend this result to the general case. Thus, revision programming turns out to be equivalent to the unitary fragment of general logic programming. Indeed, we have the following corollary.

Corollary 1. *Let P be a revision program and $\mathcal{I}$ a database. Then, a database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $T_{\mathcal{I}}(\mathcal{R})$ is an answer set for the program $dj(T_{\mathcal{I}}(P))$.*

Consider a revision program P and a database $\mathcal{I}$. A rule $r \in P$ is called a *constraint* (with respect to $\mathcal{I}$) if its head is of the form $\mathbf{in}(a)$, for some $a \in \mathcal{I}$, or $\mathbf{out}(a)$, for some $a \notin \mathcal{I}$.

Theorem 6. *Let P be a revision program and let $\mathcal{I}$ be a database. Let P' consist of all rules in P that are constraints with respect to $\mathcal{I}$. Let $P'' = P \backslash P'$. A database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P''-justified revision of $\mathcal{I}$ that satisfies all rules from P'.*

Proof. By the shifting theorem it is enough to prove the statement for the case $\mathcal{I} = \emptyset$. Let $\mathcal{I} = \emptyset$. Then, P' consists of all out-rules of P and P'' consists of all in-rules of P.

($\Rightarrow$) If $\mathcal{R}$ is a P-justified revision of $\emptyset$, then $\mathcal{R}$ is a model of P. Hence, it is a model of $P' \subseteq P$.

Theorem 1 implies that

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) = \{\mathbf{in}(a) : a \in \mathcal{R}\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}\}.$$

Let $M = NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\})$. That is, M is the least model of

$$P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\} = P' \cup P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}.$$

By the definition of inertia, $I(\emptyset, \mathcal{R}) = \{\mathbf{out}(a) : a \notin \mathcal{R}\}$.

We will now show that M is the least model of $P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$.

Let us divide P' into two disjoint parts: $P' = P'_1 \cup P'_2$, where heads of the rules from P'_1 are in $\{\mathbf{out}(a) : a \in \mathcal{R}\}$ and heads of the rules from P'_2 are in $\{\mathbf{out}(a) : a \notin \mathcal{R}\}$. For each rule $r \in P'_2$, $head(r) \in I(\emptyset, \mathcal{R})$. Hence, there exists rule $head(r) \leftarrow$ in the set $\{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$. Therefore, M is also the least model of the program

$$P'' \cup P'_1 \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}.$$

If we remove from the program some rules whose premises are false in M, M remains the least model of the reduced program. Let us show that premises

of all rules from P_1' are false in M. Indeed, let r be a rule from P_1'. Then, $head(r) \in \{\mathbf{out}(a) : a \in \mathcal{R}\}$. Assume that premises of r are true in M. Then, $head(r)$ must be true in M, since M is the model of $P'' \cup P_1' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$. Hence, $M \cap \{\mathbf{out}(a) : a \in \mathcal{R}\} \neq \emptyset$, which contradicts the fact that $M = \{\mathbf{in}(a) : a \in \mathcal{R}\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}\}$. Therefore, M is the least model of the program $P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}$. In other words,

$$NC(P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) = \{\mathbf{in}(a) : a \in \mathcal{R}\} \cup \{\mathbf{out}(a) : a \notin \mathcal{R}\}.$$

From Theorem 1 we conclude that $\mathcal{R}$ is a P''-justified revision of $\emptyset$.
($\Leftarrow$) Assume $\mathcal{R}$ is a P''-justified revision of $\emptyset$, and $\mathcal{R}$ satisfies all rules from P'. Theorem 1 implies that

$$NC(P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) = \mathcal{R}^c.$$

Let $M = \mathcal{R}^c$. Then, M is the least model of

$$P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}.$$

Clearly, M is also the least model of a modified program obtained by adding some rules that are satisfied by M. All rules in P' are satisfied by M by our assumption. Therefore, M is the least model of

$$P' \cup P'' \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\} = P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}.$$

Hence,

$$NC(P \cup \{\alpha \leftarrow : \alpha \in I(\emptyset, \mathcal{R})\}) = M = \mathcal{R}^c.$$

By Theorem 1, $\mathcal{R}$ is a P-justified revision of $\emptyset$. □

The reason for the term "constraint" is now clear. In computing P-justified revisions only "non-constraints" are used. Then, the constraint part of P is used to weed out some of the computed revisions.

Clearly, if $\mathcal{I} = \emptyset$, the constraints are exactly the out-rules of a revision program. We can extend the notion of a constraint to the case of unitary general logic programs. Namely, a unitary program rule is a *constraint* if its head is of the form *not a* (note that this notion of constraint is different from the one used in [Lif96]). Theorem 6 has the following corollary.

Corollary 2. *Let P be a unitary general logic program and let P' consists of all constraints in P. A set M is an answer set for P if and only if M is a stable model for $P \setminus P'$ that satisfies P'.*

It follows from the shifting theorem and from Theorem 5 that in order to describe updates by means of revision programming, it is enough to consider logic programs with stable model semantics and rules with *not a* in the heads that work as constraints.

Corollary 3. *Let P be a revision program and let $\mathcal{I}$ be a database. Then, a database $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $T_{\mathcal{I}}(\mathcal{R})$ is a stable model of the logic program $dj(T_{\mathcal{I}}(P) \setminus P')$ that satisfies P', where P' consists of all constraints in $T_{\mathcal{I}}(P)$.*

4.2 Disjunctive revision programs

The results of Section 4.1 imply an approach to extend revision programming to include clauses with disjunctions in the heads. Any such proposal must satisfy several natural postulates. First, the semantics of disjunctive revision programming must reduce to the semantics of justified revisions on disjunctive revision programs consisting of rules with a single literal in the head. Second, the shifting theorem must generalize to the case of disjunctive revision programs. Finally, the results of Section 4.1 indicate that there is yet another desirable criterion. Namely, the semantics of disjunctive revision programming over the empty initial database must reduce to the Lifschitz and Woo semantics for general logic programs. The construction given below satisfies all these three conditions.

First, let us introduce the syntax of disjunctive revision programs. By a *disjunctive revision rule* we mean an expression of the following form:

$$\alpha_1 | \ldots | \alpha_m \leftarrow \alpha_{m+1}, \ldots, \alpha_n \ , \tag{5}$$

where $\alpha_i,\ 1 \leq i \leq n$ are literals (that is, expressions of the form $\mathbf{in}(a)$ or $\mathbf{out}(a)$). A *disjunctive revision program* is a collection of disjunctive revision rules.

In order to specify semantics of disjunctive revision programs we first define the closure of a set of literals under a disjunctive rule. A set L of literals is *closed* under a rule (5) if at least one α_i, $1 \leq i \leq m$, belongs to L or if at least one α_i, $m+1 \leq i \leq n$, does *not* belong to L. A set of literals L is *closed* under a disjunctive revision program P if it is closed under all rules of P.

The next step involves the generalization of the notion of necessary change. Let P be a disjunctive revision program. A *necessary change* entailed by P is any minimal set of literals that is closed under P. Notice that in the context of disjunctive programs the necessary change may not be unique.

Recall that a database is a collection of atoms from universe U. A literal l is *satisfied* by a database $R \subseteq U$ if $l = \mathbf{in}(a)$ and $a \in \mathcal{R}$, or $l = \mathbf{out}(a)$ and $a \notin \mathcal{R}$, for some $a \in U$. We say that the body of a disjunctive revision rule is *satisfied* by a database R if every literal from the body is satisfied by R.

We will now introduce the notion of a reduct of a disjunctive revision program P with respect to two databases $\mathcal{I}$ (initial database) and $\mathcal{R}$ (a putative revision of $\mathcal{I}$). The reduct, denoted by $P^{\mathcal{I},\mathcal{R}}$, is constructed in the following four steps.

Step 1: Eliminate from the body of each rule in P all literals in $I(\mathcal{I}, \mathcal{R})$.
Step 2: Remove all rules r, such that $head(r) \cap I(\mathcal{I}, \mathcal{R}) \neq \emptyset$.
Step 3: Eliminate from the remaining rules every rule whose body is not satisfied by $\mathcal{R}$.
Step 4: Remove from the heads of the rules all literals that are not satisfied by $\mathcal{R}$.

We are ready now to define the notion a P-justified revision of a database $\mathcal{I}$ for the case of disjunctive revision programs. Let P be a disjunctive revision program. A database $\mathcal{R}$ is a *P-justified revision* of a database $\mathcal{I}$ if for some coherent necessary change L of $P^{\mathcal{I},\mathcal{R}}$, $\mathcal{R} = \mathcal{I} \oplus L$. Let us observe that only steps

(1) and (2) in the definition of reduct are important. Steps (3) and (4) do not change the defined notion of revision but lead to a simpler program.

The next example illustrates a possible use of disjunctive revision programming.

Example 3. Let us now represent the situation of Example 1 as a disjunctive revision program P:

$$\begin{aligned} \mathbf{in}(Ann) \mid \mathbf{in}(Bob) &\leftarrow \\ \mathbf{out}(Tom) \mid \mathbf{in}(David) &\leftarrow \\ \mathbf{out}(Ann) &\leftarrow \mathbf{in}(David) \\ \mathbf{out}(David) &\leftarrow \mathbf{in}(Bob) \end{aligned}$$

Assume that $\mathcal{I} = \{Ann, Tom\}$, $\mathcal{R} = \{Ann\}$. Then, inertia $I(\mathcal{I}, \mathcal{R}) = \{\mathbf{in}(Ann), \mathbf{out}(Bob), \mathbf{out}(David)\}$. The reduct $P^{\mathcal{I},\mathcal{R}} = \{\mathbf{out}(Tom) \leftarrow\}$. The only necessary change of $P^{\mathcal{I},\mathcal{R}}$ is $L = \{\mathbf{out}(Tom)\}$. Since L is coherent and $\mathcal{R} = \mathcal{I} \oplus L$, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$. □

The following three theorems show that the semantics for disjunctive revision programs described here satisfies the three criteria described above.

Theorem 7. *Let P be a revision program (without disjunctions). Then, $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ if and only if $\mathcal{R}$ is a P-justified revision of $\mathcal{I}$ when P is treated as a disjunctive revision program.*

Proof. For any revision program P (without disjunctions), the least model of P, when treated as a Horn program built of independent propositional atoms of the form $\mathbf{in}(a)$ and $\mathbf{out}(a)$, is closed under P. Moreover, every set of literals that is closed under P must contain the least model of P. Therefore, the notions of necessary change coincide for revision programs without disjunctions, when treated as ordinary revision programs and as disjunctive revision programs. Hence, the notions of justified revisions coincide, too. □

The definition of T_W naturally extends to the case of disjunctive revision programs.

Theorem 8 (Shifting theorem). *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be databases, and let P be a disjunctive revision program. Let $W = \mathcal{I}_1 \div \mathcal{I}_2$. Then, $\mathcal{R}_1$ is P-justified revision of $\mathcal{I}_1$ if and only if $T_W(\mathcal{R}_1)$ is $T_W(P)$-justified revision of $\mathcal{I}_2$.*

Proof. Similarly to the case of ordinary revision programs, in computing justified revisions for disjunctive revision programs we are dealing with literals. W-transformation can be viewed as renaming these literals, which does not effect the procedure. Therefore, the statement of the theorem holds. □

The embedding of (unitary) revision programs extends to the case of disjunctive revision programs. As before, each literal $\mathbf{in}(a)$ is replaced by the corresponding atom a and each literal $\mathbf{out}(a)$ is replaced by *not* a. The general logic

program obtained in this way from a disjunctive revision program P is denoted by $dj(P)$.

Theorem 9. *Let P be a disjunctive revision program. Then, $\mathcal{R}$ is a P-justified revision of $\emptyset$ if and only if $\mathcal{R}$ is an answer set for $dj(P)$.*

Proof. First notice that for every $\mathcal{R}$, $I(\emptyset, \mathcal{R})$ is equal to $\{\mathbf{out}(a) : a \notin \mathcal{R}\}$.

Observe that step 2 in the definition of the reduct $P^{\mathcal{I},\mathcal{R}}$ removes exactly those rules r for which $dj(r)$ satisfies condition $HNeg \not\subseteq \mathcal{R}$.

Step 3 removes all rules r for which $dj(r)$ satisfies condition $BNeg \cap \mathcal{R} \neq \emptyset$, as well as rules containing $\mathbf{in}(a)$ in the bodies for some $a \notin \mathcal{R}$ (corresponding disjunctive logic program rules have a in the bodies for some $a \notin \mathcal{R}$).

Step 1 eliminates from the bodies of the rules of P all literals that are in $I(\mathcal{I}, \mathcal{R})$. In disjunctive logic program it corresponds to eliminating $not(BNeg)$ parts from the bodies of the remaining rules.

Step 4 in particular corresponds to eliminating $not(HNeg)$ parts from the heads of the remaining disjunctive logic program rules.

Therefore, $dj(P)^{\mathcal{R}}$, when compared to $dj(P^{\mathcal{I},\mathcal{R}})$, may only have some extra rules, the bodies of which are not satisfied by $\mathcal{R}$, or some extra literals in the heads, which are not satisfied by $\mathcal{R}$. Hence, the statement of the theorem holds. □

We conclude this section with a simple observation related to the computational complexity of a problem of existence of justified revisions in the case of disjunctive revision programming. We will show that disjunctive revision programming is an essential extension of the unitary revision programming. In [MT98] it was proved that the problem of existence of a justified revision in the case of unitary revision programming is NP-complete. Using the results of Eiter and Gottlob [EG95] and our correspondence between disjunctive revision programs and general logic programs we obtain the following result.

Theorem 10. *The following problem is Σ_2^P-complete: Given a finite disjunctive revision program and a database $\mathcal{I}$, decide whether $\mathcal{I}$ has a P-justified revision.*

It follows that disjunctive revision programming is an essential extension of the unitary revision programming (unless the polynomial hierarchy collapses).

5 Future work

Lifschitz, Tang and Turner [LTT97] extended the answer set semantics to a class of logic programs with nested expressions permitted in the bodies and heads of rules. It can be shown that our formalism can be lifted to revision programs admitting nested occurrences of connectives as well.

The connections between revision programming and logic programming, presented in this work, imply a straightforward approach to compute justified revisions. Namely, a revision problem $(P, \mathcal{I})$ must first be compiled into a general logic program (by applying the transformation $T_{\mathcal{I}}$ to P). Then, answer sets to $T_{\mathcal{I}}(P)$ must be computed and "shifted" back by means of $T_{\mathcal{I}}$.

To compute the answer sets of the general logic program $T_{\mathcal{J}}(P)$, one might use any of the existing systems computing stable models of logic programs (for instance s-models [NS96], DeReS [CMMT95], and for disjunctive case DisLoP [ADN97], or a system `dlv` presented in [ELM+97]). Some care needs to be taken to model rules with negation as failure operator in the heads as standard logic program clauses or defaults.

In our future work, we will investigate the efficiency of this approach to compute justified revisions and we will develop related techniques tailored specifically for the case of revision programming.

References

ADN97. C. Aravindan, J. Dix, and I. Niemelä. DisLoP: Towards a disjunctive logic programming system. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 1997. 88

ALP+98. J.J. Alferes, J.A. Leite, L.M. Pereira, H. Przymusinska, and T.C. Przymusinski. Dynamic logic programming. Accepted at KR'98: Sixth International Conference on Principles of Knowledge Representation and Reasoning, Trento, Italy, June 1998. 74

AP97. J.J. Alferes and L.M. Pereira. Update-programs can update programs. In *Non-Monotonic Extensions of Logic Programming (Bad Honnef, 1996)*, volume 1216 of *Lecture Notes in Computer Science*, pages 110–131, Berlin, 1997. Springer. 74

Bar97. C. Baral. Embedding revision programs in logic programming situation calculus. *Journal of Logic Programming*, 30(1):83–97, 1997. 74

BM97. N. Bidoit and S. Maabout. Update programs versus revision programs. In *Non-monotonic extensions of logic programming (Bad Honnef, 1996)*, volume 1216 of *Lecture Notes in Computer Science*, pages 151–170, Berlin, 1997. Springer. 74

CMMT95. P. Cholewiński, W. Marek, A. Mikitiuk, and M. Truszczyński. Experimenting with nonmonotonic reasoning. In *Logic programming (Kanagawa, 1995)*, MIT Press Series in Logic Programming, pages 267–281, Cambridge, MA, 1995. MIT Press. 88

EG95. T. Eiter and G. Gottlob. On the computational cost of disjunctive logic programming: propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995. 87

ELM+97. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for non-monotonic reasoning. In *Logic programming and nonmonotonic reasoning (Dagstuhl, Germany, 1997)*, volume 1265 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 1997. 88

Lif96. V. Lifschitz. Foundations of logic programming. In *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, 1996. 80, 84

LTT97. V. Lifschitz, L. R. Tang, and H. Turner. Nested expressions in logic programs. unpublished draft, 1997. 87

LW92. V. Lifschitz and T.Y.C. Woo. Answer sets in general nonmonotonic reasoning. In *Proceedings of the 3rd international conference on principles of knowledge representation and reasoning, KR '92*, pages 603–614, San Mateo, CA, 1992. Morgan Kaufmann. 74, 77, 80, 80

MT94. W. Marek and M. Truszczyński. Revision specifications by means of programs. In *Logics in artificial intelligence (York, 1994)*, volume 838 of *Lecture Notes in Computer Science*, pages 122–136, Berlin, 1994. Springer. 74

MT95. W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *Proceedings of the 5th International Conference on Database Theory — ICDT 95*, pages 368–382. Berlin: Springer-Verlag, 1995. Lecture Notes in Computer Science 893. 74

McCT95. N. McCain and H. Turner. A causal theory of ramifications and qualifications. In *IJCAI-95, Vol. 1, 2 (Montreal, PQ, 1995)*, pages 1978–1984, San Francisco, CA, 1995. Morgan Kaufmann. 74

MT98. W. Marek and M. Truszczyński. Revision programming. *Theoretical Computer Science*, 190(2):241–277, 1998. 73, 74, 74, 74, 74, 74, 75, 77, 77, 77, 77, 77, 78, 87

NS96. I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of JICSLP-96*. MIT Press, 1996. 88

PT97. T. C. Przymusinski and H. Turner. Update by means of inference rules. *Journal of Logic Programming*, 30(2):125–143, 1997. 74, 77

SI95. C. Sakama and K. Inoue. Embedding circumscriptive theories in general disjunctive programs. In *Logic programming and nonmonotonic reasoning (Lexington, KY, 1995)*, volume 928 of *Lecture Notes in Computer Science*, pages 344–357, Berlin, 1995. Springer. 80

Tur97. H. Turner. Representing actions in logic programs and default theories: a situation calculus approach. *Journal of Logic Programming*, 31(1-3):245–298, 1997. 74

Quantifiers and the System KE: Some Surprising Results*

Uwe Egly

Institut für Informationssysteme E184.3
Technische Universität Wien
Treitlstraße 3, A–1040 Wien, Austria
emailuwe@kr.tuwien.ac.at

Abstract. In this paper, we consider the free-variable variant of the calculus KE and investigate the effect of different preprocessing activities to the proof length of variants of KE. In this context, skolemization is identified to be harmful as compared to the δ-rule. This does not only have consequences for proof length in KE, but also for the "efficiency" of some structural translations. Additionally, we investigate the effect of quantifier-shifting and quantifier-distributing rules, respectively. In all these comparisons, we identify classes of formulae for which a non-elementary difference in proof length occur.

1 Introduction

Many calculi used in the field of automated deduction rely on specific normal forms. The given closed first-order formula is transformed by appropriate translation procedures into the desired normal form. Additionally, several techniques like antiprenexing, i.e., the shifting of quantifiers inwards in order to minimize the scope of quantifiers, can be applied. Since such techniques are applied prior to the deduction process, they are summarized under the term *preprocessing activities*. But even if a calculus for the full first-order syntax is used and therefore no preprocessing is necessary, simplifications like antiprenexing are sometimes beneficial in this case.

Usually, recommendations for preprocessing activities are given with a certain (class of) calculi in mind. An example for such a recommendation is:

> If possible, apply quantifier-shifting[1] rules in order to reduce the scope of quantifiers.

This recommendation is justified by the fact that Herbrand complexity[2] (HC) is never increased as long as the distribution of quantifiers is avoided. Since HC is a

* The author would like to thank Hans Tompits for his useful comments on an earlier version of this paper.

[1] An example of such a rule is $\frac{\forall x\,(A \vee B)}{(\forall x\,A) \vee B}$, where x does not occur free in B.

[2] Let F be a formula without essentially universal (strong) quantifiers. Then HC is the cardinality of a minimal valid set of ground instances of F.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 90–104, 1999.

lower bound for the proof complexity in many cut-free calculi (like, e.g., cut-free LK, analytic tableau, various connection calculi), following the recommendation is advocated for these calculi.

A disadvantage of all the mentioned calculi is their weakness with respect to finding short proofs. For instance, it is shown in [5] that the propositional analytic tableau calculus is weaker than the well-known truth table method, which itself is always exponential in the number of propositional variables of the input formula. This means that there exist classes of formulae for which the length of *any* analytic tableau is exponential in the size of the (complete) truth table.

In order to overcome the weakness of these calculi (to some extent), a controlled integration of the cut-rule is necessary. There are many techniques which can be characterized by the cut-rule where the (structure of the) cut-formula is restricted and easily computable from the given formula (see [10] for an overview and references). As we will see below, this (and similar) recommendations have to be checked if new calculi are used because *different calculi can yield extremely different behavior!* As an example, we mention the result of Section 4.3 that shifting one (!) quantifier causes a non-elementary increase of proof length in a specific variant of KE for some class of first-order formulae.

In this paper, we use the calculus KE in a free-variable form introduced in Section 3. KE has been defined in [15] (with different quantifier rules introducing parameters and constants). The propositional fragment was studied in [5,6].

A remarkable property of KE is the necessity of the cut-rule because it is its only branching rule. Since an unrestricted use of cut is unfeasible for proof search, the cut rule has to be restricted in such a way that the subformula property is obeyed. Indeed, KE remains complete even for highly restricted variants of the cut-rule. We give new and easy soundness and completeness proofs of KE with restricted cut-rules by providing polynomial simulations with extended variants of analytic tableaux.

In Section 4, different variants of KE are compared. Moreover, skolemization is identified to be harmful as compared to the δ-rule. Additionally, we investigate the effect of quantifier-shifting and quantifier-distributing rules, respectively. In all these comparisons, we identify classes of formulae for which a non-elementary difference in proof length occur. This indicates that some techniques destroy the possibility to use (some forms of) cut resulting in a tremendous increase of proof complexity.

In Section 5, we discuss consequences of our results for other formalisms like definitional (structure-preserving) translations to normal form and different variants of circumscription.

2 Preliminaries

Throughout this paper we use a standard first-order language with function symbols. We assume familiarity with the basic concepts including skolemization (see, e.g., [12] for an introduction).

In order to distinguish between formula occurrences in the (formula tree of the) input formula and substitution instances of these subformulae, we introduce the notions of *s-subformulae* (structural subformulae) and *subformulae*, respectively.

Definition 1. *Given a formula F, we call G an* immediate s-subformula (*structural subformula*) *of F if either*

1. $F = \neg G_1$ *and* $G = G_1$, *or*
2. $F = G_1 \circ G_2$ $(\circ \in \{\wedge, \vee, \rightarrow\})$ *and* $G = G_1$ *or* $G = G_2$, *or*
3. $F = \mathsf{Q}x\ G_1$ $(\mathsf{Q} \in \{\forall, \exists\})$ *and* $G = G_1$.

The relation "is s-subformula of" *is the transitive closure of the relation* "is an immediate s-subformula of" *together with the pair (H, H) for any formula H. We call G an* immediate subformula *of F if either one of 1.–2. holds, or*

4. $F = \mathsf{Q}x\ G_1$ *and* $G = G_1{}_t^x$ *for an arbitrary term t.*

The relation "is subformula of" *is defined analogously.*

A remarkable peculiarity of subformulae is the "violation" of essentially existential quantification. Assume that $F := \forall x\ P(x)$ is the formula to be proven, i.e., $\neg F$ is refuted. The indicated quantifier is replaced by a Skolem term s (by the skolemization procedure); hence, the resulting formula is of the form $\neg P(s)$. In contrast, $P(t)$ (for an arbitrary term t) is a subformula of F, but not an s-subformula of F.

Definition 2. *The* length *of a formula F, denoted by $|F|$, is the number of symbol occurrences in the string representation of F. If $\Delta = F_1, \ldots, F_n$ then $|\Delta| = \sum_{i=1}^{n} |F_i|$. If $\Delta = \{F_1, \ldots, F_n\}$ is a set with n elements then $|\Delta| = \sum_{i=1}^{n} |F_i|$. The* length *of a tableau derivation α, denoted by $|\alpha|$, is $\sum_{S \in \mathcal{M}} |S|$, where $\mathcal{M}$ is the multiset of formulae occurring in α. By #nodes(α), we denote the* number of nodes *in (tree) α.*

The *logical complexity of a formula F* is denoted by $\mathrm{lcomp}(F)$.

Definition 3. *Let $\mathbf{2}\colon \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ be the hyper-exponential function with $\mathbf{2}_0^i = i$ and $\mathbf{2}_{n+1}^i = 2^{\mathbf{2}_n^i}$ for all $i, n \in \mathbb{N}$. Let $\mathsf{s}\colon \mathbb{N} \rightarrow \mathbb{N}$ be defined as $\mathsf{s}(n) = \mathbf{2}_n^1$ for all $n \in \mathbb{N}$.*

Definition 4. *A function $e\colon \mathbb{N} \rightarrow \mathbb{N}$ is called* elementary *if there exists a Turing machine M computing e and a number $k \in \mathbb{N}$ such that $time_M(n) \leq \mathbf{2}_k^n$ for all $n \in \mathbb{N}$, where $time_M(n)$ is the computing time of M on input n.*

Let us remark that the function s is not elementary.

The following definition of a polynomial simulation (resp. an elementary simulation) is adapted from [7] and restricted to the case that the connectives in both calculi are identical.

Definition 5. *A calculus P_1 can* polynomially simulate (elementarily simulate) *a calculus P_2 if there is a polynomial p (an elementary function e) such that the following holds. For every proof of a formula F in P_2 of length n, there is a proof of F in P_1, whose length is not greater than $p(n)$ ($e(n)$).*

α	α_1	α_2
$\mathbf{t}(A \wedge B)$	$\mathbf{t}(A)$	$\mathbf{t}(B)$
$\mathbf{f}(A \vee B)$	$\mathbf{f}(A)$	$\mathbf{f}(B)$
$\mathbf{f}(A \rightarrow B)$	$\mathbf{t}(A)$	$\mathbf{f}(B)$
$\mathbf{t}(\neg A)$	$\mathbf{f}(A)$	$\mathbf{f}(A)$
$\mathbf{f}(\neg A)$	$\mathbf{t}(A)$	$\mathbf{t}(A)$

β	β_1	β_2
$\mathbf{f}(A \wedge B)$	$\mathbf{f}(A)$	$\mathbf{f}(B)$
$\mathbf{t}(A \vee B)$	$\mathbf{t}(A)$	$\mathbf{t}(B)$
$\mathbf{t}(A \rightarrow B)$	$\mathbf{f}(A)$	$\mathbf{t}(B)$

γ	$\gamma(t)$
$\mathbf{t}(\forall x A)$	$\mathbf{t}(A^x_t)$
$\mathbf{f}(\exists x A)$	$\mathbf{f}(A^x_t)$

δ	$\delta(t)$
$\mathbf{t}(\exists x A)$	$\mathbf{t}(A^x_t)$
$\mathbf{f}(\forall x A)$	$\mathbf{f}(A^x_t)$

Fig. 1. Smullyan's uniform notation.

$$\frac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}} \qquad \frac{\beta}{\begin{array}{c}\beta_1^c\\ \hline \beta_2\end{array}} \qquad \frac{\beta}{\begin{array}{c}\beta_2^c\\ \hline \beta_1\end{array}} \qquad \frac{\gamma}{\gamma(y)} \qquad \frac{\delta}{\delta(f(\boldsymbol{z}))} \qquad \frac{}{\mathbf{t}(A) \mid \mathbf{f}(A)}$$

Fig. 2. The inference rules of KE.

3 The Calculus KE

We consider the free-variable variant of the calculus KE. The original calculus was defined in [15] (with different quantifier rules introducing parameters and constants). The propositional fragment was studied in [5,6].

For convenience, Smullyan's uniform notation is used in order to reduce the number of cases. The notation is summarized in Fig. 1. Let A be a signed formula. If A is of the form $\mathbf{f}(B)$ then A^c is $\mathbf{t}(B)$; otherwise, A^c is $\mathbf{f}(B)$. We consider the KE-rules depicted in Fig. 2. In case of the γ-type, y is a globally new variable. In case of the δ-type, $\boldsymbol{z} = z_1, \ldots, z_n$ are the free variables of A, and f is a Skolem function symbol. The right-most rule is the cut-rule; A is called the cut-formula.

Observe that cut is the *only* branching rule in KE. In contrast to sequent or tableau calculi, where cut is a redundant rule, it is necessary in KE and cannot be eliminated.

Remark 1. The KE-rules differ from the usual rules for free-variable tableaux (together with the cut-rule) in the rules for β-formulae. The other rules are identical.

The β-rule in tableau is $\frac{\beta}{\beta_1 | \beta_2}$. We will call the tableau calculus with this β-rule, and the α-, γ-, and δ-rule from above, *analytic tableau*. We will also use *analytic tableaux with the asymmetric β-rule*, which is as follows:

$$\frac{\beta}{\begin{array}{l}\beta_1 \mid \beta_2\\ \quad \mid \beta_1^c\end{array}} \qquad \frac{\beta}{\begin{array}{l}\beta_1 \mid \beta_2\\ \beta_2^c \mid\end{array}}$$

As one might expect, this asymmetric branching rule simulates (some restricted form) of analytic cut by introducing β_i $(i = 1, 2)$ in both polarities (indicated by β_i and β_i^c).

In the next two definitions, $\mathcal{X}$ stands either for tableau or KE.

Definition 6. *Let $\mathcal{S} = \{F_1, \ldots, F_m\}$ be a set of signed formulae. Then $\mathcal{T}$ is a $\mathcal{X}$-*tree for $\mathcal{S}$ *if there exists a finite sequence $\mathcal{T}_1, \ldots, \mathcal{T}_n$ such that (i) $\mathcal{T}_1$ is the initial branch consisting of $F_1, \ldots, F_m$, (ii) $\mathcal{T}_n = \mathcal{T}$, and, (iii) for each $1 \leq i < n$, $\mathcal{T}_{i+1}$ is the result of a $\mathcal{X}$-rule applied to (formulae from) $\mathcal{T}_i$.*

Definition 7. *Let $\mathcal{T}$ be a $\mathcal{X}$-tree for a set of signed formulae. A branch b is* closed *if $\mathbf{t}(A)$ and $\mathbf{f}(A)$ occur on b. If A is some atom then b is called* atomically closed. *Branches which are not closed are called* open. *$\mathcal{T}$ is closed if all branches of $\mathcal{T}$ are closed. A $\mathcal{X}$-*proof *of a formula F is a closed $\mathcal{X}$-tree for $\{\mathbf{f}(F)\}$.*

In free-variable tableaux, instantiation of variables is deferred until closing a path. Suppose that we have two (signed) atoms $\mathbf{t}(A)$ and $\mathbf{f}(B)$ on a path b such that A and B are unifiable by the most general unifier (mgu) σ, i.e., $A\sigma = B\sigma$. Then we can apply the closure operation, mark b closed (this is indicated by "$*$" at the end of b) and apply the resulting mgu σ to the whole tableau. The branch b becomes atomically closed. The same closure operation is used to close branches in KE-trees.

Remark 2. It is not really necessary to restrict the closure operation to signed atoms. Alternatively, signed formulae can be used to close branches. The necessary unification algorithm for formulae with quantifiers has been developed in [17] (see also [11]).

Lemma 1. *Let $\mathcal{T}$ be a* KE*-proof of a formula F. Then there exists a* KE*-proof $\mathcal{T}'$ of F, $\mathcal{T}'$ is atomically closed and $|\mathcal{T}'| \leq 5 \cdot |\mathcal{T}|^4$.*

Proof. We first show property (P) below by induction on the logical complexity of A.

(P) Let $\mathbf{t}(A)$ and $\mathbf{f}(A)$ be two signed formulae on the same path. Then there exists an atomically closed cut-free KE-tree $\mathcal{T}_A$ for $\{\mathbf{t}(A), \mathbf{f}(A)\}$, $\#\text{nodes}(\mathcal{T}_A) \leq 3 \cdot \text{lcomp}(A) + 2$, and $|\mathcal{T}_A| \leq \#\text{nodes}(\mathcal{T}_A) \cdot |A|^2$.

Basis. $\text{lcomp}(A) = 0$, i.e., A is an atom. Set $\mathcal{T}_A$ to the initial branch and obtain $\#\text{nodes}(\mathcal{T}_A) = 2$.
Step. (IH) Assume that (P) holds for all formulae B with $\text{lcomp}(B) < n$. Let $\text{lcomp}(A) = n$. In the following, we refer to cases 1–3 given below.
Case 1. $A = \neg B$.
Case 2. A is an α-formula (except a negation) or A is a β-formula. Without loss of generality assume that A is an α-formula.
Case 3. A is a γ-formula or A is a δ-formula. Without loss of generality assume that A is a γ-formula.

Consider the left inference figure in Fig. 3 for case 1, the middle inference figure for case 2 and the right inference figure for case 3. Let the last two signed formulae be denoted by $\mathbf{f}(B)$ and $\mathbf{t}(B)$. In all of the three cases, (IH) provides an atomically closed cut-free KE-tree $\mathcal{T}_B$ for $\{\mathbf{f}(B), \mathbf{t}(B)\}$. Then $\#\text{nodes}(\mathcal{T}_A) \leq 3 + \#\text{nodes}(\mathcal{T}_B) = 3 + 3 \cdot \text{lcomp}(B) + 2$ and (with $\text{lcomp}(B) \leq \text{lcomp}(A) - 1$) $\#\text{nodes}(\mathcal{T}_A) \leq 3 \cdot \text{lcomp}(A) + 2$. In order to estimate the length of the resulting

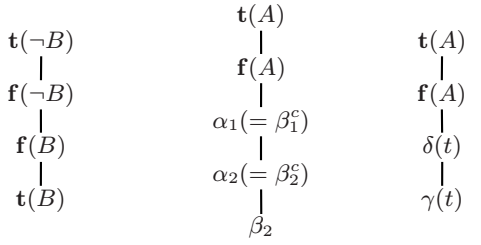


Fig. 3. Different inference figures for the proof of Lemma 1.

KE-tree, observe that any formula in $\mathcal{T}_A$ has length $\leq |A|^2$ (because Skolem terms introduced by the δ-rule can yield a quadratic increase of length in the worst case). Hence, $|\mathcal{T}_A| \leq (3 \cdot \mathrm{lcomp}(A) + 2) \cdot |A|^2$. This concludes the induction proof of (P).

Let c_m be the maximal logical complexity of a formula in $\mathcal{T}$ and let l_m be the length of the "biggest" formula of $\mathcal{T}$. Clearly, $c_m, l_m \leq |\mathcal{T}|$, $|\mathcal{T}| \geq 2$ and $|\mathcal{T}'|$ is estimated as follows.

$$|\mathcal{T}'| \leq (3 \cdot c_m + 2) \cdot l_m^2 \cdot |\mathcal{T}| \leq 3 \cdot |\mathcal{T}|^4 + 2 \cdot |\mathcal{T}|^3 \leq 5 \cdot |\mathcal{T}|^4$$

This concludes the proof of the lemma. □

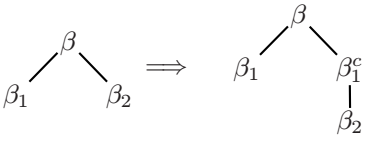
Due to Lemma 1, it is sufficient that paths are closed. We do not require that paths are atomically closed because we are interested in p-simulations between calculi.

Definition 8. *An application of cut in a tableau or* KE*-proof of F is called* analytic *if the cut-formula is a subformula or the negation of a subformula of F. An application of cut in a branch b of a* KE*-tree is called* strongly analytic *if β occurs on b, the cut formula is β_i (for some i=1,2) and neither β_1 nor β_2 occurs on b (above the cut). A* KE*-tree is called analytic (strongly analytic) if it contains only analytic (strongly analytic) applications of cut.*

Tableaux with analytic cut is the tableau calculus extended by the analytic cut rule.

Lemma 2. *The calculus* KE *can p-simulate the analytic tableau calculus (even with cut).*

Proof. It is sufficient to show that the tableau rules for β-formulae can be polynomially simulated by KE-rules. The (left) tableau β-rule is replaced by an application of cut and an application of a KE-rule for β-formulae (depicted on the right).



If, in a tableau proof of a formula F, each application of a β-rule is replaced by the corresponding KE-rules, the resulting KE-proof of F has length polynomial in the length of the tableau proof. □

Observe that the right inference figure produces the same open paths as the asymmetric tableau β-rule.

Corollary 1. *Any cut-free tableau proof of a formula F (even with applications of the asymmetric β-rule) can be transformed into a strongly analytic closed* KE*-tree for* $\{\mathbf{f}(F)\}$.

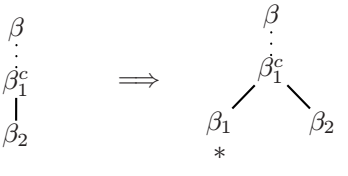
Proof. Inspecting the translation described in the proof of Lemma 2 reveals that the indicated application of cut is strongly analytic if the corresponding branch b in the tableau proof contains neither β_1 nor β_2. If b contains one or both of the latter formulae then the application of the β-rule in the tableau proof is superfluous and can be deleted. As a result, we get a simpler and shorter tableau proof which is used for the translation. □

Corollary 2. *Any tableau proof of a formula F with analytic cuts can be transformed into an analytic closed* KE*-tree for* $\{\mathbf{f}(F)\}$.

Proof. Obvious. □

Lemma 3. *The tableau calculus with cut can p-simulate the calculus* KE.

Proof. It is sufficient to show that the KE-rules for β-formulae and cut can be polynomially simulated by tableau rules and cut. Any application of cut in the KE-proof is replaced by cut using the same cut formula. The (left) KE-rule for β-formulae is replaced by an application of the β-rule (depicted on the right). We show only one KE-rule; the other rule is handled similarly.



If, in a KE-proof of a formula F, each application of a rule for β-formulae is replaced by an application of a β-rule, the resulting tableau proof of F has length polynomial in the length of the KE-proof. □

Corollary 3. *The tableau calculus with analytic cut can p-simulate analytic* KE.

Proof. Obvious. □

Corollary 4. *The tableau calculus with the asymmetric β-rule can p-simulate strongly analytic* KE.

Proof. It is sufficient to show that strongly analytic cuts can be polynomially simulated by the asymmetric β-rules. The (left) strongly analytic cut is replaced by an application of the asymmetric β-rule (depicted on the right). We show only one case; the other is handled similarly.

If, in a KE-proof of a formula F, each application of strongly analytic cuts is replaced by an application of the asymmetric β-rules, the resulting tableau proof of F has length polynomial in the length of the KE-proof. □

Theorem 1. *The calculus* KE *is sound and complete. The same holds if all applications of cut are analytic or strongly analytic.*

Proof. By Lemma 2, Corollary 1, Corollary 2, and Lemma 3. □

4 Comparisons

In this section, we compare strongly analytic KE with analytic KE. Moreover, the influence of skolemization, quantifier-distributing rules and quantifier-shifting rules with respect to the complexity of minimal strongly analytic KE-proofs are investigated. It turns out that many suggestions for the use of quantifier rules should be carefully inspected if strongly analytic KE is used.

4.1 Preparatory Results

Let us first define F_n and recapitulate some facts about the length of its proofs in sequent systems and analytic tableaux.

Definition 9. *Let* $(F_n)_{n\in\mathbb{N}}$ *be a sequence of formulae with*

$$F_n = \forall x\,((\forall w_0 \exists v_0\, P(w_0, x, v_0) \wedge C(x)) \rightarrow B_n(x))$$
$$C(x) = \forall uvw\,(\exists y\,(P(y, x, u) \wedge \exists z\,(P(v, y, z) \wedge P(z, y, w))) \rightarrow P(v, u, w))$$
$$B_n(x) = \exists v_n\,(P(x, x, v_n) \wedge \exists v_{n-1}\,(P(x, v_n, v_{n-1}) \wedge \ldots \wedge \exists v_0\, P(x, v_1, v_0))\ldots).$$

The following theorem is a corollary of Theorems 1 and 4 in [16].

Theorem 2. *Let* $F'_n := (\forall w_0\, P(w_0, b, g(w_0)) \wedge C(b)) \rightarrow B_n(b)$ *be the skolemized form of* F_n*. Let* ϕ_n *be any proof of* F_n *or* F'_n *in a cut-free analytic tableau calculus. Then,* $|\phi_n| \geq 2 \cdot \mathsf{s}(n)$*.*

In [16], Orevkov provides a short proof of F_n (in sequent systems) with exactly one application of the cut-rule and the number of sequents in this proof is linear in n.

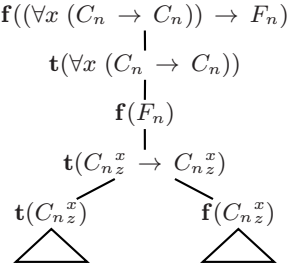
Let C_n be the cut-formula in this proof with one free variable x. The cut-formula $C_n = A_n(x)$ is defined inductively as follows.

$$
\begin{aligned}
A_0(\alpha) &= \forall w_0 \exists v_0\ p(w_0, \alpha, v_0) \\
A_{i+1}(\alpha) &= \forall w_{i+1}\ (A_i(w_{i+1}) \rightarrow \overline{A}_{i+1}(w_{i+1}, \alpha)) \\
\overline{A}_0(\alpha, \delta) &= \exists v_0\ p(\alpha, \delta, v_0) \\
\overline{A}_{i+1}(\alpha, \delta) &= \exists v_{i+1}\ (A_i(v_{i+1}) \wedge p(\alpha, \delta, v_{i+1}))
\end{aligned}
$$

It is well known that cuts can be eliminated with only a minor overhead if the change of the formula to be proven is allowed. The new formula in our case is then

$$(\forall x\ (C_n \rightarrow C_n)) \rightarrow F_n.$$

The effect of cut with cut-formula C_n, e.g., in an analytic tableau proof, can be simulated by an application of the γ-rule and an application of the β-rule. Hence, the modified formula possesses short proofs in analytic tableaux and cut-free sequent systems. The (structure of a) short proof of this formula in strongly analytic KE is as follows:



It is important that (an instance of) the cut-formula occurs in both branches indicated in the above tableau (in different polarity).

Let G'_n be constructed from F_n by adding an additional unique variable argument at the first position of any literal occurring in F_n such that this newly introduced variable z is globally new. Furthermore, the same procedure is applied to $C_n \rightarrow C_n$ resulting in $D'_n \rightarrow D'_n$. Let $D_n := \exists z \forall x\ (D'_n \rightarrow D'_n)$ and $G_n := \forall z\ G'_n$. Observe that all occurrences of the newly introduced variable z are replaced by Skolem terms if the respective subformula is decomposed in the construction of a closed analytic tableau.

We show that any analytic (or strongly analytic) KE-proof of F_n or G_n has length greater than $\mathsf{s}(n-c)$ for some constant c. We proved the following result as Lemma 5 in [10].

Lemma 4. *Let ϕ_n be a proof of F_n or G_n in tableau with analytic cut. Then, for sufficiently large n, $|\phi_n| \geq 2 \cdot \mathsf{s}(n-c)$ for some constant c.*

Due to the existence of p-simulations of tableaux with analytic cut by analytic KE (see Corollary 2) and vice versa (see Corollary 3), we get the following corollary.

Corollary 5. *Let ϕ_n be an analytic or strongly analytic* KE*-proof of G_n or F_n. Then, for sufficiently large n, $|\phi_n| \geq \mathsf{s}(n-d)$ for some constant d.*

4.2 On Skolemization and the δ-rules in KE

In variants of KE and analytic tableaux, the δ-rule is used to remove occurrences of essentially existential quantifiers dynamically. This is in contrast to most other calculi (like resolution) which require the input formula to be skolemized in advance. Skolemization is performed as a preprocessing activity prior to the "deduction process" even in analytic tableau systems like [4]. Such a preprocessing skolemization is considered to be more efficient than a dynamic δ-rule because

(i) it causes the removal of applications of the δ-rule from the search space, and
(ii) it is a computationally inexpensive operation.

In KE, however, instead of preprocessing skolemization, applications of the δ-rule can yield drastically (non-elementary) shorter proofs and search spaces. We investigate this phenomenon in the following.

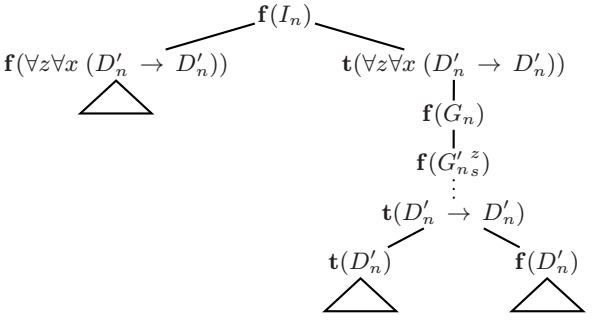
Reconsider D_n and G_n from above. We use the formula

$$I_n := (\forall z \forall x\ (D'_n \rightarrow D'_n)) \wedge G_n. \tag{1}$$

It is immediately apparent that any proof of I_n in cut-free standard analytic tableau calculi is non-elementary. There exists, however, a short proof of I_n in tableau with analytic cut.

Lemma 5. *There exists a strongly analytic* KE*-proof ϕ_n of (1) such that the length of ϕ_n is $\leq c \cdot 2^{d \cdot n}$ for some constants c, d.*

Proof. The proof ϕ_n is as follows:



The proof is divided into two main parts, namely the closed left subtableau and the closed right subtableau below $\mathbf{f}(I_n)$. Since $|D'_n|$ is exponential in length (with respect to n), an exponential length of the left closed KE-tableau follows from property (P) in the proof of Lemma 1. Observe that the two indicated variables z and x are replaced by Skolem terms by applying the δ-rule.

In the right subtableau, $\mathbf{f}(I_n)$ is decomposed resulting first in $\mathbf{f}(G_n)$ and then in $\mathbf{f}(G'^{z}_{ns})$ on the branch. Observe that s is a Skolem constant introduced by the δ-rule. By an application of the strongly analytic cut-rule, we get two branches: one with $\mathbf{f}(G'^{z}_{ns})$ and $\mathbf{t}(D'_n)$ and another one with $\mathbf{f}(G'^{z}_{ns})$ and $\mathbf{f}(D'_n)$. Now we have a similar situation as in Orevkov's short LK-proof with cut. The exponential length of the whole proof follows. □

An important feature in the KE-proof is the "change of quantification", i.e., a δ-formula in the left branch of a (strongly) analytic cut becomes a γ-formula in the right path (and vice versa). Let us consider the skolemized form of the negation of (1), namely

$$\neg I'_n := \neg((D_n^{(1)} \rightarrow D_n^{(2)})_c^z \wedge (G''_n)_d^z). \tag{2}$$

The property to "change the quantification" is lost by the introduction of Skolem terms prior to applications of (strongly) analytic cut. The following lemma can be proved with the same technique as Lemma 5 in [10].

Lemma 6. *Let ϕ_n be a (strongly) analytic* KE*-proof of I'_n. Then, for sufficiently large n, $|\phi_n| \geq 2 \cdot \mathsf{s}(n-c)$ for some constant c.*

Theorem 3. *There exists a sequence $(I_n)_{n\in\mathbb{N}}$ of first-order formulae such that, for sufficiently large n, the following holds:*

1. *There exists a (strongly) analytic* KE*-proof ϕ_n of I_n of length $\leq c \cdot 2^{d\cdot n}$ for some constants c and d.*
2. *Any analytic* KE*-proof ψ_n of the skolemized form of I_n has length $\geq \mathsf{s}(n-c)$ for some constant c.*

Although applications of the δ-rule slightly enlarge the search space, dynamic skolemization can yield a drastically better behavior. The reason is the destruction of cut-formulae by the newly introduced Skolem terms. More precisely, an application of the (strongly) analytic cut results in one path with a δ-formula and in another path with the corresponding γ-formula. If skolemization is applied as a preprocessing activity, the δ-formula has been replaced by the corresponding δ_1-formula with a Skolem term. This Skolem term is also present in the cut formula of the (strongly) analytic cut.

We stress that the result does *not* depend on a specific optimized δ-rule; even simple forms like the δ-rule suffices. The reason for the independence from the variant of the δ-rule is simple: only Skolem constants are introduced for the additional quantifier occurrences because they do not occur in the scope of essentially universal quantifiers.

4.3 Quantifier-rules and Proof Complexity

What happens if quantifier-distributing rules like

$$\frac{\forall x\,(A \wedge B)}{(\forall x\,A) \wedge (\forall x\,B)} \qquad \frac{\exists x\,(A \vee B)}{(\exists x\,A) \vee (\exists x\,B)}$$

are applied to the following formula, where $\sigma = \{z \backslash z'\}$ and $\mu = \{z \backslash f(z')\}$;

$$\forall z'\,(\forall x\,(D'_n \rightarrow D'_n)\sigma \wedge G'_n\mu) \tag{3}$$

In [3], a formula is defined for which the Herbrand complexity is slightly increased if quantifier-distributing rules are applied. Since Herbrand complexity is a lower bound for most of the analytic cut-free calculi like analytic tableau, cut-free sequent systems etc., this result implies an increase of proof complexity in all of these calculi. But even for resolution with its atomic cut-rule, an increase of proof complexity is observable. In [8], it is shown that resolution proof complexity is exponentially increased for some classes of formulae if quantifier-distributing rules are applied.

When strongly analytic KE is applied, things are extremely different. There are only *non-elementary* strongly analytic KE-proofs of (3). The reason is that

$$\forall x\ (D'_n\ \rightarrow\ D'_n)\sigma'\ \wedge\ G'_n\mu'$$

results from the elimination of $\forall z'$, where $\sigma' = \{z\backslash c\}$ and $\mu' = \{z\backslash f(c)\}$, and the instance $\forall x\ (D'_n\ \rightarrow\ D'_n)\sigma'$ cannot be used for a short simulation of cut.

In contrast, there exists a short proof (in the same calculus) of

$$\forall z'\ \forall x\ (D'_n\ \rightarrow\ D'_n)\sigma\ \wedge\ \forall z'\ G'_n\mu \tag{4}$$

with a similar structure like the proof of I_n in Lemma 5. Obviously, (4) can be obtained from (3) by an application of the quantifier-distributing rule above. As a result, the application of quantifier-distributing rules can cause a non-elementary *decrease* of proof length in strongly analytic KE!

The reason why HC or resolution proof complexity is increased is a duplication of (nearly) identical formulae or proofs. For instance, if $F = \forall x\ (p(x)\ \wedge\ p(x))\ \vee\ \exists y\ p(y)$ is the original formula, $G = (\forall x\ p(x)\ \wedge\ \forall x\ p(x))\ \vee\ \exists y\ p(y)$ the formula obtained by applying a quantifier-distribution rule, then two instances of the existentially quantified formula are required in the latter case (for the skolemized form of G), whereas only one such instance is required in the former case (for the skolemized form of F). Our result here is based on a completely different effect, namely the quantification of cut-formulae, together with the effect that δ-formulae in the "false" part become γ-formulae in the "true" part.

In the above discussion, only quantifier-distributing rules are considered. What happens if quantifier-shifting rules like

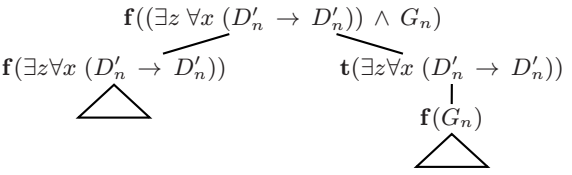
$$\frac{\forall x\ (A \circ B)}{(\forall x\ A) \circ B} \qquad \frac{\forall x\ (A \circ B)}{A \circ \forall x\ B}$$

(where x does not occur (free) in B and in A, respectively) are applied. It is shown in [3] that Herbrand complexity is not increased by applications of such rules. Here, however, we get a non-elementary increase of proof length if we apply quantifier-shifting rules in order to minimize the scope of quantifiers. Hence, antiprenexing, i.e., the movement of quantifiers inwards, is not always beneficial, even if only quantifier-shifting rules are applied.

Let us consider the formula

$$(\exists z\ \forall x\ (D'_n\ \rightarrow\ D'_n))\ \wedge\ G_n \tag{5}$$

and observe that the indicated $\exists z$ belongs to a γ-formula. Hence, we have



The indicated right KE-subtree is non-elementary in n. This is an immediate consequence of the construction of D_n, G_n, the fact that the two indicated subformulae are δ-formulae, and Lemma 4.

Consider the following formula

$$\exists z\, (\forall x\, (D'_n \rightarrow D'_n)) \wedge G_n). \tag{6}$$

and observe that formula (5) can be obtained from formula (6) by applying quantifier-shifting rules. The indicated quantifier $\exists z$ in (6) is prenexed. Surprisingly, the latter formula has a short strongly analytic KE-proof. The reason is the new "global" quantifier $\exists z$ which is outside the top-most β-formula. Observe that the elimination of this quantifier occurrence results in a new free variable but this variable does not have any effect when quantifiers in G_n are removed (by the δ-rule). Moreover, the free variable z in $E = \forall x\, (D'_n \rightarrow D'_n)$ is not bound in the course of a short strongly analytic KE-proof of E. Consequently, this free variable can be bound to the constant c introduced for the (first) quantifier occurrence $\forall z$ in G_n.

Unfortunately, there are also cases where the application of quantifier-shifting rules *decreases* proof complexity in strongly analytic KE non-elementarily. Consider the formula

$$\forall z\, (\forall x\, (D'_n \rightarrow D'_n) \wedge G_n) \tag{7}$$

and observe that different Skolem constants are introduced for z in $(\forall x\, (D'_n \rightarrow D'_n)$ and G'_n, respectively. Hence, the cut-formula also contains a Skolem term for z which is different from the Skolem term for z in G'_n. Therefore, any strongly analytic KE-proof of (7) is non-elementary. In contrast, there are short strongly analytic KE-proofs of

$$\forall z\, \forall x\, (D'_n \rightarrow D'_n) \wedge G_n \tag{8}$$

which is obtained from (7) applying a by quantifier-shifting rule.

5 Consequences for Other Formalisms

Although the results presented so far seem to be restricted to variants of the KE-calculus, they have implications for other mechanisms. In the following, we discuss two, namely structural translations to normal form and circumscription.

Let us first consider the structural translation γ_{struc} from [12]. This translation is intended to be an optimized and simplified version of the translation used in [2]. The optimization (with respect to the length of the resulting normal form!) is to transform formulae in *(skolemized) negation normal form* into a set of clauses by introducing new abbreviations for subformulae. Usually, structural

translations [7,9] are *not* restricted to negation normal form but apply to arbitrary closed first-order formulae. The important point here is that the restriction causes a non-elementary increase of proof length. This increase not only applies for (strongly) analytic KE but also for many other calculi (even with analytic cut). The main reason is that instances of $(D_n^{(1)} \rightarrow D_n^{(2)})_c^z$ from the skolemized variant of I_n no longer simulates instances of the required cut rule.

Circumscription [14] is a technique to formalize a nonmonotonic behavior of a reasoning agent. The basic principle is the completion of an implication to an equivalence. Usually, second-order logic is needed for a formalization, but there are restricted classes of formulae, e.g., the class of *solitary formulae*, for which circumscription can be formalized in first-order logic [13].

Let P be an n-ary predicate symbol, and let $\overline{x} := x_1, \ldots, x_n$ be a list of n variables. A formula is *solitary* in P iff it is equivalent to a formula of the following normal form $N(P) \wedge \forall \overline{x}\, (E(\overline{x}) \rightarrow P(\overline{x}))$. $N(P)$ is a formula which has no positive occurrences of P (but which may admit negative occurrences of P), and $E(\overline{x})$ is a formula with no occurrences of P whatsoever. Lifschitz proved that the two formulae

$$F_1 = \mathrm{CIRC}(N(P) \wedge \forall \overline{x}\, (E(\overline{x}) \rightarrow P(\overline{x})), P)$$
$$F_2 = N(P) \wedge \forall \overline{x}\, (E(\overline{x}) \leftrightarrow P(\overline{x}))$$

are equivalent.

Since the introduction of equivalences has strong connections to extensions, which in turn can be used to simulate the cut-rule [18,7,1], circumscribed formulae have (some forms of analytic) cut "compiled" into them. Since solitary formulae are those formulae which are *logically equivalent* to the form above, the concrete formula has to be determined. Here, however, care has to be taken because slightly differing variants of the cut-formula can have severe impact on proof length. Our results imply that slightly differing variants of circumscription yields formulae for which HC is extremely different.

6 Conclusion

We showed that some preprocessing activities can be harmful if strongly analytic KE-proofs are considered. This highly restricted variant of KE is a good candidate for implementing automated deduction systems. The propositional fragment can be considered as a non-clausal Davis-Putnam proof procedure, and the first-order variant corresponds to a slightly extended variant of analytic tableaux.

In contrast to what is recommended in the literature, quantifier-shifting rules have to be applied carefully. The same holds for other preprocessing activities like skolemization. In our context here, the "dynamic" δ-rule can behave "non-elementarily better" than skolemization.

The results of our comparisons indicate that preprocessing activities have to be considered together with the underlying calculus. Moreover, even slight mod-

ification of the input formula (one quantifier is shifted) can yield a tremendous non-elementary increase of (minimal) proof length.

References

1. M. Baaz, U. Egly, and A. Leitsch. Extension Methods in Automated Deduction. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume II, part 4, chapter 12, pages 331–360. Kluwer Academic Press, 1998. 103
2. M. Baaz, C. Fermüller, and A. Leitsch. A Non-Elementary Speed Up in Proof Length by Structural Clause Form Transformation. In *LICS'94*, pages 213–219, Los Alamitos, California, 1994. IEEE Computer Society Press. 102
3. M. Baaz and A. Leitsch. On Skolemization and Proof Complexity. *Fundamenta Informaticae*, 20:353–379, 1994. 101, 101
4. B. Beckert and J. Posegga. leanTAP: Lean Tableau-based Deduction. *J. Automated Reasoning*, 15(3):339–358, 1995. 99
5. M. D'Agostino. Are Tableaux an Improvement on Truth-Tables? Cut-Free Proofs and Bivalence. *J. Logic, Language and Information*, 1:235–252, 1992. 91, 91, 93
6. M. D'Agostino and M. Mondadori. The Taming of the Cut. Classical Refutations with Analytic Cut. *J. Logic and Computation*, 4:285–319, 1994. 91, 93
7. E. Eder. *Relative Complexities of First Order Calculi*. Vieweg, 1992. 92, 103, 103
8. U. Egly. On the Value of Antiprenexing. In F. Pfenning, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning*, pages 69–83. Springer Verlag, 1994. 101
9. U. Egly. On Different Structure-preserving Translations to Normal Form. *J. Symbolic Computation*, 22:121–142, 1996. 103
10. U. Egly. Cuts in Tableaux. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume I, part 1, chapter 4, pages 103–132. Kluwer Academic Press, 1998. To appear 1998. 91, 98, 100
11. M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, second edition, 1996. 94
12. A. Leitsch. *The Resolution Calculus*. Springer Verlag, 1997. 91, 102
13. V. Lifschitz. Computing Circumscription. In *Proceedings of IJCAI-85*, pages 121–127, Los Altos, CA., 1985. Morgan Kaufmann. 103
14. J. McCarthy. Circumscription – A Form of Non-Monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980. 103
15. M. Mondadori. Classical Analytical Deduction. Technical Report Annali dell'Università di Ferrara, Sezione III, Discussion Paper 1, Università di Ferrara, 1988. 91, 93
16. V. P. Orevkov. Lower Bounds for Increasing Complexity of Derivations after Cut Elimination. *Zapiski Nauchnykh Seminarov Leningradskogo Otdeleniya Matematicheskogo Instituta im V. A. Steklova AN SSSR*, 88:137–161, 1979. English translation in *J. Soviet Mathematics*, 2337–2350, 1982. 97, 97
17. W. Sieg and B. Kauffmann. Unification for Quantified Formulae. PHIL 44, Carnegie Mellon University, 1993. 94
18. G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 234–259. Seminars in Mathematics, V.A. Steklov Mathematical Institute, vol. 8, Leningrad, 1968. English translation: Consultants Bureau, New York, 1970, pp. 115–125. 103

Choice Construct and Lindström Logics

H. K. Hoang

Université de Paris 1
Centre de Recherche en Informatique
90 Rue de Tolbiac 75634 Paris Cedex 13 France
hoang@univ-paris1.fr

Abstract. A choice construct can be added to fixpoint logic to give a more expressive logic, as shown in [GH]. On the other hand, a more straightforward way of increasing the expressive power of fixpoint logic is to add generalized quantifiers, corresponding to undefinable properties, in the sense of Lindström. The paper studies the expressive power of the choice construct proposed in [GH] in its relationships to the logics defined with generalized quantifiers. We show that no extension of fixpoint logic by a set of quantifiers of bounded arity captures all properties of finite structures definable in choice fixpoint logic. Consequently, no extension of fixpoint logic with a finite set of quantifiers is more expressive than the extension of fixpoint logic with choice construct. On the other hand, we give a characterization of choice fixpoint logic by an extension of fixpoint logic with a countable set of quantifiers.

1 Introduction

An interesting open problem in finite model theory is whether there is a reasonable logic which captures exactly those properties of finite structures that are PTIME-computable. It is known that if we consider only ordered structures then fixpoint logic, FP, the extension of first-order logic by means of an inductive operator, is sufficient to solve the problem [Imm,Var]. However, on unordered structures, there are a lot of PTIME properties that are not definable by fixpoint logic. Therefore, it is essential to explore the expressive power of extensions of fixpoint logic by means of various other operations. Such extensions have not only to be strong enough to capture these properties but have also to be computationally feasible. There are several methods to define such extensions in the literature , in particular the Lindström method and the non-deterministic method.

The first one, having its origin in traditional model theory, is a well-established method which enriches a logic by adjoining to it generalized quantifiers corresponding to undefinable properties. Generalized quantifiers were first introduced by Mostowski [Mo] and the general definition for quantifiers was given by Lindström [Lin]. According to Lindström, any property of structures of some fixed finite vocabulary can be taken as the interpretation of a quantifier. For example, the Härtig quantifier $I_{X,Y}(\phi(x),\psi(y))$ is interpreted as "the number of elements satisfying ϕ is equal to the number of elements satisfying ψ". A quantifier Q can bind several variables in one or several formulas. One says that Q is n-ary if it

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 105-125, 1999

binds at most n variables in each formula. The problem of adding generalized quantifiers to fixpoint logic is discussed in [KV] and [Hel]. Usually, fixpoint logic is defined in terms of least fixpoints of formulas in which a relation symbol occurs only positively. However, in the presence of quantifiers that are not necessarily monotone, positivity is no longer a guarantee of monotonicity. Therefore, in considering extensions of inductive logic by means of generalized quantifiers, one uses the inflationary version of fixpoint logic [GS]. The resulting logic is denoted as FP(**Q**). The expressive power of extensions of fixpoint logic with various sets of generalized quantifiers has been studied intensively. Kolaitis and Väänänen showed in [KV] that there is a PTIME-computable property which is not definable in the extension of FP with all unary quantifiers. Hella showed in [Hel] that there is a logical hierarchy between FP and PTIME in the sense that for any integer n there is a PTIME computable property of finite structures which is not definable in FP(**Q**) for any set **Q** of n-ary quantifiers. In particular, this rules out the possibility of capturing PTIME by a logic obtained from fixpoint logic by adding a finite number of quantifiers.

Another method of extension is based on the introduction of non-deterministic constructs. The use of this kind of constructs in logics has a long history going back to the ε-symbol, introduced originally for proof theory by Hilbert. (See [Cai] for a discussion of its relationship to generalized quantifiers.) Roughly speaking, non-deterministic constructs provide the ability to choose, while defining a predicate, just one out of several candidates. The use of non-determinism in finite model theory is an interesting approach to obtain efficient and more expressive logics, as shown in [AB],[AV]. In fact, it is proved there that all PTIME-computable properties of finite structures can be defined by fixpoint logic extended with a choice operator. But, unfortunately, this does not mean that one obtains a logic for PTIME: this non-deterministic extension of fixpoint logic defines also formulas that may have many different interpretations for a given input structure, and it is undecidable whether a formula of this logic defines a unique interpretation on every structure. A more reasonable choice construct has been proposed in [GH] to solve this drawback. This non-deterministic construct, called symmetry-based choice, allows to choose an arbitrary tuple satisfying a given formulas, provided that for each pair of tuples satisfying it there is a definable automorphism mapping the tuples to each other. This restriction guarantees that the semantics of the choice fixpoint logic is deterministic, and all queries definable in it are computable in PTIME. Moreover, it is proved in [GH] that the closure under interpretation of the choice fixpoint logic is strictly stronger than FP+Count, the extension of fixpoint logic with the counting operator, proposed in [Imm]. This extension of fixpoint logic with symmetry-based choice construct seems therefore to be an interesting approach to study the gap between FP+Count and PTIME. In fact, it is still unknown whether there is a PTIME property of finite structures that cannot be defined by this logic.

The goal of this paper is to study the expressivity of the symmetry-based choice construct proposed in [GH] in its relationships to the logics defined with generalized quantifiers[1]. Our main result is that the logic defined with the choice construct in

[1]The relationship between choice operator and generalized quantifiers in the context of general non-determinism is studied in [Cai].

[GH] cannot be captured by any extension of fixpoint logic with a set of quantifiers of bounded arity (consequently, nor with a set of a finite number of quantifiers). This means that the logic of [GH] is not contained in the logical hierarchies defined by Hella [Hel]. Note that by the result of [Hel] a necessary condition for a logic to capture PTIME is to have such a position in these hierarchies. On the contrary, we show that the restriction of choice fixpoint logic to formulas with bounded number of variables can be captured by fixpoint logic extended by a finite number of quantifiers. This result helps us to characterize the choice fixpoint logic by an extension of fixpoint logic with a countable set of quantifiers.

2 Generalized quantifiers and logical reduction operator

In this section, we provide a brief introduction to the notion of generalized quantifiers, as defined by Lindström [Lin], and ealier results concerning the expressive power of generalized quantifiers in the context of finite model theory. We introduce also the notion of logical reduction which plays an important role in both methods of extending logics considered in this paper.

Generalized quantifiers provide the means to assert structural properties of definably interpreted structures. We therefore first define the notion of interpreted structures.

Definition 1. Let us consider a signature $\tau = <R_1,...R_n>$, with R_i of arity r_i and a tuple of formulas $\pi = (\varphi_1,...,\varphi_n)$, where each formula φ_i has a tuple of variables x of arity r_i . Over structures **A** of appropriate signature σ each φ_i defines an r_i-ary predicate $\varphi_i[\mathbf{A}] := \{\bar{a} \in A^{ri}, \mathbf{A} \models \varphi_i(\bar{a})\}$, where A is the universe of **A**. We take π to interpret the following structure of signature τ over A:

$$\pi[\mathbf{A}] := (A,\varphi_1[\mathbf{A}] \ ....,\varphi_n[\mathbf{A}])$$

As usual, the formulas may have other free variables than the ones displayed, these are then regarded as parameters in the interpretation.

A generalized quantifier Q_K is associated with an isomorphism-closed class K of structures that represents the structural property at issue. This quantifier binds formulas which are apt to interpret structures of the signature appropriate for K. Semantically such a quantifier allows to assert membership in K of the interpreted structure. More formally, let K be of the signature $\tau = <R_1,...R_n>$, with R_i of arity r_i. The syntax of any logic L can be extended to allow the construction of formula

$$\psi := Q_K\bar{x}_1,...,\bar{x}_n\ (\varphi_1(\bar{x}_1)....,\varphi_n(\bar{x}_n))$$

with semantics:

$$\mathbf{A} \models \psi \quad \text{iff} \quad \pi[\mathbf{A}] \in K$$

The logic obtained by closing L with the above formula formation is denoted by $L(Q_K)$. For a set of generalized quantifiers **Q**, we write L(**Q**) for the extension of the logic L by all the quantifiers in **Q**. ◊

The problem of adding quantifiers to fixpoint logic is dicussed in length in [KV] and [Hel]. As mentioned in the introduction, in considering extensions of inductive logic by means of generalized quantifiers, one uses the inflationary version of fixpoint logic, the resulting logic is denoted by FP(**Q**).

Let $L_{\infty\omega}$ be the usual infinitary logic which is obtained from first-order logic by allowing conjunction and disjunction over arbitrary sets of formula. Adding a set of quantifiers **Q** to $L_{\infty\omega}$ we obtain the logic $L_{\infty\omega}(\mathbf{Q})$. The logic $L^{\omega}_{\infty\omega}(\mathbf{Q})$ consists of those formulas of $L_{\infty\omega}(\mathbf{Q})$ which contain a finite number of different variables. A straightforward modification of the proof that FP is contained in $L^{\omega}_{\infty\omega}$ shows that for any set of quantifiers **Q**, FP(**Q**) is contained in $L^{\omega}_{\infty\omega}(\mathbf{Q})$.

The arity of a quantifier Q_K associated to a class K of structures of signature τ is $\max\{\text{arity}(R_i) \mid R_i \in \tau\}$. Here are some examples of generalized quantifiers, (a) *Counting quantifiers*: For each natural number n let K_n be the set of all finite structures (A,P) such that $P \subseteq A$ has at least n elements. The counting quantifier Q_{K_n} is usually written more intuitively as $\exists^{\geq n}$. (b) The *Härtig quantifier* is the quantifier which is determined by the class of all structures (A,P,S) such that $P,S \subseteq A$ and $|P|=|S|$.

It is interesting to note that one can associate to each operator T an equivalent set of generalized quantifiers:

Fact 1. Let T: Struct(σ)$\rightarrow$ Struct(Ω) be an isomorphism preserving mapping of structures over σ to structures over Ω, such that for any structure **A** over σ, $\text{dom}(T(\mathbf{A}))\subseteq \text{dom}(\mathbf{A})$. There is a finite set **Q** of quantifiers such that the mapping defined by the operator T is definable by a set of formulas of FO(**Q**).

Proof: Suppose that $\sigma = \{R_1,..,R_n\}$ and $\Omega = \{\lambda_1,..,\lambda_m\}$.
For each $1\leq i\leq m$, we define the class K_i of structures over $\sigma\cup\{\lambda_i\}$ as follows:
For any structure **A** over σ, the structure $<\mathbf{A},S_i>$ belongs to K_i iff S_i contains only one tuple which is a tuple of $T(\mathbf{A})[\lambda_i]$.
Similarly, we define the class K_0 as follows: For any structure **A** over σ, the structure $<\mathbf{A},S_0>$ belongs to K_0 iff S_0 contains only one element which is an element of dom(T(**A**)).
For each $0\leq i\leq m$, let Q_i be the quantifier associated to the class of structures K_i. Let us consider the following formulas ψ_i's of FO(**Q**):

$$\psi_i(X)= Q_{i\ Z_1,..,Z_nY}(R_1(Z_1),..,R_n(Z_n),X=Y) .$$

One can verify that for each structure **A** over σ, $\psi_0[\mathbf{A}]$ defines dom(T(**A**)) and for each $1 \leq i \leq m$, $\psi_i[\mathbf{A}] = T(\mathbf{A})[\lambda_i]$. ◊

The following result is a direct consequence of a result in [CFI] which showed that it does not suffice to add counting to fixpoint logic in order to express all PTIME properties.

Theorem 1. ([CFI]) There is a polynomial time graph property that is not expressible in $L^{\omega}_{\infty\omega}(\mathbf{C})$, where **C** is the set of all unary counting quantifiers.

Hella established the following result, which generalizes Theorem 1.

Theorem 2. ([Hel]) Given any set **Q** of generalized quantifiers of bounded arity, there is a PTIME property of structures that is not definable in $L^{\omega}_{\infty\omega}(\mathbf{Q})$.

There are some connections between results on generalized quantifiers and the notion of logical reduction. The latter means reductions between problems that can be expressed in a logical language. The notion is derived from the idea of interpretations between theories and was used in [Imm], [Daw], [GH]. Let us consider the classes of structures defined in the Definition 1. Each tuple of formulas π can be considered as an *interpretation* of τ in σ, that is, a map from structures over σ to structures over τ. If C is the class of structures over σ defined by the formula ψ in the Definition 1, then it is easily seen that the decision problem in C can be reduced to that in K, the class of structures over τ associated to the quantifiers Q_K. In general, an interpretation may not preserve the domain of structures. The following definition of interpretation and logical reducibility generalizes that of Definition 1.

Definition 2. Let σ and τ be two signatures, where $\tau = \langle R_1,...R_n\rangle$, with R_i of arity r_i. An interpretation of τ in σ is a tuple $\pi = (\varphi_1,...,\varphi_n)$ of formulas over the signature σ such that, for $1 \leq i \leq n$, each φ_i is of arity $k_i r_i$, for some k_i. This interpretation defines a map, π, from structures over σ to structures over τ as follows:

If **A** is a structure over σ, with universe A, then $\pi(\mathbf{A})$ is a structure over τ whose universe is $A^{k1} \cup ... \cup A^{kn}$ and, for $1 \leq i \leq n$,

$$\pi(\mathbf{A})(R_i) := \{(\bar{a}_1,...,\bar{a}_{r_i}) \mid \bar{a}_1,...,\bar{a}_{r_i} \in A^{ki} \text{ and } \mathbf{A} \models \varphi_i(\bar{a}_1,...,\bar{a}_{r_i})\}$$

A class C_1 of structures over σ is said to be L-reducible to a class C_2 of structures over τ, if there is an L-interpretation π of τ in σ such that $\mathbf{A} \in C_1$ iff $\pi(\mathbf{A}) \in C_2$. ◊

(Note that the above definition of interpretation is slightly different from the standard one in which one uses only one parameter k to define the universe of $\pi(\mathbf{A})$,i.e., $k_i = k_j$ for all i,j).

The notion of logical reduction may be used in different ways. On the one hand, as proposed in [Daw], this notion can be used to define for each class C of structures over $<R_1,...R_n>$ a uniform sequence of quantifiers $\{Q_k \mid k \in \omega\}$. Each Q_k of this sequence is the quantifier associated to the class C_k of structures which can be reduced to C by a reduction of size k. More precisely, C_k is the class of structures over $<R'_1,...R'_n>$, where arity(R'_i) of is k.arity(R_i) = k.r_i , such that **A** is in C_k iff the structure $\pi(\mathbf{A})$ is in C, where π is a natural reduction of **A** on the universe of k-tuples of A ;i.e, for each i,

$$\pi(\mathbf{A})(R_i) := \{(\bar{a}_1,...,\bar{a}_{r_i}) \mid \bar{a}_1,...,\bar{a}_{r_i} \in A^k \text{ and } \mathbf{A} \models R_i(\bar{a}_1,...,\bar{a}_{r_i})\} .$$

The notion of a uniform sequence of quantifiers is a natural extension of the notion of a generalized quantifier associated with a class of structures which overcomes the limitations of collections of quantifiers of bounded arity.

On the other hand, logical reduction can be considered as a natural method to extend a logic, as shown in [GH]. This method enriches a logic by the reductions which are definable by the logic itself. More precisely, the extension is based on the introduction of a so-called logical reduction operator, denoted by I (for *interpretation*), and is defined as follows.

Definition 3. [GH] Let L be a logic, σ, τ be two signatures, θ be an L-sentence over τ , and $\pi = (\varphi_1,...,\varphi_n)$ be an interpretation of τ in σ, defined as in Definition 2, such that $\varphi_1,...,\varphi_n$ are L-formulas. The syntax of L can be extended to allow the construction of formulas of the form

$$\psi := I\, x_1,...,x_n\, (\varphi_1(x_1)....,\varphi_n(x_n)\, ;\, \theta)$$

whose semantics is defined as follows:

$$\mathbf{A} \models \psi \quad \text{iff} \quad \pi(\mathbf{A}) \models \theta$$

The logic obtained by closing L with the above formula formation is denoted by L + I.

It should be noted that the closure under logical reduction is more general than the closure under substitution defined in [Ebb]. In fact, substitution is a reduction with size 1. As from an observation in [Kry], given a class of structures C, $L(Q_C)$+ I captures the extension of L with the uniform sequence of quantifiers associated to the class C defined above, while the closure of L(C) under substitution is L(C) itself if L is regular. The logical reduction operator also plays a significant role in the context of logics with choice construct, as we will see in the next section.

3 Extension with the choice construct

In this section, we review the basic concepts of the method of logical extension with the choice construct proposed in [GH].

As presented in the introduction, the use of non-determinism in finite model theory is an interesting approach to obtain more expressive logics. Indeed, one of the main drawbacks of traditional logics is the fact that they cannot distinguish between equivalent tuples of a structure, i.e. tuples that agree on all first-order formulas and therefore are logically inseparable. This aspect of traditional logics seems to strongly limit their expressive power. For example, evenness is a query that cannot be defined by fixpoint logic, although it is easily computed, simply by enumerating the elements of the structure while keeping a binary counter. Non-determinism offers a natural way to define efficiently such properties of structures, using a choice construct. In [AV] this construct is defined as follows: let $\Phi(x)$ be a formula, then $W_x\Phi(x)$ is a formula whose semantics on a structure **B** is a set of interpretations each of which corresponds to a tuple of $\Phi[\mathbf{B}]$. Roughly speaking, the choice construct applied to a formula chooses an arbitrary element among those of the structure that satisfy the formula, i.e., it distinguishes between this element and the others. It is therefore possible using fixpoint together with the choice construct to define an order on a structure. In [GH], this choice mechanism is integrated to the inflationary fixpoint operator to get an operator, called inflationary choice fixpoint and denoted by IFP_c , as follows:

Definition 4. [GH] Let $\Psi(\bar{x},S,T),\Phi(\bar{y},S,T)$ be FO-formulas over $\sigma \cup \{S,T\}$, such that the arities of S and T match the free variables $\bar{x}$ and $\bar{y}$ respectively. The formula

$$IFP_c[S,T](\Psi,\Phi)$$

defines on each input structure **A** over σ a relation which is the limit S_ω of a sequence of relations computed as follows:

$$S_0 = T_0 = \varnothing ,$$
For $i > 0$,
$$T_{i+1} = \text{choice}(\Phi_i) \ , S_{i+1} = S_i \cup \Psi_i ,$$

where $\Phi_i = \Phi[\mathbf{A}_i]$, $\Psi_i = \Psi[\mathbf{A}_i]$, $\mathbf{A}_i$ is the structure **A** expanded with S_i and T_i, and choice(R) is an operator that returns an arbitrary tuple of R. ◊

In general, a formula of $FO+IFP_c$ has many interpretations on a given structure and therefore cannot be used to define a property of structures in the traditional sense. A more reasonable non-determinism is used in [GH] to overcome this drawback. The choice mechanism is used only to distinguish between equivalent elements of a structure, i.e., elements that are interchangeable by an automorphism of the structure. Evidently, computations using this choice mechanism are usually unfeasible because they involve automorphism tests between elements of a structure.

However, a low complexity choice mechanism can be defined by including as a witness, in the choice operator, a formula which defines the required automorphism. So the choice is triggered on a set of elements only if the mappings defined by the formula are the automorphisms of the structure that interchange the elements of this set. Formally, given a structure **B** and two relations R_Φ and R_F such that the arity of R_Φ is m and the arity of R_F is 2m+2, we define a function choice($\mathbf{B}$,R_Φ,R_F) as follows:

> For any two m-tuples $\bar{u},\bar{v}$, let us denote by $F_{\bar{u},\bar{v}}$ the binary relation containing the 2-tuples (x,y) such that $R_F(\bar{u},\bar{v},x,y)$. The function choice($\mathbf{B}$,R_Φ,R_F) returns an arbitrary m-tuple of R_Φ, if for any m-tuples $\bar{a}$ and $\bar{b}$ in R_Φ, the mapping whose graph is $F_{\bar{a}\bar{b}}$ is an automorphism of **B** that maps $\bar{a}$ to $\bar{b}$, otherwise the function returns empty.

This symmetry-based choice construct is feasible because the test whether a mapping is an automorphism is realisable in polynomial time.

The inflationary choice fixpoint operator using this specified symmetry-based choice construct is denoted by $\mathrm{IFP}^{f}_{c,s}$. It is defined in the same way as IFP_c in Definition 4, except that the function choice(Φ_i) is replaced by choice($\mathbf{A}_i$,Φ_i,F_i).

Definition 5. [GH] Let $\Psi(\bar{x},S,T),\Phi(\bar{y},S,T),F(w,S,T)$ be formulas over $\sigma \cup \{S,T\}$, such that arity(S) = $|\bar{x}|$, arity(T) = $|\bar{y}|$ and $|w|=2|\bar{y}|+2$. The formula

$$\varphi(\bar{x}) := \mathrm{IFP}^{f}_{c,s}[S,T](\Psi,\Phi, F) \qquad (1)$$

defines on each input structure **A** over σ a relation which is the limit S_ω of a sequence of relations computed as follows:

$$S_0 = T_0 = \varnothing ,$$
For $i \geq 0$,
$$T_{i+1} = \mathrm{choice}(\mathbf{A}_i,\Phi_i,F_i) , S_{i+1} = S_i \cup \Psi_i ,$$

where $\mathbf{A}_i$ is the structure **A** expanded with S_i and T_i , $\Phi_i = \Phi[\mathbf{A}_i]$, $\Psi_i = \Psi[\mathbf{A}_i]$ and $F_i = F[\mathbf{A}_i]$. ◊

The following proposition states that the non-determinism implied by $\mathrm{IFP}^{f}_{c,s}$ is sound, i.e. all of its different computations are isomorphic.

Proposition 1. [GH] Let $\{\mathbf{A}_0,\mathbf{A}_1,...\}$ and $\{\mathbf{A}_0',\mathbf{A}_1',...\}$ be any two sequences of structures, defined by Definition 5, corresponding to two different computations of $\mathrm{IFP}^{f}_{c,s}[S,T](\Psi,\Phi,F)$ on an input structure **A**, then for all $i \geq 0$, $\mathbf{A}_i$ is isomorphic to $\mathbf{A}_i'$.
A boolean choice formula is a formula of the form:

$$Q_1x_1..Q_mx_m\varphi(x_1,..,x_m), \qquad (2)$$

where Q_i is a quantifier $\forall$ or $\exists$ and φ is a choice formula of the form (1) in Definition 5. The following corollary is a direct consequence of Proposition 1.

Corollary 1. Any boolean choice formula is *deterministic*, that is, it has always an unique truth value on every structure.

Any logic L can be extended with choice boolean formulas of the form (2) which, by Corollary 1, assure the determinism of the extended logic. We denote by choice-IFP this extension of IFP, i.e. the logic obtained by adding the construction of formula of the form (2) in the rules of formula construction of IFP.

Theorem 3. [GH] There is a PTIME-property of structures that is expressible by choice-IFP but not by IFP+Count.

Closure under Interpretation

The semantics of a symmetry choice construct in a formula depends on the automorphism of the structure on which the formula is evaluated (and its expansions). This semantics limits, however, the expressivity of the choice-IFP logic. For example, IFP+Count is not contained in choice-IFP. In fact, one can notice that on rigid structures choice-IFP collapses to IFP and there are classes of rigid graphs on which IFP collapses to quantifier free formulas, but IFP+Count can define a linear order and hence capture PTIME.

It is interesting therefore to define a logic in which the choice operator may refer to the automorphisms of an interpreted structure. Such a logic can be obtained by closing choice-IFP with interpretation. This closure, defined in Definition 3 of Section 2, is denoted by choice-IFP + I. In fact, in a choice-IFP+I formula of the form $I(\varphi_1,..,\varphi_n;\theta)$, where θ is a choice-IFP formula, the choice operator in θ does not refer to the automorphisms of the input structure **A** but to that of the interpreted structure $\pi(\mathbf{A})$ on which θ is evaluated. Unlike choice-IFP, the closured logic choice-IFP+I does not collapse to IFP on rigid structures. Moreover, it is shown that the closure under interpretation of choice-IFP is strictly more expressive than FP+Count.

Theorem 4. [GH] FP+Count $\subset$ choice-IFP + I $\subseteq$ PTIME

Remark. One may think to use, as an alternative to the closure under interpretation, choice formulas with explicit relations defining the structure on which the automorphism is considered. E.g., use a formula of the form $IFP^{f}_{\mathcal{L},s}(\chi;\Psi,\Phi, F)$ and require that the choice operator refers to the automorphism of the structure with the relations defined by $\chi(\mathbf{A})$. However, the semantics of this choice formula is not deterministic. This is because, while the condition of symmetry is defined w.r.t the structure $\chi(\mathbf{A})$, the computation of the formulas Ψ,Φ, F is always defined w.r.t the input structure **A**. On the contrary, with closure under interpretation, the

computation and the automorphism refer to the same structure (the interpreted structure), one can assure the determinism of the semantics.

4 Main result

In this section, we show our main results concerning the relationship in expressive power between fixpoint logics extended with generalized quantifiers and the choice fixpoint logic defined in the previous section.

Theorem 5. For each natural number n, let $\mathbf{Q}_n$ be the set of all n-ary generalized quantifiers, then there is a PTIME-computable property of finite structures that is expressible in the logic choice-IFP , but not in FP($\mathbf{Q}_n$).

The following corollary is an immediate consequence of the above theorem.

Corollary 2. There is no finite set **Q** of generalized quantifiers such that FP(**Q**) is more expressive than choice-IFP.

To show Theorem 5, we will use the class of structures defined by Hella to prove Theorem 2. We first present a brief description of this class of structures , the details can be found in [Hel].

4.1 Construction [Hel]:

Let G = (V,E) be a connected (undirected) graph such that the degree of each vertex of G is n+1. One constructs a structure G_S from G by replacing each vertex u of G by a structure C(u) defined as follows:

Let C = $\{c_0,...c_n\} \cup \{d_0,...d_n\}$ where $c_0,...c_n$,$d_0,...d_n$ are all distinct, and let ~ be the following equivalence relation of C:

$$x \sim y \text{ iff there is i such that } x, y \in \{c_i, d_i\}$$

So each pair $\{c_i, d_i\}$, $0 \le i \le n$, is an equivalence class.

We define two (n+1)-ary relations R^+ and R^- of C as follows:

- R^+ contains the tuple $(a_0,..,a_n)$ iff for all $i \ne j$, a_i and a_j are not ~-equivalent and there is an even number of d_i 's among the a_i's ,
- R^- contains the tuple $(a_0,..,a_n)$ iff for all $i \ne j$, a_i and a_j are not ~-equivalent and there is an odd number of d_i 's among the a_i's ,

We denote $\mathbf{C}^+$and $\mathbf{C}^-$ the structures (C,R^+) and (C,R^-), respectively. Clearly, if f is an automorphism of $\mathbf{C}^+$ or $\mathbf{C}^-$ or an isomorphism between these two structures, then it must preserve the equivalence relation ~:

$$a \sim b \Leftrightarrow f(a) \sim f(b)$$

The following lemma describes such automorphisms and isomorphisms:

Lemma 1. [Hel] Let f: C $\rightarrow$ C be a bijection that preserves the relation $\sim$. Then:

- f is an automorphism of $\mathbf{C}^+$ and $\mathbf{C}^-$ iff the number $|\{i\leq n \mid \exists\, j \leq n\ (f(c_i)=d_j)\}|$ of the c,d-exchanges of f is even;
- f is an isomorphism between $\mathbf{C}^+$ and $\mathbf{C}^-$ iff this number is odd.

Now, for each subset S of V we define a structure G_S as follows:

- each vertex u of G is associated to a copy of $\mathbf{C}^+$ or $\mathbf{C}^-$, denoted by C(u), depending on whether u is in S or not;
- each edge (u,v) of G is replaced by two edges linking , for some specific i,j, the component c_i (resp. d_i) of C(u) to the component c_j (resp. d_j) of C(v).

More precisely, let h be a function enumerating, for each vertex u of G, the edges adjacent to u, i.e. h(u,v)= i if (u,v) is the i-th edge adjacent to u. G_S is a structure whose signature contains the relation symbols R, of arity n+1, and E of arity 2 such that:

- the universe of G_S is V$\times$ C ,
- the relation R of G_S is the set of n+1-tuples $((u,a_0),..,(u,a_n))$ of $(V\times C)^{n+1}$ such that if u$\in$ S then $(a_0,..,a_n)\in R^-$, else $(a_0,..,a_n)\in R^+$
- the relation E of G_S is the set of pairs $((u,c_i),(v,c_j))$ and $((u,d_i),(v,d_j))$ of $(V\times C)^2$ such that (u,v) is an edge of G and i=h(u,v) and j=h(v,u).

The main property of the structures G_S 's is stated in the following lemma [Hel]:

Lemma 2. Let S,T be any two subsets of V. The structures G_S and G_T are isomorphic iff S and T have the same parity.

Remark that, by the above construction, if S and T have the same parity then the numbers of copies of $\mathbf{C}^+$ in G_S and in G_T also have the same parity.

For each vertex v of V and 0$\leq$ j <n, we will call $\{(v,c_j),(v,d_j)\}$ a pair and we will denote it by p(v,i). We say that a pair p(v,j) is connected to a pair p(u,i) if their c,d- components are related by the relation E, that is, if we have $E((u,c_i),(v,c_j))$ and $E((u,d_i),(v,d_j))$. Note that for each pair there is exactly one pair connected to it.

In what follows we suppose that there is a linear order < on the vertices of the initial graph G. This implies an order $<_p$ on the set of pairs as follows:

$$p(v,i) <_p p(w,j) \text{ iff } v<w \text{ or } (v=w \text{ and } i<j)$$

Let u be any vertex of G. We define the two following structures:

$$A_G = (G_\varnothing ,<) ,$$
$$B_G = (G_{\{u\}},<)$$

By Lemma 2, it follows that the structures A_G and B_G are not isomorphic. Furthermore, we can distinguish between them (in polynomial time) by counting the number of copies of $\mathbf{C}^+$ they contain.

Let us consider the class of structures of the type A_G or B_G for each graph G. The problem whether a structure of this class is of the type A_G (or B_G) is decidable in polynomial time. However, it is shown in [Hel] , using the technique of k-pebbles game adapted to logics with generalized quantifiers, that for each k>n there is a graph G such that A_G and B_G are equivalent on all formulas of $L^k_{\infty\omega}(\mathbf{Q}_n)$.

4.2 Proof of theorem 5

To prove that there is a PTIME computable property of finite structures that is expressible in choice-IFP but not in FP($\mathbf{Q}_n$), it suffice to show that the property of structures defined above is expressible by a formula of choice-IFP.

Let G be the connected graph used in the construction of the above structures A_G and B_G . Let D_G be a structure of the type A_G or B_G . The main idea of the proof is to use the symmetry-based choice mechanism of choice-IFP to generate an order on D_G and then use the order to express all PTIME properties. This will be done by using particular symmetries of the structure D_G. So we first focus on such automorphisms of this structure.

** Automorphisms of D_G.*

Consider a set of structures $C(v_0),...,C(v_r)$ of D_G , such that $v_0,..,v_r$ form a cycle on the graph G, that is , (v_m,v_{m+1}) , for $0\leq m <r$, and (v_r,v_0) are edges of G. Let $f:V\times C\rightarrow V\times C$ be the bijection that exchanges the components c and d corresponding to the edges that link the structures $C(v_m),C(v_{m+1})$, for $0\leq m <r$, and $C(v_0),C(v_r)$, that is:

- $f(v_m,c_l) = (v_m,d_l)$ and $f(v_m,d_l) = (v_m,c_l)$

  for : ($0<m<r$ and $l = h(v_m,v_{m+1})$ or $l = h(v_m,v_{m-1})$)
  or ($m = r$ and $l = h(v_r,v_0)$ or $l = h(v_{r-1},v_r)$)
  or ($m = 0$ and $l = h(v_0,v_r)$ or $l = h(v_0,,v_1)$)

- The other elements of $V \times C$ are invariant for f.

For each vertex w of the above cycle, let f_w: $C\rightarrow C$ be the "restriction" of f to w, that is: $f_w(a)=b$ iff $f(w,a) = (w,b)$. It can be seen that the number of c,d-exchanges of f_w is two. It follows from Lemma 1 that, for each vertex w of the above cycle, f_w is an automorphism of $\mathbf{C}^+$ or $\mathbf{C}^-$. Moreover, f preserves the relation E and the order $<_p$ of D_G , f is therefore an automorphism of D_G .

** Generation of an order on D_G by choice-IFP*

As there is an order $<_p$ between the pairs of D_G, the main problem is to order the c,d-components of each such pair. We generate such an order by iterating the following three steps:

(During this computation, we say that a pair p is ordered if its c,d-components are ordered.)

Step 1: Compute an unordered pair p(v,j) and an automorphism f of the structure that exchanges the components (v,c_j),(v,d_j) of this pair.

Step 2: Choose one of the components of p(v,j) (say (v,c_j))

Step 3: Order the c,d-components of this pair p(v,j) by $(v,c_j) < (v,d_j)$ and propagate the ordering as follows:

For each unordered p(u,i):

a- If the pair p(w,l) connected to it is ordered, then order p(u,i) as follows:
If $(w,c_l)<(w,d_l)$ and $E((u,c_i),(w,c_l))$ and $E((u,d_i),(w,d_l))$ then $(u,c_i)<(u,d_i)$;

Otherwise,

b- If the n other pairs in the structure C(u) are ordered, then order p(u,i) in the way shown in Lemma 3 below.

We give below a detailed description of the above three steps.

Description of Step 2

The choice operation in Step 2 is a symmetry-based choice: it uses the choice set and the automorphism, computed by the formulas Φ and F given in Step 1 (see below the description of step 1).

Description of Step 3

Steps 3 iterates the two steps 3-a and 3-b each of which is FO-definable. In fact, the ordering in Step 3-a is clearly FO-definable, and as shown in Lemma 3 below the ordering in Step 3-b is FO-definable too.

Lemma 3. If in a structure C(v) there are n pairs ordered then the c,d-components of the (n+1)-th pair are distinguishable and can be ordered by a FO-formula.

Proof: Suppose that the pairs p(v,0),...,p(v,n-1) of C(v) are ordered. This implies a linear order on the set X of all n-tuples of the form $((v,a_0),...,(v,a_{n-1}))$, where $a_i \in C \setminus \{c_n,d_n\}$. Let $((v,x_0),...,(v,x_{n-1}))$ be the first tuple of X such that, for some $x_n \in \{c_n,d_n\}$, $((v,x_0),...,(v,x_{n-1}),(v,x_n))$ is in the relation R of C(v). Note that, by the definition of R, the value of x_n is unique. So, we can order the c,d-components of the (n+1)-th pair as follows: $(v,c_n)<(v,d_n)$ if $x_n=c_n$ and $(v,d_n) < (v,c_n)$ if $x_n=d_n$. Clearly such an ordering is FO-definable. ◊

So we have the following lemma:

Lemma 4. There is a fixpoint formula Ψ defining the ordering in Step 3.

Description of Step 1

This step will compute a pair p(v,j) participating in a cycle of D_G such that the pairs connecting the structures C(u)'s of this cycle are all unordered. As observed above, there is an automorphism of D_G that exchanges the c,d- components of p(v,j). This cycle is computed as follows:

Let p_1 be the first pair (w.r.t the order $<_p$) that is unordered (until the pairs in D_G are not all ordered, there is always such a pair). We construct now a directed path containing only unordered pairs of D_G by the following rules:

+ p_1 is the first pair of the path;

+ if a pair p_m of a structure C(u) is in the path and has not yet a successor then:
  a- if the pair p_{m+1} connected to p_m is not yet in the path, then it is added to the path as successor of p_m ;

otherwise
  b- if there are no other pairs of C(u) that are in the path, then the first (according to $<_p$) unordered pair of C(u) that is not in the path is added to the path as the successor of p_m.

(thus if a) and b) are not satisfied then p_m is the last pair of the path).

It is clear that at each iteration of the above rules only one new pair is added to the path. The construction of the path needs at most n iterations of the above rules, where n is the number of pairs in D_G. Furthermore, as the ordering in Step 3-a assures that if a pair is unordered then so is the pair connected to it , one can see that the so constructed path contains only unordered pairs. On the other hand, following the ordering in Step 3-b, if a structure C(u) contains an unordered pair, it must contain at least two such pairs. This implies that if the condition of the above rule b) is satisfied, then p_m has a successor and is not the last pair of the path. So a pair p_m of the path is the last pair if the conditions of the rules a) and b) are not satisfied. It is easily seen that this occurs when the path returns to a structure C(u) through which it has already gone. Let p_m be the last pair and p_r be the pair of the path that belongs to the same structure C(u) as p_m. One can verify that the structures C(u) of D_G that contain the pairs of the path from p_r to p_m form the cycle that we require. Now the pair computed by Step 1 is the first pair (according to the order $<_p$) of the cycle. The automorphism computed by Step 1 is the mapping that exchanges the c,d-components of each pair participating in the connection of the cycle.

Following the description of the computation of this step, one can see that this pair and the automorphism are definable by a fixpoint formula. So we have the following lemma:

Lemma 5. There are fixpoint formulas Φ and F that define respectively the pair and the automorphisms computed in Step 1.

Finally, the ordering on D_G obtained by iterating the three steps 1,2 and 3 is defined by the formula

$$\theta : = \text{IFP}^{f}_{\mathcal{L},s}[<,T](\Psi,\Phi, F)$$

where Ψ,Φ and F are the formulas of Lemmas 4 and 5, respectively.

It is clear that the computation that iterates the above three steps terminates when all of the pairs of the structure are ordered. So, at the end of the choice fixpoint computation of the above formula we have a linear order on the structure D_G .

We have shown that there is a formula of choice-IFP defining an order < on the structure D_G . Using the fact that the property separating the structure A_G and B_G is in PTIME, one knows that there is a fixpoint formula $\mu(<)$, referring to this order, that defines this PTIME property. Therefore, the computation of this property involves at first the computation of the order by the above formula θ and then the computation of the formula μ. The whole computation can also be defined in choice-IFP. In fact, one has just to include a control mechanism that makes the computation to switch to that of μ when the computation of < terminates. For example, we can replace Ψ by the following formula Ψ^*:

$$\Psi^*(Z_1,Z_2) = (Z_2=0 \wedge \Psi(Z_1) \wedge \neg\text{End-order}) \vee (Z_1=1 \wedge \text{End-order} \wedge \mu(Z_2)) ,$$

where End-order = $\forall x,y\ (\Psi(x,y) \rightarrow x<y)$ is a predicate which is true only when no new order is generated by Ψ.

The choice-IFP formula that computes the property then has the following form:

$$\theta^* : = \text{IFP}^{f}_{\mathcal{L},s}(\Psi^*,\Phi, F) \qquad \Diamond$$

Remark. 1**-** The order computed in the above proof is the result of an iteration in which at each step one realizes first the choice computation (defined with the formulas Φ and F) and then based on the result of this choice one computes the corresponding order (supposed to be defined by the formula Ψ). This computation does not correspond exactly to the scheme of computation of a choice-IFP formula in Definition 5 in which the choice step and the inductive step are realized simultaneously at each iteration. However, one can solve this mismatch by including in the above formulas Ψ and Φ a control mechanism that helps to alternate the triggers of the order computation and the choice computation.
2- The proof makes an essential use of the assumed linear order on the graph G. Without it (and the pre-order it generates on A_G and B_G) the proof would not go through. On the other hand, the order is not needed for proving the result in [Hel]. This raises an interesting question whether A_G and B_G without the order could be separated by choice-IFP.

5 Lindström characterization of the choice construct

The previous section shows that the whole choice fixpoint logic cannot be captured by a finite number of quantifiers. In this section, we show on the contrary that, for each integer k, the restriction of choice fixpoint logic to formulas containing at most k variables can be captured by an extension of fixpoint logic with finitely many quantifiers (Theorem 7). This result allows to characterize the whole choice fixpoint logic by an extension of fixpoint logic with a countable set of generalized quantifiers (Theorem 8).

To be more precise , the *fixpoint* part of this characterization is the following FP*. With any finite structure **A** over σ associate the two-sorted structure **A*** extending the domain of **A** with the set N_A= {1,...., |A|}. That is, **A*** is a structure over σ ∪{<} such that dom(**A***)=A ∪ N_A, **A***(σ) coincides with **A** and **A***(<) is the natural ordering on N_A.

Definition 6. FP* is just full fixpoint logic for the two-sorted variants **A*** of structures **A**. We allow first-order quantification on both sorts, and fixpoint construction over mixed-sorted predicate variables. FP*(**Q**) is the extension of FP* with a set **Q** of quantifiers. ◊

The notion of k-equivalence of tuples plays an important role in our characterization. Recall that two k-tuples of a structure are k-equivalent if they satisfy exactly the same FO-formulas with k variables. It is known that the k-equivalence relation is FP-definable, and there is an FP-formula that defines an order on the k-equivalence classes of a structure. Moreover, a fixpoint computation on a structure **A** can be reduced to a fixpoint computation on the k-equivalence classes of **A**. This is the meaning of the following normal form theorem of fixpoint logic:

Theorem 6. [AV2] For each FP-formula with k variables φ over a signature Ω, there is a formula $\bar{\varphi}$ over a signature $\bar{\Omega}$, called the k-quotient of Ω, such that for any structure **A** over Ω, one can associate a structure $\bar{\mathbf{A}}$ over $\bar{\Omega}$, called the k-quotient of **A**, whose domain is the set of k-equivalence classes of **A**, and satisfying the following property:

For any k-tuple ā of **A**, **A**|= φ(ā) iff $\bar{\mathbf{A}}$ |= $\bar{\varphi}$ ([ā]) , where [ā] is the k-equivalence class of ā. ◊

In the case of fixpoint with choice, the computation cannot be reduced to the one on some fixed k-equivalence classes of the input structure, since the choice operator splits these classes. However, one can consider the computation between two consecutive choices as a computation on the k-equivalence classes of the intermediate structure. Roughly speaking, we will define for each k a generalized quantifier that simulates this form of computation of choice fixpoint formulas with k variables.

The second sort in FP* is used essentially to define the k-quotient of the intermediate structures. In fact, as there is an order on the k-equivalence classes of a

structure **A**, we can represent each k-equivalence class by its rank with respect to this order. As the number of k-equivalence classes is bounded by $|A|^k$, we will therefore represent the m^{th} k-equivalence class by the k-tuple of $N_A$of rank m. For the sake of simplicity, in what follows we will denote a k-tuple of N_A of rank m by the integer m. Moreover, we will suppose that for each formula φ of k variables, its normal form $\bar{\varphi}$ according to Theorem 6 is a formula whose variables take as values the integers m, i.e. the k-tuples of N_A.

The Lindström characterization uses a generalized quantifier whose definition is based on the operators defined below.

Let choice-IFP^k be the set of choice-IFP formulas containing at most k variables and IFP$^{f}_{\ell,s}$[S,T](Ψ,Φ, F) be a formula of choice-IFP^k. We first consider the computation of this formula at each step in terms of the k-equivalence classes of the intermediate structures $\mathbf{A}_i$. At each step i, we associate, for example, to the relation S_i computed by Ψ a k-ary relation S_i* on the second sort computed by the formula $\bar{\Psi}$, the normal form of Ψ,

S_i*(m) iff there is some tuple ā such that Ψ(ā) holds and [ā] is the m^{th} k-equivalence class of the structure $\mathbf{A}_i$.

Alternatively, following the normal form theorem 6, we obtain:

S_i* = { m | $\bar{A}_i$ |= $\bar{\Psi}$([ā]) where [ā] is the element of rank m in $\bar{A}_i$ }

To describe the computation from step 0 to step i, we define the following 2k-ary relation R^i_Ψ on the second sort:

$$R^i_\Psi = \{(j,m) \mid 0 \leq j \leq i \text{ and } S_j^*(m)\}$$

We define the similar relations R^i_Φ and R^i_F for the formulas Φ and F, respectively.

Remark: If a formula has less than k free variables, one can add to it dummy free variables to obtain an equivalent formula having exactly k free variables. So, when defining the above relations, we assume that each of the formulas Ψ,Φ, F has exactly k free variables. We will consider any m-tuple $(a_1,..,a_m)$, with m < k, as a k-tuple in which the last element a_m is repeated (k-m) times. This k-tuple will be denoted by $(a_1,..,a_m)\uparrow_k$.
The following operator P_k defines a computation similar to that of the choice fixpoint operator, using three relations playing the role of the above R^i_Ψ ,R^i_Φ and R^i_F :

Definition 7. Let **A** be a structure over σ and **A*** be the two-sorted variants **A**. For each k, let R_1, R_2 , R_3 be three 2k-ary relations on the second sort of **A***, and S,T be two relation symbols not in σ, having arity respectively k and m, where m = | (k-2) /2 |. The operator:

$$P_{k\sigma}(\mathbf{A}^*, R_1, R_2, R_3)$$

defines a structure J_ω which is the limit of a sequence of structures $J_i = (\mathbf{A},S_i,T_i)$ computed as follows:

$S_0 = T_0 = \varnothing$,

For $i \geq 0$, $T_{i+1} = \text{choice}(J_i,\Phi_i,F_i)$, $S_{i+1} = S_i \cup \Psi_i$,

where $\Phi_i = \{(a_1,..,a_m) \mid [(a_1,..,a_m)\!\upharpoonright_k]$ is the n-th k-equivalence class of J_i and $R_2(i,n)\}$,

$F_i = \{\bar{a} \mid [\bar{a}]$ is the n-th k-equivalence class of J_i and $R_3(i,n)\}$,

$\Psi_i = \{\bar{a} \mid [\bar{a}]$ is the m-th k-equivalence class of J_i and $R_1(i,m)\}$ ◊

The next lemma follows immediately from the definitions of the relations R^i_Ψ ,R^i_Φ and R^i_F and from Definition 7.

Lemma 6. Let $\mathbf{A}_i$ be the structure defined at step i in the computation of a formula $\text{IFP}_{c,s,f}[S,T](\Psi,\Phi, F)$ of choice-IFP_k on a structure $\mathbf{A}$, then

$$\mathbf{A}_i = P_{k\sigma}(\mathbf{A}^*, R^i_\Psi, R^i_\Phi, R^i_F).$$

Suppose that in the Definition 7 , we have $J_\omega = J_m$, so the limit structure is J_m . We associate to $P_{k\sigma}$ an operators $\bar{P}_{k\sigma}$ defined as follows:

$$\bar{P}_{k\sigma}(\mathbf{A}^*, R_1, R_2, R_3) = \bar{J}_m$$

where $\bar{J}_m$ is the k-quotient structure of J_m .

Remark. By definition, the operator $P_{k\sigma}$ is non-deterministic: it may associate to each input structure $\mathbf{A}$ different output structures. However, these output structures are isomorphic and therefore have the same canonical k-quotient structure (whose domain is the second sort). The operators $\bar{P}_{k\sigma}$ aims precisely to define this canonical structure. They are deterministic and one can therefore define generalized quantifiers to simulate them.

Let $\mathbf{Q}_{k\sigma}$ be the sets of quantifiers associated to the operators $\bar{P}_{k\sigma}$ (see Fact 1 of Section 2) and $\mathbf{Q}_\omega$ be the union of all $\mathbf{Q}_{k\sigma}$ for $k \in \omega$ and all σ. Let choice-$\text{IFP}^k(\sigma)$ be the set of formulas of choice-IFP^k over a vocabulary σ .

Theorem 7. If a property of structures is definable by choice-$\text{IFP}^k(\sigma)$ then it is definable by $\text{FP}^*(\mathbf{Q}_{k\sigma})$.

Proof: We show that each formula $\text{IFP}^f_{c,s}[S,T](\Psi,\Phi, F)$, where Ψ,Φ, F are fixpoint formulas of k variables over $\sigma \cup \{S,T\}$, can be simulated by a formula Γ of

FP*($\mathbf{Q}_{k\sigma}$). This formula Γ describes the computation of the choice formula on a structure $\mathbf{A}$ as follows:

Let $\mathbf{A} = \mathbf{A}_0$ be the input structure. For each i≥0, let $\mathbf{A}_i$ be the structure defined by IFP$^{f_{\mathcal{C},S}}$[S,T] (Ψ,Φ, F) at the step i. Let $\bar{\Psi}$,$\bar{\Phi}$, $\bar{F}$ be the normal forms of Ψ,Φ, F, respectively. For each step i, Γ defines the following structures and relations: M_i, $R_{1,i}$, $R_{2,i}$, $R_{3,i}$.

For i=0, M_0 is the k-quotient structure of the input $\mathbf{A}$, which is defined by a usual fixpoint formula. $R_{1,0}$,$R_{2,0}$,$R_{3,0}$ are equal to $\varnothing$.

For i ≥ 0, Γ defines M_{i+1} from M_i as follows:

Γ first defines $R_{1,i}$,$R_{2,i}$,$R_{3,i}$ by:

$R_{1,i} := R_{1,i-1} \cup \{(i,m) \mid m \in \bar{\Psi}(M_i)\}$

$R_{2,i} := R_{2,i-1} \cup \{(i,m) \mid m \in \bar{F}(M_i)\}$

$R_{3,i} := R_{3,i-1} \cup \{(i,m) \mid m \in \bar{\Phi}(M_i)\}$

Using the operator $\bar{P}_{k\sigma}$ (or, equivalently, the quantifiers of $\mathbf{Q}_{k\sigma}$), Γ defines then

$M_{i+1} := \bar{P}_{k\sigma}(\mathbf{A}^*, R_{1,i}, R_{2,i}, R_{3,i})$

It is easily seen that for each i, $R_{1,i}$,$R_{2,i}$,$R_{3,i}$ coincide with R^i_{Ψ}, R^i_{Φ}, R^i_F, respectively.

So, by Lemma 6, for each i, the structures M_i defined by formula Γ are respectively the k-quotient of the structures $\mathbf{A}_i$ defined by the formula IFP$^{f_{\mathcal{C},S}}$[S,T](Ψ,Φ, F) at step i. This implies that, for the boolean case, the two formulas give the same truth value. Moreover, it is easy to verify that the formula Γ that describes the computation in question is in FP*($\mathbf{Q}_{k\sigma}$). ◊

The converse of the above theorem is probably not true. In fact, although the definition of the quantifiers of $\mathbf{Q}_{\omega}$ is based on the operators $P_{k\sigma}$ (Definition 7) whose computation is similar to that of a choice fixpoint formula, there are two reasons for which FP*($\mathbf{Q}_{\omega}$) is more powerful than choice-IFP. Firstly, the second sort is not available in choice-IFP. Secondly, the choice process in choice-IFP is applied only to the input structure, while in FP*($\mathbf{Q}_{\omega}$) it can be applied to any definably interpreted structure. However, a complete characterization can be obtained for the full logics, i.e., the closure of choice-IFP and FP*($\mathbf{Q}_{\omega}$) under interpretation.

Theorem 8. A property of structures is definable by choice-IFP + I if and only if it is definable by FP*($\mathbf{Q}_{\omega}$) + I .

Proof: The "only if" part is immediate from the above theorem.
To show the "if" part we will show that using the reduction we can simulate by a choice formula the second sort and the quantifiers of FP*($\mathbf{Q}_\omega$).

The simulation of the second sort is as follows:

Let Γ be a formula of FP*($\mathbf{Q}_\omega$), and let k be an integer such that var(Γ) < k where var(Γ) is the number of variables in Γ, and for any quantifiers of $\mathbf{Q}_m$, $\mathbf{Q}'_m$ occurs in Γ, m < k . For any input structure **A** over $\{R_1,...,R_n\}$we can define a structure **A**' = **A**$\cup$ R, where R is a unary relation whose elements are k-tuples of A^k, by the following reduction:

$$\mathbf{A}' = \pi(\mathbf{A})$$

with $\pi = <\varphi_k , R_1,...,R_n>$, where φ_k is a formula defining the set of k-tuples of a structure.

Using choice construct on the structure **A**', one can define a total order on the elements of R. This ordered set can be used to simulate the second sort of **A***.
For the simulation of the quantifiers of $\mathbf{Q}_{m\sigma}$, $\mathbf{Q}'_{m\sigma}$ in Γ, it follows from the fact that an operator of the form $P_{m\sigma}$ (**A**, R_1 ,R_2 ,R_3) can be defined by a choice formula of the form IFP$^{f}_{\xi,s}$[S,T](Ψ,Φ, F). In fact, one can construct, for example, a formula Ψ that, for a given structure **B**, a relation R_1 and an integer i, computes the m-equivalence classes of **B** , and then outputs the m-tuples ā such that the rank of ā is n and R_1(i,n). ◊

6 Conclusion

This paper is a further step in the study of the expressive power of the choice fixpoint logic proposed in [GH]. Our results show that this logic has a promising expressive power. In fact, it cannot be captured by a finite number of generalized quantifiers although this can be done locally. The question whether this logic captures PTIME is however still open. We think that some notion like logical reductions and its relationships with generalized quantifiers plays an important role in this question and have to be studied further in the context of logics with choice construct.

Acknowledgements

I am grateful to the referees for their valuable comments and suggestions which led to many improvements in this paper.

References

[AB] V.Arvind and S.Biwas. Expressivity of First Order Logic with a Non-deterministic Operator. In Springer Lecture Notes in Computer Science vol. 247 (1987), pp.323-335.

[AV1] S.Abiteboul and V.Vianu. Non-determinism in logic-based languages. Annals of Math. and Artif. Int., 3 : 151-186, 1991.

[AV1] S.Abiteboul and V.Vianu. Generic computation and its complexity. In Proc. 23rd ACM Symp. on the Theory of Computing, 209-219, 1991.

[CFI] J. Cai, M. Furer and N. Immerman. (1992), An optimal lower bound on the number of variables for graph identification. Combinatoria 12 (4), 389-410.

[Cai] X. Caicedo. Hilbert's Epsilon-Symbol in the Presence of Generalized Quantifiers. In M. Krynicki et al (eds.), Quantifiers: Logics, Models and Computations, vol II, Kluwer Acad. Pub., 1995, pp. 63-78.

[Daw] A. Dawar. Feasible computation through model theory. Ph.D Thesis. University of Pennsylvania, Philadelphia, 1993.

[Ebb] H.D. Ebbinghass. Extended Logics: The general framwork. In J.Barwise and S.Feferman (eds.), Models-Theoric Logics, Springer, 1985, pp. 25-76.

[GH] F.Gire and H.K. Hoang. An extension of fixpoint logic with a symmetry-based choice construct. In Information and Computation, vol. 144, Number 1, July 10, 1998, pp. 40-65.

[GS] Y.Gurevich and S.Shelah. Fixed-point extension of first-order logic. In Proc. 26th IEEE Symp. on Foundation of Computer Science, 210-214. 1983.

[Hel] L. Hella. Logical Hierarchies in PTIME. Information and Computation 129, 1-19 (1996).

[Imm] N. Immerman. Relational queries computable in polynomial time. Information and Control, 68: 86-104, 1986.

[KV] Ph. G. Kolatis and J.Väänänen. Generalized quantifiers and pebble games on finite structures. In Proc. 7 th IEEE Symposium on Logic in Computer Science (1992), 348-359.

[Kry] M. Krynicki. Notion of interpretation and nonelementary languages. Zeitschrift f'ur mathematische Logik und Grundlagen der Mathematik 34 (1988) 541-552.

[Lin] P. Lindström. First order predicate logic with generalized quantifiers. Theoria, 32: 186-195, 1966.

[Mo] A. Mostowski. On a generalization of quantifiers. Fundamenca Mathematicae, 44 (1957) 12-36

[Var] M. Vardi. The complexity of relational query languages. In Proc. 14th ACM Symp. on the Theory of Computing, 137-146, 1982.

Monadic NP and Graph Minors

Martin Kreidler and Detlef Seese

Universität Karlsruhe, Institut AIFB, D-76128 Karlsruhe/Germany
Telephone: +49 721/608-6037
FAX: +49 721/693717
email{mkr,seese}@aifb.uni-karlsruhe.de

Abstract. In our paper, we prove that Graph Connectivity is not in Monadic **NP** even in the presence of a built-in relation of arbitrary degree that does not have for an arbitrary, but fixed $k \geq 2 \in \mathbb{N}$ the complete graph K_k as a minor. We obtain our result by using the method of indiscernibles and giving a winning strategy for the duplicator in the Ajtai-Fagin Ehrenfeucht-Fraïssé Game.
The result is afterwards strengthened to arbitrary forbidden minors and to minor-closed classes of binary relations.

Keywords: Monadic Second-Order Logic, Descriptive Complexity Theory, Ehrenfeucht-Fraïssé Games, Finite Model Theory.

1 Introduction

The interconnection between the logical definability of a property on finite structures and the computational complexity of a best algorithm known for deciding that property brings new questions about the expressive power of logics which are motivated by open questions from Complexity Theory. Especially, there is a kind of parallelism between Logic and Complexity: One can observe from the work of Fagin [6] and Immerman [11] that stronger logics allow to formulate problems known to be algorithmically more difficult. Thus, an investigation of those open questions in Logic is an investigation of Complexity Theory, too.
In his seminal paper [6], Fagin showed that **NP** coincides with Existential Second-Order Logic (∃SOL) on finite structures. Unfortunately, the ∃SOL is relatively hard to handle. One reason for this is that quantification over arbitrary relations allows to encode functions into relations and relations into other relations.

When we restrict ourselves to existential quantification over *unary* relations (which can be looked at as existentially quantifying over sets), i.e. consider Existential *Monadic* Second-Order Logic (∃MSOL), the above described effect cannot appear any more. In terms of Complexity Theory, the set of all problems definable in ∃MSOL establishes a subclass *Monadic* **NP** ($\mathrm{Mon}\Sigma^1_1$) of **NP**.
A nearby question is, whether $\mathrm{Mon}\Sigma^1_1$ can be ordered into the complexity class hierarchy: On the one hand, the intersection of **NPC** (i.e. the set of all **NP**-complete problems) and $\mathrm{Mon}\Sigma^1_1$ is nonempty, since Graph 3-Colorability is in $\mathrm{Mon}\Sigma^1_1$ and known to be **NP**-complete; on the other hand, Graph Connectivity

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 126–141, 1999.

is obviously in **P**, but not in MonΣ_1^1, as Fagin showed [7]. Thus, MonΣ_1^1 is not a superset of **P**. Conversely, Graph Connectivity is known to be in Monadic **co-NP**, i.e. definable by a universal MSOL formula. Hence, it is an interesting question, how much expressive power must be added to ∃MSOL such that Graph Connectivity can be defined.

One possible such enhancement of ∃MSOL is adding *built-in relations*, i.e. relations whose interpretations are a-priori fixed.
Fagin, Stockmeyer and Vardi [9] asked whether negative definability results can be proven for Graph Connectivity and MonΣ_1^1 enhanced by built-in relations of *large* degree. Several negative definability results for Graph Connectivity and enhancements of ∃MSOL by built-in relations have been proven up to now (e.g. linear order, degree $\in n^{o(1)}$) [9,12,18]; in our paper, we give a further negative result for a large class of built-in relations: The class of all binary relations that, when taken as graphs, do not have for an arbitrary, but fixed $k \in \mathbb{N}$ the complete graph K_k as a minor, i.e. don't have an induced subgraph that can be contracted and from which edges can be omitted in such a way that the obtained graph is isomorphic to the K_k.

From Wagner's characterization of planar graphs as those with no minor isomorphic to the complete graph K_5 or the bipartite graph $K_{3,3}$ to the famous proof of Wagner's Conjecture by Robertson and Seymour [15,16,17], saying that an arbitrary set M of graphs must be finite if it has the property that no graph of M is isomorphic to a minor of another graph in M, graph minors proved to be a useful tool in many areas of graph theory covering topological as well as algorithmic aspects (see [17]). The main point in all these applications is the characterization of the structure of graphs which do not have an arbitrarily given but fixed graph as minor. Graphs with no minor isomorphic to the K_k play an essential rôle there. Our main result shows that for these graphs as built-in relations the problem whether Graph Connectivity is in MonΣ_1^1 can be settled as well, i.e. that Graph Connectivity is not in MonΣ_1^1 even in the presence of such a built-in relation. Our proof works in two steps: First, we show a separation result for the built-in relation, afterwards, we give a winning strategy for the duplicator in the Ajtai-Fagin–Ehrenfeucht-Fraïssé Game based on that separation result.

We strenghten our result to arbitrary (but finite) forbidden minors and to arbitrary built-in relations which belong to a minor-closed class of irreflexive symmetric binary relations that is not equal to the class of all irreflexive symmetric binary relations. Finally, we show that for built-in relations of bounded tree-width a similar result can be proved.

Acknowledgement. We are grateful to an unknown referee for many helpful constructive suggestions.

2 Definitions

In our paper, we use standard terminology that can be looked up in textbooks in Logic, Complexity Theory and Graph Theory e.g. [4,5,13]. Below, we introduce some additional notation:

Graphs are represented as structures with one irreflexive binary relation E. In the standard way, we look at a graph G as a pair $G = (V(G), E(G))$, where $V(G)$ is the set of vertices and $E(G)$ stands for the set of edges. We do not distinguish between a binary relation and a graph. We say that the vertices $x, y \in V(G)$ of a graph $G = (V(G), E(G))$ are *adjacent*, briefly $\underset{G}{Adj}(x, y)$, if $(x, y) \in E(G)$. If $X \subset V(G)$, we call $G{\downarrow}X$ the *induced subgraph* of G by X. The *d-neighbourhood* of a vertex v in $V(G)$ is $\mathcal{N}_G^d(v) := G{\downarrow}\{y \in V(G) : \underset{G}{dist}(v, y) \leq d\}$ with $d \geq 0$ and $\underset{G}{dist}(v, y)$ denoting the distance of $v, y \in V(G)$ in G. We sometimes use a vertex set as the second argument: $\underset{G}{dist}(v, B) := \min\{\underset{G}{dist}(v, b) |\ b \in B\}$. We denote by $G - v$ the structure $G{\downarrow}(V(G)\backslash\{v\})$, analogously $G - B$ for a set $B \subset V(G)$. The *degree* $deg(v)$ of a vertex v is the number of vertices that are adjacent to v.

In the following, we assume that the vertex set $V(G)$ of any graph G are the first n natural numbers, i.e. if $|V(G)| = n$, then $V(G) = \{1, 2, \ldots, n\} = [n]$ for simplicity. Since $V(G)$ is a set of natural numbers, we can always use the canonical order for $\mathbb{N}$ also for vertices. Sequences are referred to by angles $\langle\rangle$.

A sequence $\mathbf{B} = \langle B_1, B_2, \ldots\rangle$ is a *built-in relation* if for any $n \in \mathbb{N}$ the domain of B_n is $[n] \times [n]$. We note that $\mathbf{B}$ itself is not a relation. In this paper, we only consider symmetric irreflexive relations as built-in relations. If P is a set of graphs, then a formula φ expresses P in the presence of a built-in relation $\mathbf{B}$, if for every graph G it holds that $(G, B_{|V(G)|}) \models \varphi$ iff $G \in P$. The set P is denoted as a *property*.

$\mathrm{Mon}\Sigma_1^1$ is the sublogic of $\exists$SOL where second-order variables are restricted to unary predicates. A $\mathrm{Mon}\Sigma_1^1$-formula is of the type $\exists X_1, \ldots, \exists X_j \varphi$, where φ is a First-Order (FO) formula with free unary variables $X_1, \ldots, X_j$.
A property P on graphs is said to be in $\mathrm{Mon}\Sigma_1^1$, if there is a formula $varphi \in \mathrm{Mon}\Sigma_1^1$ such that for any G it holds $G \models \varphi \leftrightarrow G \in P$.

The Ajtai-Fagin–(c, r)-Ehrenfeucht-Fraïssé Game [1] for a property P and $\mathrm{Mon}\Sigma_1^1$ is played by two players, the spoiler and the duplicator, as follows:

1. The duplicator chooses a graph $G_0 \in P$.
2. The spoiler colors G_0 with the c colors $C_1, \ldots, C_c$.
3. The duplicator chooses a graph $G_1 \notin P$.
4. The duplicator colors G_1 with the c colors $C_1, \ldots, C_c$.
5. The spoiler and the duplicator play an r-round Ehrenfeucht-Fraïssé Game on the two structures $(G_0, C_1, \ldots, C_c)$ and $(G_1, C_1 \ldots, C_c)$.

The duplicator *wins*, if the two substructures induced by the pebbled vertices wrt. their pebbling order are isomorphic. Otherwise, the spoiler wins. A player has a *winning strategy*, if he can play in such a way that he will win, whatever the other player will do. G_0 and G_1 are *FO r-equivalent* (briefly: $G_0 \sim_r G_1$) iff the duplicator has a winning strategy in the r-round Ehrenfeucht-Fraïssé Game on G_0 and G_1. A property P is not in $\mathrm{Mon}\Sigma_1^1$ iff for any $c, r \in \mathbb{N}$ the duplicator has a winning strategy in the Ajtai-Fagin–(c, r)-game for P.

3 The main result

Definition 1. *Let G be a graph. A graph H is called a minor of G, briefly $H \prec G$, if there is a subgraph of G that is either isomorphic to H or can be transformed into a graph that is isomorphic to H by successively eliminating edges and/or contracting edges and identifying the two incident vertices involved.*

Table 1 gives a brief overview of several classes of graphs with at least one forbidden minor of those classes.

class of graphs	forbidden minor
trees / forests	K_3
planar graphs	K_5 and $K_{3,3}$
graphs of tree-width at most k	K_{k+2}
partial k-trees	K_{k+2}
chordal graphs with maximal clique size k	K_{k+1}
series-parallel graphs	K_4
graphs with non-oriented genus $\leq \lceil \frac{(k-3)(k-4)}{6} \rceil$	K_{k+1}

Table 1. Classes of graphs with at least one forbidden minor.

Now, we are ready for the main result of this paper:

Theorem 1. *For every $k \in \mathbb{N}\backslash\{1\}$ is Graph Connectivity not in MonΣ^1_1, even in the presence of an irreflexive symmetric built-in relation that does not have the complete graph K_k as a minor.*

Proof: The proof proceeds in two steps: First, we show a special separation result (Lemma 1) for graphs that do not have the K_k as a minor. This allows us to look at the built-in relation as being composed of many local substructures, when at most $k-2$ vertices are removed appropriately. Especially, this k does not depend upon the *size* of the built-in relation. In the second step of the proof, we give a winning strategy of the duplicator in the Ajtai-Fagin (c,r)-game for Graph Connectivity, MonΣ^1_1 and the built-in relation. This winning strategy relies on the separation result.

Let $k \in \mathbb{N}$ be arbitrary, but fixed, let $\mathbf{B} = \langle B_1, B_2, \ldots \rangle$ be a sequence of graphs that do not have the complete graph K_k as a minor, where $V(B_n) = [n]$.

Let for any $n \in \mathbb{N}$ v_n be a vertex of B_n of maximal degree. Let $r, c \in \mathbb{N}$ (the number of rounds and the number of colors, respectively) be arbitrary, but fixed, let $C = \langle C_1, \ldots, C_c \rangle$ be the sequence of colors.

If $k = 3$, then $\mathbf{B}$ consists of forests, hence, our theorem follows from [12]. Schwentick showed [18] a negative definability result for Graph Connectivity, MonΣ^1_1 and built-in relations of degree $n^{o(1)}$. Thus, we only have to consider the case that $deg(\mathbf{B}) \in O(n)\backslash n^{o(1)}$, hence, the case that $k = 2$ is included there (since in this case all relations in $\mathbf{B}$ are empty).

We call the sequence $\mathbf{B}$ $(k;j)$-*separable*, if for any $l \in \mathbb{N}$ exists an $n \in \mathbb{N}$, a set $M_n \subset [n]$ of v_n-neighbours in B_n with $|M_n| \geq l$ and a set SV_n with $v_n \in SV_n$, $|SV_n| \leq k$, $M_n \cap SV_n = \emptyset$ and for arbitrary $x_1 \neq x_2 \in M_n$ it holds that $V(\mathcal{N}^j_{B_n - SV_n}(x_1)) \cap V(\mathcal{N}^j_{B_n - SV_n}(x_2)) = \emptyset$.

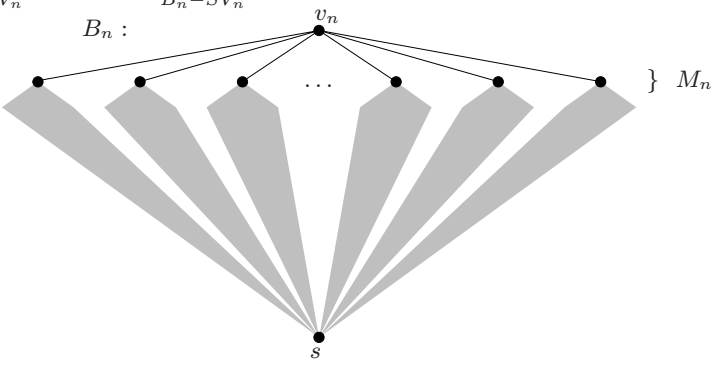


Fig. 1. A sketch of a $(2;j)$-separable Graph B_n $(j \geq 1)$

Figure 1 gives an intuitive sketch of $(2;d)$-separability, i.e. here we have that $SV_n = \{v_n, s\}$. The grey areas stand for the j-neighbourhoods of the respective v_n-neighbours in $B_n - v_n - s$.

Proposition 1. *If there exists a* $j \in \mathbb{N}$ *such that* $\mathbf{B}$ *is* $(k-1;j)$*-separable, but* $\mathbf{B}$ *is not* $(k-2;j)$*-separable, then there is an* $n \in \mathbb{N}$ *with* $K_k \prec B_n$.

Sketch of Proof: Let B_n be as above. We take for a large enough $n \in \mathbb{N}$ an appropriate $k-1$-element subset of M_n. Since $|SV_n| = k-1$ and B_n is not $(k-2,j)$-separable, there is a subgraph of B_n whose edges can be contracted in such a way that the bipartite graph $K_{k-1,k-1}$ remains ($k-1$ vertices are in SV_n, the other $k-1$ vertices are those of M_n). Our proposition follows from $K_k \prec K_{k-1,k-1}$.

Definition 2. *A set* $I_n \subseteq M_n$ *is said to be* (h,m)*-indiscernible for* $h, m \in \mathbb{N}$ *if for all* $x_1 < \cdots < x_m, x'_1 < \cdots < x'_m \in I_n$ *it holds*

$$(B_n, v_n, x_1, \ldots, x_m) \sim_h (B_n, v_n, x'_1, \ldots, x'_m).$$

We have defined this variant of the notion of indiscernibles here just for our special application, i.e. we consider only a very special case of the more general concept of indiscernibles from Model Theory (see e.g. [20]).

Theorem 2. *(see also [20], Theorem 2.4) For all* $h, l, m \in \mathbb{N}$ *and a fixed finite set of constants exists an* $n \in \mathbb{N}$*, such that* M_n *contains an* l*-element subset* I_n*, which is* (h,m)*-indiscernible.*

Proof: The proof is an obvious consequence of the finite version of the well-known theorem of Ramsey and the fact that $\sim_h$ is of finite index, since the

degree of v_n and hence the cardinality of M_n can be made large enough if n is chosen large enough.

Now, we will prove that $\mathbf{B}$ is $(k-2;2^r)$-separable. This is accomplished by a breadth-first search in B_n for a sufficiently large $n \in \mathbb{N}$ as follows: We begin with the vertices in M_n (i.e. that have distance $d=0$ from M_n) and look for a set $I_n^d \subseteq M_n$ that is $(k-2;d)$-separable. It will turn out that $k=0$ in this case. Then we will successively increase d and obtain in each step a subset $I_n^{d+1} \subseteq I_n^d$, for which we will show that it is $(k-2, d+1)$-separable. This is done by considering the behaviour of the vertices having distance $d+1$ from I_n^{d+1}.

Lemma 1. *Let* $\mathbf{B}$ *be the built-in relation as above. Then for any* $l \geq 3*k$, *any* $d \geq 0$ *and any* $h > 3*k*d$ *exists an* $n = n(d,l,h)$, *a sequence of sets* $\langle SV_n^{-1}, \ldots, SV_n^d \rangle$ *and a set* I_n^d *with*

1. $I_n^d \subset M_n$ *and* $|I_n^d| \geq l$
2. *For all* $0 \leq j \leq d$: $SV_n^{j-1} \subset SV_n^j \subset V(\underset{B_n}{\mathcal{N}^d}(I_n^d)) \setminus I_n^d$, $SV_n^{-1} = \{v_n\}$ *and* $|SV_n^d| \leq k-2$.
3. *It holds for all* $x_1 < \cdots < x_{2k}, x'_1 < \cdots < x'_{2k} \in I_n^d$

$$\Big(B_n{\downarrow}(V(\underset{B_n - SV_n^{d-1}}{\mathcal{N}^d}(I_n^d)) \cup SV_n^{d-1}) \, , SV_n^{d-1}, x_1, \ldots, x_{2k} \Big)$$
$$\sim_h$$
$$\Big(B_n{\downarrow}(V(\underset{B_n - SV_n^{d-1}}{\mathcal{N}^d}(I_n^d)) \cup SV_n^{d-1}) \, , SV_n^{d-1}, x'_1, \ldots, x'_{2k} \Big)$$

4. *For all* $0 \leq j \leq d$: $\underset{B_n - SV_n^j}{\mathcal{N}^j}(I_n^d)$ *consists of exactly* $|I_n^d|$ *disconnected components, each of which contains exactly one vertex that belongs to* I_n^d.

Proof: First, we note the difference in the right upper index of SV_n^* above (items *3)* and *4)*), what is the main "trick" for proving the lemma.

If for any $l \in \mathbb{N}$ there is an $n \in \mathbb{N}$, an l-element subset $M'_n \subset M_n$ and for any two vertices $x, y \in M'_n$ it holds $V(\underset{B_n - v_n}{\mathcal{N}^d}(x)) \cap V(\underset{B_n - v_n}{\mathcal{N}^d}(y)) = \emptyset$, we are done: Set $SV_n^j := \{v_n\}$ for $-1 \leq j \leq d$ and apply Theorem 2 on the vertices in M'_n, hence we obtain a set $I_n \subset M'_n$ that trivially fulfills the above conditions *1.-4.*

Thus, we can assume that there is a universal bound in $\mathbf{B}$ for the number of such vertices in M_n for each $n \in \mathbb{N}$. We show the lemma by induction on d, i.e. we perform a kind of breadth-first search in the graphs of $\mathbf{B}$:

For $d=0$, we obtain for any $l \geq 3*k$ and any $h > 3*k*d$ an $n = n(0,l,h)$ and I_n^0 by directly applying the standard version of Ramsey's Theorem as follows: We obtain for all $l > k$ (otherwise take $l := k+1$) an l-element set $I_n^0 \subset M_n$ with $B_n{\downarrow}I_n^0 \cong \bar{K}_l$, i.e. this induced substructure is edgeless. Hence, condition *1.* is fulfilled. Setting $SV_n^{-1} := \{v_n\}$ and $SV_n^0 := \{v_n\}$ fulfills obviously condition *2.* Clearly, condition *3.* holds for the "canonical"

$<$-relation in I_n^0, since both structures considered there are stars. As pointed out above, $B_n{\downarrow}I_n^0$ is edgeless, thus, condition *4.* also holds.

We now consider the case $d+1$, i.e. for any $l \in \mathbb{N}$ and any $h \in \mathbb{N}$ we have to find an $n = n(d+1, l, h) \in \mathbb{N}$ and sets I_n^{d+1}, SV_n^{d+1} that fulfill conditions *1.-4.* Let $l \geq 3 * k$ and $h > 3 * k * d$ be arbitrary, but fixed.

By induction hypothesis, for any $\bar{l} \in \mathbb{N}$ exist $\bar{n} = \bar{n}(d, \bar{l}, h)$, a set $I_{\bar{n}}^d \subset [\bar{n}]$ and a sequence $\langle SV_{\bar{n}}^{-1}, \ldots, SV_{\bar{n}}^d \rangle$ that fulfill conditions *1.-4.* Let $\mathbf{B'} = \langle B'_{\bar{n}} \rangle$ be the sequence of graphs where

$$B'_{\bar{n}} \;=\; B_{\bar{n}}{\downarrow}(V(\underset{B_{\bar{n}} - SV_{\bar{n}}^d}{\mathcal{N}^{d+1}}(I_{\bar{n}}^d)) \cup SV_{\bar{n}}^d)$$

We apply Theorem 2 on $\mathbf{B'}$ with $I_{\bar{n}}^d$ (instead of $M_{\bar{n}}$) and the vertices in $SV_{\bar{n}}^d$ are the additional constants. Note that $|SV_{\bar{n}}^d| \leq k-2$. From this, we obtain an ordered set $I_{\bar{n}}^{d+1} \subseteq I_{\bar{n}}^d$ of $(h, 2k)$-indiscernible vertices, which fulfills condition *3.* From Theorem 2 and the induction hypothesis follows that for l there is an $\bar{l} \in \mathbb{N}$ such that if $|I_{\bar{n}}^d| \geq \bar{l}$, then $|I_{\bar{n}}^{d+1}| \geq l$. Thus, we can choose $n(d+1, l, h) := \bar{n}$, hence, condition *1.* holds.

Let $NSV_n^{d+1} := \underset{B_n - SV_n^d}{\mathcal{N}^{d+1}}(I_n^{d+1})$, i.e. the $d+1$-neighbourhood of I_n^{d+1} in B_n without the vertices in SV_n^d and without those vertices that are only reachable from I_n^{d+1} via vertices in SV_n^d. Since the latter are distinguished, the vertices in I_n^{d+1} are also $(h, 2k)$-indiscernible in NSV_n^{d+1}. Now, we consider NSV_n^{d+1} more closely: If NSV_n^{d+1} is disconnected in such a way that no two vertices $x_1 \neq x_2 \in I_n^{d+1}$ are in the same connected component, we can set $SV_n^{d+1} := SV_n^d$ and are done: Condition *4.* follows for $j < d+1$ from the induction hypothesis, since $I_n^{d+1} \subseteq I_n^d$; for $j = d+1$, since NSV_n^{d+1} is disconnected as pointed out above. Condition *2.* also follows from its validity in the induction hypothesis and the fact that $I_n^{d+1} \subseteq I_n^d$. Note that the validity of the other two conditions was already assured above.

So, we consider the case that there exist two vertices $x_1 \neq x_2 \in I_n^{d+1}$ which are in NSV_n^{d+1} in the same connected component:

Proposition 2. *If there are pairwise different* $x_1, x_2, x_3 \in I_n^{d+1}$ *and* $y \in V(NSV_n^{d+1})$ *such that* $\underset{NSV_n^{d+1}}{dist}(x_1, y) = d+1$ *and* $\underset{NSV_n^{d+1}}{dist}(x_2, y) \leq d+2$ *and* $\underset{NSV_n^{d+1}}{dist}(x_3, y) \geq d+2$*, then* $K_k \prec NSV_n^{d+1}$*, thus* $K_k \prec B_n$*.*

Sketch of Proof: The main idea of the proof is to make use of the indiscernibility property of the vertices in I_n^{d+1}. It can be used by considering different games in which the duplicator always has a winning strategy. From the possibilities how the spoiler can play in such games (and will lose), we deduce the desired structure result about NSV_n^{d+1}.

Let $z \in V(NSV_n^{d+1})$ and $\underset{NSV_n^{d+1}}{dist}(x, z) = d+1$. Now, we look at the relative position of z to the vertices in I_n^{d+1}: Let $\kappa(z) := |\{x \in I_n^{d+1} | \underset{NSV_n^{d+1}}{dist}(I_n^{d+1}, z) = d+1\}|$. z is chosen in such a way that $\kappa(z)$ is maximal (if there are several such vertices

z, we take that with the least number) and $\kappa(z) < |I_n^{d+1}|$.
Now, we distinguish three cases: Either $\kappa(z) = 1$ and there is a vertex in I_n^{d+1} that has distance $d+2$ from z or $2 \leq \kappa(z) < 2k$ or $\kappa(z) \geq 2k$.

If $2 \leq \kappa(z) < 2k$, then the spoiler can exactly verify which vertices in I_n^{d+1} have distance $d+1$ to z, thus, he can also verify the number of these vertices; this is the key to the construction of a K_k: The duplicator has for any $2k$ vertices $x_1 < \cdots < x_k, x'_1 < \cdots < x'_k \in I_n^{d+1}$ a winning strategy in the h-round game on $(NSV_n^{d+1}, v_n, x_1, \ldots, x_k)$ and $(NSV_n^{d+1}, v_n, x'_1, \ldots, x'_k)$, since the vertices in I_n^{d+1} are $(h, 2k)$-indiscernible, thus, are (h, k)-indiscernible. The idea is that for any $\kappa(z)$-tuple in I_n^{d+1} exists a vertex z that has distance $d+1$ to all vertices in that tuple. We obtain from this a K_k, as sketched for $\kappa(z) = 3$ in Figure 2 for the first contraction step.

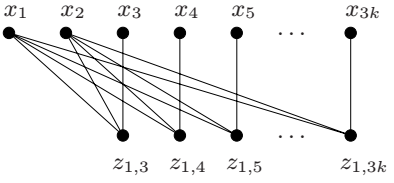


Fig. 2. The situation in the beginning.

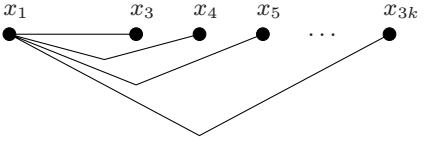


Fig. 3. Contracting the $z_{1,j}$.

Now, we treat the case $\kappa(z) \geq 2k$. In contrast to the above case, the spoiler cannot verify any more the exact number of the vertices in I_n^{d+1} to which z has distance $d+1$ in NSV_n^{d+1}. This requires a different construction of the K_k as a minor. Since $\kappa(z) < |I_n^{d+1}|$, for any such z exists at least one $x \in I_n^{d+1}$ which does not have distance $d+1$ from z. This observation ist the starting point for the construction of the K_k in this case: We now consider h-round games on NSV_n^{d+1}, in which in the first structure the $2k$ least (wrt. $<$) vertices in I_n^{d+1} are distinguished. The idea is that for any $2k$-tuple in I_n^{d+1} exists a z that has distance $d+1$ to any vertex of that tuple, and a vertex x_{i_j} that has distance $> d+1$ to z. In order to find k different such vertices z (to construct a K_k), we consider iterated games where always at least k same vertices and the x_{i_j} are distinguished. Figures 4,5 sketch the behaviour. The distance from z_j to x_{i_j} is greater than $d+1$ for $j \leq k'+1$; this is denoted in Figure 5 by a missing line.

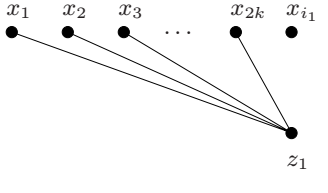


Fig. 4. After the first game.

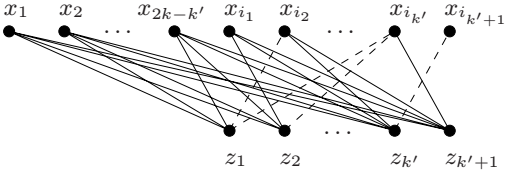


Fig. 5. After the $(k'+1)$th game.

Finally, we have to consider the case $\kappa(z) = 1$, i.e. two vertices $x_1 \neq x_2 \in I_n^{d+1}$ exist with $\underset{NSV_n^{d+1}}{dist}(x_1, z) = d+1$ and $\underset{NSV_n^{d+1}}{dist}(x_2, z) = d+2$. We refer to z as $z_{1,2}$ and to its neighbour which has distance $d+1$ from x_2 as $z_{2,1}$. The idea is to derive successively for any pair $x_i \neq x_j \in I_n^{d+1}$ two such vertices $z_{i,j}$ and $z_{j,i}$ and then constructing a K_k from this.

Thus, any vertex $y \in V(NSV_n^{d+1})$ with $\underset{NSV_n^{d+1}}{dist}(y, I_n^{d+1}) = d+1$ has either distance $d+1$ to *exactly one* vertex in I_n^{d+1} or to *all* vertices in I_n^{d+1}. From Proposition 1 follows that at most $k - 2 - |SV_n^d|$ such vertices can exist that have distance $d+1$ to all vertices in I_n^{d+1}, since otherwise $K_k \prec B_n$. Thus, we can add these at most $k - 2 - |SV_n^d|$ vertices to SV_n^d and refer to the resulting set as SV_n^{d+1}. The subset part of condition *2.* obviously holds, its second part follows from Proposition 1. We obtain condition *4.* for values $d' < d+1$ from the induction hypothesis, since $I_n^{d+1} \subseteq I_n^d$. For $d+1$, condition *4.* follows directly by the above construction of SV_n^{d+1}, what concludes our induction and hence, proves Lemma 1.

q.e.d.

Proposition 3. *Let* $\mathbf{B}$ *be as above, let* $r \in \mathbb{N}$ *be arbitrary, but fixed. Then for any* $l \in \mathbb{N}$ *there is an* $n \in \mathbb{N}$ *and sets* I_n, SV_n *such that for any* $x_1 \neq x_2 \in I_n$ *the following two assertions hold:*

a) $\mathbf{B}$ *is* $(k-2; 2^r)$*-separable.*

b) $\left(B_n{\downarrow}(V(\underset{B_n - SV_n}{\mathcal{N}^{2^r}}(x_1)) \cup SV_n)\ , SV_n, x_1 \right) \sim_{(r+2^r)} \left(B_n{\downarrow}(V(\underset{B_n - SV_n}{\mathcal{N}^{2^r}}(x_2)) \cup SV_n), SV_n, x_2 \right)$

Proof: Let $l \in \mathbb{N}$ be arbitrary, but fixed. By Lemma 1, there is an $n \in \mathbb{N}$ such that for $d := 2^r + 1$ and $h := r + 2^r$ there is a set I_n and a sequence $\langle SV_n^{-1}, \ldots, SV_n^{2^r+1} \rangle$ that fulfill conditions *1.-4.* We set $I_n := I_n^{2^r+1}$ and $SV_n := SV_n^{2^r}$.

Assertion *a)* follows from conditions *1.* and *4.* in Lemma 1 for $j = 2^r < d$. Assertion *b)* follows from condition *3.* in Lemma 1 and by the transitivity of $\sim_{(r+2^r)}$, note that by assertion *a)*, the considered structures are pairwise vertex-disjoint except the vertices in SV_n. It is important to see that I_n was chosen to be $I_n^{2^r+1}$, i.e. from condition *3.* in Lemma 1 follows something slightly stronger than we require here.

Let $l \in \mathbb{N}$ be arbitrary, but fixed, let $n \in \mathbb{N}$ as in Proposition 3, let $D := \underset{B_n - SV_n}{\mathcal{N}^{2^r}}(I_n)$. We define a function u and a set function U as follows:

$$u(x) := \begin{cases} y \in I_n & \text{if } x \text{ belongs in } D \text{ to the same component as } y \\ \perp & \text{if } x \notin V(D) \end{cases}$$

$$U(x) := \begin{cases} \{y \in V(D) \mid u(y) = u(x)\} & \text{if } u(x) \neq \perp \\ \emptyset & \text{otherwise} \end{cases}$$

We also allow the argument for U to be a set, i.e. for $X = \{x_1, \ldots, x_i\} \subset V(D)$ is $U(X) = U(x_1) \cup \cdots \cup U(x_i)$.

Now, the duplicator chooses the undirected simple connected graph $G_0 = (V(G_0), E(G_0))$ (i.e. G_0 contains no loops or multiple edges) as follows:

- $V(G_0) = V(B_n) = [n]$.
- $E(G_0)$ is defined according to *a)-f)*, let y, z be arbitrary vertices of G_0:
  - *a)* If $y \notin I_n$ and $z \notin I_n$ and $\underset{B_n}{Adj}(y, z)$: $(y, z) \in E(G_0)$.
  - *b)* If $y \in I_n$ and $z \notin I_n$ and $z \neq v_n$ and $\underset{B_n}{Adj}(y, z)$: $(v_n, z) \in E(G_0)$.
  - *c)* If $y \in I_n$ and $z \in I_n$, say, $y = x_i$, $z = x_j$: $(y, z) \in E(G_0)$ iff $i = j - 1$.
  - *d)* $(x_1, v_n) \in E(G_0)$, $(x_l, v_n) \in E(G_0)$.
  - *e)* Any disconnected component in B_n not containing v_n is connected by an undirected edge from one arbitrary vertex of that component to v_n.
  - *f)* $(y, z) \notin E(G_0)$ in all remaining cases.

Intuitively, all B_n-edges are covered by G_0-edges except those being incident to vertices in I_n; these are connected to v_n and any I_n-vertex is connected to both its neighbours wrt. the order of I_n. The largest and the least vertex of I_n are connected to v_n.

Now, the spoiler colors the graph G_0 with the c colors $C_1, \ldots, C_c$.

Proposition 4. *There exist sets* $T = \{t_0, \ldots, t_{2^r+1}\} \subset I_n$, $W = \{w_0, \ldots, w_{2^r+1}\} \subset I_n$, *and two vertices* t_{-1}, w_{-1} *with* $t_i < t_j, w_i < w_j$ *for* $-1 \leq i < j \leq 2^r + 1$, $t_{2^r+1} < w_{-1}$ *and* $\underset{G_0}{Adj}(t_{i-1}, t_i)$, $\underset{G_0}{Adj}(w_{i-1}, w_i)$ *for* $0 \leq i \leq 2^r + 1$ *and*

$$\Big((G_0, B_n, C)\!\downarrow\!(U(T) \cup SV_n) \ , SV_n, t_0, \ldots, t_{2^r+1} \Big)$$
$$\sim_{(r+2^r)}$$
$$\Big((G_0, B_n, C)\!\downarrow\!(U(W) \cup SV_n) \ , SV_n, w_0, \ldots, w_{2^r+1} \Big)$$

Proof: First, we note that T, W are ordered, since they are subsets of the ordered set I_n, furthermore, by $t_{2^r+1} < w_{-1}$ and the ordering of T, W, we have that $(T \cup \{t_{-1}\}) \cap (W \cup \{w_{-1}\}) = \emptyset$.

By Proposition 3 *b)*, all considered structures belong without coloring to the same $\sim_{(r+2^r)}$equivalence class (note that $\sim_{(r+2^r)}$is an equivalence relation of finite index). Since c is finite, there are only finitely many $\sim_{(r+2^r)}$equivalence classes, when the coloring is respected, thus, our claim follows, because $l \in \mathbb{N}$ (the cardinality of I_n) can be chosen arbitrarily large by Proposition 3 *a)*.

Let $TW := T \cup W$, let $H^0 := \{t_0, t_1, w_0, w_1\}$. We define a special distance measure δ^0, let $x \in [n]$:

$$\delta^0(x) := \begin{cases} \min\{ \underset{G_0 - t_{-1} - w_{-1}}{dist}(y, H^0) + \underset{B_n - SV_n}{dist}(x, y) \ , \ 2^r + 1 \} \\ \qquad \text{iff there is } y \in TW : \ u(x) = y \\ 2^r + 1 \text{ otherwise} \end{cases}$$

Thus, if $x, y \in U(TW)$ then $|\delta^0(x) - \delta^0(y)| > 1 \ \Rightarrow \ \neg \underset{(G_0, B_n)}{Adj}(x, y)$.

Corollary 1. *The following equivalence holds:*

$$\Big((G_0, B_n, C, \delta^0)\!\downarrow\!(U(T) \cup SV_n)\ , SV_n, t_0, \ldots, t_{2^r+1} \Big)$$
$$\sim_r$$
$$\Big((G_0, B_n, C, \delta^0)\!\downarrow\!(U(W) \cup SV_n)\ , SV_n, w_0, \ldots, w_{2^r+1} \Big)$$

Proof: By definition, δ^0 can maximally have $2^r + 2$ different values. Thus, the above abbreviation stands for the addition of $2^r + 2$ new unary relations, where each vertex is captured by exactly one of these new relations. From Proposition 4 follows that the duplicator has a winning strategy in the $(r + 2^r)$-round game on the two structures considered here without the new relations encoding δ^0. The proof is accomplished by showing that the spoiler can easily verify in the last 2^r rounds of that game the δ^0-distances of any vertex pebbled during the first r rounds. Hence, the duplicator's pebbling strategy respects the $2^r + 2$ new relations in the first r rounds.

Now, the duplicator chooses the disconnected graph G_1 as follows:

- $V(G_1) = V(G_0) = V(B_n) = [n]$
- $E(G_1) = E(G_0) \setminus \{ (t_0, t_1)\ ,\ (w_0, w_1) \} \cup \{ (t_0, w_1)\ ,\ (t_1, w_0) \}$

G_1 is disconnected, since it contains a cycle $Q = \{t_1, t_2, \ldots, t_{2^r+1}, \ldots, w_{-1}, w_0\}$.

Afterwards, the duplicator colors G_1 identically to the coloring of G_0, i.e. vertices with the same number obtain the same color.

Let $H^1 := H^0$, define δ^1 in (G_1, B_n) analogously to δ^0 in (G_0, B_n), replace the occurrence of H^0 by H^1 and the occurrence of G_0 by G_1. Obviously, for any $x \in [n]$ it holds that $\delta^0(x) = \delta^1(x)$. Furthermore, it is worth to note that the two considered structures only differ by four edges (between t_0, t_1, w_0, w_1); thus, we say that both structures are *nearly isomorphic*.

Now, the spoiler and the duplicator play an r-round Ehrenfeucht-Fraïssé Game that respects the edges of B_n, G_0/G_1 and the vertex coloring. Our game uses partially the method invented by Schwentick [18]; the main idea is that the two structures to be played on can both be divided into two substructures such that the duplicator has winning strategies for each pair and these two winning strategies are then combined to a global winning strategy. In contrast to [18], we have additionally to take closer care of the vertices in $\{t_0, w_0\} \cup SV_n$; note that for any $y \in SV_n$ it holds $\delta^{0/1}(y) = 2^r + 1$.

We need two special distance bounds; these will be defined sucessively during the game, in the beginning of the game (i.e. before the first round) they have the following values: $d(0) := 0$, $D(0) := 2^r + 1$. Using these bounds, we can define two *areas* J^0, H^0 in (G_0, B_n, δ^0) and J^1, H^1 in (G_1, B_n, δ^1), respectively, for $0 \leq q \leq r$ as follows:

$H^{0/1}(q) := \{ x \in [n] \mid \delta^{0/1}(x) \leq d(q) \} \cup SV_n$
$J^{0/1}(q) := \{ x \in [n] \mid \delta^{0/1}(x) \geq D(q) \} \cup SV_n \cup \{t_0, w_0\}$

Lemma 2. *The duplicator can play in such a way that after the qth round* $(0 \leq q \leq r)$ *for pebbled vertices* $A_q := \{a_1, \ldots, a_q\} \subset [n]$ *in* (G_0, B_n) *and* $B_q := \{b_1, \ldots, b_q\} \subset [n]$ *in* (G_1, B_n)*, the following assertions (i)-(iii) hold:*

(i) *For* $\{q_1,\ldots,q_j\} \subseteq [q]$ *it holds* $\{a_{q_1},\ldots,a_{q_j}\} = A_q \cap J^0(q)$ *iff* $\{b_{q_1},\ldots,b_{q_j}\} = B_q \cap J^1(q)$ *and if* $\{a_{q_1},\ldots,a_{q_j}\} = A_q \cap J^0(q)$ *then the identity function maps*

$$\Big((G_0, B_n, C, \delta^0)\!\downarrow\!(([n]\backslash H^0(q)) \cup SV_n \cup \{t_0, w_0\}) \ , SV_n, t_0, w_0, a_{q_1},\ldots,a_{q_j} \Big)$$

isomorphically to

$$\Big((G_1, B_n, C, \delta^1)\!\downarrow\!(([n]\backslash H^1(q)) \cup SV_n \cup \{t_0, w_0\}) \ , SV_n, t_0, w_0, b_{q_1},\ldots,b_{q_j} \Big)$$

(ii) *For* $\{q'_1,\ldots,q'_{j'}\} \subseteq [q]$ *it holds* $\{a_{q'_1},\ldots,a_{q'_{j'}}\} = A_q \cap H^0(q)$ *iff* $\{b_{q'_1},\ldots,b_{q'_{j'}}\} = B_q \cap H^1(q)$ *and if* $\{a_{q'_1},\ldots,a_{q'_{j'}}\} = A_q \cap H^0(q)$ *then*

$$\Big((G_0, B_n, C, \delta^0)\!\downarrow\!([n]\backslash J^0(q)) \cup SV_n \cup \{t_0, w_0\}) \ , SV_n, t_0, w_0, a_{q'_1},\ldots,a_{q'_{j'}} \Big)$$

$$\sim_{(r-q)}$$

$$\Big((G_1, B_n, C, \delta^1)\!\downarrow\!([n]\backslash J^1(q)) \cup SV_n \cup \{t_0, w_0\}) \ , SV_n, t_0, w_0, b_{q'_1},\ldots,b_{q'_{j'}} \Big)$$

(iii) $D(q) - d(q) \ > \ 2^{r-q}$ *and* $(J^0(q) \cap A_q) \ \cup \ (H^0(q) \cap A_q) \ = \ A_q$.

Proof: We show the claim by induction on q:

For $q = 0$, the first part of assertion *(i)* is trivial; the second follows, since both structures are nearly isomorphic. For assertion *(ii)*, the first part is again trivially fulfilled. The second part is obtained as follows: The two substructures of (G_0, B_n, C, δ^0) considered in Corollary 1 are $\sim_r$-equivalent. Since both structures are nearly isomorphic, and $\delta^0(x) = \delta^1(x)$ for any $x \in [n]$, we obtain that (G_0, B_n, C, δ^0) and (G_1, B_n, C, δ^1) are *isomorphic* (not only indistinguishable) to each other except the edges in $H^{0/1}$. Thus, the two structures considered in assertion *(ii)* are without the edges in $H^{0/1}$ $\sim_r$-equivalent. But for these edges, the assertion follows directly from the construction of G_0 and G_1. The first part of assertion *(iii)* follows directly from the definitions of $d(0)$ and $D(0)$, the second part holds, since no vertex is pebbled yet.

Assume that assertions *(i)-(iii)* hold after $q < r$ rounds. We now consider the $q+1$st round, say, the spoiler pebbles a_{q+1} in (G_0, B_n, C, δ^0):

If $\delta^0(a_{q+1}) > d(q) + 2^{r-(q+1)}$ and $a_{q+1} \notin SV_n$, then the duplicator pebbles the same vertex in (G_1, B_n, C, δ^1). We set $D(q+1) := \min\{D(q), \delta^0(a_{q+1})\}$ and $d(q+1) := d(q)$.

The first part of assertion *(iii)* is straightforward. Its second part follows from the induction hypothesis and the fact that $a_{q+1} \in J^0(q+1)$.

Assertion *(i)* follows directly from the induction hypothesis and the fact that b_{q+1} was pebbled identically to a_{q+1} by the duplicator.

Assertion *(ii)*: The first part follows directly from the induction hypothesis, since $A_{q+1} \cap H^0(q+1) = A_q \cap H^0(q)$. The second part of the assertion follows directly from the induction hypothesis, if $D(q+1) = D(q)$. Otherwise, by assertion *(ii)* of the induction hypothesis, the duplicator has a winning strategy that respects the additional $2^r + 2$ relations encoding $\delta^{0/1}$ in an $(r-q)$-round game

on the structures considered there. If $D(q+1) < D(q)$, then the structures considered now in assertion *(ii)* are proper substructures of those in the induction hypothesis *(ii)*. Since the substructures considered now result from the former in a way that respects δ^0, the strategy of the duplicator in the induction hypothesis is, when applied now, again a winning strategy. Thus, also in this case assertion *(ii)* follows from its validity in the induction hypothesis. Note that we have something slightly stronger than required: The duplicator has a winning strategy in an $r-q$–round game on the structures considered now, but we only need a winning strategy in an $(r-(q+1))$-round game.

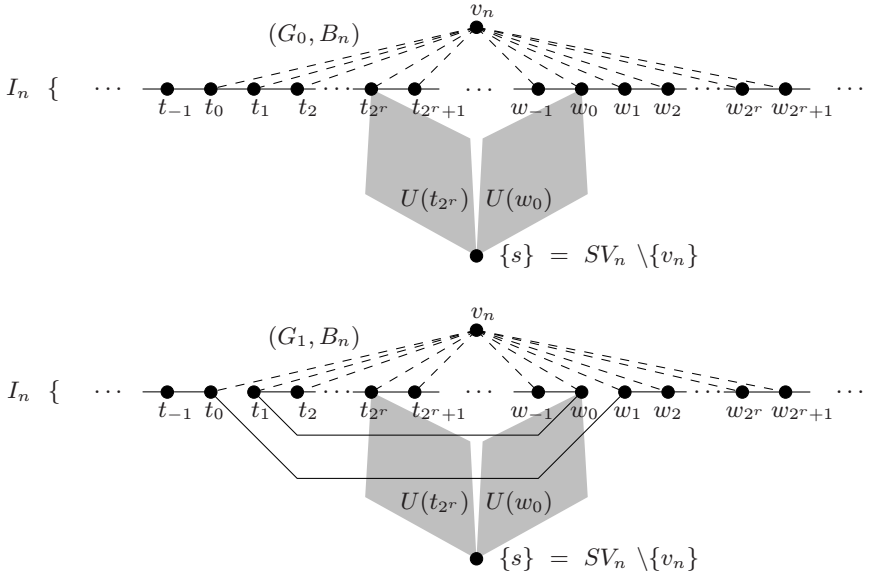


Fig. 6. A sketch of the structures (G_0, B_n) and (G_1, B_n) for the case $|SV_n| = 2$.

If $\delta^0(a_{q+1}) \leq d(q)+2^{r-(q+1)}$ and $a_{q+1} \notin \{t_0, w_0\}$, the duplicator pebbles b_{q+1} in (G_1, B_n, C, δ^1) according to his winning strategy from assertion *(ii)* of the induction hypothesis. We set $d(q+1) := \max\{d(q), \delta^0(a_{q+1})\}$, $D(q+1) := D(q)$.

The first part of assertion *(iii)* is again straightforward, the second part of it follows from the induction hypothesis and the fact that $a_{q+1} \in H^0(q+1)$.

The first part of assertion *(i)* follows directly from the induction hypothesis, since $a_{q+1} \in H^0(q+1)$ and $b_{q+1} \in H^1(q+1)$. The second part of assertion *(i)* follows directly from the induction hypothesis, since we now consider (not necessarily proper) substructures of those considered in the induction hypothesis.

For assertion *(ii)* the first part follows, since b_{q+1} was pebbled according to the winning strategy of the induction hypothesis, hence, we obtain $\delta^1(b_{q+1}) = \delta^0(a_{q+1})$ (note: $\delta^{0/1}$ is encoded into both structures considered in the induction hypothesis), thus especially it holds that $b_{q+1} \in H^1(q+1)$. The second part of assertion *(ii)* follows directly from the induction hypothesis, since the duplicator

pebbled b_{q+1} according to the winning strategy supposed there.

If $a_{q+1} \in (SV_n \cup \{t_0, w_0\})$, the duplicator pebbles the vertex b_{q+1} in (G_1, B_n, C, δ^1) with the same number. We set $d(q+1) := d(q)$, $D(q+1) := D(q)$.

Assertion *(iii)* follows immediately.

The first part of assertion *(i)* holds, since by definition of $J^{0/1}(q+1)$ it holds that $J^{0/1}(q+1) \supset (SV_n \cup \{t_0, w_0\})$. The second part of assertion *(i)* follows, since the duplicator pebbled identically.

The first part of assertion *(ii)* holds, since by definition of $H^{0/1}(q+1)$ it holds that $H^{0/1}(q+1) \supset (SV_n \cup \{t_0, w_0\})$. The second part of assertion *(ii)* holds directly by the induction hypothesis, since a_{q+1} is a distinguished vertex in the first structure considered there, thus, since b_{q+1} was pebbled identically to a_{q+1}, the former is the corresponding distinguished vertex in the second structure considered there.

If the spoiler pebbles in the $(q+1)$-th round a vertex b_{q+1} in (G_1, B_n, C, δ^1), the duplicator pebbles a_{q+1} in (G_0, B_n, C, δ^0) analogously to the above case distinction.

The validity of the assertions *(i)-(iii)* follows in the analogous way as above (note that for the second part of assertion *(iii)*, one has first to verify the first parts of assertions *(i)* and *(ii)*), what concludes our induction.

Figure 6 gives a small sketch how both structures look for the special case $SV_n = \{v_n, s\}$, hence, $|SV_n| = 2$. The dashed lines stand for B_n-edges, the other lines for edges of G_0 or G_1, respectively.

Finally, we have to show that for $q \neq q'$ it holds that $((G_0, B_n, C)\downarrow\{a_q, a_{q'}\}, a_q, a_{q'}) \cong ((G_1, B_n, C)\downarrow\{b_q, b_{q'}\}, a_q, a_{q'})$:

If $\{a_q, a'_q\} \subset J^0(r)$, this follows from assertion *(i)* of Lemma 2.
If $\{a_q, a'_q\} \subset H^0(r)$, then this follows from assertion *(ii)* of Lemma 2.
If $\{a_q, a'_q\} \not\subset H^0(r)$, and $\{a_q, a'_q\} \not\subset J^0(r)$, then from the second part of assertion *(iii)* of Lemma 2 follows that either $a_q \in H^0(r)\backslash(SV_n \cup \{t_0, w_0\})$ and $a_{q'} \in J^0(r)\backslash(SV_n \cup \{t_0, w_0\})$ or vice versa. Thus, we have by the first part of assertion *(iii)* of Lemma 2 (note that from $\{a_q, a'_q\} \not\subset J^0(r)$ follows that $\{a_q, a_{q'}\} \subset U(TW)$) that $|\delta^0(a_q) - \delta^0(a_{q'})| > 1$, thus, we obtain $\neg \underset{(G_0,B_n)}{Adj}(a_q, a_{q'})$ and $\neg \underset{(G_1,B_n)}{Adj}(b_q, b_{q'})$, and we are done.

All in all, we obtain that the duplicator has a winning strategy in the r-round game for Graph Connectivity and MonΣ^1_1 enriched by the built-in relation $\mathbf{B}$, what concludes our proof.

q.e.d.

4 Summary

Our result yields an affirmative answer in the direction of the question of Fagin, Stockmeyer and Vardi [9], whether Graph Connectivity is not in $\mathrm{Mon}\Sigma_1^1$ even in the presence of built-in relations of large degree.

Using the idea that any graph $G = (V(G), E(G))$ is a minor of the complete graph $K_{|V(G)|}$, we obtain the following:

Theorem 3. *Graph Connectivity is not in $\mathrm{Mon}\Sigma_1^1$ even in the presence of an irreflexive symmetric built-in relation that does not have an arbitrary, but fixed finite graph $H = (V(H), E(H))$ as a minor.*

Thus, we have negative definability results for all classes of graphs given in Table 1, since all classes of graphs listed there have at least one forbidden minor.

When we consider classes of simple graphs which are minor-closed (i.e. such a class contains all minors of isomorphic copies of its members), it follows from the results of Robertson and Seymour [16,17] (see also [4], Corollary 12.5.3) that the minimal set of forbidden minors of that class is finite. Thus, we can derive a more general result for minor-closed classes of symmetric irreflexive binary relations:

Corollary 2. *Graph connectivity is not in $\mathrm{Mon}\Sigma_1^1$, even in the presence of an arbitrary built-in relation which belongs to a minor-closed class of irreflexive symmetric binary relations which is not equal to the class of all irreflexive symmetric binary relations.*

As already mentioned in Table 1, the class of graphs that do not have the complete graph K_k as a minor for a certain $k \in \mathbb{N}$ contains many other classes of graphs. For these, our result also holds. We give one corollary:

Corollary 3. *Graph Connectivity is not in $\mathrm{Mon}\Sigma_1^1$ even in the presence of an irreflexive symmetric built-in relation that has bounded tree-width.*

Classes of Graphs	Tree-Width
Trees / Forests	1
Chordal Graphs with maximal clique size k	$< k$
Partial k-Trees	$< k$
Series-Parallel Graphs	< 3
Outerplanar Graphs	< 3
Halin Graphs	< 4
Complete Graph K_k	$k - 1$

Table 2. Collection of some known lower bounds for tree-width.

Table 2 gives an overview of classes of graphs that are captured by the class of graphs of tree-width k.

References

1. M. Ajtai and R. Fagin: *Reachability is harder for directed than for undirected finite graphs*, in Journal of Symbolic Logic, 55 (1), pp. 113-150, 1990. 128
2. S. Arnborg, J. Lagergren and D. Seese: *Easy Problems for Tree-Decomposable Graphs*, Journal of Algorithms vol. 12, pp. 308-340, 1991.
3. C. Chang and H. Keisler: *Model Theory* North-Holland, 1974, 3rd edition, 1990.
4. R. Diestel: *Graph Theory*, Springer-Verlag, 1997. 127, 140
5. H.-D. Ebbinghaus and J. Flum: *Finite Model Theory*, Springer-Verlag, 1995. 127
6. R. Fagin: *Generalized First-Order spectra and polynomial-time recognizable sets*, in Complexity of Computation, (ed. R. Karp) SIAM-AMS Proc. 7, pp. 27-41, 1974. 126, 126
7. R. Fagin: *Monadic generalized spectra*, in Zeitschrift für mathematische Logik und Grundlagen der Mathematik vol. 21, pp. 89-96, 1975. 127
8. R. Fagin: *Comparing the Power of Games on Graphs*, in Mathematical Logic Quarterly vol. 43, pp. 431-455, 1997.
9. R. Fagin, L. Stockmeyer and M. Vardi: *On Monadic NP vs. Monadic co-NP*, in Information and Computation vol. 120 (1), pp. 78-92, 1995. 127, 127, 140
10. W. Hodges: *Model Theory*, Cambridge University Press, 1993.
11. N. Immerman: *Languages That Capture Complexity Classes*, in SIAM Journal of Computing vol. 16 (4), pp. 760-778, 1987. 126
12. M. Kreidler and D. Seese: *Monadic NP and Built-in Trees*, in Proc. CSL'96, LNCS 1258, Springer-Verlag, pp. 260-274, 1997. 127, 129
13. C. H. Papadimitriou: *Computational Complexity*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1994. 127
14. N. Robertson and P. Seymour: *Graph minors - a survey*, in I. Anderson (Ed.), Surveys in Combinatorics, Cambridge University Press, pp. 153-171, 1985.
15. N. Robertson and P. D. Seymour: *Graph Minors IV: Tree-Width and Well-Quasi-Ordering*, Journal of Comb. Theory Series B, vol. 48, pp. 227-254, 1990. 127
16. N. Robertson and P. Seymour: *Graph minors XII: Excluding a non-planar graph*, in Journal of Combinatorial Theory Series B vol. 64, pp. 240-272, 1995. 127, 140
17. N. Robertson and P. Seymour: *Graph minors XIII: The disjoint path problem*, in Journal of Combinatorial Theory Series B vol. 63, 1995, pp. 65-110, 1995. 127, 127, 140
18. T. Schwentick: *On Winning Strategies in Ehrenfeucht Games and Monadic NP*, in Annals of Pure and Applied Logic vol. 79, pp. 61-92, 1996. 127, 129, 136, 136
19. D. Seese and W. Wessel: *Graphminoren und Gitter: Zu einigen Arbeiten von N. Robertson und P. Seymour*, in *Graphentheorie III* (edts. K. Wagner and R. Bodendiek), BI Wissenschaftsverlag, 1993.
20. S. Shelah: *Classification Theory and the Number of Non-Isomorphic Models*, Studies in Logic series, North-Holland, 1978. 130, 130
21. K. Wagner: *Über eine Eigenschaft der ebenen Komplexe*, in Mathematische Annalen vol. 114, pp. 570-590, 1937.

Invariant Definability and P/*poly*

J.A. Makowsky *

Department of Computer Science
Technion—Israel Institute of Technology
IL-32000 Haifa, Israel
janos@cs.technion.ac.il

Abstract. We look at various uniform and non-uniform complexity classes within $\mathbf{P}/poly$ and its variations $\mathbf{L}/poly$, $\mathbf{NL}/poly$, $\mathbf{NP}/poly$ and $\mathbf{PSpace}/poly$, and look for analogues of the Ajtai-Immerman theorem which characterizes $\mathbf{AC}_0$ as the non-uniformly First Order Definable classes of finite structures. We have previously observed that the Ajtai-Immerman theorem can be rephrased in terms of *invariant definability*: A class of finite structures is FOL invariantly definable iff it is in $\mathbf{AC}_0$. Invariant definability is a notion closely related to but different from *implicit definability* and *Δ-definability*. Its exact relationship to these other notions of definability has been determined in [Mak97].
Our first results are a slight generalization of similar results due to Molzan and can be stated as follows: let $\mathbf{C}$ be one of $\mathbf{L}, \mathbf{NL}, \mathbf{P}$, $\mathbf{NP}$, $\mathbf{PSpace}$ and $\mathcal{L}$ be a logic which captures $\mathbf{C}$ on ordered structures. Then the non-uniform $\mathcal{L}$-invariantly definable classes of (not necessarily ordered) finite structures are exactly the classes in $\mathbf{C}/poly$. We also consider uniformity conditions imposed on invariant definability and relate them to uniformity conditions on the advice sequences. This approach is different from imposing uniformity conditions on the circuit families.
The significance of our investigation is conceptual, rather than technical: We identify exactly the logical analogue of uniform and non-uniform complexity classes.

* Supported by the German Israeli Foundation (GIF) and the Technion Foundation for Promotion of Research in the framework of the project *Definability and Computability*

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 142–158, 1999.

1 Introduction and survey of results

In this paper we propose a way to unify various notions of definability from model theoretic logic, formal language theory and descriptive complexity theory. Their common theme is definability using *auxiliary* predicates. This paper continues the analysis in [Mak97] of the notion of *invariant definability*, the definition of which will be given below. Special cases of invariant definability have been used, more or less explicitly, in various contexts of finite model theory and descriptive complexity theory. We assume the reader is familiar with the basics of complexity theory as given in [Joh90,BS90] and of descriptive complexity theory as given in [EF95,Imm9x]. Unless otherwise stated all structures considered are *finite*.

We see the main merit of this paper in its synthesis of various observations concerning the logical description of uniform and non-uniform circuit complexity classes within the polynomial hierarchy and their relationship with the corresponding Turing complexity classes. The purely logical aspects of invariant definability and its relationship with Turing complexity classes were already presented in [Mak97]. Related and overlapping work was incdependently obtained in the widely overlooked[1] *beautiful paper by Molzan [Mol90], and more recently, in [AB97].*

We deal with logics $\mathcal{L}$ such as First Order Logic FOL, possibly extended by a determinsitic (non-determinsitic) transitive closure operator DTC (TC), denoted by $FOL{+}DTC$ ($FOL{+}TC$); (Existential) Second Order Logic ($ESOL$) SOL; and Inflationary (Partial) Fixed Point Logic IFP (PFP). For detailed definitions we refer to the textbook [EF95]. Vocabularies are sets of relation symbols. They are denoted by τ, σ or $\bar{R}, \bar{S}$ with $\bar{R} = \{R_1, \ldots R_n\}$, depending whether we emphasize the vocabulary as an abstract entity or as an explicit set of relation symbols. For a logic $\mathcal{L}$ we denote by $\mathcal{L}(\tau)$ ($\mathcal{L}(\bar{R})$) the set of τ-sentences of $\mathcal{L}$, by for $\phi \in \mathcal{L}(\tau)$ by $MOD(\phi)$ the class of finite τ-structures $\mathfrak{A}$ such that $\mathfrak{A} \models \phi$, and by $DEF(\mathcal{L}(\tau))$ the classes K of finite τ-structures of the form $K = MOD(\phi)$ for some sentence $\phi \in \mathcal{L}(\tau)$. For two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ we write $\mathcal{L}_1 \subseteq \mathcal{L}_2$ if $DEF(\mathcal{L}_1(\tau)) \subseteq DEF(\mathcal{L}_2(\tau))$ for every vocabulary τ.

1.1 Invariant definability: Definitions

Let $\bar{R}$ and $\bar{S}$ be two disjoint sets of relation symbols and let $\mathcal{L}$ be a logic. Furthermore, let K_0 and K be two classes of (finite) $\bar{R}$, respectively $\bar{S}$ structures, both closed under isomorphisms. If $\mathfrak{A}$ is an $\bar{R} \cup \bar{S}$-structure we denote by $\mathfrak{A}_{|\bar{S}}$ the *restriction of* $\mathfrak{A}$ *to interpretations of* $\bar{S}$. Conversely, $\mathfrak{A}$ is called an *expansion of* $\mathfrak{A}_{|\bar{S}}$. To introduce our notions of invariant definability we first make precise what we mean by the statement *a formula* $\phi(\bar{R}, \bar{S})$ *with predicate symbols* $\bar{R}$ *and* $\bar{S}$ *does not depend on the interpretation of* $\bar{R}$ *over* K_0. This holds if for any two (finite) $\bar{R} \cup \bar{S}$-structures $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A}|_{\bar{S}} \cong \mathcal{B}|_{\bar{S}}$ and $\mathcal{A}|_{\bar{R}}, \mathcal{B}|_{\bar{R}} \in K_0$ we have that $\mathcal{A} \models \phi$ iff $\mathcal{B} \models \phi$.

[1] Our result was also obtained independently of [Mol90]. The paper was pointed out to us by an anonymous referee of CCC'98

Definition 1. *Let $\phi(\bar{R}, \bar{S})$ a formula in $\mathcal{L}(\bar{R} \cup \bar{S})$.*

(i) ϕ defines a class of $\bar{S}$-structures K invariantly over K_0 (or just K_0-invariantly) if for any (finite) $\bar{R} \cup \bar{S}$-structure $\mathcal{A}$ with $\mathcal{A}|_{\bar{R}} \in K_0$ we have that $\mathcal{A} \models \phi$ iff $\mathcal{A}|_{\bar{S}} \in K$.

(ii) K is invariantly definable over K_0, if there is a ϕ which defines K invariantly over K_0.

(iii) If, additionally, K_0 is definable by an $\mathcal{L}(\bar{R})$ formula ψ, we say that K is strictly invariantly definable over K_0.
If we want to stress that K_0 is not required to be $\mathcal{L}(\bar{R})$-definable, we say that K is weakly invariantly definable over K_0.

K_0 gives us the auxiliary relations which interpret the auxiliary predicates $\bar{R}$. One easily sees that if ϕ defines a class of $\bar{S}$-structures K invariantly over K_0 then ϕ does not depend on the interpretation of $\bar{R}$ over K_0. This is the reason we call this notion of definability *invariant*.

Let $\mathfrak{H}$ be a class of classes of finite structures which all are closed under isomorphisms. Typically, $\mathfrak{H}$ could be given as the class of all K definable in some logic $\mathcal{L}$ or the class of languages (or classes of finite structures) recognizable in some complexity class. K is $\mathfrak{H}$-invariantly definable in $\mathcal{L}$ if there is a $K_0 \in \mathfrak{H}$ and a ϕ such that ϕ defines K K_0-invariantly. $\mathfrak{H}$ defines the range of possible K_0's which are allowed as auxiliary classes K_0.

Various $\mathfrak{H}$ have been considered in the literature. Typically $\mathfrak{H}$ could be one of the following:

(i) $\mathfrak{H}$ consists of all *numerical relations (predicates)*. Numerical relations are relations whose isomorphism type is uniquely determined by the cardinality of the underlying universe. In model theoretic terms these are the *fin-categorical* classes K_0 of τ-structures where τ consists of a single relation symbol. A class of finite τ-structures is *fin-categorical* if any two structures of the same finite cardinality are τ-isomorphic. Immerman, [Imm87, theorem 6.2], considers this case for the notion of FOL definability using auxiliary predicates in his characterization of $\mathbf{AC}_0$. Molzan [Mol90, theorem 3.3] considers this case in his characterization of non-uniform $\mathbf{L}, \mathbf{NL}$ and $\mathbf{P}$. Gurevich and Lewis, [GL84] use a notion of definability using free predicate variables, which is equivalent to this case as well.

(ii) $\mathfrak{H}$ consists of $ISO(\tau)$, i.e. all classes K_0 of finite τ-structures closed under τ-isomorphisms. Hence it includes the numerical relations. Ajtai uses this notion of invariant $\mathfrak{H}$-definability in FOL for his non-definability results in [Ajt83].

(iii) $\mathfrak{H}$ contains exactly one numerical class K_0. In [Imm87] this notion is used with $K_0 = SUCC$ the class of finite successor relations, to characterize the complexity classes $\mathbf{L}, \mathbf{NL}$ and $\mathbf{P}$. With $K_0 = ORD$, the class of linear orderings the same characterizations are obtained.

(iv) $\mathfrak{H}$ consists of all the classes K_0 which are closed under substructures. In [Mak97] we have characterized the numerical predicates which are closed under substructures and have found that the linear orderings are in a certain sense the only such class.

(v) $\mathfrak{H}$ consists of numerical predicates which are constructible or recognizable within some complexity bound, such as polynomial time or logarithmic space. Such *uniformity* conditions were considered in a series of papers by Barrington, Compton, Gurevich, Immerman, Lewis, Straubing, Thérien, e.g. [GL84,BCST92,BIS90,BI94,Str94].

(vi) $\mathfrak{H}$ consists of $DEF(\mathcal{L}_1)$, i.e. all the classes definable in a logic $\mathcal{L}_1$ (without auxiliary predicates). ORD is FOL-definable in this sense ($ORD \in DEF(FOL)$), but $SUCC$ is not (as we can not express the absence of cycles). In section 4 we shall discuss the impact of this choice for various $\mathcal{L}_1$. We shall see that this notion is rather robust under variations of $\mathcal{L}_1$. For $\mathcal{L}_1$ with $DEF(FOL) \subseteq DEF(\mathcal{L}_1) \subseteq DEF(ESOL)$ we get the same $DEF(\mathcal{L}_1)$-invariantly definable classes in $\mathcal{L}$.

1.2 Guiding examples

Recall that denote by ORD the class of finite linear orders.

Example 1. **E***ven* is the class of finite sets (= structures over the empty vocabulary) of even cardinality. **E***ven* is in $\mathbf{AC}_0$, but is not definable in IFP or even PFP, cf. [EF95]. It is easy to see that **E***ven* is K_0-invariantly FOL-definable with K_0 the class of ordered structures with an additional unary predicate P which colors every second element in the order.

Example 2. Let K be a class of $\bar{S}$-structures and K_{ORD} the class of $\bar{S} \cup \{R_<\}$-structures which are ordered expansions of structures in K. Assume that $\phi(\bar{S}, R_<)$ defines K ORD-invariantly in a logic $\mathcal{L}$. In particular we assume w.l.o.g. that ϕ also says that $R_<$ is a linear order. Then $\phi(\bar{S}, R_<)$ defines K_{ORD}. The contrapositive is useful: $\mathbf{E}even_{ORD}$ is known not to be FOL-definable. A simple Pebble game will establish this, cf. [EF95]. We conclude that **E***ven* is not ORD-invariantly FOL-definable.

Example 3. **P***arity* is the class of finite structures for one unary predicate which has even cardinality. Clearly **P***arity* is in **L**, but it is not in $\mathbf{AC}_0$, cf. [BS90], by the famous result of Ajtai [Ajt83] and Furst, Saxe and Sipser [FSS84]. As in the previous example, it is easy to see that **P***arity* is not ORD-invariantly FOL-definable. In contrast, **P***arity* is ORD-invariantly $FOL + DTC$-definable.

Example 4. **s-t-conn** (also called **Gap**) is the class of directed graphs $\langle V, E, s, t\rangle$ with two distinguished vertices $s, t \in V$ and edge relation E such that there is an E-path from s to t. **s-t-conn** is $FOL + TC$-definable, and hence IFP-definable. Ajtai [Ajt89] has shown that **s-t-conn** is not in $\mathbf{AC}_0$.

1.3 The Immerman–Sazonov–Vardi Theorem revisited

The by now classical results in descriptive complexity theory, cf. the monograph [EF95] and [Saz80], show that various complexity classes such a polynomial time (space) can be captured by logics in the presence of a linear order or labeling of the underlying structures. Here we understand under $\mathcal{L}$ *captures a complexity class* $\mathbf{C}$ *on ordered structures* that for every class of ordered $\bar{S}$-structures K we have that $K \in \mathbf{C}$ iff there is an $\mathcal{L}(\bar{S})$-sentence ϕ with $\mathfrak{A} \in K$ iff $\mathfrak{A} \models \phi$.

The following argument shows that capturing complexity classes by logics in the presence of ordering can be recast in terms of invariant definability.

Let $\mathcal{L}$ be a logic (not weaker than FOL), $\mathbf{C}$ a complexity class (not below Logspace $\mathbf{L}$, say). If $\mathcal{L}$ captures $\mathbf{C}$ on ordered structures, then it just follows that a class K of (not necessarily ordered) $\bar{S}$-structures is in $\mathbf{C}$ iff it is ORD-invariantly $\mathcal{L}(\bar{S} \cup \{R_<\})$-definable. For the interesting direction, assume $K \in \mathbf{C}$. Then also $K_{ORD} \in \mathbf{C}$, hence K_{ORD} (from Example 2) is $\mathcal{L}(\bar{S} \cup \{R_<\})$-definable by a sentence ϕ. But by the definition of K_{ORD} it contains all the possible ordered expansions of structures in K, hence ϕ defines K invariantly over ORD. Conversely, if K is ORD-invariantly definable in $\mathcal{L}$ by some ϕ which also asserts that $R_<$ is a linear order, the same ϕ defines K_{ORD}. Therefore K_{ORD}, and a fortiori K, are in $\mathbf{P}$. So we have

Theorem 1 ((Immerman, Sazonov, Vardi)). *Let K_0 be the class of finite linear orders, $K_0 = ORD$, and K be a class of $\bar{R}$ structures. Then K is in $\mathbf{P}$ iff there is a $\phi \in IFP(\bar{R} \cup \{R_<\})$ which defines K ORD-invariantly.*

1.4 The Ajtai–Immerman Theorem revisited

In [Mak97] we showed the following variation of [BS90, theorem 3.21], there attributed to Ajtai and Immerman. Actually, Immerman showed this in [Imm87, theorem 6.2] for definability over numerical relations.

Theorem 2 ((Ajtai, Immerman)). *A class of τ-structures is in $\mathbf{AC}_0$ iff it is $ISO(\sigma)$-invariantly definable in First Order Logic.*

To see this we exploited a close relationship between the notion of invariant definability and a notion of definability previously studied by Ajtai [Ajt83,Ajt89]. Ajtai's notion of definability was introduced to prove negative results in circuit complexity and is stronger than invariant definability, cf. [BS90]. It is easy to see that even non-recursive languages may be Ajtai-definable (as well as ISO-invariantly definable). In particular Ajtai showed that neither **Parity** nor **s-t-conn** (**G***ap*) are $\mathbf{AC}_0$-definable, hence they are not weakly invariantly FOL-definable.

1.5 Capturing non-uniform complexity classes

We want to explore the connection between invariant definability and descriptive complexity for non-uniform circuit complexity classes. Let $f : \mathbb{N} \to \mathbb{N}$. An $f(n)$

advice sequence is a sequence of binary strings $A = (a_1, a_2, \ldots)$ where $\mid a_n \mid \leq f(n)$. For a language $B \subseteq \{0,1,\sharp\}^*$ let $B@A = \{x : x\sharp a_{|x|} \in B\}$. For a space or time bounded complexity class $\mathbf{C}$ we define, following [BS90],

$$\mathbf{C}/f = \{B@A : B \in \mathbf{C} \text{ and } A \text{ is an } f(n) \text{ advice sequence for some } A\}$$

Finally $\mathbf{C}/poly = \bigcup_k \mathbf{C}/n^k$. We are mostly interested in the non-uniform analogues of $\mathbf{P}$ (deterministic polynomial time), $\mathbf{NP}$ (non-deterministic polynomial time), $\mathbf{L}$ (deterministic logarithmic space), $\mathbf{NL}$ (non-deterministic logarithmic space) and $\mathbf{PSpace}$ (deterministic polynomial space), which are denoted by $\mathbf{P}/poly$, $\mathbf{NP}/poly$, $\mathbf{L}/poly$, $\mathbf{NL}/poly$ and $\mathbf{PSpace}/poly$ respectively.

We generalize this to finite structures as follows. Let $\bar{R}$ and $\bar{S}$ be two disjoint sets of relation symbols. Let $c : Str \to \{0,1\}^*$ be a coding of finite structures as words and K a class of $\bar{S}$-structures. An $\bar{R}$-advice sequence is is a sequence of $\bar{R}$-structures $(\mathfrak{A}_1, \mathfrak{A}_2, \ldots)$ such that the universe of $\mathfrak{A}_n$ has cardinality n. Let $B_K = \{c(\mathfrak{B}) : \mathfrak{B} \in K\}$ and $A = \{c(\mathfrak{A}_i) : i \in \mathbb{N}\}$. We now say that a class of $\bar{S}$-structures $K \in \mathbf{C}/poly$ if there exists an $\bar{R}$-advice sequence such that $B_K@A \in \mathbf{C}$ Furthermore we note that $B_K@A$ can be viewed as the encodings of $\bar{R} \cup \bar{S}$-structures $\mathfrak{C}$ such that $\mathfrak{C}_{|\bar{R}} \cong \mathfrak{A}_i$ for some $i \in \mathbb{N}$ and $\mathfrak{C}_{|\bar{S}} \in K$. Clearly, $\bar{R}$-advice sequences are always polynomially bounded.

The first result of this paper is the following (easy) theorem. For numerical relations this was already shown by Molzan [Mol90], a paper which has been widely overlooked.

Theorem 3. *Let K be a class of finite $\bar{R}$-structures.*

(i) K is in $\mathbf{P}/poly$ iff K is $ISO(\bar{S})$-invariantly definable in IFP.
(ii) This generalizes to:
K is in $\mathbf{L}/poly$ ($\mathbf{NL}/poly$, $\mathbf{NP}/poly$, $\mathbf{PSpace}/poly$) iff K is $ISO(\bar{S})$-invariantly definable in $FOL + DTC$ ($FOL + TC$, $ESOL$, PFP).

The proof is given in section 2 and uses two lemmas on advice sequences and the Immerman–Sazonov–Vardi theorem.

1.6 Implicit and Δ-definability

The reason we introduced in [Mak97] $\mathfrak{H}$-invariant $\mathcal{L}$-definability for its own sake lies in the fact that in the case with $\mathfrak{H} = DEF(\mathcal{L})$ and $\mathcal{L}$ a sublogic of Second Order Logic, it gives rise to well defined logics with its definable classes of structures lying well within the polynomial hierarchy (in $\mathbf{NP} \cap \mathbf{CoNP}$ in the case of first order logic). In section 3 we shall define the notion of implicit definability and Δ-definability and state precisely the relationships between these three.

1.7 Uniformity conditions

It is now natural to ask what happens if we restrict our advice sequences to definable or computable sequences. As an advice sequence $A = (a_1, a_2, \ldots)$ is a

function the range of which could be finite, we look at the complexity (definability) of the set $\{1^n a_n : n \in \mathbb{N}\}$, i.e.the complexity of the graph of the advice sequence. Let us denote by $\mathbf{P}/\mathbf{P}$ ($\mathbf{P}/\mathbf{NP}$, $\mathbf{P}/FOL$) the class of languages recognized by polynomial time Turing machines with advice sequences in $\mathbf{P}$ ($\mathbf{NP}$, FOL-definable). We can generalize this notation to $\mathbf{C}/\mathbf{D}$, for pairs of complexity classes $\mathbf{C}, \mathbf{D}$ or to $\mathbf{C}/\mathcal{L}$ for a complexity class $\mathbf{C}$ and a logic $\mathcal{L}$. In our general notation this is just $\mathbf{C}/\mathfrak{H}$ with $\mathfrak{H} = \mathbf{D}$ or $\mathfrak{H} = DEF(\mathcal{L})$.

Theorem 3 can easily be generalized to

Theorem 4. *A class of $\bar{S}$-structures K is in $\mathbf{P}/\mathbf{P}$ ($\mathbf{P}/\mathbf{NP}$, $\mathbf{P}/FOL$) iff K is $\mathbf{P}$- ($\mathbf{NP}$-, FOL-) -invariantly definable in IFP.*

In section 4 we shall further investigate how the restrictions on the advice sequences affect the circuit complexity classes.

1.8 Significance of the results

The results of this paper are technically not difficult and they are primarily of *conceptual interest* inasmuch as they exhibit a new feature which allows us to capture both a Turing complexity class, say $\mathbf{P}$, and its non-uniform circuit complexity analogue, here $\mathbf{P}/poly$, with variations of invariant definability in a logic, here IFP.

In particular we can see now the uniformity/non-uniformity of $\mathbf{P}$ and $\mathbf{P}/poly$ in the following way:

Uniformly order invariantly definable: **There is a class** K_0, actually $K_0 = ORD$, such that **for every class** of finite $\bar{R}$-structures K we have $K \in \mathbf{P}$ iff K is K_0 invariantly definable in IFP. In other words $\mathbf{P} = \mathbf{P}/\mathfrak{H}$ with $\mathfrak{H} = \{ORD\}$.

Non-uniformly invariantly definable: **For every class** of finite $\bar{R}$-structures K we have $K \in \mathbf{P}/poly$ iff **there is class** K_0 such that K is K_0 invariantly definable in IFP.

Related and similar results have been obtained by various authors and will be discussed in more detail in section 5.

Ajtai has succeeded in showing that **Parity** is not in $\mathbf{AC}_0$ using logico-combinatorial methods, [Ajt83]. In fact he showed that **Parity** is not invariantly FOL-definable. The question is whether Ajtai's methods can be extended to some other logics such as IFP ? A positive answer would allow us to exhibit (recursive, low complexity) classes K of structures which were not in $\mathbf{P}/poly$.

Remark 1. L. Hella has noted that every class of finite graphs closed under isomorphism is ORD-invariantly definable in $\mathcal{L}^2_{\infty,\omega}$.

Acknowledgments

I am indebted to Eric Rosen with whom I have discussed this work extensively during his stay at the Technion as a Postdoctoral fellow. He pointed out to me the relevance of Ajtai's work to invariant definability. I also wish to thank A. Dawar, G. Gottlob, Y. Gurevich, Y. Ishay, K.-J. Lange, L. Hella, M. Otto and E. Ravve for various insightful comments on earlier drafts of this paper. Finally, I would like to thank an insistent referee for his efforts and insight, which contributed to the present form of the paper.

2 Characterizing P/*poly* and its friends

In this section we sketch a proof of theorem 3. The proof uses two lemmas on advice sequences. A polynomial advice sequence W is a sequence of binary strings $w_i, i \in \mathbb{N}$ where the size of w_i is polynomially bounded in i. We think of w_i as coding a set of relations interpreting the relation symbols of $\bar{R}_0$ on the universe $\{1, \ldots, i\}$.

Let $\bar{R} = \bar{R}_0 \cup \{R_<\}$ where $R_<$ is a binary relation symbol whose interpretation is the natural linear order on the universe of the structure. Let K_W be the class of (ordered) $\bar{R}$-structures with the interpretation of the relation symbols in $\bar{R}_0$ encoded by W. K_W has, up to isomorphism, exactly one member in each cardinality.

Let **C** be a complexity class. In the notation of the previous section, we say that a class of $\bar{S}$-structures K *is in* **C**/*poly with polynomial advice sequence* W if $B_K@W \in \mathbf{C}$.

Lemma 1. *Let K be a class of finite $\bar{S}$-structures. If $K \in \mathbf{P}/poly$ with polynomial advice sequence W then K is K_W-invariantly definable in IFP.*

Proof. (Sketch) Assume $K \in \mathbf{P}/poly$ with polynomial advice sequence W. So the language $B_K@W \in \mathbf{P}$ and a word $x \in B_K@W$ codes an (ordered) $\bar{R} \cup \bar{S}$-structure $\mathfrak{A}_x$ for a suitable chosen set of relation symbols $\bar{R}$. By the Immerman-Vardi-Sazonov Theorem there is an $\bar{R} \cup \bar{S}$-sentence in IFP which defines $\{\mathfrak{A}_x : x \in B_K@W\}$. As W is an advice sequence, ϕ does not depend on $\bar{R}$ over $K_W = \{(\mathfrak{A}_x)_{|\bar{R}} : \mathfrak{A}_x \models \phi\}$ and ϕ defines K K_W-invariantly. □

Recall that a class K of finite $\bar{R}$-structures is *fin-categorical* if K has models in each finite cardinality and any two $\mathfrak{A}, \mathfrak{B} \in K$ of the same cardinality are isomorphic. If K is fin-categorical, it can be used as an advice sequence. Conversely, every $\bar{R}$-advice sequence can be viewed as an fin-categorical class of finite structures. From a model theoretic point of view, there is a one-one correspondence between advice sequences and fin-categorical classes, because the structure $\mathfrak{A}_i$ depends only on the cardinality of its universe. Clearly, any class K with models in every finite cardinality contains a fin-categorical subclass K_1.

Lemma 2. *Let K be a class of finite $\bar{S}$-structures and K_0 be a class of finite $\bar{R}$-structures. If K is K_0-invariantly definable in IFP and $K_1 \subseteq K_0$ is fin-categorical then $K \in \mathbf{P}/poly$ with $\bar{R}$-advice sequence defined by K_1.*

Proof. (Sketch) Let ϕ be the $IFP(\bar{R} \cup \bar{S})$-sentence which defines K K_0-invariantly. Then the same formula defines K K_1-invariantly. Hence $B_K@W_{K_1} \in \mathbf{P}$. As K_1 is fin-categorical, it can be viewed as an $\bar{R}$-advice sequence. □

Assume $\mathbf{C}$ is any complexity class which is captured by a logic $\mathcal{L}$ over ordered structures. Then both lemmas hold also if we replace $\mathbf{P}$ by $\mathbf{C}$ and IFP by $\mathcal{L}$. Theorem 3 now follows immediately.

3 Implicit, invariant and Δ-definability

We recall the following standard definitions, cf. [BF85]

Definition 2. *Let $\mathcal{L}$ be a logic. Let K be a class of $\bar{S}$-structures.*

(i) K *is a* $\mathcal{L}$-projective class *if there is a vocabulary $\bar{R}$ and a sentence ϕ in $\mathcal{L}(\bar{R} \sqcup \bar{S})$ such that $\mathfrak{A} \in K$ iff there is an expansion of $\mathfrak{A}$ to a $\bar{R} \cup \bar{S}$-structure $\bar{\mathfrak{A}}$ such that $\bar{\mathfrak{A}} \models \phi$.*

(ii) $\Delta(\mathcal{L})$ is the family of classes K of σ-structures such that both K and its complement are $\mathcal{L}$-projective classes.

(iii) A logic $\mathcal{L}$ is Δ-closed *if every $K \in \Delta(\mathcal{L})$ is definable in $\mathcal{L}$.*

FOL is Δ-closed if we allow finite and infinite structures. FOL is not Δ-closed if we only allow finite structures. IFP on finite structures is not Δ-closed: It is well known, cf. [EF95], that $EVEN$ is not definable in IFP, but it is easily seen to be in $\Delta(FOL)$, and hence in $\Delta(IFP)$. Second Order Logic is Δ-closed on finite structures. This follows from the closure under quantification over relation variables.

First we note that our strict invariantly definable classes are special cases of projective classes.

Lemma 3. *Let K_0 be $\mathcal{L}(\bar{R})$-definable by a sentence θ. If a class of $\bar{S}$-structures K is strictly K_0-invariantly $\mathcal{L}$-definable by the $\mathcal{L}(\bar{R} \sqcup \bar{S})$-sentence ϕ then*

(i) $\mathfrak{A} \in K$ *iff* $\mathfrak{A} \models \exists \bar{R}(\theta(\bar{R}) \wedge \phi(\bar{R}, \bar{S}))$

(ii) $\mathfrak{A} \in K$ *iff* $\mathfrak{A} \models \forall \bar{R}(\theta(\bar{R}) \rightarrow \phi(\bar{R}, \bar{S}))$

In other words, if K is strictly K_0-invariantly $\mathcal{L}$-definable, then both K and its complement are $\mathcal{L}$-projective classes. Clearly, **Parity** *is in $\Delta(FOL)$, but by example 3, it is not strictly invariantly FOL-definable.*

Given a logic $\mathcal{L}$ the logic $IMP(\mathcal{L}) \subseteq \Delta(\mathcal{L})$ is defined in [EF95] as follows:

Definition 3 ((The logic $IMP(\mathcal{L})$)). *An $IMP(\mathcal{L})(\tau)$-formula $\phi(\bar{x})$ with free variables among $\bar{x}$ is a tuple*

$$(\psi_1(R_1), \ldots, \psi_m(R_m), \psi(\bar{x}, \bar{R}))$$

where each ψ_i is an $\mathcal{L}(\tau \cup \{R_i\})$-sentence and ψ is an $\mathcal{L}(\tau \cup \{\bar{R}\})$-formula with free variables among $\bar{x}$. Furthermore, we require that every finite τ-structure $\mathcal{A}$

has exactly one expansion to an $(\tau \cup \{\bar{R}\})$*-structure* $\bar{\mathcal{A}}$ *satisfying the conjunction* $\psi_1(R_1) \wedge \ldots \wedge \psi_m(R_m)$*. Now the meaning of* ϕ *is given by* $\mathcal{A} \models \phi(\bar{a})$ *iff* $\bar{\mathcal{A}} \models \psi(\bar{a}, \bar{R})$ *where the* R_i *in* $\mathcal{A}$ *are interpreted to satisfy* $\psi_i(R_i)$*.*

It is easy to see that $IMP(\mathcal{L}) \subseteq \Delta(\mathcal{L})$.

Now we want to show that invariant definability is incomparable with implicit definability.

Definition 4. *A* τ*–structure* $\mathcal{A}$ *is* trivial *if every permutation of the universe of* $\mathcal{A}$ *is a* τ*–automorphism. A relation* R^A *on* A *is trivial if the structure* $< A, R^A >$ *is trivial.*

The following is well known and can be proved using pebble games:

Lemma 4. *Assume the vocabulary* τ *is empty and* $\mathcal{L}$ *is a sublogic of finite variable infinitary logic* $\mathcal{L}^{\omega}_{\omega_1,\omega}$*. Then the implicit definitions just define trivial relations. Furthermore, every implictly* FOL*-definable relation is already* FOL*-definable.*

The following theorem was proved by Kolaitis [Kol90], cf. also [EF95][corollary 7.5.9].

Theorem 5 ((Kolaitis)). $IFP \subseteq IMP(FOL)$ *over finite structures, i.e. every* IFP*-definable class of finite structures* K *is also* $IMP(FOL)$*-definable.*

Remark 2. Theorem 5 is false if we allow infinite structures. To see this we note that IFP on infinite structures properly extends FOL and is not compact, whereas $IMP(FOL) = FOL$.

We denote by $INV(FOL)$ the class of classes of finite structures which are strictly invariantly definable in FOL. We already know that both $IMP(FOL) \subseteq \Delta(FOL)$ and $INV(FOL) \subseteq \Delta(FOL)$.

In [Mak97] we showed

Proposition 1. $INV(FOL)$ *and* $IMP(FOL)$ *are incomparable. More precisely:* **Even** $\in INV(FOL) - IMP(FOL)$ *and* **s-t-conn** $\in IMP(FOL) - INV(FOL)$.

Proof. **Even** $\in INV(FOL) - IMP(FOL)$ follows from example 1 and lemma 4. **s-t-conn** $\in IMP(FOL) - INV(FOL)$ follows from example 4, theorem 2 and theorem 5. □

The notion of invariant definability is not interesting for traditional First Order Logic (FOL) over arbitrary τ-structures, as the following shows:

Proposition 2. *If a class of* σ*-structures* K *is strictly* K_0*-invariantly* FOL*-definable on all (finite and infinite) structures then* K *is already* FOL*-definable. Moreover, If* $\mathcal{L}$ *is a* Δ*-closed logic, i.e. if* $\Delta(\mathcal{L}) = \mathcal{L}$*, then every class* $K \in INV(\mathcal{L})$ *is already* $\mathcal{L}$*-definable.*

Proof: By lemma 3, both K and its complement are projective classes in FOL, and therefore, by Craig's Interpolation Theorem for FOL, K is FOL-definable. □

4 Uniformity conditions

As we said in the introduction it is now natural to ask what happens if we restrict our advice sequences to *definable* or *computable* sequences. In view of the one-one correspondence between fin-categorical classes and advice sequences, we say that an $\bar{R}$-advice sequence $(\mathfrak{A}_i, i \in \mathbb{N})$ is in a complexity class $\mathbf{C}$ if the class of structures $K = \{\mathfrak{A}_i : i \in \mathbb{N}\} \in \mathbf{C}$. Similarly, we say that the $\bar{R}$-advice sequence $(\mathfrak{A}_i, i \in \mathbb{N})$ is definable in a logic $\mathcal{L}$ if there exists a $\bar{R}$-sentence ϕ in $\mathcal{L}$ such that $\mathfrak{B} \in K$ iff $\mathfrak{B} \models \phi$.

Let $\mathbf{C}, \mathbf{D}$ be complexity classes and $\mathcal{L}$ be a logic. Let us denote by $\mathbf{C}/\mathbf{D}$ ($\mathbf{C}/\mathcal{L}$) the class of languages recognized by machines from $\mathbf{C}$ with advice sequences in $\mathbf{D}$ (definable in $\mathcal{L}$).

Note that this notion of uniformity imposed on the complexity of the advice sequence is *different* from the usual uniformity imposed on the complexity of the circuits.

It is well known, [BS90, Theorem 2.2], that $K \in \mathbf{P}/poly$ iff K has polynomial circuit complexity. Our uniformity condition concerns in the complexity or definability of the advice sequence, not of the circuits. A priori, this is a weaker assumption, as it does not affect the circuit size. For uniformity conditions imposed on the circuit families, cf. [BIS90,BI94]. However, the proof of [BS90, theorem 2.1] shows that for polynomial size circuits the two uniformity conditions are related, as the circuits can be computed from the Turing machine. We shall elaborate on the exact relationship between the two kinds of uniformity in subsequent paper.

Theorem 3 can easily be generalized to

Theorem 6. *If a logic $\mathcal{L}$ captures the complexity class $\mathbf{C}$ over ordered structures, then $K \in \mathbf{C}/\mathbf{C}$ iff K is strictly invariantly $\mathcal{L}$-definable.*

Proof. We prove it for $\mathbf{C} = \mathbf{P}$ and $\mathcal{L} = IFP$, leaving the generalization to the reader. As ORD is FOL-definable it is also IFP-definable (this holds also for other $\mathcal{L}$, as logics are assumed to be extensions of FOL). Now for K strictly invariantly definable in IFP we have $K \in \mathbf{P}/\mathbf{P}$. Conversely if $K \in \mathbf{P}/\mathbf{P}$, there is a $\bar{R}$-advice sequence A with $\{1^n a_n : n \in \mathbb{N}\} \in \mathbf{P}$ such that $B_K @ A \in \mathbf{P}$. Hence there is a $\phi \in IFP(\bar{R} \cup \bar{S})$, which defines K invariantly using the class of ordered $\bar{R}$-structures K_A and also $K_A \in \mathbf{P}$. Using the Immerman–Sazonov-Vardi Theorem, we get that K_A is definable in IFP. □

The same idea gives also the following generalization:

Theorem 7. *Let $\mathcal{L}_1, \mathcal{L}_2$ be logics which capture complexity classes $\mathbf{C}_1, \mathbf{C}_2$ respectively on ordered structures. A class of $\bar{S}$-structures K is in $\mathbf{C}_1/\mathbf{C}_2$ iff K is $\mathcal{L}_2$-invariantly definable in $\mathcal{L}_1$.*

Furthermore, advice sequences which are in **NP** do not help much in the following sense:

For $\mathcal{L} \subseteq SOL$ let $Exist\mathcal{L}$ be the the class of existential second order formulas $\exists \bar{S} \phi(\bar{R}, \bar{S})$ with $\phi \in \mathcal{L}$. Recall that, by Fagin's theorem, cf. [EF95], $K \in \mathbf{NP}$ iff K is definable in $ExistFOL$.

Theorem 8. *For a regular logic $\mathcal{L}$, a class K is strictly invariantly $\mathcal{L}$-definable iff K is $Exist\mathcal{L}$-invariantly $\mathcal{L}$-definable.*
In particular, a class K is strictly invariantly FOL-definable iff K is $ExistFOL$-invariantly FOL-definable iff K is **NP***-invariantly FOL-definable.*

Proof. From left to right this is trivial. For the other direction we use lemma 3. If K is $Exist\mathcal{L}$-invariantly definable, there is a formula $\exists \bar{U}\theta(\bar{R}, \bar{U})$ with $\theta \in \mathcal{L}$ and a formula $\phi(\bar{S}, \bar{R} \in \mathcal{L}$ such that

$$\mathfrak{A} \in K \text{ iff } \mathfrak{A} \models \exists \bar{R} \left(\exists \bar{U}\theta(\bar{R}, \bar{U}) \wedge \phi(\bar{S}, \bar{R})\right)$$

and equivalently

$$\mathfrak{A} \in K \text{ iff } \mathfrak{A} \models \forall \bar{R} \left(\exists \bar{U}\theta(\bar{R}, \bar{U}) \rightarrow \phi(\bar{S}, \bar{R})\right)$$

By moving the quantification of $\bar{U}$ outside we obtain

$$\mathfrak{A} \in K \text{ iff } \mathfrak{A} \models \exists \bar{R} \exists \bar{U} \left(\theta(\bar{R}, \bar{U}) \wedge \phi(\bar{S}, \bar{R})\right)$$

and, similarly,

$$\mathfrak{A} \in K \text{ iff } \mathfrak{A} \models \forall \bar{R} \forall \bar{U} \left(\theta(\bar{R}, \bar{U}) \rightarrow \phi(\bar{S}, \bar{R})\right)$$

As $\bar{U}$ does not appear in ϕ this shows that K is strictly invariantly $\mathcal{L}$-definable. □

How does the computational power of circuit complexity classes change with changing restrictions on the complexity or definability of the advice sequences?

Proposition 3.

(i) $\mathbf{P}/\mathbf{NP} \subseteq \mathbf{NP} \cap \mathbf{CoNP}$*;*
(ii) On unary languages (tally languages) the equality

$$\mathbf{L}/FOL = \mathbf{P}/FOL = \mathbf{NP} \cap \mathbf{CoNP}$$

holds, hence $\mathbf{NP} \cap \mathbf{CoNP} \subseteq \mathbf{P}/poly$ *on unary languages.*
(iii) $\mathbf{PSpace}/\mathbf{PSpace} = \mathbf{PSpace}$*, and hence,* $\mathbf{PSpace}/\mathbf{P} \subseteq \mathbf{PSpace}$*.*
(iv) Therefore, if $\mathbf{NP} \neq \mathbf{CoNP}$*, then* $\mathbf{NP} \not\subseteq \mathbf{P}/\mathbf{NP}$

Proof. For (i) we use theorem 8 and Fagin's characterization of **NP** as the class of $ESOL$-definable classes of structures, cf. [EF95].
To see (ii) we observe that tally languages can be viewed as classes K of $\bar{S}$-structures with $\bar{S} = \emptyset$. We use again Fagin's theorem. If $K \in \mathbf{NP} \cap \mathbf{CoNP}$, there are two formulas $\phi_1 \in FOL(\bar{R}_1)$ and $\phi_2 \in FOL(\bar{R}_2)$ such that $\mathfrak{A} \in K$ iff $\mathfrak{A}$ has an $\bar{R}_1$-expansion satisfying ϕ_1 and $\mathfrak{A} \notin K$ iff $\mathfrak{A}$ has an $\bar{R}_2$-expansion satisfying ϕ_2 Now let K_0 be the class of $\bar{R}_1 \cup \bar{R}_2$-structures which satisfy $\phi_1 \vee \phi_2$. It is now clear that $\phi_1 \in FOL(\bar{R}_1 \cup \bar{R}_2)$ defines K K_0-invariantly.
To see (iii)a we use Savitch's theorem which asserts that $\mathbf{NPSpace} = \mathbf{PSpace}$ and theorem 8. Finally, (iv) is follows from (i). □

5 Outlook and comparison with related work

5.1 Non-uniform case

The non-uniform classes have been characterized logically already before, using auxiliary numerical predicates, with a variation of it called also *Ajtai-definability* in [Mak97]. Here the interpretation of the auxiliary predicates $\bar{R}$ depend only on the cardinality of the structure (hence the name numerical). The exact relationship between K-invariant definability and K-Ajtai definability was studied in [Mak97]. Immerman characterized $\mathbf{AC}_0$ in [Imm87] and Molzan states and proves in [Mol90, Theorem 3.3] the analogue of theorem 3 for definability with numerical relations (predicates) which are unary relations. Although our prove contains the same ingredients as his, our treatement has several, not only cosmetical, advantages: Our framework allows us

(i) to treat also numerical relations with richer vocabularies, including linear order ORD;
(ii) to include Ajtai's notion of definability as part of the general unified picture;
(iii) to compare the various notions of invariant definability with other notions of definability, linking non-uniform circuit complexity with traditional definability theory, as explained in section 3.

Extensions of Molzan's work may be found in [AB97], where $\mathbf{C}/log$ is characterized for logarithmically bounded advice sequences. Although we were not aware of [Mol90] and [AB97] when proving our results, the technical content is very similar. Our results were first presented in August 1997 at ESSLLI'97, [Mak].

From our proposition 3 we know that on

$$\begin{array}{c} \textbf{unary} \ \text{ languages } (\textbf{tally} \text{ languages}) \\ \mathbf{NP} \cap \mathbf{CoNP} \subseteq \mathbf{L}/poly \subseteq \mathbf{P}/poly. \end{array}$$

Problem 1. What can we say about $\mathbf{P}/poly - \mathbf{L}/poly$ in general, and in particular on unary languages?

For binary languages the situation is wide open. It is known, cf. [BS90], that $\mathbf{P}/poly - \mathbf{NP} \neq \emptyset$, and that if $\mathbf{NP} \subseteq \mathbf{P}/poly$ then the polynomial hierarchy collapses to the second level.

Problem 2. Can we find a language or class of finite structures K with

$$K \in \mathbf{L}/poly - \mathbf{NP} \cap \mathbf{CoNP}$$

or could it be that

$$\mathbf{NP} \cap \mathbf{CoNP} \subseteq \mathbf{P}/poly?$$

5.2 Uniform cases

Uniform cases of circuit complexity classes have been studied extensively, starting with [GL84] and continuing with [BIS90,Lin92,BI94]. But in all these papers the uniformity required concerns the computability/definability of the circuits used to compute K. In our definition of $\mathbf{C}/\mathbf{D}$ we look at classes K computable in $\mathbf{C}$ using advice sequences computable in $\mathbf{D}$. This latter notion is a priori weaker, but, assuming that $\mathbf{NP} \cap \mathbf{CoNP} - \mathbf{P}$ contains a unary language, we have that $\mathbf{P}/\mathbf{P} - \mathbf{P} \neq \emptyset$, cf. proposition 3.

Our preference of implicit definabilty over definability with numerical predicates is more of a logical nature: The generalization of definability via numerical predicates to infinite structures leads to an artificial concept which is non-logical in its very nature. Choosing a unique well-ordering on a set of fixed cardinality (e.g. the initial ordinal) such as to make it into a 'numerical' predicate is a set theoretic, rather than a logical choice. In contrast to this the notion of invariant definability is a notion which is *inherently logical*. We often establish certain properties of mathematical objects using well-orderings invariantly, i.e. independtly of the particular choice of the well-ordering which we use in the proof. The logical character of invariant definability can also be seen in the fact, cf. prioposition 2, that on finite and infinite structures, strict invariant definability for First Order Logic is provably the same as explicit definability for First Order Logic (and the same holds for any other logic which satisfies Craig's Interpolation Theorem). The difference between invariant and explicit definability (for FOL) becomes only apparent when we restrict our framework to finite structures.

5.3 Oracles vs. advice

In our comparisons we want to make a final remark on oracle complexity classes. We have seen that in general $\mathbf{P}/\mathbf{NP}$ does not contain $\mathbf{NP}$ unless $\mathbf{NP} = \mathbf{CoNP}$. However, if we consider the class of languages $\mathbf{P}^{\mathbf{NP}}$ ($\mathbf{L}^{\mathbf{NL}}$) accepted by polynomial time (logarithmic space) Oracle Turing machines using oracles from $\mathbf{NP}$ ($\mathbf{NL}$), the picture is as follows:

$\mathbf{P}^{\mathbf{NP}}$ does contain $\mathbf{NP}$ and is closed under complements. Hence we have

$$\mathbf{P}/\mathbf{NP} \subseteq \mathbf{NP} \cap \mathbf{CoNP} \subseteq \mathbf{NP} \cup \mathbf{CoNP} \subseteq \mathbf{P}^{\mathbf{NP}}.$$

On the other hand our analysis in proposition 3 shows that on unary languages we have

$$\mathbf{L}^{\mathbf{NL}} \subseteq \mathbf{P} \subseteq \mathbf{L}/\mathbf{NL}.$$

In other words $\mathbf{NP}$ oracles are stronger than $\mathbf{NP}$ advice, whereas, on unary languages, $\mathbf{NL}$ advice is stronger than $\mathbf{NL}$ oracles.

Oracle complexity classes have been studied from a logical point of view by Gottlob, Grädel, Makowsky and Pnueli, Stewart, and more recently by Frick and Frick, Makowsky and Pnueli, [Got97,Grä90,Ste91,Ste92,Ste93b,Ste93a], [MP94,MP95] and [Fri97,FMP99]

5.4 Conclusions and further research

For the cases of the complexity classes $\mathbf{C} \in \{\mathbf{L}, \mathbf{NL}, \mathbf{P}, \mathbf{NP}, \mathbf{PSpace}\}$ we have shown that the notion of *invariant definability* in a logic $\mathcal{L}$ fits exactly to capture a non-uniform complexity class $\mathbf{C}/poly$ whenever $\mathcal{L}$ captures $\mathbf{C}$ over ordered structures. We have also seen that ORD-invariant definability in $\mathcal{L}$ suffices to capture $\mathbf{C}$.

It remains an intriguing question whether ORD-invariant definability can be replaced by some other class of $\bar{R}$-structures K_0, such that if $K \in \mathbf{P}$ then K is K_0-invariantly definable in IFP.

Eric Rosen has observed, cf. [Mak97], that if we require that K_0 be fin-categorical and closed under substructures then either K_0 contains only trivial structures or ORD is *parametrically FOL-definable in* K_0, there is a formula $\phi(x, y, \bar{z}) \in FOL(\bar{R})$ such that whenever $\mathfrak{A} \in K_0$ there are $(a_1, a_2, \ldots, a_k) = \bar{a}$ such that $\phi(x, y, \bar{a})$ defines a linear order on $\mathfrak{A}$.

We conjectured in [Mak97]:

Conjecture 1. If K_0 is a class of $\bar{R}$-structures such that whenever $K \in \mathbf{P}$ then K is K_0-invariantly definable in IFP, then ORD is parametrically IFP-definable in K_0.

Ajtai has succeded in showing that **Parity** is not in $\mathbf{AC}_0$ using logico-combinatorial methods, [Ajt83]. In fact he showed that **Parity** is not invariantly FOL-definable. The question is whether Ajtai's methods can be extended to some other logics such as IFP? A positive answer would allow us to exhibit classes K of structures which were not in $\mathbf{P}/poly$. If such a K were in **NP** we would get $\mathbf{NP} \not\subseteq \mathbf{P}/poly$.

An obvious (but difficult) candidate for such a class is **Ham**, the class of finite undirected graphs with a Hamiltonian cycle.

The following is known:

(i) **Ham** is not invariantly FOL-definable.
(ii) **Ham** is not definable in IFP without order.
(iii) If **Ham** is FOL-invariantly definable in IFP, then $\mathbf{NP} = \mathbf{CoNP}$.

The first two statements can be seen by using non-relativizing FOL-reductions to **Parity**. The last statement follows from the Immerman–Sazanov–Vardi Theorem.

Conjecture 2. **Ham** is not invariantly (not FOL-invariantly) definable in IFP. Note however, that **Ham** is ORD-invariantly definable in the finite variable logic $\mathcal{L}^2_{\infty,\omega}$, cf. remark in section 1.

As the conjecture implies $\mathbf{P} \neq \mathbf{NP}$, we admit, that we are very far from such an application, and we are not too optimistic, that a breakthrough could be achieved using **Ham**. But it might be possible, and very instructive, to show that for some explicitly given $K \notin \mathbf{NP}$, K is not invariantly (not FOL-invariantly) definable in IFP.

References

AB97. A. Atserias and J. L. Balcázar. Refining logical characterizations of advice complexity classes. In *First Panhellenic Symposium on Logic*, Cyprus, 1997, to appear

Ajt83. M. Ajtai. Σ_1^1–formulae on finite structures. *Annals of Pure and Applied Logic*, 24:1–48, 1983. 143, 154, 154 144, 145, 146, 148, 156

Ajt89. M. Ajtai. First-order definability on finite structures. *Annals of Pure and Applied Logic*, 45:211–225, 1989. 145, 146

BCST92. D. A. M. Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44:478–499, 1992. 145

BF85. J. Barwise and S. Feferman, editors. *Model-Theoretic Logics*. Perspectives in Mathematical Logic. Springer Verlag, 1985. 150

BI94. D.A.M. Barrington and N. Immerman. Time, hardware and uniformity. In *Proccedings of the 9th Structure in Complexity Theory*, pages 176–185. IEEE Computer Society, 1994. 145, 152, 155

BIS90. D.A.M. Barrington, N. Immerman, and H. Straubing. On uniformity in nc^1. *JCSS*, 41:274–306, 1990. 145, 152, 155

BS90. R.B. Boppana and M. Sipser. The complexity of finite functions. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 14. Elsevier Science Publishers, 1990. 143, 145, 146, 146, 147, 152, 152, 154

EF95. H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995. 143, 143, 145, 145, 146, 150, 150, 151, 152, 153

FMP99. M. Frick, J.A. Makowsky, and Y.B. Pnueli. Oracles and Lindström quantifiers on ordered structures. submitted to *Information and Computation*, in revision. in revision. 155

Fri97. M. Frick. *Oracles and quantifiers*. PhD thesis, Department of Mathematics, University of Freiburg, Freiburg, Germany, 1997. 155

FSS84. M. Furst, J. Saxe, and M. Sipser. Parity, circuits, and the polynomial hierarchy. *J. Math. Systems Theory*, 17:13–27, 1984. 145

GL84. Y. Gurevich and H. Lewis. A logic for constant-depth circuits. *Information and Control*, 61:65–74, 1984. 144, 145, 155

Got97. G. Gottlob. Relativized logspace and generalized quantifiers over finite ordered structures. *Journal of Symbolic Logic*, 62.2:545–574, 1997. 155

Grä90. E. Grädel. On logical descriptions of some concepts in structural complexity theory. In *CSL'89*, volume 440 of *Lecture Notes in Computer Science*, pages 163–175, 1990. 155

Imm87. N. Immerman. Languages that capture complexity classes. *SIAM Journal on Computing*, 16(4):760–778, Aug 1987. 144, 144, 146, 154

Imm9x. N. Immerman. *Descriptive Complexity*. Springer, 1998. 143

Joh90. D.S. Johnson. A catalog of complexity classes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume 1, chapter 2. Elsevier Science Publishers, 1990. 143

Kol90. P.G. Kolaitis. Implicit definability on finite structures and unambiguous computations. In *LiCS'90*, pages 168–180. IEEE, 1990. 151

Lin92. S. Lindell. A purely logical characterization of circuit uniformity. In *Proceedings of the 7th conference on Structures in Complexity*, 1992. 155

Mak. J.A. Makowsky. Translations, interpretations and reductions. Lecture Notes of a course given at ESSLLI'97 in Aix-en-Provence, August 1997. 154

Mak97. J.A. Makowsky. Invariant definability. In *Computational Logic and Proof Theory*, volume 1289 of *Lecture Notes in Computer Science*, pages 186–202. Springer Verlag, 1997. 142, 143, 143, 144, 146, 147, 151, 154, 154, 156, 156

Mol90. B. Molzan. Expressibility and nonuniform complexity classes. *SIAM Journal of Computing*, 19.3:411–423, 1990. 143, 143, 144, 147, 154, 154

MP94. J.A. Makowsky and Y.B. Pnueli. Oracles and quantifiers. In *CSL'93*, volume 832 of *Lecture Notes in Computer Science*, pages 189–222. Springer, 1994. 155

MP95. J.A. Makowsky and Y. Pnueli. Logics capturing oracle complexity classes uniformly. In *Logic and Computational Complexity (LCC'94)*, volume 960 of *Lecture Notes in Computer Science*, pages 463–479. Springer Verlag, 1995. 155

Saz80. V. Sazonov. Polynomial computability and recursivity in finite domains. *Elektronische Informationsverarbeitung und Kybernetik*, 16:319–323, 1980. 146

Ste91. I.A. Stewart. Comparing the expressibility of languages formed using NP-complete operators. *Journal of Logic and Computation*, 1.3:305–330, 1991. 155

Ste92. I.A. Stewart. Using the Hamilton path operator to capture NP. *Journal of Computer and Systems Ssciences*, 45:127–151, 1992. 155

Ste93a. I.A. Stewart. Logical characterization of bounded query classes i: Polynomial time oracle machines. *Fundamenta Informaticae*, 18:93–105, 1993. 155

Ste93b. I.A. Stewart. Logical characterizations of bounded query classes II: Polynomial–time oracle machines. *Fundamenta Informaticae*, 18:93–105, 1993. 155

Str94. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Progress in Theoretical Computer Science. Birkhäuser, 1994. 145

Computational Complexity of Ehrenfeucht–Fraïssé Games on Finite Structures

Elena Pezzoli

Boston College, Chesnut Hill, MA 02467, USA
pezzoli@csli.stanford.edu

Abstract. We show that deciding the winner of the r–moves Ehrenfeucht–Fraïssé game on two finite structures A and B, over any fixed signature Σ that contains at least one binary and one ternary relation, is PSPACE complete. We consider two natural modifications of the EF game, the one–sided r–moves EF game, where the spoiler can choose from the first structure A only, and therefore the duplicator wins only if B satisfies all the existential formulas of rank at most r that A satisfies; and the k–alternations r–moves EF game (for each fixed k), where the spoiler can choose from either structure, but he can switch structure at most k times, and therefore the duplicator wins iff A and B satisfy the same first order formulas of rank at most r and quantifier alternation at most k (defined in the paper). We show that deciding the winner in both the one–sided EF game and the k–alternations EF game is also PSPACE complete.

1 Introduction

Two structures A and B are r–equivalent ($A \equiv_r B$) if they satisfy the same first order formulas of quantifier depth at most r; A and B are $L^s_{\infty\omega}$–equivalent ($A \equiv^s_\infty B$) if they satisfy the same $L_{\infty\omega}$ (or even just first order, for finite structures) formulas with at most s variables (free or bound). Equivalently A and B are r–equivalent or $L^s_{\infty\omega}$–equivalent if the duplicator wins the r–moves Ehrenfeucht–Fraïssé game (EF game) or the s–pebbles pebble game on A and B, respectively. r–equivalence and $L^s_{\infty\omega}$–equivalence play an important role in logic and computer science, and in particular in descriptive complexity theory, which aims to characterize the queries in a given complexity class by means of a logic in which they can be described. To show that a class K of structures is not axiomatizable in first order logic (or in $L_{\infty\omega}$), in fact, it is sufficient to show that for every r (s) there are structures $A \in K$ and $B \notin K$, with $A \equiv_r B$ ($A \equiv^s_\infty B$). See for example [1]. We are interested in determining the complexity of deciding whether two finite structures are r–equivalent, and we will call this the Ehrenfeucht–Fraïssé problem (EF problem), or $L^s_{\infty\omega}$–equivalent, and we will call this the pebble problem. If the number r of moves is fixed then it is easy to see that the EF problem is in LOGSPACE. Martin Grohe [4] has shown that, if the number s of pebbles is fixed, the pebble problem is complete for PTIME. If r and s are part of the input, the exact complexity of both the EF problem and

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 159–170, 1999.

the pebble problem was so far open. It is easy to see that the EF problem is in PSPACE, and that the pebble problem is in EXPTIME, but neither had been proved complete yet. In this paper we show that the EF problem is complete for PSPACE. We also consider two natural modifications of the EF problem on two finite structures A and B: the one-sided EF problem, where the spoiler can choose from the first structure A only, and therefore the duplicator wins only if B satisfies all the existential formulas of rank at most r that A satisfies; and the k–alternations r–moves EF game (for each fixed k), where the spoiler can choose from either structure, but he can switch structure at most k times, and therefore the duplicator wins iff A and B satisfy the same first order formulas of rank at most r and quantifier alternation (defined below) at most k, and show that they are also PSPACE complete. One can view these EF problems and the pebble problem as an approximation to the structure isomorphism problem [6], whose complexity is very problematic; instead of asking whether two finite structures are isomorphic, that is whether they agree on all first order formulas, we change the class of formulas they have to agree on, and consider the complexity of the corresponding problem.

In this paper we will provide only sketches of the proofs. Full proofs will appear elsewhere [2],[3].

2 Background

We give here the definition of Ehrenfeucht–Fraïssé game, one sided Ehrenfeucht–Fraïssé game, and k–alternations Ehrenfeucht–Fraïssé game.

Definition 1. *Let* (G, H) *be a pair of structures over a given vocabulary, and* $\boldsymbol{\gamma}, \boldsymbol{\theta}$ *strings of elements in* G *and* H *respectively ; the* r–moves EF game (EF game) *on* $(G, \boldsymbol{\gamma}, H, \boldsymbol{\theta})$ *is played by two players called the* spoiler *and the* duplicator. *Each player has to make* r *moves in the course of the play. The players take turns. In the* i^{th} *round the spoiler selects one of the structure* G *or* H *and an element from that structure; the duplicator answers by choosing an element from the structure not chosen by the spoiler. At the end of* r *rounds the duplicator wins iff* $(\boldsymbol{\gamma}, g_1, \ldots g_r) \rightarrow (\boldsymbol{\theta}, h_1, \ldots h_r)$ *is a partial isomorphism from* G *to* H*, where* $g_1, \ldots, g_r$ *are the elements chosen from* G *by either the spoiler or the duplicator in round* $1, \ldots, r$ *and* $h_1, \ldots h_r$ *are the elements chosen from* H*. Otherwise the spoiler wins.*

The connection with logic comes from the following:

Definition 2. *The* quantifier rank *(or simply* rank*) of a first order formula* ϕ *is the maximum number of nested quantifiers in it, defined by induction by:*

1. $qr(\phi) = 0$ *if* ϕ *is atomic.*
2. $qr(\neg\phi) = qr(\phi)$.
3. $qr(\phi \vee \psi) = qr(\phi \wedge \psi) = \max\{qr(\phi), qr(\psi)\}$.
4. $qr(\exists x\psi) = qr(\forall x\phi) = 1 + qr(\phi)$.

Theorem 1. *The duplicator wins the r–moves EF game on* $(G, \boldsymbol{\gamma}, H, \boldsymbol{\theta})$ *iff* $\boldsymbol{\gamma}$ *and* $\boldsymbol{\theta}$ *satisfy the same formulas of rank at most r, that is iff*

$$G \models \phi(\boldsymbol{\gamma}) iff H \models \phi(\boldsymbol{\theta})$$

for all ϕ *of quantifier rank at most r.*

For a proof see [1].
We also want to consider the following game:

Definition 3. *Let* G, H, $\boldsymbol{\gamma}$, $\boldsymbol{\theta}$ *be as in Definition 1. The* r–moves one–sided EF game *on* $(G, \boldsymbol{\gamma}, H, \boldsymbol{\theta})$ *is played like the EF game, except that the spoiler can choose from structure G only. The winning condition is as in the EF game.*

The only difference between the one–sided EF game and the ordinary EF game is that in the one–sided EF game the spoiler is restricted to choose from the first input structure only. It is possible to give a logical charcterization of the one–sided EF game in the spirit of Theorem 1.

Definition 4. Existential formulas *are defined by the clauses:*

1. *All quantifier free formulas are existential.*
2. *If* ϕ *and* ψ *are existential then* $\phi \vee \psi$ *and* $\phi \wedge \psi$ *are existential.*
3. *If* ϕ *is existential then* $\exists x \phi$ *is existential*

Theorem 2. *The duplicator wins the n–moves one–sided EF game on* (G, γ, H, θ) *iff* θ *satisfies all the existential formulas of rank* $\leq n$ *that* γ *satisfies.*

Proof: the proof is similar to that of Theorem 1.
A generalization of the one–sided EF game consists in allowing the spoiler to choose from either structure, but with a bound k on the number of times he alternates, that is chooses in one round from a different structure than the previous round.

Definition 5. *Let* k *be a natural number. Let* G, H, $\boldsymbol{\gamma}$, $\boldsymbol{\theta}$ *be as in Definition 1. The* k alternations r–moves one–sided EF game *on* $(G, \boldsymbol{\gamma}, H, \boldsymbol{\theta})$ *is played like the EF game, except that the spoiler is allowed at most k alternations. The winning condition is as in the EF game.*

Again it is possible to relate this game to logic.

Definition 6. *The* external quantifier set and alternation number *of a first order formula are defined by induction as follows:*

1. *The external quantifier set of an atomic formula is empty; the alternation number is 0.*
2. *The external quantifier set* Q_ξ *of a formula* $\xi = \phi\, b\, \psi$, *where* $b = \wedge$ *or* $\vee$ *is equal to* $Q_\phi \cup Q_\psi$; *the alternation number is the maximum of the alternation numbers of* ϕ *and* ψ.

3. *The external quantifier set of* $\neg\phi$ *is obtained by replacing* $\exists$ *with* $\forall$ *and vice versa in the external quantifier set of* ϕ*; the alternation number is the same as that of* ϕ*.*
4. *The external quantifier set of* $\exists x\,\phi$ *is* $\{\exists\}$*; the alternation number is the same as that of* ϕ *if* Q_ϕ *does not contain* $\forall$*, and it is one plus the alternation number of* ϕ *otherwise.*
5. *The external quantifier set of* $\forall x\,\phi$ *is* $\{\forall\}$*; the alternation number is the same as that of* ϕ *if* Q_ϕ *does not contain* $\exists$*, and it is one plus the alternation number of* ϕ *otherwise.*

Theorem 3. *The duplicator wins the* r*-moves,* k *alternation EF game on* $(G, \boldsymbol{\gamma}, H, \boldsymbol{\theta})$ *iff* $\boldsymbol{\gamma}$ *and* $\boldsymbol{\theta}$ *satisfy the same formulas of rank at most* r *and alternation number at most* k*.*

Proof: A proof is given in [2]

Definition 7. *Given a fixed signature* Σ*, we will call the EF problem (for* Σ*), the one sided EF problem (for* Σ*) and the* k*–alternations EF problem (for* Σ*) the following problems, respectively: given as input two finite structures* A *and* B *over* Σ*, and a number* r*, determine who wins the* r*–moves EF game, one–sided EF game,* k*–alternations EF game on* A *and* B*.*

3 Complexity of the EF problem

It is easy to show that the EF problem is in PSPACE (for any signature Σ, actually even if Σ is part of the input); our goal here is to show hardness for PSPACE. We will use a reduction from the following game theoretic version of Quantified Boolean Formula (QBF).

Definition 8. The Quantified Boolean Formula game *is played by two players, I and II; on input a formula of the form* $\phi = \exists x_1 \forall x_2 \ldots \exists x_{2r-1} \forall x_{2r} (C_1 \wedge \ldots \wedge C_n)$ *the game continues for* r *rounds. In round* i *player I chooses a truth assignment for variable* x_{2i-1} *and then player II chooses a truth value for* x_{2i}*. At the end of the* r *rounds, player I wins iff the assignment that has been produced makes* $C_1 \wedge \ldots \wedge C_n$ *true.*

We call *the QBF problem* the problem of deciding who wins the QBF game on a certain input formula.

Theorem 4 ([7]). *The QBF problem is PSPACE complete.*

Theorem 5. *The EF problem (for finite structures over any fixed signature* Σ *that contains at least one binary and one ternary relation) is PSPACE complete.*

The plan of the proof is to show that the QBF problem reduces to the EF problem. Given any quantified boolean formula of the form

$$\phi = \exists x_1 \forall x_2 \ldots \exists x_{2r-1} \forall x_{2r} (C_1 \wedge \ldots \wedge C_n)$$

we will show how to construct structures A and B over $\Sigma = \{E, H\}$, where E is a binary relation and H is a ternary relation, such that player I wins the QBF game on input ϕ iff the spoiler wins the $2r + 1$–moves EF game on (A, B). The main difficulty originates from the fact that in each round of the QBF game player I and II can assign a truth value to one given variable only, while the spoiler and the duplicator have much more freedom and the spoiler ,in particular, in an y round of the EF game can choose any vertex he wants. The proof will proceed by first imposing additional constraints on the spoiler and the duplicator, for example requiring them in each round to choose from a certain subset of the vertices of A and B only, as explained below, so that the EF game with constraints will closely reflect the QBF game. Then it will be easy to show that player I wins QBF game on input ϕ iff the spoiler wins the $2r + 1$–moves EF game with constraints on (A, B). To complete the proof it will be necessary to show that the first player that does not respect the constraints is going to lose the EF game. Both structures A and B consist of r blocks; the idea is that choosing some of the vertices in block i of A or B in a move of the EF game corresponds to assigning truth value T or F to variable x_i or to variable x_{i+1} of ϕ, (we will label such vertices by $T(x_i)$,$F(x_i)$,$T(x_{i+1})$ or $F(x_{i+1})$ or sometimes simply by T and F), choosing other vertices in block i will correspond to recording that a certain truth value has been assigned to x_i in a previous move, and to choosing a truth value for x_{i+1}, (we will label such elements by $TF(x_ix_{i+1}), TT(x_ix_{i+1}), \dots$); then there are also vertices in block i that do not correspond to any truth assignment to the variables of ϕ (we will denote any such vertex simply by v*). We then consider constraining the spoiler and the duplicator to play from block i in round i and $i + 1$, i odd, and in the following way:

round i	round $i + 1$	
$s : T(x_i)$	$d : F(x_i)$	A
$d : TF(x_ix_{i+1})$	$s : v*$	B

This means that in round i the spoiler must first assign a truth value to variable x_i (by choosing an element $T(x_i)$ or $F(x_i)$ in block i of structure A), duplicator must record the spoiler's assignment and assign a truth value to variable x_{i+1} (by choosing an element $TT(x_ix_{i+1})$ or $TF(x_ix_{i+1})$ or $FT(x_ix_{i+1})$ or $FF(x_ix_{i+1})$ in block i of structure B); then the spoiler must play some vertex $v*$, in block i of structure B, which does not correspond to a truth assignment, and the duplicator must record the truth assignment to variable x_{i+1} in structure A as well (by choosing some element $T(x_{i+1})$ or $F(x_{i+1})$ in block i of A). At the end of the first $2r$ rounds played in this fashion, a truth assignment of the variables of ϕ has been determined by the two players of the EF game. In the last move the spoiler will have a chance to win iff the assignment makes ϕ true. The main difficulty is so as to ensure that the spoiler loses if he does not follow the rules.

In the next section we will construct preliminary structures $\bar{A}_k$ and $\bar{B}_k$ over a signature containing a binary relation only, and prove some useful facts about them. Structures A and B will be obtained by introducing a ternary relation on

the vertices of $\bar{A}_k$ and $\bar{B}_k$ (after some minor modification). $\bar{A}_k$ and $\bar{B}_k$ consist of r blocks; the blocks are gadgets I_j (see Figure 3), introduced in the next section, which are in turn built out of other gadgets J_m and L_n.

We will say that in the s–moves EF game on two structures, *the spoiler forces a pair* (y, y') if he can play y (or y') and the duplicator must answer with y' (or y) not to lose the game.

3.1 The structures $\bar{A}_k$ and $\bar{B}_k$

We first need to introduce gadgets J_k and L_k shown in Fig. 1. They are somehow similar to gadgets used in [5] and [4]. An edge between a vertex v and the symbol k or $k-1$ means that v has k or $k-1$ additional neighbours, not shown in the figure. They will be called *special neighbours*. Special neighbours of distinct vertices are all distinct. Vertices with special neighbours will be called *vertices in the middle*. Then gadget I_k is built using gadgets J_k and L_k as follows (see Fig. 2): I_k has vertices x, x', y, y'; x and x' have each 16 neighbours; again we will call them *vertices in the middle*; the vertices in the middle have k or $k-1$ additional neighbours (besides x or x'); again they will be called *special neighbours*; in addition each vertex v in the middle is glued to a separate copy of a gadget J_{k-1} or L_{k-1}, disjoint from all others, so that v coincides with z and y with t and y' with t' as follows:

1. Eight of the vertices in the middle connected with x have k special neighbours; four of them are glued to an L_{k-1} gadget, and four to an J_{k-1}.
2. Eight of the vertices in the middle connected with x have $k-1$ special neighbours; four are glued to a L_{k-1} gadget, and four to a J_{k-1} gadget.
3. Four of the vertices in the middle connected with x' have k special neighbours, and they are glued to a L_{k-1} gadget.
4. The remaining 12 vertices in the middle connected with x' have $k-1$ special neighbours; four of them are glued to a L_{k-1} gadget, and eight to a J_{k-1} gadget.

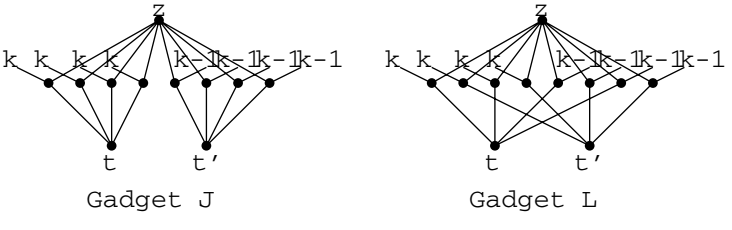


Fig. 1. The gadgets J_k and L_k

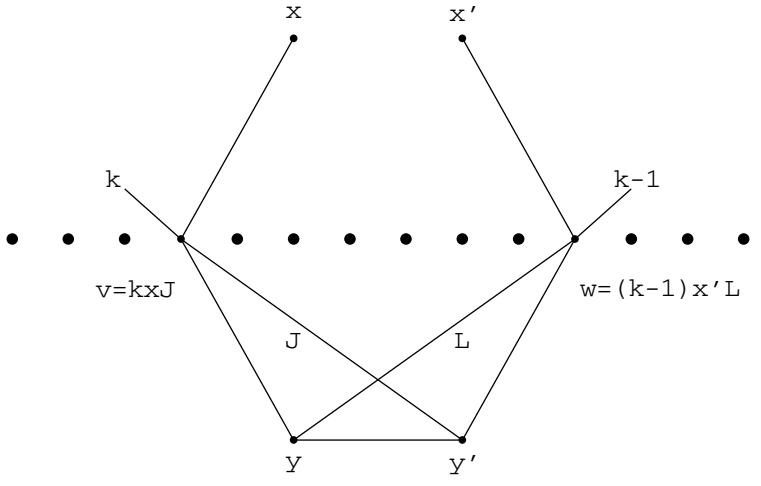


Fig. 2. The gadget I_k

We will denote a vertex in the middle of gadget I J and L by giving its neighbours, or the gadget it is glued to; so, for example, if v is a vertex in the middle of gadget I_k, $v = kxJ$ means that v is any of the 4 vertices with k special neighbours and glued to an J_{k-1} gadget; $(k-1)vy$ stands for a vertex in the middle of gadget J, having $k-1$ special neighbours and connected to v and y. We need the following lemma:

Lemma 1. *In the $k+1$–moves EF game on (I_k, x, I_k, x'), the spoiler can force the pair (y, y'), but the duplicator has a strategy to win the k– moves EF game that allows him to answer y with y and y' with y'.*

Proof:(sketch) In the $k+1$–moves game the spoiler can start by playing $v = kxJ$ and the duplicator must answer with $w = kx'L$, then the spoiler can select $w(k-1)y'$ in gadget L_{k-1} and the duplicator must answer $v(k-1)y$ in gadget J_{k-1}. With only k moves the duplicator can follow a partial isomorphism that maps x to x', y to y and y' to y'. The only problem maybe if the spoiler plays all k special neighbours of, say, some kxJ; but then the duplicator can play all $k-1$ neighbours of $(k-1)x'J$ and any additional vertex not connected to x'.

Definition 9. *Let k be even and let $\bar{A}_k$ consist of $k/2$ copies of I gadgets, I_k, I_{k-2}, $I_{k-4}, \ldots, I_2$ with the y (y') vertex of the i^{th} gadget coinciding with the x (x') vertex of the $(i+1)^{th}$ gadget, plus an additional vertex connected to the x vertex of the first gadget I_k, as shown in figure 3, and $\bar{B}_k$ be the same as $\bar{A}_k$ except that it has an additional vertex connected to x'_1 and not x_1.*

We will say that gadget $I_{k-2(i-1)}$ is the i^{th} block of $\bar{A}_k$ or $\bar{B}_k$. Now we want to consider the $k+1$–moves EF game on $\bar{A}_k$ and $\bar{B}_k$, and show that the spoiler can

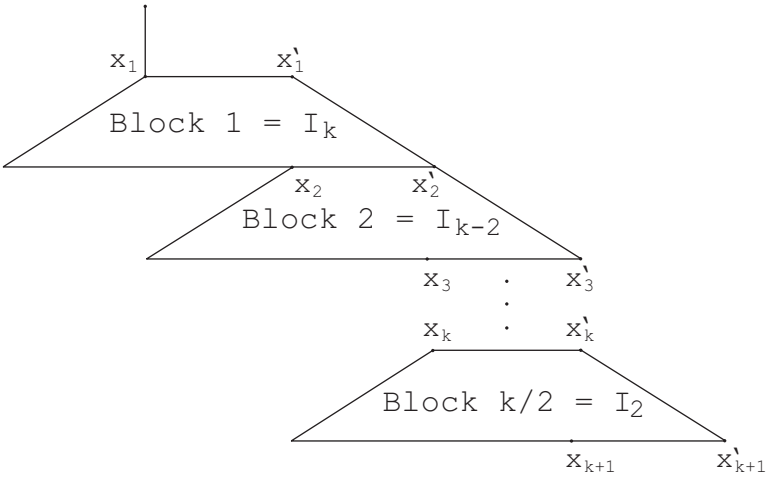


Fig. 3. The structure $\bar{A}_k$

force the pair (x_{k+1}, x'_{k+1}) only if he follows a precise strategy of first choosing certain elements in block 1, then certain elements in block 2, and so on. Once we have constructed structures A and B form $\bar{A}$ and $\bar{B}$ this strategy will correspond to assigning truth value to the variables of ϕ, and so on as explained at the end of Section 2.

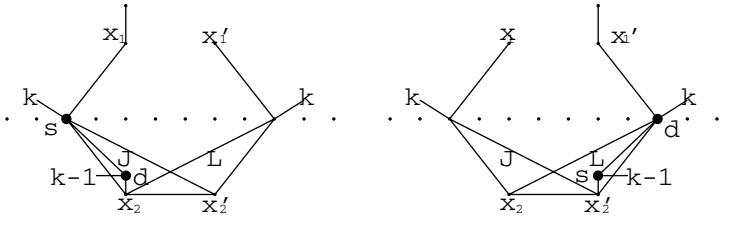


Fig. 4. The first two lawful moves

Definition 10. *Consider the $k+1$–moves EF game on $\bar{A}_k$ and $\bar{B}_k$; we will say that* the players play lawfully *if in round i, $i < k+1$ and i odd, the spoiler chooses a vertex v in the i^{th} block, $v = (k-2(i-1))x_iJ$ (or a special neighbour of v) from the middle of gadget $I_{k-2(i-1)}$ and the duplicator answers with some $w = (k-2(i-1))x'_iL$ (or a special neighbour of w). In the next round the spoiler plays lawfully if he chooses vertex $(k-2(i-1)-1)wx'_{i+1}$ (or a special neighbour) or vertex $(k-2(i-1)-2)wx_{i+1}$ in gadget L (or a special neighbour), and the*

duplicator plays lawfully if he plays vertex $(k-2(i-1)-1)vx_{i+1}$ *(if the spoiler has played* $(k-2(i-1)-1)wx'_{i+1}$ *or a special neighbour if the spoiler has played a special neighbour of* $(k-2(i-1)-1)wx'_{i+1}$*) or vertex* $(k-2(i-1)-2)vx'_{i+1}$ *(if the spolier has played* $(k-2(i-1)-2)wx_{i+1}$ *or a special neighbour if the spoiler has played a special neighbour of* $(k-2(i-1)-2)wx_{i+1}$*) in gadget* J*; (See Fig 4, where the spoiler's moves are marked s, the duplicator 's d.) In the last round the spoiler plays lawfully if he chooses* x_{k+1} *or* x'_{x+1}*, from either* A *or* B*, and the duplicator if he plays* x'_{k+1} *(if the spoiler has played* x_{k+1}*) or* x_{k+1} *(if the spoiler has played* x'_{k+1}*) from* B *or* A*.*

Theorem 6. *In the the* $k+1$*–moves EF game on* $\bar{A}_k$ *and* $\bar{B}_k$ *we have that if the spoiler plays lawfully, he can force a pair* (x_{k+1}, x'_{k+1})*; but if he does not play lawfully the duplicator has a strategy to win the game and answer* x_{k+1} *with* x_{k+1} *and* x'_{k+1} *with* x'_{k+1}*.*

Proof(sketch): let move j be the first unlawful move of the spoiler. Then for the rest of the game the duplicator can play according to the partial isomorphism that maps x_j to x'_j and x_l to x_l, x'_l to x'_l for $l > j$.

3.2 The Main Theorem

We are now ready to describe the structures A and B. Recall that our goal is to show that player I wins the QBF game on input

$$\phi = \exists x_1 \forall x_2 \ldots \exists x_{2r-1} \forall x_{2r} (C_1 \wedge \ldots \wedge C_n).$$

iff the spoiler wins the $2r+1$–moves EF game on (A, B).

A and B are obtained by introducing a ternary relation H on the vertices of structures $\bar{A}_k$ and $\bar{B}_k$ (after minor modifications), with $k = 2r$. In order to motivate the definition of H, consider the $k+1$–moves EF game on $\bar{A}_k$ and $\bar{B}_k$. For simplicity , consider a *lawful strategy* defined as in Definition 10, except that the players must play vertices in the middle, and not special neighbours. If the players play according to such a lawful strategy a run of the game may look like:

$s : T(x_1)$	$d : F(x_2)$	$\ldots$	$s : F(x_{2r-1})$	$d : F(x_{2r})$	$d : x'_{k+1}$
$d : TF(x_1x_2)$	$s : v*$	$\ldots$	$d : FF(x_{2r-1}x_{2r})$	$s : v*$	$s : x_{k+1}$

That is, the spoiler has first chosen an element in the middle of gadget I_k labelled T, the duplicator has answered with an element labelled TF , then the spoiler has chosen an element with no label, and so on. The first k (lawful) rounds determine a truth assignment for the variables of ϕ. Of course the duplicator wins the game on $\bar{A}_k$ and $\bar{B}_k$, while we want the duplicator to win a run of the game iff the truth assignment determined by the first k rounds of the run does not make ϕ true. We will achieve this by taking advantage of the fact that in the last move the spoiler can force a pair (x_{k+1}, x'_{k+1}). We first replace x_{k+1} and x'_{k+1} with two sets $V_{x_{k+1}}$ and $V_{x'_{k+1}}$, each having $k+1$ new vertices labelled C_1, $k+1$ new vertices labelled C_2,...,$k+1$ new vertices labelled C_n, where $C_1, \ldots, C_n$

are the clauses of ϕ (again, no unary relation corresponding to these labels are present in the signature).$V_{x_{k+1}}$ has also an additional new vertex $w*$, with no label. There are no edges between vertices in the same set $V_{x_{k+1}}$ or $V'_{x_{k+1}}$, and there is an edge between any vertex $v \in V_{x_{k+1}}$ ($V_{x'_{k+1}}$) and any vertex w that was connected to x_{k+1} (x'_{k+1}) in $\bar{A}_k$ or $\bar{B}_k$. So we have:

- $V_{x_{k+1}} = \{w*, C_1, \ldots, C_1, \ldots, C_n, \ldots, C_n\}$.
- $V_{x'_{k+1}} = \{C_1, \ldots, C_1, \ldots, C_n, \ldots, C_n\}$.

Definition 11. *Let C_k and D_k be the structures obtained by replacing x_{k+1} and x'_{k+1} with the sets $V_{x_{k+1}}$ and $V_{x'_{k+1}}$, as described above.*

On C_k and D_k a lawful run of the $k+1$–moves EF game may look like:

$s : T(x_1)$	$d : F(x_2)$	$\ldots$	$s : F(x_{2r-1})$	$d : F(x_{2r})$	$d : C_j$
$d : TF(x_1x_2)$	$s : v*$	$\ldots$	$d : FF(x_{2r-1}x_{2r})$	$s : v*$	$s : w*$

In the last round the spoiler has played an unlabelled element $w*$ from set $V_{x_{k+1}}$ and the duplicator has responded with an element of $V_{x'_{k+1}}$, that is has exhibited a clause C_j of ϕ. The duplicator must lose the run if C_j is not falsified by the assignment. To ensure this, we add a ternary relation H; no triple $(*, *, w*)$ is in H, but for example we will have $H(T(x_1), F(x_2), C_j)$ iff assigning T to x_1 and F to x_2 in ϕ makes clause C_j true. For the sake of exposition in introducing ternary relation H we will first label explicitly some of the vertices in A_k and B_k with the labels T,F,TT,TF,FT,FF; this is just to facilitate the exposition and introduce ternary relation H below; no unary relations are part of the signature Σ. First we label the vertices in the middle of gadget I_i and gadgets J_{i-1} and L_{i-1} for each $i = k, k-2, \ldots, 2$.

1. Of the four vertices ixJ, two are labelled T and the other two F; of the four vertices $(i-1)xJ$, ixL, $(i-1)xL$,or $ix'J$,$(i-1)x'J$, $(i-1)x'L$, one is labelled TT, one TF, one FF, one FT.
2. Of the four verices in the middle of any gadget J_{i-1} with $i-1$ special neighbours, or $i-2$ special neighbours, two are labelled T and two F;
3. In gadget L_{i-1} the two vertices $(i-1)zt'$ and the two vertices $(i-2)zt$ are not labelled; of the two remaining verices $(i-1)zt$ and the two $(i-2)zt'$, one is labelled T and the other F.

We will say that two vertices v and w are consecutive in block i iff v is a vertex in the middle of gadget $I_{k-2(i-1)}$ and w is a vertex in the middle of the L or J gadget glued to v.

Definition 12. *Relation H is defined as follows on the vertices of C_k and D_k:*

- *$H(u, v, C_j)$ iff u is labelled a (a =T or F) v is labelled b, u and v are consecutive in block i, and assigning x_i to a and x_{i+1} to b makes clause C_j true in ϕ.*
- *$H(w, v*, C_j)$ w is labelled ab (a, $b = T$ or F), w and $v*$ are consecutive in block i and assigning x_i to a and x_{i+1} to b makes clause C_j true in ϕ.*

– $H(z, v, C_j)$ *z is labelled ac, v is labelled b, z and v are consecutive in block i and assigning x_i to a and x_{i+1} to b makes clause C_j true in ϕ.*

Definition 13. *Structures A* and *B are obtained from C_k and D_k by introducing ternary relation H.*

Now the proof of Theorem 5 that is the proof that the spoiler wins the $k+1$–moves EF game on A and B iff player I wins the QBF game on ϕ proceeds as follows; we have to consider three cases:

– The spoiler plays lawfully: he wins the game iff ϕ is true.
– The spoiler does not play lawfully, but he can still force pair (x_{k+1}, x'_{k+1}) in the last move; the spoiler has not gained anything by not playing lawfully, he could have played lawfully.
– The spoiler does not play lawfully and cannot force pair (x_{k+1}, x'_{k+1}). In this case the duplicator can follow a winning strategy for the game on $\bar{A_k}$ and $\bar{B_k}$ as in Theorem 6. The only thing we need to check is that the relation H does not cause any problem.

Question: Can we eliminate relation H and show that the EF problem is complete if played on structures over a signature containing a binary relation only?

4 The one–sided and the k–alternations EF games

We show here that the one–sided EF game and the k–alternations EF game are also PSPACE complete.

Theorem 7. *The one–sided EF game is PSPACE complete.*

Proof (sketch): We just show hardness. We reduce the r–moves EF game on structures A and B over some signature Σ to the $r+n$-moves one–sided EF game on structures C and D, on a signature $\Sigma' = \Sigma \cup \{R, B\}$ where R and B are new unary relations, and $n = |V_A| + |V_B|$.(here V_A , V_B are the sets of vertices of structures A and B) Order the elements of $V_A \cup V_B$, so that $V_A = \{1, \ldots, m\}$ and $V_B = \{m+1, \ldots, m+k\}$ for some m and k,then:

– For any $x \in V_A \cup V_B$ there is an element $\bar{x}$ in V_C with x additional new neighbours colored R and n additional new neighbours colored B. V_C also contains many elements with x new neighbours colored R and $n-1$ neighbours colored B. All relations except for R and B are empty in C.
– For any element $(x, y) \in V_A \times V_B \cup V_B \times V_A$ there is a vertex $(\bar{x}, \bar{y})$ in V_D with x new neighbours colored R and n new neighbours colored B. V_D also contains many elements with $\bar{x}$ new neighbours colored R and $n-1$ neighbours colored B. We have $E(\bar{a}, \bar{b})(\bar{c}, \bar{d})$ iff Exy in A but not Ezt in B, or vice versa; where x is the cohordinate a or b of the pair (a, b) that belongs to A, and y is the one that belongs to B. Similarly for all other relations in Σ.

Theorem 8. *The one–sided EF game reduces to the k alternations EF game , for any $k \geq 0$.*

A proof is given in [2].

References

1. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer, Berlin (1995). 159, 161
2. E. Pezzoli. *On the computational complexity of type-two functionals and logical games on finite structures.* Ph.D thesis. Stanford University. June 1998. 160, 162, 169
3. E. Pezzoli *Classical and paramteric complexity of logical games.* In preparation. 160
4. M. Grohe. *Equivalence in finite-variable logics is complete for polynomial time.* Proceeding FOCS 96. 159, 164
5. N. Immerman, D. Kozen. *Definability with a bounded number of variables.* Information and Computation vol 83 (1989) pp. 121–139. 164
6. J. Köbler, U. Schöning,J. Torán. *The Graph Isomorphism Problem.* Birkhäuser 1993. 160
7. L. J. Stockmeyer and A. R. Meyer *Word problems requiring exponential time.* Fifth Annual ACM Symposium on theory of computing. pp. 1–9, Association for Computing Machinery, New york, 1973. 162

An Upper Bound for Minimal Resolution Refutations

Hans Kleine Büning

Department of Mathematics and Computer Science
University of Paderborn
33095 Paderborn (Germany)
kbcsl@uni-paderborn.de

Abstract. We consider upper bounds for minimal resolution refutations of propositional formulas in *CNF* . We show that minimal refutations of minimal unsatisfiable formulas over n variables and $n+k$ clauses consist of at most $2^{k-1}n^2$ applications of the resolution rule.

Keywords: propositional formulas, length of proofs, minimal unsatisfiability, resolution

1 Introduction

A resolution refutation or a proof of an unsatisfiable formula is a sequence of applications of the resolution rule generating the empty clause. The number of resolution steps is the length of the proof and a minimal refutation is a proof with minimal length. Instead of counting the number of applications of the resolution rule often the size of the refutation is considered, where the size is the number of generated clauses. With respect to minimal refutations both measures do not differ.

Here, we restrict ourselves to minimal unsatisfiable formulas. A formula in conjunctive normal form (CNF) is called minimal unsatisfiable if, and only if the formula is unsatisfiable and removing an arbitrary clause leads to a satisfiable formula. Most of the hard examples for the resolution presented in the literature are minimal unsatisfiable.

In [6] it is shown that so called pigeonhole formulas, formulas over $n(n+1)$ variables and $O(n^3)$ clauses, have a minimal proof size of c^n for some $c > 1$ and based on Haken's proof method other hard examples have been invented, see for example [7]. In connection with average case analysis interesting lower bounds habe been established (for example [3], [2]), but besides the obvious upper bound 2^{2n} for refutations of formulas over n variables only few results with respect to upper bounds are known [5].

Our upper bound $2^{k-1}n^2$ for minimal resolution refutations has two parameters. One of them is the number of variables, n, and the other one is the difference between the number of clauses and the number of variables, k. For fixed k there exist unsatisfiable formulas with minimal proofs exponential in the

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 171–178, 1999.

number of variables, but these formulas are not minimal unsatisfiable. Take for example a result shown in [3], that there is some $e > 0$ and that there are unsatisfiable formulas over n variables with about 5.6 clauses for which a resolution refutation requires at least $(1 + e)^n$ steps. In order to prove that there are unsatisfiable formulas with $n + 1$ clauses and exponential minimal proofs, we have to add only some satisfiable clauses over new variables to the formula, such that the generated formula has $m + 1$ clauses and m variables for some appropriate $m > n$.

For minimal unsatisfiable formulas the difference between the number of clauses and the number of variables is always positive, because there exists no minimal unsatisfiable formulas over n variables with less or equal than n clauses. A proof can be found in [1] or one can make use of Hall's Theorem.

We present a resolution procedure which requires not more than $2^{k-1}n^2$ resolution steps where k is the difference between the number of clauses and the number of variables, n. The procedure makes use of a so called splitting theorem. The theorem says that, if each literal occurs at least twice in a minimal unsatisfiable formula, the formula can be splitted into two minimal unsatisfiable formulas where the difference between the number of clauses and the number of variables is less than k. An iterative application of this splitting leads to tree. The size of the tree is bounded depending on k and the tree itself can be used to generate a resolution refutation.

Our upper bound $2^{k-1}n^2$ is only of interest for minimal unsatisfiable formulas with less than $3n$ clauses, because 2^{2n} is the trivial upper bound for resolution refutions of formulas over n variables. As an immediate consequence of our result we see e.g. that for relatively few clauses, say $n + \log n$ clauses, resolution refutations of length not greater than n^3 exists. For minimal unsatisfiable formulas with $1.5n$ clauses we obtain that the size of a minimal proof is bound by $(\sqrt{2})^n n^2$.

2 Notation

A literal is a propositional variable or a negated propositional variable. $var(F)$ is the set of variables of a formula F and $\#var(F)$ is the number of variables. Clauses are sets of literals without multiple occurrences of literals. $\sqcup$ denotes the empty clause. $\#cl(F)$ is the number of clauses for a formula F in CNF. Since formulas with multiple occurrences of clauses are not minimal unsatisfiable, we consider formulas in CNF not as set of clauses but as multi–set of clauses. Often a formula $F = f_1 \wedge \cdots \wedge f_m$ will be written as $F = [f_1, \cdots, f_m]$. Note that the order in which the clauses occur does not play any role.

MU is the set of minimal unsatisfiable formulas in CNF, and for a fixed $k \geq 1$ $MU(k)$ is the set of minimal unsatisfiable formulas over n variables with $n + k$ clauses.

3 Upper Bound

The proof of the upper bound $2^{k-1}n^2$ for minimal resolution refutations of minimal unsatisfiable formulas is based on an induction on k, where k is the difference between the number of clauses and the number of variables.

At first we give an short outline of the proof. Minimal unsatisfiable formulas over n variables with $n+1$ clauses can be refuted in n resolution steps. That follows from the fact that in a minimal unsatisfiable formula over n variables and $n+1$ clauses a variable occurs exactly once positively and exactly once negatively (see [4]). After resolving the clause with x and the clause with $\neg x$, we obtain a minimal unsatisfiable formula over $n-1$ variables and n clauses, where the parent clauses has been removed and the resolvent be added.

For $k > 1$ in a first step we resolve upon all literals occurring exactly once. Then we split the formula into two unsatisfiable formulas by setting an arbitrarily given variable to true resp. false. The generated formulas must contain minimal unsatisfiable subformulas, for which we can show that now the difference between the number of clauses and the number of variables is less than k. By the induction hypothesis the desired upper bound holds for these minimal unsatisfiable formulas. Finally, we combine the resolution proofs and obtain the upper bound.

Let $k > 1$ be given. If a literal L occurs exactly once in a formula $F \in MU(k)$, then we can resolve the clause in which L occurs with all clauses containing the literal $\neg L$ preserving the minimal unsatisfiability. After adding the resolvents and removing the parent clauses the resulting formula is again in $MU(k)$, but now with $n-1$ variables and $n-1+k$ clauses.

This resolution procedure resolving upon literals occurring exactly once is called $(1,*)$–resolution procedure and can be described formally as

$(1,*)$–**resolution procedure:**

while F contains a literal L exactly once and $\neg L$ is a literal in F **do**
$\{F = [(L \vee f), (\neg L \vee g_1), \cdots, (\neg L \vee g_r), F_{rest}]$, *resolve* F *on* $L\}$
$F := [(f \vee g_1), \cdots, (f \vee g_r), F_{rest}]$;
end(while);
return(F)

For $k > 1$ performing these resolution steps as long as such single literals exists cannot lead to the empty clause, because otherwise in the final step a formula $x \wedge \neg x \in MU(1)$ would occur. That would be a contradiction to the fact, that the $(1,*)$–resolution procedure preserves the difference between the number of clauses and the number of variables. Thus, the $(1,*)$–resolution procedure resolves on not more than $q \leq n-1$ variables and the maximal number of resolution steps is bounded from above by $\sum_{1 \leq i \leq q}(n+k-i)$. We summerize these observations in the following Lemma.

Lemma 1. *For $k > 1$ let F be a formula in $MU(k)$:*
The $(1,*)$–resolution procedure returns a formula in $MU(k)$. The procedure resolves on at most $q \leq n-1$ variables and performes at most $\sum_{1 \leq i \leq q}(n+k-i)$ resolution steps.

Now let us recall some results about minimal unsatisfiable formulas with $n+1$ clauses. In [1] it is shown that every minimal unsatisfiable formula over n variables consists of at least $n+1$ clauses. For our proof of the upper bound we need the fact that for formulas in $MU(1)$ a resolution refutation requires not more than n steps where n is the number of variables. This property is an immediate consequence of the first statement of the following theorem.

Theorem 1. *[4]*
1. If $F \in MU(1)$ then there exists a variable occuring exactly once positively and once negatively in F.
2. For formulas in $MU(1)$ over n variables there exists a resolution refutation with exactly n resolution steps.

Later on we make use of a technical Lemma for which we need the following definition.

Definition 1. *Let F be a formula in CNF and X a non–empty subset of $var(F)$. Then $F(X)$ is the result of removing all clauses not containing a variable in X and deleting in the remaining clauses all variables not in X. Note that multiple occurrences of clauses will not be deleted.*

Lemma 2. *Suppose $F \in MU(k)$ for some $k \geq 2$ and each literal occurs at least twice in F. Then for all non–empty $X \subseteq var(F) : \#cl(F(X)) \geq \mid X \mid +2$*

Proof. Induction on the number of variables m in X.
Since $F \in MU$ the formula $F(X)$ must be unsatisfiable. If $m = 1$ then $F(X)$ contains two clauses x and and two clauses $\neg x$, because each literal occurs at least twice in F. Note that we consider F and also $F(X)$ as multi–sets of clauses. Thus, we have $\#cl(F(X)) \geq 1 + 2$.
For $m > 1$ let $X = \{x_1, \cdots, x_m\}$ be the set of variables. Then there is some minimal unsatisfiable formula G with $G \subseteq F(X)$.
If $var(G) = \{x_1, \cdots, x_m\}$ then $\#cl(G) \geq m+1$ (see [1]). Further, if $G \in MU(1)$ then in G a variable occurs exactly once positively and once negatively. Since each literal occurs at least twice in F and no tautological clause is in F, $F(X)$ contains at least two clauses more than G. Hence, we obtain, that $F(X)$ consists of at least $m+1+2$ clauses. If $G \in MU(t)$ for some $t > 1$, then G consists of $m + t \geq m + 2$ clauses.
If $var(G) = \{x_1, \cdots, x_s\}$ for some $s < m$ then $\#cl(F(X)) \geq \#cl(G)+$
$\#cl(F(X - \{x_1, \cdots, x_s\}))$. Since G is minimal unsatisfiable, the formula G consists of at least $s+1$ clauses (see [1]). By the induction hypothesis we get $\#cl(F(X - \{x_1, \cdots, x_s\})) \geq m - s + 2$. Altogether we have $\#cl(F(X)) \geq s + 1 + m - s + 2 \geq m + 2$.

We split a minimal unsatisfiable formula in $MU(k)$ into two minimal unsatisfiable formulas. For a variable x we remove the clauses with literal $\neg x$ (set $\neg x = 1$) resp. x (set $x = 1$). In the remaining clauses we delete the occurrences of the literal x resp. $\neg x$. The formulas are unsatisfiable and contain therefore

some minimal unsatisfiable subformulas, say $F_x \in MU(k_x)$ and $F_{\neg x} \in MU(k_{\neg x})$ for some k_x and $k_{\neg x}$. Now we interested in the size of k_x and $k_{\neg x}$ with respect to k. For arbitrary formulas in $MU(k)$ such a splitting may lead to a minimal unsatisfiable formula F_x with $k_x = k$. Take for example the formula
$F = [\neg x \vee \neg a, a, y \vee z, \neg y \vee z, y \vee \neg z, x \vee \neg y \vee \neg z] \in MU(2)$. A splitting on x leads to minimal unsatisfiable formulas $F_{\neg x} = \neg a \wedge a \in MU(1)$ and $F_x = [y \vee z, \neg y \vee z, y \vee \neg z, \neg y \vee \neg z] \in MU(2)$. That means $k_x = k = 2$, and $k_{\neg x} = 1$.

But as we will see for $k > 1$ and after the application of the $(1, *)$–resolution procedure k_x as well as $k_{\neg x}$ must be less than k.

For example the $(1, *)$–resolution applied with the formula F generates the formula $F^* = [y \vee z, \neg y \vee z, y \vee \neg z, \neg y \vee \neg z] \in MU(2)$. Now a splitting on the variable y leads to minimal unsatisfiable formulas $F^*_y = z \wedge \neg z \in MU(1)$ and $F^*_{\neg y} = z \wedge \neg z \in MU(1)$.

Theorem 2 ((Splitting)). *Suppose $F \in MU(k)$ for some $k \geq 2$ and each literal occurs at least twice in F. Furthermore, we suppose*
$F = [(x \vee f_1), \cdots, (x \vee f_s), B_x, C, B_{\neg x}, (\neg x \vee g_1), \cdots, (\neg x \vee g_t)]$, where $B_x, C, B_{\neg x}$ are some conjunctions of clauses without occurrences of x and $\neg x$, such that
$F_x := [f_1, \cdots, f_s, B_x, C] \in MU(k_x)$, $F_{\neg x} := [g_1, \cdots, g_t, B_{\neg x}, C] \in MU(k_{\neg x})$ for some k_x and $k_{\neg x}$.
Then we have $k_x, k_{\neg x} < k$.

Proof. Suppose F is a formula over n variables. We define $v_x := \#var(F_x)$ and $v_{\neg x} := \#var(F_{\neg x})$. That means F_x resp. $F_{\neg x}$ consists of $v_x + k_x$ resp. $v_{\neg x} + k_{\neg x}$ clauses.

Since the variable x occurs at least twice positively and at least twice negatively in F, we get $s, t \geq 2$ and therefore $v_x + k_x \leq n + k - 2$ and $v_{\neg x} + k_{\neg x} \leq n + k - 2$. That implies for $v_x = n - 1$ resp. $v_{\neg x} = n - 1$ the inequality $k_x \leq k - 1$ resp. $k_{\neg x} \leq k - 1$.
Now we proceed by a case distinction:
Case 1: $v_x = n - 1$ and $v_{\neg x} < n - 1$.
Then we know $k_x \leq k - 1$. Note that $var(F_{\neg x}) \subseteq var(F_x) = var(F) - \{x\}$. For $\{a_1, \cdots, a_l\} = var(F_x) - var(F_{\neg x})$ we have $n = v_{\neg x} + l + 1$.
Since $\#cl(F(\{a_1, \cdots, a_l\}) = \#cl([f_1, \cdots, f_s, B_x](\{a_1, \cdots, a_l\}))$, by Lemma 4 we get $\#cl([f_1, \cdots, f_s, B_x](\{a_1, \cdots, a_l\})) \geq l+2$ and therefore $\#cl([f_1, \cdots, f_s, B_x]) \geq l + 2$. Hence we obtain $v_{\neg x} + k_{\neg x} + l + 2 \leq n + k$.

Assuming $k_{\neg x} \geq k$ leads to $v_{\neg x} + k_{\neg x} + l + 2 \leq n + k_{\neg x}$ and therefore to $v_{\neg x} + l + 2 \leq n$ in contradiction to $v_{\neg x} + l + 1 = n$. Thus we have $k_{\neg x} < k$.
Case 2: $v_x < n - 1$ and $v_{\neg x} = n - 1$ analogue to case 1.
Case 3: $v_x, v_{\neg x} < n - 1$.
For $\{a_1, \cdots, a_l\} = var(F_x) - var(F_{\neg x})$ and $\{b_1, \cdots, b_r\} = var(F_{\neg x}) - var(F_x)$ we have $l, r \geq 1$. Otherwise $var(F_x)$ resp. $var(F_{\neg x})$ would be a subset of $var(F_{\neg x})$ resp. $var(F_x)$. That would imply $\#var(F_{\neg x}) = n - 1$ resp. $\#var(F_x) = n - 1$ in contradiction to our case assumption. Further, a_i and b_j do not occur in C.

By means of Lemma 4 we obtain
$\#cl([f_1, \cdots, f_s, B_x](\{a_1, \cdots, a_l\})) = \#cl(F(\{a_1, \cdots, a_l\})) \geq l + 2$ and
$\#cl([g_1, \cdots, g_t, B_{\neg x}](\{b_1, \cdots, b_r\})) = \#cl(F(\{b_1, \cdots, b_r\})) \geq r + 2$, and therefore
$v_{\neg x} + k_{\neg x} + l + 2 \leq n + k$ and $v_x + k_x + r + 2 \leq n + k$.
The inequalities imply $k_x, k_{\neg x} < k$, because of $v_x + r + 1 = n$ and $v_{\neg x} + l + 1 = n$.

Now we are able to prove our desired upper bound for minimal resolution refutations of minimal unsatisfiable formulas.

Theorem 3. *Let F be a minimal unsatisfiable formula over n variables with $n + k$ clauses. Then a minimal resolution refutation of F requires not more than $2^{k-1}n^2$ resolution steps.*

Proof. (Induction on k)
For $k = 1$ the upper bound is n, because of Theorem 2.

For $k > 1$ and a formula $F \in MU(k)$ we first apply the $(1, *)$–resolution procedure to F. Suppose the $(1, *)$–resolution procedure resolves on q variables. Please, note that q must be less than n. Then an upper bound for the number of these resolution steps is $\sum_{1 \leq i \leq q}(n + k - i)$ and the generated formula is in $MU(k)$, see Lemma 1.
Say F^S is the formula we obtain after applying the $(1, *)$–resolution procedure to F. Then in F^S each literal occurs at least twice. Due to Theorem 5 we split the formula F^S over a variable x into two minimal unsatisfiable formulas $F_x \in MU(k_x)$ and $F_{\neg x} \in MU(k_{\neg x})$ for some k_x and $k_{\neg x}$. Then it holds $k_x, k_{\neg x} < k$. Further, the formulas F_x and $F_{\neg x}$ contain at most $n - q - 1$ variables. W.l.o.g. we suppose that $k_x = k_{\neg x} = k - 1$ and both formulas have the maximal number of variables $n - q - 1$.

By the induction hypothesis we obtain for the minimal resolution refutation of F_x and $F_{\neg x}$ the upper bound $2^{k-2}(n - q - 1)^2$. Now we add to each clause in F_x resp. $F_{\neg x}$ the removed variable x resp. $\neg x$. Say F_x^x and $F_{\neg x}^{\neg x}$ are the resulting formulas. Then x and $\neg x$ can be derived in $2 \cdot 2^{k-2}(n - q - 1)^2$ resolution steps. Finally, in one step we obtain the empty clause.

Since F_x^x and $F_{\neg x}^{\neg x}$ are subformulas of F^S we obtain as an upper bound for a minimal resolution refutation for F:

$$\sum_{1 \leq i \leq q} (n + k - i) + 1 + 2^{k-1}(n - q - 1)^2 \leq 2^{k-1}n^2 \tag{1}$$

The last inequality can be shown by an induction on k. Please, note that $k \geq 2, n \geq 1$, and $0 \leq q \leq n - 1$.
For $k = 2$ we start with the inequality

$$0 \leq 3q^2 + 3q + 2 \tag{2}$$

Then we add $11q + 3q^2 + 6$ to the inequality

$$11q + 3q^2 + 6 \leq 6q^2 + 6q + 8q + 8 \tag{3}$$

$$11q + 3q^2 + 6 \leq (q+1)(6q+8) \tag{4}$$

Since $q < n$ we obtain

$$11q + 3q^2 + 6 \leq n(6q+8) \tag{5}$$

$$11q/2 + 3q^2/2 + 3 \leq 3nq + 4n \tag{6}$$

The inequality is equivalent to

$$nq + 6q - q/2 + 2q^2 - q^2/2 + 3 - 4nq - 4n \leq 0 \tag{7}$$

Then we obtain

$$\sum_{1 \leq i \leq q} (n+2-i) + 4q + 2q^2 + 2 - 4nq - 4n + 2n^2 + 1 \leq 2n^2 \tag{8}$$

and finally

$$\sum_{1 \leq i \leq q} (n+2-i) + 2(n-q-1)^2 + 1 \leq 2n^2 \tag{9}$$

For $k+1$ we start with the following inequality (please, note that $q < n$).

$$q \leq \sum_{1 \leq i \leq q} (n+k-i) + 1 \tag{10}$$

Now we add $\sum_{1 \leq i \leq q}(n+k-i) + 1 + 2^k(n-q-1)^2$ and obtain

$$\sum_{1 \leq i \leq q} (n+k-i) + q + 1 + 2^k(n-q-1)^2 \leq 2 \sum_{1 \leq i \leq q} (n+k-i) + 2 + 2^k(n-q-1)^2 \tag{11}$$

Using (11) and the induction hypothesis (IH) we obtain our desired result for $k+1$.

$$\sum_{1 \leq i \leq q} (n+k+1-i) + 1 + 2^k(n-q-1)^2 = \tag{12}$$

$$\sum_{1 \leq i \leq q} (n+k-i) + q + 1 + 2^k(n-q-1)^2 \leq_{(11)} \tag{13}$$

$$2 \sum_{1 \leq i \leq q} (n+k-i) + 2 + 2^k(n-q-1)^2 = \tag{14}$$

$$2 \cdot [\sum_{1 \leq i \leq q} (n+k-i) + 1 + 2^{k-1}(n-q-1)^2] \leq_{IH} 2(2^{k-1}n^2) \leq 2^k n^2 \tag{15}$$

4 Conclusion and Future Work

The upper bound $2^{k-1}n^2$ is not sharp, because for example for $k = 1$ minimal resolution refutations require exactly n resolution steps. Maybe a more careful analysis of the number of $(1, *)$–resolutions and the number of variables occurring in the splitting leads to a better bound. Otherwise as far as we know no minimal unsatisfiable formulas with minimal proof length of at least $2^{k-1}n$ and relatively few clauses, say for example for fixed k or for $n + t \cdot logn$ clauses are known.

In order to solve these problems and (more generally) in order to prove complexity of different calculi, we must reference more information about the structure of minimal unsatisfiable formulas. We expect an investigation of the particular structure of minimal unsatisfiable formulas with a fixed difference between the number of clauses and the number of variables and a better characterization of minimal unsatisfiable formulas to be the key to more accurate results. Note that the structure of formulas in MU(1) and MU(2) is well–known, e.g. formulas in MU(2) in which each each literal occurs at least twice are of the form $(x_1 \vee \cdots \vee x_n), (\neg x_1 \vee x_2), \cdots, (\neg x_n \vee x_1), (\neg x_1 \vee \cdots \vee \neg x_n)$. Obviously these formulas can be refuted by means of the resolution operation in $2n$ steps. Similar results would be desirable for arbitrary MU(k).

References

1. Aharoni, R., Linial, N.: Minimal Non–Two–Colorable Hypergraphs and Minimal Unsatisfiable Formulas. Journal of Combinatorial Theory **43** (1986), 196–204 172, 174, 174, 174
2. Beame, P., Pitassi, T.: Simplified and Improved Resolution Lower Bounds. FOCS 96 (1996) 274–282 171
3. Chvatal, V., Szemeredi, T.: Many hard Exanples for Resolution. Journal of the Association for Computing Maschinery **35** (1988) 759–768 171, 172
4. Davydov, G., Davydova, I., Kleine Büning, H.: An Efficient Algorithm for the Minimal Unsatisfiability Problem for a Subclass of CNF. to appear in Annals of Mathematics and Artificial Intelligence 173, 174
5. Fu, X.: On the Complexity of Proof Systems. University of Toronto, PhD Thesis, (1995) 171
6. Haken, A.: The intractibility of Resolution. Theoretical Computer Science **39** (1985) 297–308 171
7. Urquhart, A.: Hard Examples for Resolution. Journal of ACM **34** (1987) 209 –219 171

On an Optimal Deterministic Algorithm for SAT

Zenon Sadowski

Institute of Mathematics,
University of Białystok
15-267 Białystok, ul. Akademicka 2, Poland
sadowski@math.uwb.edu.pl

Abstract. J. Krajíček and P. Pudlák proved that an almost optimal deterministic algorithm for $TAUT$ exists if and only if there exists a p-optimal proof system for $TAUT$. In this paper we prove that an almost optimal deterministic algorithm for SAT exists if and only if there exists a p-optimal proof system for SAT. Combining Krajíček and Pudlák's result with our result we show that an optimal deterministic algorithm for SAT exists if and only if both p-optimal proof systems for $TAUT$ and for SAT exist.

1 Introduction

A deterministic algorithm recognizing SAT is optimal if no other algorithm recognizing SAT has more than a polynomial speed-up over its running time (see [5]). Two versions of optimality appear in Computational Complexity: Levin's optimality and Krajíček - Pudlák's optimality. In this paper we are mainly concerned with an optimal algorithm possessing Krajíček - Pudlák's optimality property. If the optimality property is stated only for any input string x which belongs to SAT and nothing is claimed for other x's, we name such an algorithm as an almost optimal deterministic algorithm for SAT.

A proof system for a language L is a polynomial time computable function whose range is L. This notion was defined by S. Cook and R. Reckhow in [3]. In order to compare the efficiency of different proof systems, they considered the notion of p-simulation. Intuitively a proof system h p-simulates a second one g if there is a polynomial time computable function t translating proofs in g into proofs in h. A proof system is called p-optimal for L when it p-simulates any proof system for L. The question whether a p-optimal proof system for $TAUT$ (the set of tautologies in Propositional Logic) exists, posed by J. Krajíček and P. Pudlák, is an important one in the field (see [5], [7], [4]).

The problem of the existence of an optimal deterministic algorithm for SAT has been considered as early as the beginning of the **NP**-era. In [6] L. Levin observed that there exists an almost optimal deterministic algorithm for the functional version of SAT (see [8]). Using Levin's construction we can also build an almost optimal deterministic algorithm for the decision version of SAT with the optimality property named by us as Levin's optimality (see [9]).

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 179–187, 1999.

Nearly two decades later J. Krajíček and P. Pudlák considered the notion of an almost optimal deterministic algorithm for $TAUT$ (see [5]). Their concept of optimality differs from the one attributed to L. Levin and is named by us as Krajíček-Pudlák's optimality. They proved that an almost optimal deterministic algorithm for $TAUT$ exists if and only if there exists a p-optimal proof system for $TAUT$.

The question whether it is also possible to characterize an optimal deterministic algorithm for $TAUT$ (or equivalently an optimal deterministic algorithm for SAT) in p-optimal proof systems terms arises naturally from Krajíček and Pudlák's work. We answer this question. We prove that an almost optimal deterministic algorithm for SAT exists if and only if there exists a p-optimal proof system for SAT. Combining Krajíček and Pudlák's result with our result we show that an optimal deterministic algorithm for SAT exists if and only if both proof systems for $TAUT$ and for SAT exist.

In the last section we discuss the problem of the existence of an optimal deterministic algorithm for SAT with Levin's optimality property.

The inspiration to write this paper comes from [5,4].

2 Preliminaries

We assume some familiarity with basic complexity theory, see [1]. The symbol Σ denotes, throughout the paper, a certain fixed finite alphabet. The set of all strings over Σ is denoted by $\Sigma^\star$. For a string x, $|x|$ denotes the length of x. The symbol FP denotes the class of functions that can be computed in polynomial time.

We use Turing machines (acceptors and transducers) as our basic computational model. For a deterministic Turing machine M and an input w TIME(M, w) denotes the computing time of M on w.

We consider clocked deterministic polynomial time Turing transducers (PTM for short). In the sequel we will not distinguish between a machine and its code. We impose some restrictions on our encoding of PTMs. From the code of any PTM we can detect easily (in polynomial time) the natural k such that $n^k + k$ is its polynomial time bound. We consider only languages over the alphabet Σ (this means that, e. g. , boolean formulas have to be suitably encoded). The symbol $TAUT$ denotes the set (of all encodings) of propositional tautologies, SAT denotes the set of all satisfiable propositional formulas and $UNSAT$ denotes the set of all unsatisfiable propositional formulas.

Finally, $< ., \ldots, . >$ denotes some standard polynomial time computable tupling function.

3 Optimal algorithms and p-optimal proof systems

The notion of an almost optimal deterministic algorithm for $TAUT$ was introduced by J. Krajíček and P. Pudlák. We propose to consider almost optimal

and optimal deterministic algorithms for a language L with Krajíček-Pudlák's optimality property.

Definition 1. *(cf. [5]) An almost optimal deterministic algorithm for L is a deterministic Turing machine M which recognizes L and such that for every deterministic Turing machine M' which recognizes L there exists a polynomial p such that for every $x \in L$*

$$TIME(M; x) \leq p(|x|, TIME(M'; x))$$

If we state the optimality conditions for both "yes" and "no" inputs we obtain the definition of an optimal deterministic algorithm for L.

Definition 2. *An optimal deterministic algorithm for L is a deterministic Turing machine M which recognizes L and such that for every deterministic Turing machine M' which recognizes L there exists a polynomial p such that for any $x \in \Sigma^\star$*

$$TIME(M; x) \leq p(|x|, TIME(M'; x))$$

Fact 1. If an optimal deterministic algorithm for SAT exists then $\mathbf{P} = \mathbf{NP}$ if and only if this algorithm works in polynomial time.

Fact 2. The following statements are equivalent.

(i) There exists an optimal deterministic algorithm for SAT.
(ii) There exists an optimal deterministic algorithm for $TAUT$.
(iii) There exists an optimal deterministic algorithm for $UNSAT$.

Fact 3. The following statements are equivalent.

(i) There exists an almost optimal deterministic algorithm for $TAUT$.
(ii) There exists an almost optimal deterministic algorithm for $UNSAT$.

A systematic study of complexity of proof systems for Propositional Logic was started by S. Cook and R. Reckhow in [3]. They introduced the abstract notion of a proof system for $TAUT$. The question of the existence of a polynomially bounded proof system for $TAUT$ is connected with the problem **NP=co-NP?** (see [3,7]). The existence of an optimal proof system for $TAUT$ is an important open question introduced by J. Krajíček and P. Pudlák in [5].

J. Köbler and J. Messner defined the abstract notion of a proof system for a language L in the following way:

Definition 3. *(see [4]) A proof system for L is a polynomial time computable function $h : \Sigma^\star \stackrel{onto}{\longrightarrow} L$.*

If $h(w) = x$ we say that w is a proof of x in h.

It follows from the above definition that for any proof system h there exists a PTM M_h which computes h. Proof systems for L can be treated as nondeterministic algorithms accepting L.

The notion of p-simulation is useful in comparing the strength of different proof systems for L.

Definition 4. *(see [3]) Let h, h' be two proof systems for L. We say that h p-simulates h' if there exists a polynomial time computable function $\gamma : \Sigma^\star \longrightarrow \Sigma^\star$ such that for every $x \in L$ and every $w \in \Sigma^\star$, if w is a proof of x in h', then $\gamma(w)$ is a proof of x in h.*

Definition 5. *(see [4]) A proof system is p-optimal for L if it p-simulates any proof system for L.*

The family of proof systems for SAT contains a proof system which can be named natural. Any truth assignment satisfying a formula α is its proof in this system. If the natural proof system for SAT was p-optimal then the assertion PRIMES belongs to **P** would be equivalent to the assertion FACTORING is in FP. Since FACTORING seems much harder than PRIMES, hence the natural proof system for SAT is not likely p-optimal.

4 Test formulas

In this section we construct boolean formulas which can be used to verify for a given PTM M and an input w that M on input w produces a satisfiable boolean formula. We use these formulas in the proof of Theorem 1.

Definition 6. *We say that a PTM M behaves well on input w if M on w outputs a satisfiable boolean formula.*

To any PTM M and any $w \in \Sigma^\star$ we shall assign the boolean formula $TEST_{M,w}$ such that:
The formula $TEST_{M,w}$ is satisfiable if and only if M behaves well on input w.

Our construction of the formula $TEST_{M,w}$ is adapted from Cook's proof that SAT is **NP**-complete (cf. [2]). Let S be a fixed nondeterministic Turing machine working in polynomial time and accepting SAT. Let M' be the Turing machine which on any input x runs M and then runs S on the output produced by M. The formula $TEST_{M,w}$ is just Cook's formula for the pair $< M', w >$.

From the construction of test formulas it follows that they possess the following properties:

(1) Global uniformity property
There exists a function $f \in FP$ such that for any PTM N with time bound $n^k + k$ and for any $w \in \Sigma^\star$

$$f(\langle N, w, 0^{|w|^k+k}\rangle) = TEST_{N,w}$$

(2) Local uniformity property
Let M be any fixed PTM. There exists a function $f_M \in FP$ such that for any $w \in \Sigma^\star$

$$f_M(\langle M, w\rangle) = TEST_{M,w}$$

Let T be an almost optimal deterministic algorithm for SAT. The following lemma intuitively says that T easily accepts test formulas for any fixed proof system for SAT.

Let $k : \Sigma^\star \stackrel{onto}{\longrightarrow} SAT$ be a proof system for SAT. Then there exists a certain PTM M with time bound $n^l + l$ and such that M computes k.

Lemma 1. *For every PTM M such that M is the machine of a certain proof system $k : \Sigma^\star \stackrel{onto}{\longrightarrow} SAT$ there exists a polynomial p such that for every $w \in \Sigma^\star$ it holds: $TIME(T, TEST_{M,w}) \leq p(|w|)$.*

Proof. Since M is the machine of the proof system k, $TEST_{M,w}$ is a satisfiable boolean formula for every $w \in \Sigma^\star$. The machine M is fixed, so there exists a deterministic Turing machine which recognizes in polynomial time the language $\{TEST_{M,w}: w \in \Sigma^\star\}$. Combining this machine with the "brute force" algorithm for SAT we obtain the deterministic machine N recognizing SAT and such that the following condition holds: There exists a polynomial q such that for every $w \in \Sigma^\star$
$TIME(N, TEST_{M,w}) \leq q(|w|)$. The desired conclusion follows from the definition of an almost optimal deterministic algorithm for SAT.

5 Consistent Turing machines

The notion of a consistent Turing machine will be used in the proof of Theorem 1.

Let M be a deterministic Turing machine.

Definition 7. *We say that M is consistent if M accepts only satisfiable boolean formulas (if M accepts w, then $w \in SAT$).*

Fact 4. For every consistent Turing machine M there exists the proof system $g_M : \Sigma^\star \stackrel{onto}{\longrightarrow} SAT$ such that for every w accepted by M
$g_M(x) = w$, where x is the computation of M accepting w.

The function g_M is the proof system for SAT generated by M. Because we impose conditions only on w's accepted by M, there are many proof systems generated by M.

6 Krajíček - Pudlák's optimality

In this section we prove the equivalence between the problem of the existence of an optimal deterministic algorithm for SAT and the problem of whether SAT and $TAUT$ have p-optimal proof systems. This is the main result of our paper.

Theorem 1. *Statements (i) - (ii) are equivalent.*

(i) There exists an almost optimal deterministic algorithm for SAT.
(ii) There exists a p-optimal proof system for SAT.

Proof. $(i) \rightarrow (ii)$
Let T be an almost optimal deterministic algorithm for SAT. We say that a string $v \in \Sigma^\star$ is in good form if

$$v = \langle M, w, 0^{|w|^k+k}, Comp \rangle$$

where: M is a PTM with $n^k + k$ time bound, $w \in \Sigma^\star$, $0^{|w|^k+k}$ is the sequence of zeros, $Comp$ is the computation of T accepting the formula $TEST_{M,w}$.

Let us notice, that if v is in good form then $TEST_{M,w}$ is a satisfiable boolean formula, hence PTM M behaves well on input w producing a certain satisfiable boolean formula α. The string v is a natural candidate for the proof of the formula α in a p-optimal proof system for SAT.

Let α_0 be a certain fixed satisfiable boolean formula. We define
$h : \Sigma^\star \longrightarrow \Sigma^\star$ in the following way:

$h(v) = \alpha$ if v is in good form ($v = \langle M, w, 0^{|w|^k+k}, Comp \rangle$) and α is a boolean formula produced by M on input w, otherwise $h(v) = \alpha_0$.

Clearly, $h : \Sigma^\star \stackrel{onto}{\longrightarrow} SAT$. It follows from global uniformity condition from Section 4 that $h \in FP$.This proves that h is a proof system for SAT.

We have to show that h p-simulates any proof system for SAT. Let g be a proof system for SAT computed by PTM M with time bound $n^k + k$.

The function $t : \Sigma^\star \longrightarrow \Sigma^\star$

$$t(x) = \langle M, x, 0^{|x|^k+k}, Comp \rangle$$

translates proofs in g into proofs in h.

The word $Comp$ in the definition of t is the computation of T acccepting the formula $TEST_{M,x}$. It follows from local uniformity property and Lemma 1 (see Section 4) that $Comp$ can be constructed from x in polynomial time. This proves that $t \in FP$.

$(ii) \rightarrow (i)$ Our proof technique is adapted from [5].

Let Opt be the machine of a p-optimal proof system for SAT (Opt works in polynomial time). Let M_0, M_1, M_2, M_3, ... be an enumeration of all deterministic Turing machine acceptors such that M_0 is the trivial "brute force" algorithm recognizing SAT. Let T_1, T_2, T_3, ... be an enumeration of all Turing machine transducers.

The desired machine T, which is an almost optimal deterministic algorithm for SAT, is constructed as follows. It has two distinguished worktapes QUEUE FOR VERIFICATION and PARTIAL RESULTS. These tapes are blanked at first.

On an input w, $|w| = n$, T simulates the work of M_0, M_1, M_2, M_3, ..., M_n and T_1, T_2, T_3, ... T_n in several rounds. At the m-th round the machine T performs the following three groups of operations:

1. One additional computational step of M_0, M_1, M_2, M_3, ..., M_n on w. The histories of computations of M_1, M_2, M_3, ..., M_n are stored on the PARTIAL RESULTS tape, so performing computational steps of M_1, M_2, M_3, ..., M_n, the machine T updates the contents of the PARTIAL RESULTS tape. If a certain machine M_i, $1 \leq i \leq n$, accepts w and x is the computation of M_i accepting w, then $\langle M_i, x\rangle$ is stored on the QUEUE FOR VERIFICATION tape ($\langle M_i, x\rangle$ is added to QUEUE FOR VERIFICATION).
2. One additional computational step of $T_1, T_2, T_3, \ldots, T_n$ on every input x such that $\langle M_i, x\rangle$ for a certain M_i is stored on QUEUE FOR VERIFICATION at that time.
3. T checks whether there are strings x, y and integers $i, k \leq n$ such that:
   (a) M_i has accepted w and $\langle M_i, x\rangle$ is on the QUEUE FOR VERIFICATION tape.
   (b) T_k has produced on input x a string y.
   (c) y is a proof of w in *Opt*.

   If this is the case, T halts and accepts w, otherwise it continues operating till M_0 halts and delivers YES or NO result.

First we shall show that T recognizes SAT. If T finishes the simulation of M_0, then this is clear. So suppose that M accepts w because the situation in (3) occurs. Since w has an *Opt*-proof, w is a satisfiable boolean formula.

Let N be any deterministic Turing machine which recognizes SAT. $N = M_i$ for a certain i. Since N is consistent, there exists a proof system generated by N, and there exists j such that T_j translates proofs in this system into *Opt*-proofs. There exists also a polynomial p bounding the running time of T_j.

Let w be any input such that $|w| \geq \max\{i, j\}$ and $w \in SAT$. Let $k = TIME(M_i, w)$. T accepts w in $p(k) + k$ round or sooner. Since the mth round of T takes polynomially many steps in n and m, there is a polynomial q such that $TIME(T, w) \leq q(n, k)$. This proves that T is almost optimal deterministic algorithm for SAT.

The following result was proved in [5].

Theorem 2. *(Krajíček, Pudlák) The following statements are equivalent.*
(i) There exists a p-optimal proof system for TAUT.
(ii) There exists an almost optimal deterministic algorithm for TAUT.

Using Theorem 1, Theorem 2, Fact 2, and Fact 3 we can prove our main theorem.

Theorem 3. *The following statements are equivalent.*
(i) TAUT and SAT have p-optimal proof systems.
(ii) There exists an optimal deterministic algorithm for SAT.
(iii) There exists an optimal deterministic algorithm for TAUT.

7 Levin's optimality

The following theorem is attributed to L. A. Levin in [9].

Theorem 4. *(Levin)*
There exists a deterministic Turing machine L which recognizes SAT and such that for every deterministic Turing machine N which recognizes SAT there exists a polynomial p such that for any $w \in SAT$

$$TIME(L, w) \leq p(|w|, \max\{TIME(N, v) : \ v \in SAT, |v| \leq |w|\})$$

The machine L is named by us as an almost optimal deterministic algorithm for SAT with Levin's optimality property. We propose to introduce the following notion.

Definition 8. *An optimal deterministic algorithm for SAT with Levin's optimality property is a deterministic Turing machine M which recognizes SAT and such that for every deterministic Turing machine M' which recognizes SAT there exists a polynomial p such that for any $w \in \Sigma^\star$*

$$TIME(M, w) \leq p(|w|, max\{TIME(M', v) : \ |v| \leq |w|\})$$

It follows from Theorem 2, Fact 3 and Theorem 4 that the existence of a p-optimal proof system for $TAUT$ implies the existence of an optimal deterministic algorithm for SAT with Levin's optimality property.

Acknowledgements

The author wishes to thank an anonymous referee for the information, presented at the end of section 3, about the consequences of the p-optimality of the natural proof system for SAT.

References

1. Balcazar, J.L., Díaz, J., Gabarró, J.: Structural complexity I. 2nd edn. Springer-Verlag, Berlin Heidelberg New York (1995) 180
2. Cook, S.A.: The complexity of theorem proving procedures. In: Proc. 3rd ACM Symposium on Theory of Computing. (1971) 151–158 182
3. Cook, S.A., Reckhow R.A.: The relative efficiency of propositional proof systems. J. Symbolic Logic 44 (1979) 36–50 179, 181, 181, 182
4. Köbler, J., Messner, J.: Complete Problems for Promise Classes by Optimal Proof Systems for Test Sets. In Proceedings of the 13th Annual IEEE Conference on Computational Complexity.(1998) 132–140 179, 180, 181, 182
5. Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. J. Symbolic Logic 54 (1989) 1063–1079 179, 179, 180, 180, 181, 181, 184, 185
6. Levin, L.A.: Universal sorting problems. Problems of Information Transmission 9 (1973) 265 –266 179

7. Messner, J., Torán, J.: Optimal proof systems for Propositional Logic and complete sets. Electronic Colloquium on Computational Complexity. (1997) available via http://www.eccc.uni-trier.de/eccc/ 179, 181
8. Trakhtenbrot, B.A.: A survey of Russian Approaches to Perebor (Brute-Force Search) Algorithms. Annals of the History of Computing 6 (1984) 384–400 179
9. Verbitskii, O.V.: Optimal algorithms for co-NP sets and the EXP=NEXP problem. Matematicheskie Zametki M.V. Lomonosov Moscow State University 50 (1991) 37–46 179, 186

Characteristic Properties of Majorant-Computability over the Reals *

M. V. Korovina[1] and O. V. Kudinov[2]

[1] Insitute of Informatics Systems, Lavrent'ev pr., 6,
Novosibirsk, Russia
rita@ssc.nsu.ru
[2] Institute of Mathematics, University pr., 4,
Novosibirsk, Russia
kud@math.nsc.ru

Abstract. Characteristic properties of majorant-computable real-valued functions are studied. A formal theory of computability over the reals which satisfies the requirements of numerical analysis used in Computer Science is constructed on the base of the definition of majorant-computability proposed in [13]. A model-theoretical characterization of majorant-computability real-valued functions and their domains is investigated. A theorem which connects the graph of a majorant-computable function with validity of a finite formula on the set of hereditarily finite sets on $\bar{\mathbb{R}}$, $\mathbf{HF}(\bar{\mathbb{R}})$ (where $\bar{\mathbb{R}}$ is a proper elementary enlargement of the standard reals) is proven. A comparative analysis of the definition of majorant-computability and the notions of computability earlier proposed by Blum et al., Edalat, Sünderhauf, Pour-El and Richards, Stoltenberg-Hansen and Tucker is given. Examples of majorant-computable real-valued functions are presented.

1 Introduction

In the recent time, attention to the problems of computability over uncountable structures, particularly over the reals, is constantly raised. The theories proposed by Barwise [1], Scott [18], Ershov [7], Grzegorczyk [9], Moschovakis [15], Freedman [8] got further development in the works of Blum, Shub, Smail [4], Poul-El, Richards [16], Edalat, Sünderhauf [6], Stoltenberg-Hansen, Tucker [19] Korovina, Kudinov [13] and others. This work continues the investigation of the approach to computability proposed in [13].

Developing our approach we took into consideration the following requirements:

1. Our notion of computability should involve only effective processes.
2. Its definition should contain minimal number of limitations.

* This reseach was supported in part by the RFBR (grant N 96-15-96878) and by the Siberian Division of RAS (a grant for young reseachers, 1997)

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 188–203, 1999.

3. The class of computable real-valued functions should have clear and exact classifications in logical and topological terms.
4. This class should also contain the classes of computable real-valued functions proposed in earlier works by Blum Shub, Smail; Poul-El, Richards; Stoltenberg-Hansen, Tucker; Edalat, Sünderhauf as subclasses.

According to these requirements we construct a notion of computability over the reals with the following properties:

1. In our approach, the computation of a real-valued function is an infinite process that produces approximations closer and closer to the result.

2. This approach does not depend on the way of representing the reals. The definition of computability does not limit the class of considered functions by the property of continuity and the property of being total. Also uniform convergence of processes that produce approximations to a computed function is not required. The use of nonstandard models of the first-order theory of the reals enables us to investigate properties of computability of partial real-valued functions.

3. Special attention is paid to definability of majorant-computable real-valued functions and their domains. A theorem which connects the graph of a majorant-computable function with validity of a finite formula in the set of hereditarily finite sets $\mathbf{HF}(\bar{\mathbb{R}})$ (where $\bar{\mathbb{R}}$ is a proper elementary enlargement of the standard real numbers) is proven. The property of definability can be considered as a denotational semantics of computational processes. Also we give a characterization of the domains of majorant-computable functions via formulas.

4. A comparative analysis of the definition of majorant-computability and the notions of computability earlier proposed by Blum et. al., Poer-El and Richards, Edalat and Sünderhauf is given. For continuous total real-valued functions, the class of majorant-computable functions coincides with the class of computable functions introduced by Pour-El, Richards, and with the class introduced by Edalat and Sünderhauf. For partial real-valued functions, the class of computable functions introduced by Edalat and Sünderhauf is contained in the class of majorant-computable functions.

In our approach, the majorant-computable functions include an interesting class of real-valued total functions that admit meromorphic continuation onto $\mathbb{C}$. This class, in particular, contains functions that are solutions of well-known differential equations.

2 Basic Notions

Throughout the article, $< \mathbb{R}, 0, 1, +, \cdot, \leq >$ is the standard model of the reals, denoted also by $\mathbb{R}$, where $+$ and $\cdot$ are regarded as predicate symbols. Let $\mathrm{Th}(\mathbb{R})$ be the first order theory of $\mathbb{R}$. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{Q}$ the set of rational numbers.

We use definability as one of the basic conceptions. Montague [14] proposed to consider computability from the point of view of definability. Later, many authors among them Ershov [7], Moschovakis [15] paid attention to properties

of this approach applied to various basic models. We base on the following notion of definability in a structure $\mathbf{M} = \langle M, \sigma_0 \rangle$, where σ_0 is a finite first-order language. Bold face indicates sequences, in particular, $\mathbf{x} = x_1, \ldots, x_n$.

A formula $\Phi(\bar{x})$ is said to *define* the set $\{\mathbf{r} \in M^n | \mathbf{M} \models \Phi(\mathbf{r})\}$, and a set of this form is called a *definable* subset of M^n. A function $F : M^m \to M$ is said to be *definable* if its graph is definable (in $\mathbf{M}$).
Recall some properties of definable sets and functions in $\mathbb{R}$. The following statements are proven in [5, 12]. Denote by $\mathbb{R}_0$ the real closure of the rational numbers. We use standard notations for real intervals (a, b), $[a, b]$, $(a, b]$, $[a, b)$.
In addition, $\langle a, b \rangle$ denotes any of these intervals.

Proposition 1 ((0-minimality property)).

1. *A set $B \subseteq \mathbb{R}$ is definable if and only if there exist $n \in \omega$ and $a_1, b_1, \ldots, a_n, b_n \in \mathbb{R}_0 \cup \{+\infty, -\infty\}$ such that $B = \bigcup_{i \leq n} \langle a_i, b_i \rangle$.*
2. *Each definable subset of $\mathbb{R}^n$ is the disjoint union of finitely many cells, each of which is also definable. (A cell is a space that homeomorfic to some $\mathbb{R}^k$, where $k \leq n$.)*

Proof. See [2, 5].

Since the theory $\mathrm{Th}(\mathbb{R})$ admits elimination of quantifiers, it follows that 0-minimality property holds on every model of $\mathrm{Th}(\mathbb{R})$.
For a function f let us denote

$$\begin{aligned} \mathrm{dom}(f) &= \{x \mid \exists y\, (f(x) = y)\}, \\ \mathrm{im}(f) &= \{y \mid \exists x\, (f(x) = y)\}, \\ \Gamma_f &= \{(x, y) \mid [(x, y) \in \mathrm{dom}(f) \times \mathrm{im}(f)] \wedge [f(x) = y]\} \ . \end{aligned}$$

A function $f : M \to L$ is called *total* if $\mathrm{dom}(f) = M$.

Definition 2. *A partial real-valued function $f : \mathbb{R} \to \mathbb{R}$ is said to be algebraic if for some $a, b, c, d \in \mathbb{R}_0 \cup \{-\infty, +\infty\}$ and a polynomial $p \in \mathbb{Q}[x, y]$, the following conditions hold:*

1. $\mathrm{dom}(f) = \langle a, b \rangle$;
2. $\mathrm{im}(f) = \langle c, d \rangle$;
3. $\Gamma_f = \{(x, y) \mid [(x, y) \in \langle a, b \rangle \times \langle c, d \rangle] \wedge [p(x, y) = 0]\}$.

Proposition 3. *A real-valued function $f : \mathbb{R} \to \mathbb{R}$ is definable if and only if the following conditions hold.*

1. *There exist $n \in \omega$ and $a_1, b_1, \ldots, a_n, b_n \in \mathbb{R}_0 \cup \{+\infty, -\infty\}$ such that $\mathrm{dom}(f) = \bigcup_{i \leq n} \langle a_i, b_i \rangle$ and $\langle a_j, b_j \rangle \bigcap \langle a_i, b_i \rangle = \emptyset$ for $i \neq j$.*
2. *f is an algebraic function on the interval $\langle a_i, b_i \rangle$ for all $i \leq n$.*

Proof. See [2, 12].

The previous proposition implies that a real-valued function f is definable if and only if f is the finite union of algebraic functions defined on disjoint intervals with algebraic endpoints. In order to study richer classes of real-valued functions then the above one we need an enlargement of our basic model.

Let us construct the set of hereditarily finite sets $\mathrm{HF}(M)$ over a model $\mathbf{M}$. This structure is rather well studied in the theory of admissible sets [1, 7] and permits us to define the natural numbers, to code, and to store information via formulas.

Let $\mathbf{M}$ be a model whose language σ_0 contains no function symbols and whose carrier set is M. We construct the set of hereditarily finite sets, $\mathrm{HF}(M)$, as follows:

1. $\mathrm{S}_0(M) \rightleftharpoons M$, $\mathrm{S}_{n+1}(M) \rightleftharpoons \mathcal{P}_\omega(\mathrm{S}_n(M)) \cup \mathrm{S}_n(M)$, where $n \in \omega$ and for every set B, $\mathcal{P}_\omega(B)$ is the set of all finite subsets of B.
2. $\mathrm{HF}(M) = \bigcup_{n\in\omega} \mathrm{S}_n(M)$.

We define $\mathbf{HF}(\mathbf{M}) \rightleftharpoons \left\langle \mathrm{HF}(M), M, \sigma_0, \emptyset_{\mathbf{HF(M)}}, \in_{\mathbf{HF(M)}} \right\rangle$, where $\emptyset_{\mathbf{HF(M)}}$ and the binary predicate symbol $\in_{\mathbf{HF(M)}}$ have the set-theoretic interpretation. Denote $\sigma = \sigma_0 \cup \{\emptyset_{\mathbf{HF(M)}}, \in_{\mathbf{HF(M)}}\}$. The notions of a term and an atomic formula are given in a standard manner.

The set of Δ_0-*formulas* is the closure of the set of atomic formulas in the language σ under $\wedge, \vee, \neg, \exists x \in t$ and $\forall x \in t$, where $\exists x \in t\ \varphi$ denotes $\exists x(x \in t \wedge\ \varphi)$ and $\forall x \in t\ \varphi$ denotes $\forall x(x \in t \to \varphi)$. The set of Σ-*formulas* is the closure of the set of Δ_0 formulas under $\wedge, \vee, \exists x \in t, \forall x \in t$, and $\exists$. We define Π-*formulas* as negations of Σ-formulas.

Definition 4. *1. A set $B \subseteq \mathrm{HF}(M)$ is Σ-definable if there exists a Σ-formula $\Phi(x)$ such that $x \in B \leftrightarrow \mathbf{HF}(\mathbf{M}) \models \Phi(x)$.*

2. A function $f : \mathrm{HF}(M) \to \mathrm{HF}(M)$ is Σ-definable if there exists a Σ-formula $\Phi(x, y)$ such that $f(x) = y \leftrightarrow \mathbf{HF}(\mathbf{M}) \models \Phi(x, y)$.

In a similar way, we define the notions of Π-*definable functions* and *sets*. The class of Δ-*definable functions* (*sets*) is the intersection of the class Σ-definable functions (sets) and the class of Π-definable functions (sets).

Note that the sets M and M^n are Δ_0–definable. This fact makes $\mathbf{HF}(\mathbf{M})$ a suitable domain for studying functions from M^k to M. To introduce the definition of majorant-computability we use a class of Σ-definable real-valued functions as a basic class. So, we recall properties of Σ-definable real-valued functions and subsets of $\mathbb{R}^n$. The following propositions give important properties of Σ-, Δ-, Π- definable sets and functions.

Proposition 5. *Let $\tilde{\mathbb{R}}$ be a model of $\mathrm{Th}(\mathbb{R})$ and σ_0 be the language of $\mathrm{Th}(\mathbb{R})$.*

1. *A set $B \subseteq \tilde{\mathbb{R}}^n$ is Σ-definable if and only if there exists an effective sequence of quantifier-free formulas in the language σ_0, $\{\Phi_s(x)\}_{s\in\omega}$, such that $x \in B \leftrightarrow \tilde{\mathbb{R}} \models \bigvee_{s\in\omega} \Phi_s(x)$.*
2. *A set $B \subseteq \tilde{\mathbb{R}}^n$ is Π-definable if and only if there exists an effective sequence of quantifier-free formulas in the language σ_0, $\{\Phi_s(x)\}_{s\in\omega}$, such that $x \in B \leftrightarrow \tilde{\mathbb{R}} \models \bigwedge_{s\in\omega} \Phi_s(x)$.*

Proof. The claim is immediate from the properties of the set of hereditarily finite sets and the fact that $\mathrm{Th}(\mathbb{R})$ admits elimination of quantifiers. □

Proposition 6. *Let $\mathbb{R}$ be the standard reals.*
The following statements hold in $\mathbf{HF}(\mathbb{R})$.

1. *The set $\mathbb{N}$ is Δ-definable.*
2. *The class of Σ-definable real-valued total functions is closed under composition.*
3. *There is an universal real-valued Σ-definable function in the class of Σ-definable real-valued functions.*

Proof. See [7, 12, 11] and Proposition 5.

Proposition 7. *A real-valued function $f : \mathbb{R} \to \mathbb{R}$ is Σ-definable if and only if the following conditions hold.*

1. *There exist effective sequences $\{a_i\}_{i\in\omega}$, $\{b_i\}_{i\in\omega}$, where $a_i, b_i \in \mathbb{R}_0 \cup \{+\infty, -\infty\}$ for $i \in \omega$, such that*
$$\mathrm{dom}(f) = \bigcup_{i\in\omega} \langle a_i, b_i\rangle \text{ and } \langle a_j, b_j\rangle \bigcap \langle a_i, b_i\rangle = \emptyset \text{ for } i \neq j \ .$$
2. *There exists effective sequences of algebraic functions $\{g_i\}_{i\in\omega}$ such that f coincides with g_i on the interval $\langle a_i, b_i\rangle$ for all $i \in \omega$.*

Proof. See [12] and Proposition 5.

Theorem 8 ((Uniformization)). *Let $\tilde{\mathbb{R}}$ be a model of $\mathrm{Th}(\mathbb{R})$.*
For any subset of $\tilde{\mathbb{R}}^n$ which is Σ–definable by a Σ–formula Φ there exists an Σ–definable function $f : \tilde{\mathbb{R}}^{n-1} \to \tilde{\mathbb{R}}$ such that

1. $\mathrm{dom}(f) = \{x | \mathrm{HF}(\tilde{\mathbb{R}}) \models \exists y \Phi(x, y)\}$.
2. *for $x \in \mathrm{dom}(f)$ we have $\mathrm{HF}(\tilde{\mathbb{R}}) \models \Phi(x, f(x))$.*

Proof. See [12] and Proposition 7.

To introduce the notion of majorant-computability for partial real-valued functions we need a notion of prime enlargement of $\mathbb{R}$.

Definition 9. *A model $\bar{\mathbb{R}}$ of $\mathrm{Th}(\mathbb{R})$ is called a prime enlargement of $\mathbb{R}$ if there exists $t \in \bar{\mathbb{R}}$ such that $t > n$ for every natural n (we write $t > \mathbb{N}$) and $\bar{\mathbb{R}}$ is the real closure of the ordered field $\mathbb{R}(t)$.*
In addition, we write $t > \mathbb{R}$ if $t > r$ for all $r \in \mathbb{R}$.

Definition 10. *Let $\bar{\mathbb{R}}$ be a proper elementary enlargement of $\mathbb{R}$ and let $t > \mathbb{N}$. We define the following function $\mathrm{sp} : \bar{\mathbb{R}} \to \mathbb{R} \cup \{-\infty, +\infty\}$*

$$\mathrm{sp}(x) = \begin{cases} x^* \in \mathbb{R} \text{ if for } | x - x^* | = \varepsilon \text{ the condition } 0 < \varepsilon < \mathbb{R}^+ \text{ holds} , \\ +\infty \quad \text{if } x > \mathbb{R}, \\ -\infty \quad \text{if } x < \mathbb{R} \ . \end{cases}$$

An element $x \in \bar{\mathbb{R}}$ is finite if and only if $-\infty < \mathrm{sp}(x) < +\infty$. The set of all finite elements of $\bar{\mathbb{R}}$ is denoted by $\mathrm{Fin}(\bar{\mathbb{R}})$.

Since $\bar{\mathbb{R}} \models \mathrm{Th}(\mathbb{R})$, $\bar{\mathbb{R}}$ is a proper elementary enlargment of $\mathbb{R}$ with the same true first-order formulas. Recall properties of prime enlargements and proper elementary enlargements of $\mathbb{R}$.

Proposition 11. 1. *Every two prime enlargements of* $\mathbb{R}$ *are isomorphic.*
2. *There exists a nonstandard element* $t > \mathbb{N}$ *in a proper elementary enlargement of* $\mathbb{R}$.

Proof. See [10, 17].

Lemma 12. *Let* $p(x, \mathbf{y})$ *be a countable set of first-order formulas with one variable* x *and real parameters* $\mathbf{y}$. *Given proper elementary enlargements* $\mathbb{R}_1$, $\mathbb{R}_2$ *of* $\mathbb{R}$, $p(x, \mathbf{y})$ *holds in* $\mathbb{R}_1$ *if and only if* $p(x, \mathbf{y})$ *holds in* $\mathbb{R}_2$.

Proof. The claim is immediate from 0-minimality of $\mathbb{R}_1$ and $\mathbb{R}_2$. □

Lemma 13. *Let* $\bar{\mathbb{R}}$ *be a prime enlargement of* $\mathbb{R}$.
For every Σ*-formula* $\varphi(z, \mathbf{x})$ *there exists a* Σ*-formula* $\varphi^*(\mathbf{x})$ *such that, for all* $\mathbf{x} \in \mathbb{R}^n$ *and* $t > \mathbb{N}$, $\mathbf{HF}(\bar{\mathbb{R}}) \models \varphi(t, \mathbf{x}) \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models \varphi^*(\mathbf{x})$.

Proof. The claim is immediate from 0-minimality of $\bar{\mathbb{R}}$. □

Lemma 14. *Let* $\bar{\mathbb{R}}$ *be a model of* $\mathrm{Th}(\mathbb{R})$ *and let* B *be a countable subset of* $\bar{\mathbb{R}}$. *If* B *is* Σ*-definable in* $\mathbf{HF}(\bar{\mathbb{R}})$ *then there exists a* Σ*-definable function* $h : \mathbb{N} \to \bar{\mathbb{R}}$ *numbering* B.

Proof. The claim is immediate from Proposition 1 and Proposition 5. □

3 Majorant-Computable Functions

Let us recall the notion of majorant-computability for real-valued functions presented in [13]. We would use as a basic class the class of Σ-definable total functions of type $f : \bar{\mathbb{R}}^n \to \bar{\mathbb{R}}$, where $\bar{\mathbb{R}}$ is a proper elementary enlargement of $\mathbb{R}$. A real-valued function is said to be majorant-computable if we can construct a special kind of nonterminating process computing approximations closer and closer to the result.

Definition 15. *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is called majorant-computable if there exist effective sequences of* Σ*-formulas* $\{\Phi_s(a, \mathbf{x}, y)\}_{s\in\omega}$ *and* $\{G_s(a, \mathbf{x}, y)\}_{s\in\omega}$, *with a parameter* a, *a proper elementary enlargement* $\bar{\mathbb{R}}$ *of* $\mathbb{R}$ *such that the following conditions hold.*

1. *There exists* $t \in \bar{\mathbb{R}}$ *such that* $t > \mathbb{N}$;
2. *For all* $s \in \omega$, *the formulas* $\Phi_s(a, \mathbf{x}, y)$ *and* $G_s(a, \mathbf{x}, y)$ *define total functions* f_s *and* g_s *as follows:*
   a. $f_s : \bar{\mathbb{R}}^n \to \bar{\mathbb{R}}$ *and* $g_s : \bar{\mathbb{R}}^n \to \bar{\mathbb{R}}$,
   b. $f_s(\mathbf{x}) = y \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models \Phi_s(t, \mathbf{x}, y)$,
      $g_s(\mathbf{x}) = y \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models G_s(t, \mathbf{x}, y)$;

3. *For all* $\mathbf{x} \in \bar{\mathbb{R}}^n$, *the sequence* $\{f_s(\mathbf{x})\}_{s\in\omega}$ *increases monotonically; the sequence* $\{g_s(\mathbf{x})\}_{s\in\omega}$ *decreases monotonically ;*
4. *For all* $s \in \omega$, $\mathbf{x} \in \mathrm{dom}(f)$, $f_s(\mathbf{x}) \leq f(\mathbf{x}) \leq g_s(\mathbf{x})$ *and, for all* $\mathbf{x} \in \bar{\mathbb{R}}^n$, $f_s(\mathbf{x}) \leq g_s(\mathbf{x})$;
5. $f(\mathbf{x}) = y \leftrightarrow \lim_{s\to\infty} \mathrm{sp}(f_s(\mathbf{x})) = y$ *and* $\lim_{s\to\infty} \mathrm{sp}(g_s(\mathbf{x})) = y$.

The sequence $\{f_s\}_{s\in\omega}$ in Definition 15 is called a *sequence of lower Σ-approximations* for f. The sequence $\{g_s\}_{s\in\omega}$ is called a *sequence of upper Σ-approximations* for f.

As we can see the process which carries out the computation is represented by two effective procedures. These procedures produce Σ-approximations closer and closer to the result. If the computational process converges to infinity the procedures produce nonstandard elements like $-t$ or t, where $t > \mathbb{R}$.

So, using a proper elementary enlargement of $\mathbb{R}$ in the Definition 9 we admit only precise computations at finite steps. Since a prime enlargement is effectively constructed over $\mathbb{R}$, this approach admit us to consider computability of partial functions in a natural way.

The following theorem connects the graph of a majorant-computable function with validity of a finite formula in the set of hereditarily finite sets, $\mathbf{HF}(\bar{\mathbb{R}})$ (where $\bar{\mathbb{R}}$ is a proper elementary enlargement of the standard real numbers).

Definition 16. *Let* $\bar{\mathbb{R}}$ *be a proper elementary enlargement of* $\mathbb{R}$. *A formula* Φ *is said to determine a function* $f : \bar{\mathbb{R}}^n \to \bar{\mathbb{R}}$ *in the model* $\mathbf{HF}(\bar{\mathbb{R}})$ *if the following statement holds* $f(\mathbf{x}) = y \leftrightarrow$
$\big(\mathbf{HF}(\bar{\mathbb{R}}) \models \Phi(\mathbf{x}, y)$ *for* $\mathbf{x} \in \mathbb{R}$ *and* $\{\mathrm{sp}(z) \mid \Phi(\mathbf{x}, z)\} = \{y\}$ *for* $y \in \mathbb{R}\big)$.

Theorem 17. *For all functions* $f : \mathbb{R}^n \to \mathbb{R}$, *the following assertions are equivalent*:

1. *The function* f *is majorant-computable.*
2. *There exist a prime enlargement* $\bar{\mathbb{R}} \succ \mathbb{R}$ *and a* Π*-formula that determines a function* F *in* $\mathbf{HF}(\bar{\mathbb{R}})$ *with the property* $F \mid_{\mathbb{R}} = f$.
3. *There exists a* Π*-formula that in any proper elementary enlargement* $\hat{\mathbb{R}} \succ \mathbb{R}$ *determines a function* F *with the property* $F \mid_{\mathbb{R}} = f$.

Proof. $1 \to 2$) Let f be majorant-computable. By Proposition 11, without loss of generality, we may assume that there exist a prime enlargement $\bar{\mathbb{R}}$ of $\mathbb{R}$, a sequence $\{f_s\}_{s\in\omega}$ of lower Σ-approximations for f, and a sequence $\{g_s\}_{s\in\omega}$ of upper Σ-approximations for f. Let $t > \mathbb{N}$. Denote $B = \mathbb{Q} \cup \{\pm t^n \mid n \in \mathbb{Q}\}$.

Using $\{f_s\}_{s\in\omega}$, $\{g_s\}_{s\in\omega}$, and cofinality of B in $\bar{\mathbb{R}}$, we construct a sequence $\{f_s^*\}_{s\in\omega}$ of new lower Σ-approximations for f and a sequence $\{g_s^*\}_{s\in\omega}$ of new upper Σ-approximations for f such that, for all $s \in \omega$, the ranges of f_s and g_s are subsets of B. Denote:

$$D_1(\mathbf{x}) = \{z \in B \mid \exists s\,(f_s(\mathbf{x}) \geq z)\}, D_2(\mathbf{x}) = \{z \in B \mid \exists s\,(g_s(\mathbf{x}) \leq z)\}\ .$$

The sets $D_1(\mathbf{x})$ and $D_2(\mathbf{x})$ are Σ-definable and countable; so, by Lemma 14, there exist a function $h_1(n, \mathbf{x})$ numbering $D_1(\mathbf{x})$ and a function $h_2(n, \mathbf{x})$ numbering $D_2(\mathbf{x})$. Next, put

$$f_s^*(\mathbf{x}) = max_{n\leq s} h_1(n, \mathbf{x}),\ g_s^*(\mathbf{x}) = min_{n\leq s} h_2(n, \mathbf{x})\ .$$

By construction, the sequences $\{f_s^*\}_{s\in\omega}$ and $\{g_s^*\}_{s\in\omega}$ are what we seek.
Let

$$f_s^*(\mathbf{x}) = y \rightleftharpoons \mathbf{HF}(\bar{\mathbb{R}}) \models \Phi_s^*(t,\mathbf{x},y),\ g_s^*(\mathbf{x}) = y \rightleftharpoons \mathbf{HF}(\bar{\mathbb{R}}) \models G_s^*(t,\mathbf{x},y)\ ,$$

where Φ_s^*, G_s^* are Σ-formulas. From Lemma 12 it follows that, for all $s \in \omega$, there exist Σ-formulas Φ_s^{**} and G_s^{**} such that, for every $\mathbf{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$:

$$\mathbf{HF}(\bar{\mathbb{R}}) \models G_s^*(t,\mathbf{x},y) \leftrightarrow G_s^{**}(\mathbf{x},y),\ \mathbf{HF}(\bar{\mathbb{R}}) \models \Phi_s^*(t,\mathbf{x},y) \leftrightarrow \Phi_s^{**}(\mathbf{x},y)\ .$$

Let

$$d_s = \{\mathbf{x} \mid \mathbf{x} \in \mathrm{Fin}^n(\bar{\mathbb{R}}),\ f_s(\mathbf{x}),\ g_s(\mathbf{x}) \in \mathrm{Fin}(\bar{\mathbb{R}})\} \text{ for } s \in \omega.$$

By Σ-definability of $\mathrm{Fin}(\bar{\mathbb{R}})$, the set d_s is Σ-definable. It is easy to see that $d_s \to_{s\to\omega} \mathrm{dom}(f)$. Put

$$\Phi(\mathbf{x},y) \rightleftharpoons \forall s \forall y_1 \forall y_2\, [(x \in d_s) \to ((\Phi_s^{**}(\mathbf{x},y_1) \wedge G_s^{**}(\mathbf{x},y_2)) \to (y_1 \le y \le y_2))]\ .$$

The Π-formula Φ determines a function $F : \bar{\mathbb{R}} \to \bar{\mathbb{R}}$.
Let us prove that $F \mid \mathbb{R} = f$.

Let $f(\mathbf{x}) = y$. Suppose the contrary: $F(\mathbf{x}) \neq y$. By Definition 16, there exist $s \in \omega,\ y_1,\ y_2 \in \bar{\mathbb{R}}$ such that

$$\mathbf{HF}(\bar{\mathbb{R}}) \models (\mathbf{x} \in d_s) \wedge \Phi_s^{**}(\mathbf{x},y_1) \wedge G_s^{**}(\mathbf{x},y_2) \wedge (y \notin [y_1,y_2])\ .$$

By the constructions of f_s, g_s and the definition of d_s, there exist $z_1,\ z_2 \in \mathbb{R}$ such that

$$f_s^*(\mathbf{x}) = z_1 \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models \Phi_s^*(t,\mathbf{x},z_1),\ g_s^*(\mathbf{x}) = z_2 \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models G_s^*(t,\mathbf{x},z_2)\ .$$

Note that $\mathbf{x} \in \mathbb{R}^n$ and $z_1,\ z_2 \in \mathbb{R}$. So, we have

$$f_s^*(\mathbf{x}) = z_1 \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models \Phi_s^{**}(\mathbf{x},z_1),\ g_s^*(\mathbf{x}) = z_2 \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models G_s^{**}(\mathbf{x},z_2)$$

and

$$\mathbf{HF}(\bar{\mathbb{R}}) \models (\mathbf{x} \in d_s) \wedge \Phi_s^{**}(\mathbf{x},z_1) \wedge G_s^{**}(\mathbf{x},z_2) \wedge (y \in [y_1,y_2])\ .$$

Note that if a formula $G(\mathbf{x},y)$ defines the function f in $\mathbf{HF}(\mathbb{R})$ then $\mathbf{HF}(\bar{\mathbb{R}}) \not\models G(\mathbf{x},y)$ for $x \in \mathrm{dom}(f)$, $y \notin \mathbb{R}$. It is easy to see that the formulas Φ_s^{**}, G_s^{**} define functions in $\mathbf{HF}(\mathbb{R})$, so $y_1 = z_1$ and $y_2 = z_2$; a contradiction.

If $F(\mathbf{x}) = y$ for $\mathbf{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$,

$$\{\mathrm{sp}(z) \mid \bigwedge_{i\in\omega} (f_i^*(\mathbf{x}) \le z \le g_i^*(\mathbf{x}))\} = \{y\}\ .$$

It follows that $\lim_{s\to\infty} \mathrm{sp}(f_s^*(\mathbf{x})) = y$ and $\lim_{s\to\infty} \mathrm{sp}(g_s^*(\mathbf{x})) = y$. So, $f(\mathbf{x}) = y$ and the formula Φ is what we seek.
$2 \to 3$) This is obvious.

$3 \to 1)$ Let $\bar{\mathbb{R}}$ be a prime enlargement of $\mathbb{R}$ and let a Π-formula $\Phi(\mathbf{x}, y)$ determine the function F such that $F \mid_{\mathbb{R}} = f$. From Proposition 5 it follows that there exists an effective sequence of quantifier-free formulas $\{\varphi_i(\mathbf{x}, y)\}_{i\in\omega}$ such that

$$\mathbf{HF}(\bar{\mathbb{R}}) \models \Phi(\mathbf{x}, y) \leftrightarrow \mathbf{HF}(\bar{\mathbb{R}}) \models \bigwedge_{i\in\omega} \varphi_i(\mathbf{x}, y) \ .$$

Denote

$$\Phi_s(\mathbf{x}, y) \rightleftharpoons \bigwedge_{i\le s} \varphi_i(\mathbf{x}, y) \ .$$

Put

$$f_s(\mathbf{x}) = \begin{cases} \inf\{y \mid \Phi_s(\mathbf{x}, y)\} \ \textit{if} \ \inf\{y \mid \Phi_s(\mathbf{x}, y)\} \ \textit{exists}, \\ -t \text{ otherwise}, \end{cases}$$

$$g_s(\mathbf{x}) = \begin{cases} \sup\{y \mid \Phi_s(\mathbf{x}, y)\} \ \textit{if} \ \sup\{y \mid \Phi_s(\mathbf{x}, y)\} \ \textit{exists}, \\ t \text{ otherwise}, \end{cases}$$

where $t > \mathbb{N}$.

Then $\{f_s\}_{s\in\omega}$ and $\{g_s\}_{s\in\omega}$ are the sought sequences. □

We remark that if f is a total function then the sequences $\{f_s\}_{s\in\omega}$ and $\{g_s\}_{s\in\omega}$ can be constructed to converge uniformly to f on $\mathbb{R}$.

As a corollary from the previous theorem we note that whenever we want to prove some statements to define a majorant-computable function, without loss of generality, we can assume that the proper elementary enlargement of $\mathbb{R}$ in Definition 15 is a prime enlargement of $\mathbb{R}$.

Proposition 18. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be majorant-computable.*
Then $\mathrm{dom}(f)$ is Π_2-definable by a formula of type $\big((\forall z \in \mathbb{R})\, \Phi(z, \mathbf{x})\big)$, where Φ is a Σ-formula.

Proof. Let $\{f_s\}_{s\in\omega}$ be a sequence of lower Σ-approximations for f, and let $\{g_s\}_{s\in\omega}$ be a sequence of upper Σ-approximations for f.

We obtain:

$$x \in \mathrm{dom}(f) \leftrightarrow \mathrm{HF}(\mathbb{R}) \models (\forall \varepsilon \in \mathbb{R})(\exists s \in \mathbb{N})\big|f_s(\mathbf{x}) - g_s(\mathbf{x})\big| \le \varepsilon \ .$$

By Σ-definability of the natural numbers, the set $\mathrm{dom}(f)$ is Π_2-definable. □

Corollary 19. *The domain of a majorant-computable function is the effective intersection of intervals with algebraic endpoints.*

Proof. The claim is immediate from Proposition 5 and Proposition 18. □

Definition 20. *Let a function $f : \mathbb{R}^n \to \mathbb{R}$ be total.*
The epigraph of f is defined to be the set $U = \{(\mathbf{x}, y) \mid f(\mathbf{x}) < y\}$. The ordinate set of f is defined to be the set $D = \{(\mathbf{x}, y) \mid f(\mathbf{x}) > y\}$.

Corollary 21. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a total function.*
The function f is majorant-computable if and only if the epigraph of f and the ordinate set of f are Σ-definable sets.

Proof. It is clear that

$$(\mathbf{x}, z) \in U \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \exists s \exists y \, [\, (s \in \mathbb{N}) \wedge G^{**}(\mathbf{x}, y) \wedge (z > y)] \ ,$$

$$(\mathbf{x}, z) \in D \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \exists s \exists y \, [\, (s \in \mathbb{N}) \wedge F^{**}(\mathbf{x}, y) \wedge (z < y)] \ ,$$

where the formulas G^{**} and F^{**} are those in the proof of Theorem 17. □

Corollary 22. *Let $f : \mathbb{R} \to \mathbb{R}$.*

1. *If f is a Σ-definable real-valued function, then f is majorant-computable.*
2. *If $\mathrm{dom} f \subset \mathbb{N}$ and $\mathrm{im} f \subset \mathbb{N}$, then f is majorant-computable if f is a partial recursive function.*
3. *If f is a majorant-computable total function, then f is a piecewise continuous function.*

Proof. The claims follow from the definition of majorant-computability. □

4 Majorant-Computability and PR-, ES-Computabilities over the Reals

Let us denote the computability proposed in [16] as PR-computability.
Recall the definition of PR-computability for real-valued functions.

Definition 23. *A total function $f : \mathbb{R} \to \mathbb{R}$ is PR-computable if and only if*

1. *it maps computable sequences to computable sequences,*
2. *it is effectively uniformly continuous on intervals $[-n, n]$.*

Let us denote the computability proposed in [6] as ES-computability.
Recall the definition of ES-computability for real-valued functions.

We recall the definition of the *interval domain* I :
$I = \{[a, b] \subseteq \mathbb{R} \mid a, b \in \mathbb{R},\ a \leq b\} \cup \{\perp\}$.
The relation $\ll$ is defined as follows: $\perp \ll J$ for all $J \in I$ and $[a, b] \ll [c, d]$ if and only if $a < c$ and $b > d$.

Definition 24. *Let $I_0 = \{b_1, \ldots, b_n, \ldots\} \cup \{\perp\}$ be the effective enumerated set of all intervals with rationals endpoints.*
A continuous function $f : I \to I$ is computable, if the relation $b_m \ll f(b_n)$ is r.e. in n, m, where $b_m,\ b_n \in I_0$.

Definition 25. *A function $f : \mathbb{R} \to \mathbb{R}$ is ES-computable if and only if there is an enlargement $g : I \to I$ (i.e., $g(\{x\}) = \{f(x)\}$ for all $x \in \mathrm{dom}(f)$) which is computable in the sense of the Definition 24.*

Let us define $g_{x,y} = \Pi_{[x,y]}$, $\hat{g}_{x,y} = \Pi_{[x,y)}$,
where $\Pi_M(x) = \begin{cases} 1 & \text{if } x \in M, \\ \text{undefined} & \text{otherwise .} \end{cases}$

Definition 26. *A continuous function $h : [a, b] \to \mathbb{R}$ is called a piecewise linear function if there exist $x_0, \ldots, x_n \in \mathbb{R}$ such that $a = x_0 < x_1 < \ldots x_n = b$, and $h\,|_{[x_i,x_{i+1}]}$ is a linear function for all $0 \leq i \leq m$. We define the code $[h]$ of h as follows: $[h] = \{< x_i, h(x_i) > |\ 0 \leq i \leq m\}$.*

Let us consider functions from $< a, b >$ to $\mathbb{R}$.

Proposition 27. *Let $f :< a, b > \to \mathbb{R}$ be a continuous majorant-computable function. The function f is PR-computable if and only if the relation*

$$R_f = \{< x, y, z > \in [a, b]^2 \times \mathbb{R} \mid x < y \wedge f|_{[x,y]} > zg_{x,y}\}$$

is Σ-definable in $\mathrm{HF}(\mathbb{R})$.

Proof. $\rightarrow$) By the definition of PR-computability, f is an effectively uniformly continuous function. It follows that the set

$$Q_f = \left\{< x, y, z, \epsilon > \in [a, b] \times \mathbb{R}^2 \mid ||f - zg_{x,y}||_{[x,y]} < \epsilon\right\}$$

is Σ-definable.

For $x, y \in \mathbb{R}$ such that $x < y$, the following equivalence holds.

$$||f - zg_{x,y}||_{[x,y]} \leq \epsilon \leftrightarrow \left(\exists s \geq \frac{3}{\epsilon}\right)\left(\exists < b_1, \ldots, b_n > \in \mathbb{Q}^{<\omega}\right)$$
$$\Big(\bigwedge_{i=0}^{n-1} |b_i - b_{i+1}| < \delta = w_f(\frac{\epsilon}{3}) \wedge b_0 = x \wedge b_n = y \wedge \bigwedge_{i=0}^{n-1} |f_s(b_i) - z| < \frac{\epsilon}{3}\Big)\ ,$$

where w is an effective modulus of continuity for f. It follows that $< x, y, z > \in R_f$ if and only if there exist $\epsilon > 0$ and a step-function $h : [x, y] \to \mathbb{Q}$ such that $h > zg_{a,b} + \epsilon$ and $||f - h||_{[x,y]} < \epsilon$.
For arbitrary $x_0 < \ldots < x_i < \ldots < x_m = y$ and $h = \bigcup_{i=0}^{m-1} \left(z_i \hat{g}_{x_i,x_{i+1}}\right)$,
$||f - h||_{[x,y]} = \max_{0 \leq i \leq m-1} ||f - z_i \hat{g}_{x_i,x_{i+1}}||$.
So the set R_f is Σ-definable in $\mathbf{HF}(\mathbb{R})$.

$\leftarrow$) Let R_f be Σ-definable in $\mathbf{HF}(\mathbb{R})$. For every $\epsilon = \frac{1}{s} > 0$, where $s \in \omega$, we will effectively construct piecewise linear functions $f_s : [a, b] \to \mathbb{R}$, $g_s : [a, b] \to \mathbb{R}$ with the following conditions:
1. $f_s(x) \leq f(x) \leq g_s(x)$ for all $x \in [a, b]$; 2. $||f_s - g_s|| \leq \epsilon$.

In fact, by Σ-definability of R_f, the set

$$T_f = \left\{< x, y, z, t > \in [a, b]^2 \times \mathbb{R}^2 \mid x < y \wedge \forall u \in [x, y]\left(f(u) > z + \frac{u(t - z)}{y - x}\right)\right\}$$

is Σ-definable. It follows that the set

$$L_f = \{< s, [f], [g] >| \ s \in \omega, \ f, \ g \text{ are piecewise linear with the condition 1-2}\}$$

is Σ-definable. Using uniformization (see Theorem 8) we can construct the required functions $f_s, \ g_s$.
Obviously, by the code $[h] \subseteq \mathbb{Q}^2$ of a piecewise linear function, we can effectively construct a function $w_h : [0,1] \to \mathbb{R}$ with the following properties:

1. w_h is a linear function;
2. $(\forall \epsilon \in (0,1])\,(\forall x, \ y \in [0,1])\,(|x-y| < w_h(\epsilon) \to |h(x) - h(y)| < \epsilon)$

As we can see, w_h is a modulus of continuity for h.

Let us define the function w_f as follows: $w_f = w_{f_{s(\epsilon)}}$, where $s(\epsilon) = [\frac{3}{\epsilon}] + 1$. It is easy to see that w_f is Σ-definable, and it is a modulus of continuity for f. By the definition of PR-computability, f is PR-computable. □

It is easy to see that the continuous majorant-computable total function f is PR-computable if and only if the intersection $R_f \cap \mathbb{Q}$ is Σ-definable in $\mathrm{HF}(\mathbb{R})$.

Proposition 28. *If B is an open Σ-definable set then B is the effective union of open intervals.*

Proof. Let B be an open Σ-definable set. By Proposition 3,

$$B = \bigcup_{i \in \omega} < \alpha_i, \beta_i > .$$

We represent B as follows:

$$B = \bigcup_{i \in I_1} [\alpha_i, \beta_i) \cup \bigcup_{i \in I_2} (\alpha_i, \beta_i] \cup \bigcup_{i \in I_3} (\alpha_i, \beta_i) \cup \bigcup_{i \in I_4} [\alpha_i, \beta_i].$$

Put $I_4' = \{i \in I_4 \mid \alpha_i \neq \beta_i, \ \alpha_i \neq \beta_j, \ j \in I_4\}$. Because B is open, we can effectively choose $\gamma_i, \ \nu_i, \ \delta_i$, and θ_i as follows

$$\gamma_i < \alpha_i \text{ and } \gamma_i \in (\alpha_j, \beta_j), \text{ where } i \in I_1, \ j \in I_2 \cup I_3 \cup I_4',$$

$$\nu_i > \beta_i \text{ and } \nu_i \in (\alpha_j, \beta_j), \text{ where } i \in I_2, \ j \in I_1 \cup I_3 \cup I_4',$$

$$\delta_i < \alpha_i \text{ and } \delta_i \in (\alpha_j, \beta_j), \text{ where } i \in I_4, \ j \in I_1 \cup I_2 \cup I_3,$$

$$\theta_i > \beta_i \text{ and } \beta_i \in (\alpha_j, \beta_j), \text{ where } i \in I_4, \ j \in I_1 \cup I_3.$$

So,

$$B = \bigcup_{i \in I_1} (\nu_i, \beta_i) \cup \bigcup_{i \in I_2} (\alpha_i, \beta_i) \cup \bigcup_{i \in I_3} (\alpha_i, \beta_i) \cup \bigcup_{i \in I_4} (\delta_i, \theta_i).$$

This means that B is the effective union of open intervals. □

Proposition 29. *Let $f : [0,1] \to \mathbb{R}$ be a majorant-computable continuous function.*
The set $R_f = \{< a, b, c > |\ 0 \leq a < b \leq 1 \wedge f|_{[a,b]} > c\}$ is Σ-definable in $\mathbf{HF}(\mathbb{R})$.

Proof. Without loss of generality, let us consider the case $f : [0,1] \to [0,1]$.

Let us define $x \in A_i^s \leftrightarrow |f(x) - \frac{i}{s}| < \frac{1}{s}$, for $i = 0, \ldots, s$.
We have the following properties:
1. $\bigcup A_i^s = [0,1]$, 2. A_i^s is an open Σ-definable set.
By Proposition 28, $A_i^s = \bigcup_{j \in \omega} (\alpha_{i,s}^j, \beta_{i,s}^j)$ for some rational numbers $\alpha_{i,s}^j \leq \beta_{i,s}^j$.

Let $[a,b] \subseteq [0,1]$. Let us consider the case $c = 0$. By continuity of f, $f|_{[a,b]} > 0$ if and only if there exists s such that $[a,b] \subseteq \bigcup_{i=1}^s A_i^s$.

Really, if there exists s such that $[a,b] \subseteq \bigcup_{i=1}^s A_i^s$ and $x \in [a,b]$ then $x \in A_i^s$ for some i. By the definition of A_i^s, $|f(x) - \frac{i}{s}| < \frac{1}{s}$, so $f(x) > 0$.
If $f|_{[a,b]} > 0$, then there exists s_1 such that $f|_{[a,b]} > \frac{1}{s_1}$. Put $s = s_1$.
For some $i > 0$, we have $|f(x) - \frac{i}{s}| < \frac{1}{s}$, i.e., $x \in A_i^s$. So, $[a,b] \subseteq \bigcup_{i=1}^s A_i^s$.

By compactness of $[a,b]$, $f_{[a,b]} > 0$ if and only if there exists s such that $[a,b] \subseteq \bigcup_{i=1}^s \hat{A}_i^s$, where $\hat{A}_i^s = \bigcup_{j \in J_{i,s}} (\alpha_{i,s}^j, \beta_{i,s}^j)$, and $J_{i,s}$ is some finite subset of ω, $0 < i < s$, i.e., $f_{[a,b]} > 0$ if and only if the Σ-condition holds.

In the general case, for $[a,b]$ and c, using similar considerations we check the following condition:

$$[a,b] \subseteq \bigcup_{i : |\frac{i}{s} - c| \geq \frac{1}{s}} A_i^s.$$

□

Theorem 30. *Let $f : [a,b] \to \mathbb{R}$ be a continuous total function.*
The function f is majorant-computable if and only if f is PR-computable.

Proof. $\rightarrow$) The claim is immediate from Proposition 27 and Proposition 29.
$\leftarrow$) The claim is immediate from Corollary 21. □

Corollary 31. *Let $f : [a,b] \to \mathbb{R}$ be a total continuous function.*
The function f is majorant-computable if and only if f is ES-computable.

Proof. The claims are immediate from Corollary 30 in [6]. □

Theorem 32. *The class of ES-computable functions is contained in the class of majorant-computable functions.*

Proof. Let $f : \mathbb{R} \to \mathbb{R}$ be ES-computable.
For $n \in \omega$, we define $A_n = \{x \in \mathbb{R} \mid \mu(f([x])) < \frac{1}{n}\}$, where μ is the natural measure defined on I. It is easy to see that A_n is Σ-definable open set, and $\mathrm{dom}(f) = \bigcap_{n \in \omega} A_n$.
By Proposition 28, $A_1 = \bigcup_{i \in \omega} (\alpha_i, \beta_i)$, where $\alpha_i,\ \beta_i \in \mathbb{Q}$ and $\alpha_i \leq \beta_i$.

By Lemma 14, there exist a Σ-definable function $h : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ numbering the Σ-definable set $\{z \in \mathbb{Q} \mid z < f([x])\}$, and a Σ-definable function $H : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ numbering the Σ-definable set $\{z \in \mathbb{Q} \mid z > f([x])\}$.

Put

$$f_s(\mathbf{x}) = \begin{cases} \max_{k\le s} h(x,k) \ if \ x \in \bigcup_{n\le s}(\alpha_n, \beta_n), \\ -t \text{ otherwise,} \end{cases}$$

$$g_s(\mathbf{x}) = \begin{cases} \min_{k\le s} H(x,k) \ if \ x \in \bigcup_{n\le s}(\alpha_n, \beta_n), \\ t \text{ otherwise,} \end{cases}$$

where $t > \mathbb{N}$.

Then $\{f_s\}_{s\in\omega}$ and $\{g_s\}_{s\in\omega}$ are the sought sequences. So, f is majorant-computable.

There exist majorant-computable functions that are not ES-computable. For example, a total step-function with a computable set of discontinuities is the sought one. □

Corollary 33. *If f is ES-computable, then the domain of f is the effective intersection of open intervals.*

Proof. The claim is immediate from the proof of Theorem 32. □

Corollary 34. *For real-valued functions, we have the following inclusions:*

1. *The class of Σ-definable functions is contained in the class of majorant-computable functions.*
2. *The class of computable functions introduced by Moschovakis is contained in the class of majorant-computable functions.*
3. *The class of functions definable by finite dimensional machines of Blum et al. without real constants is contained in the class of majorant-computable functions.*
4. *The class of PR-computable functions is contained in the class of majorant-computable functions.*
5. *The class of ES-computable functions is contained in the class of majorant-computable functions.*
6. *The class of computable functions introduced by Stoltenberg-Hansen, Tucker is contained in the class of majorant-computable functions.*

5 Examples

Consider an interesting class of real-valued total functions possessing meromorphic continuation onto $\mathbb{C}$. This subclass contains, for example, solutions of known differential equations.

Let $\mathbb{C}$ be the set of complex numbers.

Definition 35. *A function $f : \mathbb{R} \to \mathbb{R}$ is said to admit meromorphic continuation onto $\mathbb{C}$ if there exists a meromorphic function $f^* : \mathbb{C} \to \mathbb{C}$ such that $f^* \mid_{\mathbb{R}} = f$.*

We recall a fact concerning meromorphic functions in complex numbers theory [3].

Proposition 36. *Let $f : \mathbb{C} \to \mathbb{C}$ be a total meromorphic function. There exist entire functions $a(z)$ and $b(z)$ such that*

1. $f(z) = \frac{a(z)}{b(z)}$,
2. *$a(z)$ and $b(z)$ have Taylor expansions with coefficients in $\mathbb{R}$.*

Proof. The claim follows from the Schwartz principle [3]. □

Theorem 37. *Let $f : \mathbb{R} \to \mathbb{R}$ be a total function and let f^* be its meromorphic continuation onto $\mathbb{C}$. Suppose that f^* can be written as $f^*(z) = \frac{a(z)}{b(z)}$, where functions $a(z)$ and $b(z)$ are as in Proposition 36. The function f is majorant-computable if the following conditions on f hold:*

1. *There exist Σ-definable real-valued functions A and B such that*

$$max_{|z|\leq w} \mid a(z) \mid \leq A(w),$$
$$max_{|z|\leq w} \mid b(z) \mid \leq B(w), \text{ where } w \in \mathbb{R} ;$$

2. *The coefficients of the Taylor expansions for $a(x), b(x)$ are majorant-computable as constant functions.*

Proof. Let f satisfy the conditions of the theorem. We shall prove that the epigraph of f and the ordinate set of f are Σ-definable. Denote the Taylor expansions for $a(z)$ and $b(z)$ at the point 0 as follows:

$$a(z) = \sum_{k\in\omega} a_k z^k, \quad b(z) = \sum_{k\in\omega} b_k z^k.$$

Show that $a \mid_{\mathbb{R}}$ and $b \mid_{\mathbb{R}}$ are majorant-computable. Let us consider an arbitrary disk $\mid z \mid \leq R$.

From the Cauchy inequality [3], for all $k \in \omega$, we have

$$|a_k| \leq \frac{M}{R^k}, \quad \text{where} \quad M = \max_{|z|\leq R} |a(z)| .$$

This remark admits us to construct two sequences of required approximations for $a \mid_{\mathbb{R}}$. It is easy to see that $a \mid_{\mathbb{R}}$ is majorant-computable. Similarly, $b \mid_{\mathbb{R}}$ is majorant-computable. It is easy to proof that if $a \mid_{\mathbb{R}}$ and $b \mid_{\mathbb{R}}$ are majorant computable, then $|a\|_{\mathbb{R}}$ and $|b\|_{\mathbb{R}}$ are majorant-computable. From now it is evident that the epigraph and the ordinate set of the function f are Σ-definable sets. Corollary 2 implies that f is a majorant–computable function. □

Corollary 38. *If an analytical function f has Taylor expansion with majorant-computable coefficients and the function $\mid f \mid$ is bounded by an Σ-definable total real function, then f is a majorant–computable function.*

Since the considered language contains equality, majorant–computable functions can be discontinuous. The natural question is raised describe the situation without equality. Our forthcoming results would be devoted to this problem.

References

1. J. Barwise, Admissible sets and structures, Berlin, Springer–Verlag, 1975.
2. J. Bochnak, M. Coster and M.-F. Roy, Géometrie algébrique réelle Springer–Verlag, 1987.
3. A. V. Bitsadze, The bases of analytic function theory of complex variables, Nauka, Moskow, 1972.
4. L. Blum and M. Shub and S. Smale, On a theory of computation and complexity over the reals:NP-completeness, recursive functions and universal machines, Bull. Amer. Math. Soc., (N.S.) , v. 21, no. 1, 1989, pages 1–46.
5. L. van den Dries, Remarks on Tarski's problem concerning $(\mathbb{R}, +, *, \exp)$, Proc. Logic Colloquium'82, 1984, pages 240–266.
6. A. Edalat, P. Sünderhauf, A domain-theoretic approach to computability on the real line, Theoretical Computer Science. To appear.
7. Yu. L. Ershov, Definability and computability, Plenum, New York, 1996.
8. H. Freedman and K. Ko, Computational complexity of real functions, Theoret. Comput. Sci. , v. 20, 1982, pages 323–352.
9. A. Grzegorczyk, On the definitions of computable real continuous functions, Fund. Math., N 44, 1957, pages 61–71.
10. N. Jacobson, Basic Algebra II, W.E. Freedman and Company, New York, 1995.
11. M. Korovina, Generalized computability of real functions, Siberian Advance of Mathematics, v. 2, N 4, 1992, pages 1–18.
12. M. Korovina, About an universal recursive function and abstract machines on the list superstructure over the reals, Vychislitel'nye Sistemy, Novosibirsk, v. 156, 1996, pages 3–20.
13. M. Korovina, O. Kudinov, A New Approach to Computability over the Reals, SibAM, v. 8, N 3, 1998, pages 59–73.
14. R. Montague, Recursion theory as a branch of model theory, Proc. of the third international congr. on Logic, Methodology and the Philos. of Sc., 1967, Amsterdam, 1968, pages 63–86.
15. Y. N. Moschovakis, Abstract first order computability, Trans. Amer. Math. Soc., v. 138, 1969, pages 427–464.
16. M. B. Pour-El, J. I. Richards, Computability in Analysis and Physics, Springer-Verlag, 1988.
17. A. Robinson, Nonstandard Analysis, North-Holland, Studies in Logic and foundation of Mathematics, 1966.
18. D. Scott, Outline of a mathematical theory of computation, In 4th Annual Princeton Conference on Information Sciences and Systems, 1970, pages 169–176.
19. V. Stoltenberg-Hansen and J. V. Tucker, Effective algebras, Handbook of Logic in computer Science, v. 4, Clarendon Press, 1995, pages 375–526.

Theorems of Péter and Parsons in Computer Programming

Ján Komara and Paul J. Voda

Institute of Informatics, Comenius University Bratislava,
Mlynská dolina, 842 15 Bratislava, Slovakia,
{komara,voda}@fmph.uniba.sk
www.fmph.uniba.sk/~voda

1 Introduction

This paper describes principles behind a declarative programming language CL (*Clausal Language*) which comes with its own proof system for proving properties of defined functions and predicates. We use our own implementation of CL in three courses in the first and second years of undergraduate study. By unifying the domain of LISP's S-expressions with the domain $\mathbb{N}$ of natural numbers we have combined the LISP-like simplicity of coding with the simplicity of semantics. We deal just with functions over $\mathbb{N}$ within the framework of formal Peano arithmetic. We believe that most of the time this is as much as is needed. CL is thus an extremely simple language which is completely based in mathematics.

We illustrate in Sect. 2 how simple schemas of recursion and/or induction lead to definitions of functions which are not necessarily the most efficient ones to compute. For efficient computation one needs stronger schemas of recursion and/or induction. This calls for a delicate balance between the strength of the language and that of its formal system. The formal system must be strong enough to prove the recurrences of strong recursive schemas as theorems. On the other hand, the language must be *specification complete* and admit for every *specification* formula $\forall x \exists y \phi(x, y)$ (ϕ is quantifier-free) proved in the formal system a function f *witnessing* the formula: $A(x, f(x))$. Moreover, this must be provable in the formal system.

In Sect. 3 we introduce a very general schema of *recursion with measure* which combines expressivity with efficient computation. The justification of the schema is easy, it is defined by transfinite recursion. The *characterization* problem, i.e. the question of what class of functions is admitted by the schema, is far from trivial. We rely on a theorem of Péter [Pét32] and show that the schema admits exactly the primitive recursive functions.

The proof system of CL is a fragment of Peano arithmetic and its soundness is obviously guaranteed by the standard model $\mathcal{N}$ in natural numbers. In order to achieve the specification completeness we select in Sect. 4 the fragment called $I\Sigma_1$ whose witnessing functions are primitive recursive. A difficult problem arises: we wish to extract functions defined by the general schema from proofs where induction with Π_2-formulas is naturally called for. Unfortunately,

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 204–223, 1999.

$I\Sigma_1$ admits induction **axioms** with at most Π_1-formulas. Fortunately, a theorem of Parsons [Par70] comes to our rescue: induction **rules** for Π_2-formulas are admissible in proofs of Π_2-specifications.

What is amazing to us is that the relatively minor theorems of Péter and Parsons turn out to be crucially important in the characterization problem of a really existing and usable declarative programming language. What is even more amazing is that the theorems are closely connected. Namely, the proof of the special case of theorem of Parsons in Thm. 4.15 calls for functions from Thm. 3.7 by Péter. We hope that apart from the two quite technical proofs (which are both new), the rest of the paper will be accessible to any computer scientist with interest in logic but without the specialized knowledge in recursion and proof theory.

We wish to stress that the connection between the theorems of Péter and Parsons or new proofs of both theorems are not the main results of this paper. Our main goal is the demonstration of how the classical theory of recursive functions and the formal Peano arithmetic can be employed to give a simple semantics to a programming language.

1.1 **Pairing function.** CL combines the domain of S-expressions of LISP with the domain of natural numbers $\mathbb{N}$ in order to achieve a productive symbiosis of easy coding in the first domain with the standard setting of theory of computation and formal arithmetic in the second domain. This is done with the help of a *pairing function* $\langle x, y\rangle$ such that 0 is the only *atom*, i.e. $0 \neq \langle x, y\rangle$ and every positive number is in the range of the pairing function. The function satisfies the *pairing property*: $\langle x_1, x_2\rangle = \langle y_1, y_2\rangle \rightarrow x_1 = y_1 \wedge x_2 = y_2$ and we have $x < \langle x, y\rangle$ and $y < \langle x, y\rangle$. From the last two conditions we get *pair induction*:

$$\phi(0) \wedge \forall x \forall y(\phi(x) \wedge \phi(y) \rightarrow \phi\langle x, y\rangle) \rightarrow \forall x\, \phi(x)\ . \tag{1}$$

The simplest pairing function is the Cantor's *diagonal* function offset by one:

$$\langle x, y\rangle = \frac{(x+y)(x+y+1)}{2} + x + 1\ .$$

Projection functions H (*Head*) and T (*Tail*) satisfy $H(0) = T(0) = 0$, $H\langle v, w\rangle = v$, and $T\langle v, w\rangle = w$. All three functions are primitive recursive (see [Dav85]). *Pair size* $|x|$ of a number x is the number of pairing operations needed to construct x and we have $|0| = 0$ and $|\langle v, w\rangle| = |v| + |w| + 1$. This is a definition by *course of values* which is reducible to primitive recursion.

For $n \geq 3$ we abbreviate $\langle x_1, \langle x_2, \ldots, x_n\rangle\rangle$ to $\langle x_1, x_2, \ldots, x_n\rangle$. Because 0 is the only atom, every natural number is uniquely a *list*, i.e. a code of a finite sequence of natural numbers where the empty sequence is coded by the empty list 0 (LISP's *nil*) and a sequence $x_1\ x_2\ \ldots x_n$ is coded by the list $\langle x_1, x_2, \ldots, x_n, 0\rangle$. The *length* $L(x)$ of the list x is defined by $L(0) = 0$ and $L\langle v, w\rangle = L(w) + 1$.

In order to obtain a simple recursive characterization of subelementary complexity classes (such as P) one should use a pairing function such that $|x| = \Omega(\log(x))$ [Vod94]. CL uses such a function but for the purposes of this paper

this requirement is not important and we use the simpler Cantor's function which does not satisfy the requirement.

1.2 **Contractions.** In the presence of pairing it suffices to deal only with unary functions because to each n-ary function f we can define its unary *contraction* function $\langle f \rangle$ such that

$$f(x_1, \ldots, x_n) = \langle f \rangle \langle x_1, \ldots, x_n \rangle \ .$$

The use of contractions in CL frees the symbol comma from its role of the separator of arguments. CL uses the infix pairing notation x, y instead of $\langle x, y \rangle$ and deals with unary functions and predicates only. However, such an use of comma requires some practice in reading definitions which we do not want to impose on a casual reader of this paper and so we continue to use here n-ary functions and denote the pairing by $\langle x, y \rangle$.

2 Examples of Clausal Definitions

In this section we give example function definitions in the form of clauses of CL. We do not dwell on the syntax of clauses as we consider them almost self-explanatory. The only requirement imposed on clausal definitions is that they should be mechanically transformable into a closed form of definitions discussed in the following section. We intend to demonstrate that simple recursion and/or induction does not always lead to a definition which, when used as rewriting rules, is efficient. A computationally optimal definition usually needs a more complex recursion and/or induction.

2.1 **Pair recursion.** The function $x \oplus y$ concatenating lists x and y has a following clausal definition by course of values recursion:

$0 \oplus y = y$
$\langle v, w \rangle \oplus y = \langle v, w \oplus y \rangle \ .$

The function Rt has a similar clausal definition:

$Rt(0) = 0$
$Rt\langle v, w \rangle = \langle 0, Rt(v) \oplus Rt(w) \rangle \ .$

The function Rt takes a number x and yields a list satisfying $L\, Rt(x) = |x|$. Lists in the range of Rt contain at most zeros. Declarative programmers will recognize $Rt(x)$ as a function *flattening* the tree x. Recursion from $\langle v, w \rangle$ to v and/or w is called *pair recursion*.

Alternatively, the definition of Rt can be automatically extracted as a *witness* from the proof of its *specification*: $\exists z\, L(z) = |x|$. The proof is by pair induction on x. If $x = 0$ take $z = 0$. If $x = \langle v, w \rangle$ obtain z_1 and z_2 from IH and set $z = \langle 0, z_1 \oplus z_2 \rangle$. We have

$$L(z) = L\langle 0, z_1 \oplus z_2 \rangle = L(z_1) + L(z_2) + 1 \stackrel{IH}{=} |v| + |w| + 1 = |\langle v, w \rangle| \ .$$

The function Rt has a simple definition by pair recursion or it can be extracted from a proof by pair induction. Unfortunately, this is not a very efficient definition to use as rewrite rules for computation. The number of operations needed to evaluate $Rt(x)$ is quadratic in $|x|$ due to the repeated concatenation. We will remedy this situation in Par. 2.4.

2.2 **Structural induction.** As every worker in the field of declarative programming knows there is a need for a special kind of *structural* induction for recursive data structures. A typical example is a definition of the type of *binary trees with labels of type* T found in practically all functional languages: $Bt = E \mid Nd(T, Bt, Bt)$. Semantically a new sort freely generated by this definition is added to the domain of such a language. We cannot extend the domain $\mathbb{N}$ of CL but we can code binary trees into natural numbers. For that we need two *constructors*: a constant $E = \langle 0, 0\rangle$ and a function $Nd(x) = \langle 1, x\rangle$. The type Bt is in CL a predicate holding exactly of codes of binary trees:

$Bt(E)$
$Bt\, Nd\langle n, a, b\rangle \leftarrow T(n) \wedge Bt(a) \wedge Bt(b)$.

Such a clausal definition is *minimal* in the sense that unless the truth of $Bt(x)$ follows from the clauses the predicate does not hold for x by *default*. The predicate is primitive recursive by course of values pair recursion because $a < Nd\langle n, a, b\rangle$. We have the following principle of *structural induction* which can be used to prove that a property $\phi(x)$ holds of all (codes of) binary trees:

$$\phi(E) \wedge \forall n \forall a \forall b(T(n) \wedge \phi(a) \wedge \phi(b) \rightarrow \phi(Nd\langle n, a, b\rangle)) \wedge Bt(x) \rightarrow \phi(x) \ .$$

The principle is reduced to complete induction as follows. We assume its first two conjuncts and continue by complete induction on x with the induction formula $Bt(x) \rightarrow \phi(x)$. We take any x s.t. $Bt(x)$ and consider two cases (which follow from the minimality of Bt). If $x = E$ we use the first assumption. If $x = Nd\langle n, a, b\rangle$ then $T(n)$, $Bt(a)$, and $Bt(b)$ and so, since $a < x$ and $b < x$, we get $\phi(a)$ and $\phi(b)$ from IH. Thus we get $\phi(x)$ from the second assumption.

2.3 **Substitution in parameters.** Consider the following clausal definition of the minimum function $\min(x, y)$:

$\min(0, y) = 0$
$\min(x + 1, 0) = 0$
$\min(x + 1, y + 1) = \min(x, y) + 1$.

This is a definition by primitive recursion (on x) with *substitution in parameter* (y) which does not lead outside of primitive recursion. The reduction to primitive recursion is important extensionally for knowing that min is primitive recursive. Intensionally, the function should be computed directly from its clauses used as rewrite rules. Colson [Col91] has proved that this kind of intensional behavior is impossible with any primitive recursive derivation of the minimum function

when the identities are used intensionally as rewriting rules. The basic property of min:

$$\forall y\,(x \geq y \wedge \min(x,y) = y \vee x < y \wedge \min(x,y) = x) \tag{1}$$

is proved by induction on x. Note that the quantifier is necessary because min is defined by substitution in parameter.

We observe that the above definition of min is computationally inefficient because the minimum can be computed exponentially faster by *recursion on notation* which goes from $2 \cdot x$ and $2 \cdot x + 1$ to x.

2.4 **Nested recursion.** We can obtain a more efficient definition of Rt by defining an auxiliary *accumulator* function Rta which does not apply concatenation. We will obtain the accumulator function as a witness of (1) to get $Rta(x,a) = Rt(x) \oplus a$. We can then explicitly define the function Rt by $Rt(x) = Rta(x,0)$. The accumulator function substitutes in the accumulator a so we prove

$$\forall a \exists z\; z = Rt(x) \oplus a \tag{1}$$

by pair induction on x. If $x = 0$ then it suffices to take $z = a$. If $x = \langle v, w\rangle$ then for a given a we obtain by IH a z_1 such that $z_1 = Rt(w) \oplus a$. By another IH we obtain z_2 such that $z_2 = Rt(v) \oplus z_1$. It now suffices to set $z = \langle 0, z_2\rangle$ because

$$z = \langle 0, z_2\rangle \overset{IH}{=} \langle 0, Rt(v) \oplus z_1\rangle \overset{IH}{=} \langle 0, Rt(v) \oplus Rt(w) \oplus a\rangle = Rt\langle v, w\rangle \oplus a\ .$$

The witness can be now automatically extracted from the proof to satisfy:

$Rta(0,a) = a$
$Rta(\langle v, w\rangle, a) = \langle 0, Rta(v, Rta(w,a))\rangle\quad .$

This is a definition by *nested* pair recursion.

2.5 **Assignments.** The following definition by primitive recursion shows a construct which makes computation more efficient:

$f(0) = 1$
$f(x+1) = \langle y, y\rangle \leftarrow f(x) = y\ .$

We could have defined the same function with the second clause $f(x+1) = \langle f(x), f(x)\rangle$. The latter definition needs 2^x recursions to compute $f(x)$ whereas the *assignment* of $f(x)$ to the auxiliary variable y reduces the number of recursions to x.

2.6 **Non-primitive recursive function.** The well-known non-primitive recursive Ackermann-Péter function (see [Pét67,Ros82]) has the following clausal definition:

$A(0,y) = y+1$
$A(x+1,0) = A(x,1)$
$A(x+1,y+1) = A(x, A(x+1,y))\ .$

By Thm. 3.9 this is a definition not permitted in CL.

3 Recursion with Measure

3.1 **Syntax of recursion with measure.** CL restricts the syntax of clausal definitions in such a way that the clauses can be mechanically transformed into *closed form*:

$$f(x_1, \ldots, x_n) = \tau(f, x_1, \ldots, x_n) \ . \tag{1}$$

The term τ is composed from the constant 0 and variables by applications of the successor function $S(x) = x + 1$, pairing $\langle x, y \rangle$, previously defined functions, recursive applications of f, and by three operators:

$$\textit{if } \tau_1 = S(x) \textit{ then } \tau_2(x) \textit{ else } \tau_3 \tag{2}$$

$$\textit{if } \tau_1 = \langle x, y \rangle \textit{ then } \tau_2(x, y) \textit{ else } \tau_3 \tag{3}$$

$$\textit{let } \tau_1 = x \textit{ in } \tau_2(x) \ . \tag{4}$$

The *pattern* variables on the right-hand-side of the identities must be new and they are *bound* in terms with indicated variables. Such terms τ are well-known from LISP and from functional languages and they allow a fine degree of control over evaluation.

The actual implementation of CL has a slightly richer syntax of terms τ (for instance built-in arithmetic operators). The syntax of terms τ is not minimal because we wish to demonstrate that the coding of S-expressions into natural numbers brings the benefits of LISP-like programming without any deterioration of efficiency (see Paragraphs 3.4 and 3.5).

3.2 **Semantics of recursion with measure.** We assign the meaning to the three operators from Par. 3.1 as follows. The operator 3.1(4) denotes the same as the term $\tau_2(\tau_1)$. For the two *if* operators we introduce the *case discrimination* function D by primitive recursion: $D(0, y, z) = z$ and $D(x+1, y, z) = y$. The operator 3.1(2) denotes the same as the term $D(\tau_1, \tau_2(\tau_1 \dot{-} 1), \tau_3)$ and the operator 3.1(3) denotes the same as the term $D(\tau_1, \tau_2(H(\tau_1), T(\tau_1)), \tau_3)$. The remaining constructs in terms τ have their standard meaning.

The definition 3.1(1) can be either *explicit* if f is not applied in τ or else it must be *regular*, in that there is a measure function $m : \mathbb{N}^n \mapsto \mathcal{O}$ into an initial segment of ordinal numbers. The recursion in τ *must go down in the measure* m in the sense that for every recursive application $f(\overline{\tau}_1, \ldots \overline{\tau}_n)$ in τ we have $m(\overline{\tau}_1, \ldots \overline{\tau}_n) < m(x_1, \ldots, x_n)$ under the assumption of all *conditions governing* the recursive application. If the application occurs in τ after some *then* (*in*) of an operator then it is governed by the condition after *if* (*let*). If the application occurs after some *else* of an operator then it is governed by $\tau_1 = 0$ for τ_1 from that operator.

The regularity condition guarantees that we have

$$\tau(f, x_1, \ldots, x_n) = \tau([f]_{x_1, \ldots, x_n}, x_1, \ldots, x_n)$$

where we denote by $[g]_{x_1,\ldots,x_n}$ the *m-restriction* of g, i.e the function satisfying:

$$[g]_{x_1,\ldots,x_n}(y_1,\ldots,y_n) = \begin{cases} g(y_1,\ldots,y_n) & \text{if } m(y_1,\ldots,y_n) < m(x_1,\ldots,x_n) \\ 0 & \text{otherwise} \end{cases}$$

It is well-known from set theory that there is a unique function f defined by transfinite recursion such that $f(x_1,\ldots,x_n) = \tau([f]_{x_1,\ldots,x_n}, x_1,\ldots,x_n)$ and f is thus the unique function satisfying 3.1(1).

3.3 **Measures of example functions.** The measure for the primitive recursion in both clausal definitions of min from Par. 2.3 as well as in the definition of f from Par. 2.5 is the identity function $I(x) = x < \omega$.

We can also use I as a measure for definitions by pair recursion of functions $|x|$, L, $\oplus$, Rt, and Rta but the measure $|x|$ is superior as the recursion in CL runs exponentially faster due to $|x| = \Omega(\log(x))$ (this, however, is not true for Cantor's function used here).

The Ackermann-Péter function has the measure $m(x,y) = \omega \cdot x + y < \omega^2$. For instance, for the outer recursive application in the third clause we have

$$m(x, A(x,y)) = \omega \cdot x + A(x,y) < \omega \cdot (x+1) + y + 1 = m(x+1, y+1) \ .$$

3.4 **Numerals.** *Unary numerals* are terms composed from the constant 0 by the application of the successor function $S(x) = x + 1$. The numeral $S^n(0)$, i.e. $\overbrace{S \ldots S}^{n}(0)$, denotes the number n. *Pair numerals* are terms composed from 0 by pairing $\langle \cdot, \cdot \rangle$. Clearly, to every natural number there is exactly one pair numeral denoting the number.

In order to facilitate not only effective but also *efficient* computation we introduce *mixed numerals* as terms composed from 0 by the successor function and pairing. It is now possible that two different mixed numerals denote the same number. Mixed numerals have a simple representation in computers. Unary numerals are represented in binary (say by the Bignum's of LISP). The mixed numeral $\langle \tau_1, \tau_2 \rangle$ is represented by a pointer to a LISP-cell with pointers to the representations of mixed numerals τ_1 and τ_2.

Conversions between mixed numerals are effectively computable in the sense that we can effectively find the mixed numerals denoting $\langle \tau_1, \tau_2 \rangle \dot{-} 1$ (this is used in 3.1(2)), $H\,S(\tau_1)$, and $T\,S(\tau_1)$ (this is used in 3.1(3)). The conversions are effective because the pairing and projection functions are primitive recursive.

3.5 **Effective computability of recursion with measure.** We say that an n-ary function f is *effectively computable* if there is a mechanical process which, given the mixed numerals $\tau_1, \ldots, \tau_n$ denoting the numbers $k_1, \ldots k_n$ respectively, yields after a finite time a mixed numeral denoting the number $f(k_1, \ldots, k_n)$.

We now assume that all functions (except f) applied in a term τ from 3.1(1) are effectively computable. Suppose that we are given mixed numerals $\tau_1, \ldots, \tau_n$

as arguments for the (free) variables of τ. If the definition is explicit one can easily prove by induction on construction of τ that it is effectively computable into a mixed numeral. If the definition is recursive we assume that f can be computed for all mixed numerals $\overline{\tau}_1, \ldots, \overline{\tau}_n$ such that $m(\overline{\tau}_1, \ldots, \overline{\tau}_n) < m(\tau_1, \ldots, \tau_n)$ and show similarly that τ can be effectively computed into a mixed numeral (the outer induction is thus a transfinite induction on m).

Definitions of CL functions are almost always well-behaved in the sense that they perform list operations on numerals with outermost pairings and arithmetic operations on unary numerals. The need for the time consuming conversions is exceptional.

3.6 **Multiply recursive functions.** The Ackermann-Péter function A is an example of a 2-*recursive function* in the hierarchy of *multiply recursive functions* by Péter [Pét67,Ros82]. Its definition is by *nested double recursion*.

The function f is 1-*recursive* if it is defined by *nested simple recursion*:

$$f(0, \overline{x}) = g(x_1, \overline{x}) \tag{1}$$

$$f(z+1, \overline{x}) = \tau(f(z, \cdot), z, \overline{x}) \tag{2}$$

where we denote by $\overline{x}$ the n-tuple of variables $x_1, \ldots, x_n$. The right-hand-side of the second identity should be understood as the term obtained from $\tau(f, z, \overline{x})$ by replacing all applications of $f(\overline{\tau}_1, \ldots, \overline{\tau}_n)$ by $f(z, \overline{\tau}_1, \ldots, \overline{\tau}_n)$. Nested simple recursion is a special case of recursion with measure where $m(z, \overline{x}) = z$. Note that the term τ does not need to be regular because the regularity is enforced by the restriction $f(z, \cdot)$.

For a measure function $m(\overline{x}) < \omega$ we can reduce a regular definition 3.1(1) into nested simple recursion by defining an auxiliary function $\overline{f}$:

$$\overline{f}(0, \overline{x}) = \tau(Z(\cdot), \overline{x})$$

$$\overline{f}(z+1, \overline{x}) = \tau(\overline{f}(z, \cdot), \overline{x})$$

where $Z(\overline{x}) = 0$. We then explicitly set $f(\overline{x}) = \overline{f}(m(\overline{x}), \overline{x})$.

We now present a proof of slightly generalized Péter's theorem [Pét32]. The proof is done in a way which can be readily formalized by Thm. 4.9 in a small fragment of Peano arithmetic.

3.7 **Péter's theorem.** *Primitive recursive functions are closed under nested simple recursion.*

Proof. Because of pairing we may assume that the function f defined by a nested simple recursion is of the form $f(z, x)$. We assume that the function f is applied in τ at least once because otherwise there is nothing to prove. Finally, we assume that the *if* and *let* operators have been removed from the term τ as suggested in Par. 3.2. We 'unnest' from inside out and left to right the applications of f in τ by writing the definition in the clausal form:

$$f(0,x) = g(x)$$
$$f(z+1,x) = \tau_k(z,x,t_0,t_1,\ldots,t_{k-1}) \leftarrow$$
$$\quad f(z,\tau_0(z,x)) = t_0 \wedge f(z,\tau_1(z,x,t_0)) = t_1 \wedge \ldots \wedge$$
$$\quad f(z,\tau_{k-1}(z,x,t_0,t_1,\ldots,t_{k-2})) = t_{k-1} \quad .$$

Here the terms τ_i for $0 \leq i \leq k$ contain at most the indicated variables and do not apply f. Let us until the end of the proof call a list t such that $L(t) \leq k$ a *table for* $f(z+1,x)$ if for all $0 \leq i < L(t)$ we have $(t)_i = f(z,\tau_i(z,x,(t)_0,(t)_1,\ldots,(t)_{i-1}))$. Here $(t)_i$ denotes the *list indexing* function $(t)_i = H\,T^i(t)$. The table is *partial* if $L(t) < k$ and *full* otherwise. Note that 0 is a partial table for every $f(z+1,x)$. We will use tables to hold the values of the auxiliary recursive applications needed in the evaluation of $f(z+1,x)$. A partial table t for $f(z+1,x)$ can be used to determine the value of the parameter $\tau_{L(t)}$ for the next recursive application to be evaluated. A full table determines the value of the term τ_k and hence, of the application $f(z+1,x)$. This is captured by a primitive recursive function V with an explicit clausal definition:

$$V(z,x,0) = \tau_0(z,x)$$
$$V(z,x,\langle t_0,0\rangle) = \tau_1(z,x,t_0)$$
$$V(z,x,\langle t_0,t_1,0\rangle) = \tau_2(z,x,t_0,t_1)$$
$$\vdots$$
$$V(z,x,\langle t_0,t_1,\ldots,t_{k-2},0\rangle) = \tau_{k-1}(z,x,t_0,t_1,\ldots,t_{k-2})$$
$$V(z,x,\langle t_0,t_1,\ldots,t_{k-2},t_{k-1},0\rangle) = \tau_k(z,x,t_0,t_1,\ldots,t_{k-2},t_{k-1}) \ .$$

If t is a table for $f(z+1,x)$ then $V(z,x,t) = \tau_{L(t)}(z,x,(t)_0,(t)_1,\ldots,(t)_{L(t)-1})$. Hence, $V(z,x,t)$ yields for a partial t the parameter of the next recursive application of f to be evaluated. If t is full then we have $f(z+1,x) = V(z,x,t)$.

In order to evaluate the application $f(z,x)$ we will keep *lists* $\overline{t}$ *of tables for* $f(z,x)$ where $\overline{t}$ is a list of length z such that $(\overline{t})_i$ is a partial table for $f(z-i,x_i)$ for each $i < z$. The parameters x_i are such that $x_0 = x$ and for $i+1 < z$ we have $x_{i+1} = V(z-(i+1),x_i,(\overline{t})_i)$. Note that for $i < z$ the parameters can be determined as $x_i = B(i,x,\overline{t})$ by a primitive recursive function B:

$$B(0,x,\overline{t}) = x$$
$$B(i+1,x,\overline{t}) = V(L(\overline{t}) \dot{-} (i+1), B(i,x,\overline{t}), (\overline{t})_i) \ .$$

We note that $x_z = B(z,x,\overline{t})$ is the parameter for the application $f(0,x_z)$ to be computed next in the last table $(\overline{t})_{z-1}$ of the list of tables $\overline{t}$ for $f(1,x_{z-1})$. We also note that $\overline{t}_0 = \sigma(z)$, where $\sigma(0) = 0$ and $\sigma(x+1) = \langle 0,\sigma(x)\rangle$, is a list of tables for every $f(z,x)$ and that every list of tables for $f(0,x)$ must be 0.

In order to evaluate $f(z,x)$ for $z > 0$ we need the values of k recursive applications $f(z-1,\tau_i)$. If $z > 1$ each such an application needs again k evaluations of f. Thus we need k^z evaluations of f in order to compute $f(z,x)$. This suggests a computation strategy for f where we start the evaluation of $f(z,x)$ with the list of tables $\overline{t}_0 = \sigma(z)$ and after $j < k^z$-steps we will have list of tables for $f(z,x)$: $\overline{t}_j$ with each partial table $(\overline{t}_j)_i$ for $i < z$ having the length given by the i-th digit of the expansion of j in the positional representation with k digits 0 through

$k-1$. We do not suppress the leading zeros and count the digits from left to right starting with 0 and ending with $z-1$.

The step $j+1$ of the evaluation will thus resemble the addition of one to the number j in the k-ary notation. We will add the value of $v = f(0, x_z)$ to the partial table $t = (\overline{t}_j)_{z-1}$ thereby obtaining a new table $t \oplus \langle v, 0\rangle$. Should this table become full, it will be reset to the empty table 0 in the list of tables $\overline{t}_{j+1}$ and we will propagate as a 'carry' the value $V(0, x_{z-1}, t \oplus \langle v, 0\rangle) = f(1, x_{z-1})$. If $j+1 < k^z$ then the process of carries rippling through will terminate in some partial table of $\overline{t}_{j+1}$. If $j+1 = k^z$ then $\overline{t}_{j+1}$ will be the empty list of tables $\sigma(z)$ again and the carry out will yield the value of $f(z, x)$.

We indicate no carry by 0 and a carry of the value v by $\langle 0, v\rangle$. The list of tables $\overline{t}_{j+1}$ is be produced by the function *Incr* which yields a pair consisting of a carry and of a list of tables such that $Incr(\overline{t}_j, x, \overline{t}_j) = \langle c, \overline{t}_{j+1}\rangle$. The function has a clausal definition by pair recursion:

$$
\begin{aligned}
&Incr(\overline{s}, x, 0) = \langle\langle 0, g\, B(L(\overline{s}), x, \overline{s})\rangle, 0\rangle\\
&Incr(\overline{s}, x, \langle t, \overline{t}\rangle) = \langle 0, t, \overline{t}_1\rangle \leftarrow Incr(\overline{s}, x, \overline{t}) = \langle 0, \overline{t}_1\rangle\\
&Incr(\overline{s}, x, \langle t, \overline{t}\rangle) = \langle 0, t \oplus \langle v, 0\rangle, \overline{t}_1\rangle \leftarrow Incr(\overline{s}, x, \overline{t}) = \langle\langle 0, v\rangle, \overline{t}_1\rangle \wedge L(t) < k-1\\
&Incr(\overline{s}, x, \langle t, \overline{t}\rangle) = \langle\langle 0, V(L(\overline{t}), B(L(\overline{s}) \dot{-} (L(\overline{t})+1), x, \overline{s}), t \oplus \langle v, 0\rangle)\rangle, 0, \overline{t}_1\rangle \leftarrow\\
&\qquad Incr(\overline{s}, x, \overline{t}) = \langle\langle 0, v\rangle, \overline{t}_1\rangle \wedge L(t) = k-1\ .
\end{aligned}
$$

The function $\overline{Incr}$ iterates the function *Incr* in such a way that $\overline{Incr}(i, x, \overline{t}_0) = \langle c, \overline{t}_i\rangle$ and it is defined by primitive recursion:

$$
\begin{aligned}
&\overline{Incr}(0, x, \overline{t}) = \langle 0, \overline{t}\rangle\\
&\overline{Incr}(i+1, x, \overline{t}) = Incr(\overline{t}_1, x, \overline{t}_1) \leftarrow \overline{Incr}(i, x, \overline{t}) = \langle c, \overline{t}_1\rangle\ .
\end{aligned}
$$

The function f defined by nested simple recursion has thus a primitive recursive derivation by an explicit clausal definition:

$$f(z, x) = v \leftarrow \overline{Incr}(k^z, x, \sigma(z)) = \langle\langle 0, v\rangle, \overline{t}\rangle\ . \qquad \square$$

3.8 **CL-definable functions.** Already the elementary functions (for definition see [Kal43,Ros82]), which are but a tiny subclass of primitive recursive functions, are computationally unfeasible. It is generally accepted that the computationally *feasible* functions are the functions computable in polynomial time. The class P of such functions was characterized by Cobham [Cob65,Ros82] with the help of *recursion on notation* which goes from $2x$ and $2x+1$ to x. We have characterized P in [Vod94] with the help of pair recursion. The class P is a proper subset of elementary functions. We have decided to restrict CL to primitive recursive functions because they have simpler closure properties than the elementary functions or the functions in P. Primitive recursive functions have also a simple formal theory (see Sect. 4) for proving the properties of defined functions. The reader might wish to consult [Wai97] for a similar argument in favor of primitive recursive functions.

We say that a function is *CL-definable* if it has a clausal definition from previously CL-defined functions such that the clauses can be converted to an explicit or regular recursion with measure 3.1(1). If the definition is regular then

also the measure function must be CL-defined (and so it must be into finite ordinals $\mathbb{N}$).

3.9 **Theorem** *(Characterization of primitive recursive functions). CL-definable functions are exactly the primitive recursive functions.*

Proof. That CL-definable functions are primitive recursive follows from Péter's theorem and from the reduction of regular definitions to nested simple recursion in Par. 3.6. Vice versa, the *initial* primitive recursive functions, i.e. the successor, zero and, projection functions, have explicit definitions $S(x) = S(x)$, $Z(x) = 0$, and $I_i^n(x_1, \ldots, x_i, \ldots, x_n) = x_i$ respectively. Functions defined by substitution have explicit definitions: $f(\overline{x}) = g(h_1(\overline{x}), \ldots, h_m(\overline{x}))$ and functions defined by primitive recursion have a regular clausal definition with the measure $I_1^{n+1}(x, \overline{y}) = x$:

$f(0, \overline{y}) = g(\overline{y})$
$f(x+1, \overline{y}) = h(x, \overline{y}, f(x, \overline{y}))$. □

3.10 **Characterization of recursive functions.** By permitting arbitrary ordinal-valued measure functions in regular definitions (even non-computable ones and without ordinal notations in natural numbers) we can define by explicit and regular recursive definitions exactly the *recursive* functions. That every recursive function can be thus defined can be seen from the clausal derivation of *regular minimalization*:

$$f(\overline{x}) = \mu_y[g(y, \overline{x}) = 0]$$

where for arbitrary $\overline{x}$ there is a y such that $g(y, \overline{x}) = 0$. For the clausal derivation of f we define an auxiliary function $\overline{f}$:

$\overline{f}(y, \overline{x}) = z \leftarrow \mu_{z \leq y}[g(z, \overline{x}) = 0] = z \wedge g(z, \overline{x}) = 0$
$\overline{f}(y, \overline{x}) = \overline{f}(y+1, \overline{x}) \leftarrow \mu_{z \leq y}[g(z, \overline{x}) = 0] = z \wedge g(z, \overline{x}) > 0$.

This is a regular clausal definition using *bounded minimalization* which is reducible to primitive recursion. The measure is: $m(y, \overline{x}) = f(\overline{x}) \dot{-} y$ because when the recursion takes place we must have $g(z, \overline{x}) > 0$ for all $z \leq y$ and so $y < f(\overline{x})$. The function f is then explicitly derived as $f(\overline{x}) = \overline{f}(0, \overline{x})$.

We do not prove here the converse that every regular clausal definition 3.1(1) is recursive in the functions applied in τ. We just suggest the proof by doing it for the Ackermann-Péter function. We redo the definition in Par. 2.6 and add a new argument z:

$\overline{A}(0, 0, y) = y + 2$
$\overline{A}(0, x+1, y) = 0$
$\overline{A}(z+1, 0, y) = y + 2$
$\overline{A}(z+1, x+1, 0) = \overline{A}(z, x, 1)$
$\overline{A}(z+1, x+1, y+1) = \overline{A}(z, x, a) \leftarrow \overline{A}(z, x+1, y) = a + 1$
$\overline{A}(z+1, x+1, y+1) = 0 \leftarrow \overline{A}(z, x+1, y) = 0$.

The argument z of $\overline{A}(z,x,y)$ plays the role of a counter. If z is sufficiently large then $\overline{A}(z,x,y) = A(x,y)+1$, if z is not large enough to count the number of recursions in A then $\overline{A}(z,x,y) = 0$ (note the second clause for $\overline{A}$). The function $\overline{A}$ is primitive recursive because it is defined by nested simple recursion. Moreover, we have $\exists z\, \overline{A}(z,x,y) = A(x,y)+1$ as can be proved by transfinite induction with the measure $m(x,y) = \omega\cdot x+y$. Hence, we can define $\overline{m}(x,y) = \mu_z[\overline{A}(z,x,y) > 0]$ by regular minimalization and set $A(x,y) = \overline{A}(\overline{m}(x,y),x,y) \dot{-} 1$. Alternatively, the definition of A in Par. 2.6 has the measure $\overline{m}(x,y) < \omega$, and so A is primitive recursive in $\overline{m}$ which thus cannot be primitive recursive. The last shows the *collapse* of any measure function, no matter how large its ordinal range is, into a measure function $< \omega$ by a single regular minimalization.

4 Formal Arithmetic

4.1 **Peano arithmetic.** *Peano arithmetic* is obtained by the addition of *axioms of induction*:

$$\phi(0) \wedge \forall x(\phi(x) \to \phi(x+1)) \to \forall x \phi(x) \tag{1}$$

to the six axioms of *Robinson arithmetic*:

$$\begin{gathered} S(x) \neq 0 \\ S(x) = S(y) \to x = y \\ x \neq 0 \to \exists y\, x = S(y) \\ 0 + y = y \\ S(x) + y = S(x+y) \\ 0 \cdot y = 0 \\ S(x) \cdot y = x \cdot y + y\ . \end{gathered}$$

The language $\mathcal{L}$ of both arithmetics consists of the symbols 0, S, $+$ and $\cdot$. Axioms of both arithmetics have the *standard model* $\mathcal{N}$ whose domain is $\mathbb{N}$ and the symbols of the language have the standard interpretation. We write $T \vdash \phi$ when ϕ is provable from axioms T. The reader is invited to use his favorite (complete) proof system. In the following discussion we state many properties of Peano arithmetic without proofs. The reader is advised to consult [HP93] for more details.

4.2 **Fragments of Peano arithmetic.** If we restrict the formulas $\phi(x)$ in axioms of induction 4.1(1) to the formulas of certain quantifier complexity we obtain *fragments* of Peano arithmetic.

The class of $\Sigma_0 = \Pi_0$*-formulas* is defined to consist of formulas with at most *bounded quantifiers* $\exists x \leq s\, \phi(x)$ and $\forall x \leq s\, \phi(x)$. Bounded quantifiers are abbreviations for the formulas $\exists x \exists z(x+z = s \wedge \phi(x))$ and $\forall x \forall z(x+z = s \to \phi(x))$ respectively (z is a new variable). Σ_{n+1}*-formulas* are the formulas of the form

$\exists x_1 \ldots \exists x_k \phi$ where $\phi \in \Pi_n$ and *Π_{n+1}-formulas* are the formulas of the form $\forall x_1 \ldots \forall x_k \phi$ where $\phi \in \Sigma_n$ and $k \geq 1$.

We denote by $I\Sigma_{n+1}$ the fragment of Peano arithmetic whose axioms are axioms of Robinson arithmetic and the induction axioms for Σ_{n+1}-formulas.

Let T be a fragment of Peano arithmetic. We say that a formula ϕ is Σ_n (Π_n) *in* T if there is a formula $\psi \in \Sigma_n$ ($\psi \in \Pi_n$) such that $T \vdash \phi \leftrightarrow \psi$.

4.3 **Witnessing theorem.** Consider a Σ_1-formula $\phi(x_1, \ldots, x_n, y)$ with all of its free variables among the indicated ones and assume that a fragment T is such that

$$T \vdash \exists y\, \phi(x_1, \ldots, x_n, y)\ . \tag{1}$$

Then clearly, there is a *witness* to the existential quantifier, i.e. a function f such that for all numbers $k_1, \ldots, k_n$ we have

$$\mathcal{N} \vDash \phi(S^{k_1}(0), \ldots, S^{k_n}(0), S^{f(k_1,\ldots,k_n)}(0))\ .$$

The formula ϕ has a form $\exists z_1 \ldots \exists z_m \phi_1$ where the Σ_0-formula ϕ_1 is constructed by primitive recursive operations. The witness, which for given $x_1, \ldots, x_n$ looks for the least number v of the form $\langle y, z_1, \ldots, z_m \rangle$ whose projections make the formula ϕ_1 true and then yields $H(v)$, is clearly a recursive function.

The fragment which we have adopted as the formal system of CL is $I\Sigma_1$-arithmetic. We have chosen this fragment because the *witnessing* theorem for $I\Sigma_1$ [Kre52,Par70,Min73] says that every Σ_1-formula ϕ for which (1) holds has a primitive recursive witness. Moreover, a primitive recursive derivation of the witness can be mechanically extracted from the proof.

4.4 **Recursive extensions.** CL as a programming language allows incremental addition of new definitions of functions and predicates. This has a correspondence in the formal system where a fragment T of Peano arithmetic can be *extended* to the fragment T_1 by a *recursive extension*. For functions this means that when

$$T \vdash \forall \overline{x} \exists! y\, \phi(\overline{x}, y)$$

with $\phi \in \Sigma_1$ the language $\mathcal{L}_1$ of T_1 is obtained by adding a new n-ary function symbol f to the language of T. Axioms of T_1 are those of T plus the *defining* axiom for f:

$$f(\overline{x}) = y \leftrightarrow \phi(\overline{x}, y)\ . \tag{1}$$

With the addition of a new function symbol to T we uniquely *expand* the standard model $\mathcal{N}$.

Classes of formulas $\Sigma_n(f)$ and $\Pi_n(f)$ in the language $\mathcal{L}_1$ are defined similarly as the corresponding classes Σ_n and Π_n (see Par. 4.2).

The *graph* of f, i.e. the left-hand side of the above equivalence, is given by a Σ_1-formula. The fragment T_1 clearly proves that the graph is equivalent to

$\forall z(\phi(x_1,\ldots,x_n,z) \to y = z)$ which is easily shown to be Π_1 in T. This fact permits an *elimination* of all occurrences of f from formulas of $\mathcal{L}_1$ by replacing every atomic formula $\psi_y[f(\overline{x})]$ by formulas $\exists y(\phi(\overline{x},y) \wedge \psi)$ or $\forall y(\phi(\overline{x},y) \to \psi)$ (depending on the polarity of the replaced formula) either of which is equivalent in T_1 to the replaced formula. As a consequence every $\Sigma_1(f)$ $(\Pi_1(f))$ formula is a Σ_1 (Π_1) formula in T_1. For this reason T_1 is *conservative* over T, i.e. if for a formula ψ of $\mathcal{L}$ we have $T_1 \vdash \psi$ then also $T \vdash \psi$. In other words, recursive extensions do not add anything new to the extended theories except increased expressivity. In situations where there can be no confusion we will use the symbol $I\Sigma_1$ to denote also the axioms of recursive extensions. We can use introduced symbols in induction axioms and in the formulas introducing new symbols because they can be always eliminated without increasing the quantifier complexity.

4.5 **Primitive recursive functions in $I\Sigma_1$.** For every primitive recursive function f there is a recursive extension of $I\Sigma_1$ which proves the identities used in the derivation of f (i.e. those in the proof of Thm. 3.9) as theorems. Vice versa, if we extend $I\Sigma_1$ with suitable recursive identities for primitive recursive functions as axioms we can derive their defining axioms 4.4(1) as theorems. This equivalence has the advantage that the axioms in recursive extensions can be taken as open formulas (i.e. formulas without quantifiers).

The pairing function $\langle x,y\rangle$ and its projections H and T are primitive recursive. So the functions can be introduced into a recursive extension of $I\Sigma_1$ in such a way that their properties given in Par. 1.1 are theorems.

4.6 **Pair induction.** The principle of pair induction 1.1(1) for any Σ_1-formula ϕ can be proved in a recursive extension containing the pairing function. We reason in the extension, assume the antecedent of 1.1(1), and continue by induction on z with the formula $\forall x < z\, \phi(x)$ (which is Σ_1 in the extension). For the base case there is nothing to prove. If $x < z+1$ then we distinguish two cases. If $x = 0$ we have $\phi(x)$ from the assumption. Otherwise there are v and w such that $v,w < \langle v,w\rangle = x \le z$ and so $\phi(v)$ and $\phi(w)$ by IH. Hence, $\phi\langle v,w\rangle$ from the assumption. Now, since $x < x+1$, we have $\phi(x)$ for any x.

4.7 **Π_{n+1}-induction in $I\Sigma_{n+1}$.** For every $\phi(x) \in \Pi_{n+1}$ the theory $I\Sigma_{n+1}$ proves the Π_{n+1}-induction principle 4.1(1). We reason in $I\Sigma_{n+1}$ and assume the antecedent of 4.1(1) as well as $\neg\forall x\phi(x)$. For some x we then have $\neg\phi(x)$ and we prove

$$z \le x \to \neg\phi(x \dot{-} z)\ , \tag{1}$$

which is Σ_{n+1} in $I\Sigma_{n+1}$, by induction on z. For $z = 0$ this follows from $\neg\phi(x)$. If $z+1 \le x$ then $z < x$ and $\neg\phi(x \dot{-} z)$ from IH. From $x \dot{-} z > 0$ we have $x \dot{-} (z+1) + 1 = x \dot{-} z$ and so $\neg\phi(x \dot{-} (z+1))$ from the inductive assumption of 4.1(1). By taking $z = x$ we get from (1) a contradiction $\neg\phi(0)$.

4.8 **Clausal definitions in $I\Sigma_1$.** Every clausal definition admissible in CL can be mechanically transformed into the form 3.1(1). The demonstration that the two forms are equivalent does not require induction and can be formalized in any fragment of arithmetic.

We wish to prove the property 3.1(1) as a theorem in a suitable recursive extension of $I\Sigma_1$. This is a technically quite demanding proof because we have to make sure that the inductions needed in the reduction of 3.1(1) first to nested simple recursion in Par. 3.6 and then in the proof of Péter's theorem 3.7 are available. This is indeed the case and the formalization of Péter's theorem goes through. Obviously, we cannot do the proof here and so we just state the formalized version of Peter's theorem:

4.9 **Formalized Péter's theorem.** *If the function f is defined by nested simple recursion from primitive recursive functions then there is a recursive extension of $I\Sigma_1$ which proves the identities 3.6(1)(2) as theorems.* □

4.10 **Proof obligations.** Before a recursive clausal definition is accepted its regularity must be proved in CL (this is called a *proof obligation*). The proof turns the semantic condition of regularity (in general undecidable) to a syntactic one (which is decidable). It should be clear that by replacing the semantic condition by a syntactic one we do not lose any primitive recursive function extensionally, we may just lose some definitions intensionally, i.e. as rewriting rules. It can happen that $I\Sigma_1$ is not strong enough to prove a condition of regularity which is true in $\mathcal{N}$.

4.11 **Ackermann-Péter function revisited.** By the witnessing theorem we have $I\Sigma_1 \nvdash \exists z\, \overline{A}(z,x,y) > 0$ for the function $\overline{A}$ from Par. 3.10 because the primitive recursive witnessing function $\overline{m}(x,y)$ would make also the Ackermann-Péter function A primitive recursive. However, we can introduce A into $I\Sigma_2$ by proving

$$\forall y \exists z\, \overline{A}(z,x,y) > 0 \tag{1}$$

by Π_2-induction on x. For $x = 0$ we take $z = 0$. For $x+1$ we proceed by induction on y with the Σ_1-formula $\exists z \overline{A}(z, x+1, y) > 0$. In the base case we get a z_1 such that $\overline{A}(z_1, x, 1) > 0$ from the outer IH and set $z = z_1 + 1$. In the inductive case we get a z_1 s.t. $\overline{A}(z_1, x+1, y) > 0$ from the inner IH and a z_2 s.t. $\overline{A}(z_2, x, \overline{A}(z_1, x+1, y) \dot{-} 1) > 0$ from the outer IH. We then set $z = \max(z_1, z_2) + 1$.

4.12 **Theorem of Parsons.** Because Π_1-induction is available in $I\Sigma_1$ the theory proves the property 2.3(1). On the other hand, Π_2-induction is not available in $I\Sigma_1$ and we cannot directly prove the property 2.4(1) needed for the extraction of the witnessing function *Rta*. Fortunately, a theorem by Parsons [Par70,Bus94,Sie91] helps us here. The theorem says that

Robinson arithmetic plus Π_{n+2}-rules inferring $\forall x\phi(x)$ from $\phi(0)$ and $\forall x(\phi(x) \to \phi(x+1))$ for any $\phi \in \Pi_{n+2}$ are Π_{n+2}-conservative over $I\Sigma_{n+1}$.

In our special case $n = 1$ this means that we can derive the Π_2-formula 2.4(1) by a Π_2-induction rule and then use the conservativity to derive the same formula in $I\Sigma_1$. This is possible because the proof of 2.4(1) is without any side assumptions. Note that this is not the case in the proof of 4.11(1) because the inner induction uses the outer IH as a side assumption and thus cannot be turned into a rule.

4.13 **Formal proof system of CL.** We will outline the proof of a liberalization of the case $n = 1$ of Parsons theorem in Thm. 4.15. As this is a proof-theoretic argument we now present the proof system of CL which are *signed tableaux* of Smullyan [Smu68]. We write $\phi*$ instead of Smullyan's $F\phi$ and ψ instead of $T\psi$. We can view formulas $\phi*$ as *goals* to be proved under the *assumptions* given by the formulas ψ.

A branch of a signed tableau can be viewed as a *sequent* where the conjunction of assumptions implies the disjunction of goals. This interpretation of signed tableaux gives them a flavor of *natural deduction* which can be readily translated into English description of proofs (as it is the case in CL). We just mention here that this aspect of signed tableaux seems to have escaped the workers in the field and CL is probably the first formal system with such an interpretation.

If we wish to prove a formula ϕ we write it down as a goal $\phi*$ and then proceed to build a tableau as a downwards growing tree with the following expansion rules.

Propositional rules:

$$\frac{\phi \to \psi*}{\begin{array}{c}\phi\\ \psi*\end{array}} \quad \frac{\phi \vee \psi*}{\begin{array}{c}\phi*\\ \psi*\end{array}} \quad \frac{\phi \wedge \psi}{\begin{array}{c}\phi\\ \psi\end{array}} \quad \frac{\phi \wedge \psi*}{\phi* \mid \phi*} \quad \frac{\phi \vee \psi}{\phi \mid \psi} \quad \frac{\phi \to \psi}{\psi \mid \phi*} \quad \frac{\neg\phi*}{\phi} \quad \frac{\neg\phi}{\phi*}\,.$$

Equational rules:

$$\frac{}{s = s} \quad \frac{s = t}{t = s} \quad \frac{s = t \quad t = u}{s = u}$$

$$\frac{s_1 = t_1 \cdots s_n = t_n}{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)} \quad \frac{s_1 = t_1 \cdots s_n = t_n \; R(s_1, \ldots, s_n)}{R(t_1, \ldots, t_n)}\,.$$

Quantifier rules (†: y is an eigen-variable):

$$\frac{\forall x\phi(x)*}{\phi(y)*}\,\dagger \quad \frac{\exists x\phi(x)}{\phi(y)}\,\dagger \quad \frac{\exists x\phi(x)*}{\phi(s)*} \quad \frac{\forall x\phi(x)}{\phi(s)}\,.$$

Axiom (†) and **Cut** (‡) **rules**:

$$\frac{}{\phi}\,\dagger \qquad \frac{}{\phi \mid \phi*}\,\ddagger\,.$$

$I\Sigma_1$-induction rule for $\exists\overline{x}\phi(z,\overline{x})$ where $\phi(z,\overline{x}) \in \Sigma_0$, z, $\overline{y}$ eigen-variables:

$$\frac{\exists\overline{x}\phi(s,\overline{x})*}{\exists\overline{x}\phi(0,\overline{x})* \mid \begin{array}{c}\phi(z,\overline{y})\\ \exists\overline{x}\phi(z+1,\overline{x})*\end{array}}.$$

Axioms are the axioms of Robinson arithmetic as well as the defining axioms for introduced function and predicate symbols. A tableau is *closed* if every of its branches is *closed*, i.e. contains ϕ and $\phi*$ for some formula.

By a straightforward argument, similar to the one discussed in [KV95], we can show that a Σ_1-induction axiom can be *eliminated* by replacing it with a Σ_1-induction rule. Vice versa, every Σ_1 induction rule can be replaced by a Σ_1-induction axiom. This means that the CL proof system is equivalent to any complete proof system with $I\Sigma_1$ axioms, i.e. we have $I\Sigma_1 \vdash \phi$ iff there is a closed tableau for $\phi*$.

By the standard *cut elimination* argument adapted to tableaux (see [KV95]) we can eliminate from a closed tableau all cuts except the ones *anchored* to an induction rule, i.e. such where the cut formula $\phi*$ is the premise of an induction rule.

4.14 **Weak Π_2-induction rule.** All tableau expansion rules introduced in Par. 4.13 are local in the sense that one expansion rule does not interfere with another one. The following rules are not such. Let $\phi(z,\overline{x},\overline{y}) \in \Sigma_0$. The *weak Π_2-induction rule for the formula* $\forall\overline{x}\exists\overline{y}\phi(z,\overline{x},\overline{y})$ is

$$\frac{\exists\overline{y}\phi(s,\overline{t},\overline{y})*}{\exists\overline{y}\phi(0,\overline{u},\overline{y})* \mid \begin{array}{c}\forall\overline{x}\exists\overline{y}\phi(z,\overline{x},\overline{y})\\ \exists\overline{y}\phi(z+1,\overline{u},\overline{y})*\end{array}}$$

where z and $\overline{u}$ are eigen-variables. We attach an additional non-local condition to this rule because otherwise the rule would be equivalent to the corresponding Π_2-induction axiom. The condition is that every tableau expansion by a weak Π_2-induction rule as well as **every** expansion of an induction rule below it (including Σ_1-induction rules) may have at most *weak side formulas*. We say that an induction rule has a *side formula* ϕ if ϕ appears in the branch above the conclusion and it is a premise of a rule used below this conclusion. *Weak* side formulas are Σ_1-formulas in goals, Π_1-formulas in assumptions and Σ_0-formulas anywhere.

4.15 **Theorem** *(Elimination of weak Π_2-induction rules). Theorems proved by tableaux with weak Π_2-induction rules are provable in $I\Sigma_1$.*

Proof. In the tableau proof p of the following Witnessing lemma we may assume without loss of generality that the bounded quantifiers are eliminated by going into a suitable extension, that adjacent quantifiers of the same kind are contracted by pairing, that the cuts are anchored to induction, i.e. they are with Σ_1-formulas, and that all axioms are open. All formulas named by Greek letters

can be thus assumed to be open. The idea of the proof is similar to the very terse outline given in [Bus94] but it is done here in more detail because unlike [Bus94] we have the necessary primitive recursive apparatus developed. In the Witnessing lemma we show just one weak side formula corresponding to a Σ_1-goal, remaining side formulas are indicated by $\cdots$ but the reader should bear in mind that the witness found for the shown side formula is to be found also for every not shown quantified formula in exactly the same way.

Witnessing lemma: *If p is a closed tableau with weak Π_2-induction rules for assumptions and goals corresponding to the sequent*

$$\forall x \exists y \phi_1(\overline{w}_1, x, y) \rightarrow \cdots \exists v \psi(\overline{w}, v) \cdots \ , \tag{1}$$

where all free variables are indicated and $\overline{w}_1$ is a subsequence of $\overline{w}$, then a suitable recursive extension of $I\Sigma_1$ proves for a new function symbol f_1:

$$\forall x \phi_1(\overline{w}_1, x, f_1(\overline{w}_1, x)) \rightarrow \cdots \psi(\overline{w}, t(f_1(\overline{w}_1, \cdot), \overline{w})) \cdots \ . \tag{2}$$

The proof is similar to the proof of lemma needed for the $I\Sigma_1$ witnessing theorem and it is by induction on the construction of p. We do just two new cases. If the first expansion in p is by the instantiation $\exists y \phi_1(\overline{w}_1, s, y)$ of the Π_2-assumption in (1) then we may assume without loss of generality that the next expansion introduces the eigen-variable y: $\phi(\overline{w}_1, s, y)$. We use IH with the last formula as a new assumption and y as a new parameter and so a suitable extension of $I\Sigma_1$ proves

$$\forall x \phi_1(\overline{w}_1, x, f_1(\overline{w}_1, x)) \wedge \phi_1(\overline{w}_1, s, y) \rightarrow \cdots \psi(\overline{w}, t_1(f_1(\overline{w}_1, \cdot), \overline{w}, y)) \cdots \ .$$

By setting y to $f_1(\overline{w}_1, s)$ we may discharge the second assumption and get (2).

Let the first expansion in p be by a weak Π_2-induction rule for the formula $\forall x \exists y \phi(\overline{w}, z, x, y)$ with the premise $\exists y \phi(\overline{w}, s_1(\overline{w}), s_2(\overline{w}), y)$. Without loss of generality we may assume that the premise is not used in p and so we remove it from the side formulas. We also remove the Π_2-assumption $\forall x \exists y \phi_1$ because it cannot be used in p. For the base-case branch of p we add a new goal $\exists y \phi(\overline{w}, 0, x, y)$ with the eigen-variable x. A suitable extension of $I\Sigma_1$ proves by IH:

$$\phi(\overline{w}, 0, x, s_3(\overline{w}, x)) \vee \cdots \psi(\overline{w}, t_1(\overline{w}, x)) \cdots \ .$$

For the inductive-case branch of p we add a new Π_2-assumption $\forall x \exists y \phi(\overline{w}, z, x, y)$ and a new goal $\exists y \phi(\overline{w}, z+1, x, y)$ for two eigen-variables z and x. A suitable extension of $I\Sigma_1$ proves by IH:

$$\forall x \phi(\overline{w}, z, x, f(\overline{w}, z, x)) \rightarrow \\ \phi(\overline{w}, z+1, x, s_4(f(\overline{w}, z, \cdot), \overline{w}, z, x)) \vee \cdots \psi(\overline{w}, t_2(\overline{w}, z, x)) \cdots \ . \tag{3}$$

We extend the theory unifying both theories from IH by f defined by nested simple recursion:

$f(\overline{w}, 0, x) = s_3(\overline{w}, x)$
$f(\overline{w}, z+1, x) = s_4(f(\overline{w}, z, \cdot), \overline{w}, z, x)$.

This is a slight misuse of notation because we use the same symbol f as the one in (3) but nothing happens because we now substitute in it the new symbol for the old one. Formula (3) is now Σ_1 in the current extension and by the Witnessing theorem for $I\Sigma_1$ we get for a function $h(\overline{w}, z, x)$

$$\phi(\overline{w}, z, h(\overline{w}, z, x), f(\overline{w}, z, h(\overline{w}, z, x))) \rightarrow \\ \phi(\overline{w}, z+1, x, s_4(f(\overline{w}, z, \cdot), \overline{w}, z, x)) \vee \cdots \psi(\overline{w}, t_2(\overline{w}, z, x)) \cdots$$

We define by (un)nested simple recursion the function $g(\overline{w}, z, x)$:

$g(\overline{w}, 0, x) = t_1(\overline{w}, x)$
$g(\overline{w}, z+1, x) = t_2(\overline{w}, z, x) \leftarrow \phi(\overline{w}, z, h(\overline{w}, z, x), f(\overline{w}, z, h(\overline{w}, z, x)))$
$g(\overline{w}, z+1, x) = g(\overline{w}, z, h(\overline{w}, z, x)) \leftarrow \neg\phi(\overline{w}, z, h(\overline{w}, z, x), f(\overline{w}, z, h(\overline{w}, z, x)))$.

By a not too difficult Π_1-induction on z we prove

$$\forall x\, (\phi(\overline{w}, z, x, f(\overline{w}, z, x)) \vee \cdots \psi(\overline{w}, g(\overline{w}, z, x)) \cdots) \ .$$

We now substitute s_1 for z and s_2 for x and get (2). This ends the proof of the Witnessing lemma.

Now, given a closed tableau p with weak Π_2-induction rules, we replace all top uses of such rules by tableaux without the Π_2-rules. Each such tableau is obtained from the Witnessing lemma without a Π_2 assumption but with all weak side formulas used in the corresponding Π_2-rule. □

4.16 **Induction with measure.** The schema of recursion with measure 3.1(1) calls for the corresponding schema of *induction with measure*:

$$\forall x\, (\forall y(m(y) < m(x) \rightarrow \phi(y)) \rightarrow \phi(x)) \rightarrow \forall x \phi(x) \ .$$

We just state here without a proof that this can be reduced into a Π_1-induction formula for $\phi \in \Pi_1$ and into a weak Π_2-induction rule for $\phi \in \Pi_2$.

5 Conclusion

We have hopefully demonstrated that CL is a simple language and proof system with natural semantics in $\mathbb{N}$. In this paper we have presented only the most simplest form of CL. The full language contains many built-in functions and automatically offers derived induction schemas like the structural induction (see Par. 2.2).

Our two year's experience with teaching CL shows that the first and second year students have no problems defining functions in CL and seem to enjoy doing the formal proofs of their properties. On the other hand, the characterization of CL requires deep mathematical knowledge and relies on two almost forgotten theorems of logic. Obviously, we do not teach the characterization in the introductory courses.

The interested reader will find in our home-page the executable file of a PC implementation of CL as well as a partially finished text on CL which will eventually contain in detail the material of this paper. We are also preparing lecture notes for the courses we teach with CL.

References

Bus94. S. R. Buss. The witness function method and provably recursive functions of Peano arithmetic. In D. Prawitz, B. Skyrms, and D. Westerstahl, editors, *Logic, Methodology and Philosophy of Science IX, 1991*. North-Holland, 1994.

Cob65. A. Cobham. The intristic computational difficulty of functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science II*, pages 24–30. North-Holland, 1965.

Col91. L. Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83(1):57–69, 1991.

Dav85. M. Davis. *Computability and Unsolvability*. McGraw-Hill, second edition, 1985.

HP93. P. Hájek and P. Pudlák. *Metamathematics of First-Order Arithmetic*. Springer Verlag, 1993.

Kal43. L. Kalmár. A simple example of an undecidable arithmetical problem (Hungarian with German abstract). *Matematikai és Fizikai Lapok*, 50:1–23, 1943.

Kre52. G. Kreisel. On the interpretation of non-finitist proofs II. *Journal of Symbolic Logic*, 17:43–58, 1952.

KV95. J. Komara and P. J. Voda. Syntactic reduction of predicate tableaux to propositional tableaux. In P. Baumgartner, R. Haehnle, and J. Posegga, editors, *Proceedings of TABLEAUX '95*, number 918 in LNAI, pages 231–246. Springer Verlag, 1995.

Min73. G. Mints. Quantifier-free and one-quantifier systems. *Journal of Soviet Mathematics*, 1:71–84, 1973.

Par70. C. Parsons. On a number-theoretic choice schema and its relation to induction. In *Intuitionism and Proof Theory: proceedings of the summer conference at Buffalo N.Y. 1968*, pages 459–473. North-Holland, 1970.

Pét32. R. Péter. Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion. *Mathematische Annalen*, 110:612–632, 1932.

Pét67. R. Péter. *Recursive Functions*. Academic Press, 1967.

Ros82. H. E. Rose. *Subrecursion: Functions and Hierarchies*. Number 9 in Oxford Logic Guides. Clarendon Press, Oxford, 1982.

Sie91. W. Sieg. Herbrand analyses. *Archive for Mathematical Logic*, 30:409–441, 1991.

Smu68. R. Smullyan. *First Order Logic*. Springer Verlag, 1968.

Vod94. P. J. Voda. Subrecursion as a basis for a feasible programming language. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, number 933 in LNCS, pages 324–338. Springer Verlag, 1994.

Wai97. S. S. Wainer. Basic proof theory and applications to computation. In H. Schwichtenberg, editor, *Logic of Computation*, volume 157 of *Series F: Computer and Systems Sciences*, pages 349–394. NATO Advanced Study Institute, International Summer School held in Marktoberdorf, Germany, July 25–August 6, 1995, Springer Verlag, 1997.

Kripke, Belnap, Urquhart and Relevant Decidability & Complexity

"Das ist nicht Mathematik. Das ist Theologie."

Jacques Riche[1] and Robert K. Meyer[2]

[1] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
riche@cs.kuleuven.ac.be
[2] Automated Reasoning Project, RSISE, Australian National University, Australia
rkm@arp.anu.edu.au

Abstract. The first philosophically motivated sentential logics to be proved *undecidable* were relevant logics like **R** and **E**. But we deal here with important decidable fragments thereof, like $\mathbf{R}_{\rightarrow}$. Their decidability rests on S. Kripke's gentzenizations, together with his central combinatorial lemma. Kripke's lemma has a long history and was reinvented several times. It turns out equivalent to and a consequence of Dickson's lemma in number theory, with antecedents in Hilbert's basis theorem. This lemma has been used in several forms and in various fields. For example, Dickson's lemma guarantees termination of Buchberger's algorithm that computes the Gröbner bases of polynomial ideals. In logic, Kripke's lemma is used in decision proofs of some substructural logics with contraction. Our preferred form here of Dickson-Kripke is the Infinite Division Principle (IDP). We present our proof of IDP and its use in proving the finite model property for $\mathbf{R}_{\rightarrow}$.

1 Introduction

The Patron Saint of the Logicians Liberation League is Alasdair Urquhart, whose SECOND miracle was his proof that the major relevant logics of [AB75,ABD92] are *undecidable*. But our chief concerns in this paper are pre-Urquhart. We shall examine decidable fragments of the major relevant logics (especially **R**) and dwell on the combinatorics underlying their decision procedures, mainly the Dickson's, Kripke's and Meyer's lemmas and their ancestor, Hilbert's finite basis theorem. The main proofs in this paper are that of the IDP and the finite model property for some decidable fragments. We shall also mention briefly Urquhart's further results on the computational complexity of the decision procedures based on such principles.

The FIRST substructural logic was the system $\mathbf{R}_{\rightarrow}$ of pure relevant implication.[1] And no sooner had $\mathbf{R}_{\rightarrow}$ been seriously proposed than the question of

[1] $\mathbf{R}_{\rightarrow}$ was introduced by Moh-Shaw-Kwei and by Church circa 1950, e.g. in [Ch51]. But Došen [Do92] dug up [Or28], which basically already had relevant arrow, negation and necessity.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 224–240, 1999.

its decidability arose. Indeed, Curry's early review [CuC53] with Craig of [Ch51] claimed (mistakenly) that decidability of $\mathbf{R}_{\rightarrow}$ was straightforward. But this was *not* confirmed by Belnap, when he successfully accepted the challenge of Robert Feys to "Gentzenize" the kindred pure calculus $\mathbf{E}_{\rightarrow}$ of entailment. For while Belnap did at that time invent the "merge" consecution calculi for various relevant logics (see [AB75]), the usual (trivial?) consequences (such as decidability) of successful Gentzenization were NOT forthcoming.

Enter then young Saul Kripke, who interested himself in these decision problems at about this time (late 1950's). He just dropped the weakening rule *K from a Gentzenization $\mathbf{LJ}_{\rightarrow}$ for pure intuitionist implication, producing one $\mathbf{LR}_{\rightarrow}$ for $\mathbf{R}_{\rightarrow}$.[2] But the usual moves from Kripke's $\mathbf{LR}_{\rightarrow}$ *failed* to produce decidability for $\mathbf{R}_{\rightarrow}$. Clearly some stronger medicine was in order. We look in succeeding sections at that medicine.

2 Gentzen and Hilbert Systems.

2.1 Gentzenizing $\mathbf{R}_{\rightarrow}$ at first

Let our first formulation $\mathbf{LR1}_{\rightarrow}$ of $\mathbf{LR}_{\rightarrow}$ be as follows:[3] We adopt the usage of [AB75] in speaking of *consecutions*, which are of the form

$$\alpha \vdash A \tag{1}$$

where α is a *multiset* of formulas and A is a formula.[4] Axioms are all formulas

(AX) $\quad A \vdash A$

While the classical and intuitionist rule of weakening *K has been chopped from among the rules,[5] the structural rule *W of contraction remains.

(*W) $\quad \alpha, A, A \vdash C \Longrightarrow \alpha, A \vdash C$

We state for the record what the rule *C of permutation would look like (had we chosen sequences rather than multisets as the data type of α).

[2] We follow the conventions of [Cu63] in naming rules. And we present 2 formulations of $\mathbf{LR}_{\rightarrow}$ in the next section.

[3] We also follow [Cu63] on matters notational. I.e., we use $\vdash$ as our Gentzen predicate and commas for multiset union; we associate equal connectives to the LEFT. This frees $\Longrightarrow$ for another purpose, whence we write things like "P and $Q \Longrightarrow R$" to indicate the RULE that has P and Q as premisses and R as conclusion.

[4] It is possible to follow Gentzen and Curry in taking α as a *sequence* of formulas. But in view of the rule *C of permutation for sequences it is more elegant to choose a data type that builds in *C. Multisets fill the bill, as we suggested with McRobbie in [MM82] and implemented with him and Thistlewaite in [TMM88].

[5] *K is the rule $\alpha \vdash C \Longrightarrow \alpha, A \vdash C$.

(*C) $\alpha, A, B, \beta \vdash C \Longrightarrow \alpha, B, A, \beta \vdash C$

Operational rules *→* introduce → on left and right respectively.

(*→) $\beta \vdash A$ and $\alpha, B \vdash C \Longrightarrow \alpha, A \to B, \beta \vdash C$

(→ *) $\alpha, A \vdash B \Longrightarrow \alpha \vdash A \to B$

The VALID CONSECUTIONS of $\mathbf{LR}_{\to}$ are just those that may be obtained from (AX) using rules from among *W and *→*. The following CUT rule is moreover admissible:

(CUT) $\beta \vdash A$ and $\alpha, A \vdash C \Longrightarrow \alpha, \beta \vdash C$

Using CUT, it is straightforward to establish the equivalence of $\mathbf{LR}_{\to}$ and the Hilbert-style axiomatic version of $\mathbf{R}_{\to}$ set out in [Ch51] and below.

Alas, it is also easy to see why decidability remains a problem. For consider a candidate consecution, say $A \vdash C$. Of course this could have come by *W from $A, A \vdash C$, and, continuing in this vein, we are clearly "off to the races".

If we are to get a decision procedure for $\mathbf{LR}_{\to}$, some combinatorial insight is in order. But we are pleased nonetheless by Kripke's invention of $\mathbf{LR1}_{\to}$, which we take to agree with the philosophical motivation for dropping the "thinning" rule *K. The mate of *K in the Hilbert-style axiomatization of $\mathbf{R}_{\to}$ is the Positive Paradox axiom A7 below.

2.2 Automating the search for an *associative* operation.

One main early use of theorem proving techniques in relevant logics was related to the decision problem for **R**, a logically important, technically difficult and long open problem. The full system **LR**, closely related to **R**, was known to be decidable. There were reasons to think that **R** was undecidable and there were reasons to think that a theorem prover based on **LR** could help to establish this negative solution for the decision problem of **R**.

For more on relevant automated reasoning, see [TMM88].[6] The leading idea for showing **R** undecidable relied on a technique suggested by Meyer for coding the word problem for semigroups into **R**. The hope was that, since the former is undecidable, so also is the latter.[7] Translating the semigroup operations into the **R** connectives, the problem reduced to finding a binary connective $\oplus$ corresponding to semigroup multiplication which is *freely* associative. That is, the

[6] This book is based on the ANU Ph.D. thesis of Paul Thistlewaite, in which with McRobbie and Meyer he describes his computer program *KRIPKE*. The program was so named by McRobbie, because it implements Kripke's decision procedure for $\mathbf{R}_{\to}$ and other systems related to **LR**.

[7] The hope has *paid off*, though not quite as envisaged. For the argument of [Ur84] eventually finds an undecidable semigroup hidden in **R**. But Urquhart found it *more deeply* hidden than Meyer ever supposed.

subalgebra of the Lindenbaum algebra of **R** generated by the first n (congruence classes of) propositional variables under $\oplus$ was to be the *free* semigroup with n generators. Of course the $\oplus$ thus found cannot be commutative nor idempotent, nor have any other special properties beyond associativity.
Possible definitions of such a connective were generated. Those proved in decidable **LR** could not define the free connective wanted for **R**. The remaining candidates had then to be checked to see whether they define an associative connective in **R**. The theorem prover *KRIPKE* had already proved associative 16 candidate definitions of such a connective in **LR**. But then A. Urquhart's proof of undecidability of **R** suspended this specific research and made of **R**, as Urquhart notes, the first independently motivated undecidable propositional logic.
Nevertheless, the technical problem of defining an associative connective (i.e. to automatically prove associativity of all candidates in **LR** and **R**) still remains a challenge for the automated theorem proving community. Granted, it is important not to miss the *point* of the exercise. But Urquhart, who did see its point, continues to hold it an interesting question whether a freely associative operation is definable in **R**. He, and we, do not know.

2.3 Tuning $\mathbf{LR}_{\rightarrow}$ for decidability.

When its authors put together [BW65], they extended to the system with negation work that Kripke had done for the pure system $\mathbf{E}_{\rightarrow}$. While this extension is itself non-trivial,[8] the hardest and most imaginative part of [BW65] seems to us the work reported in [Kr59].

We shall mainly follow the Kripke-Belnap-Wallace strategy here, "building *W into the rules" to set out a consecution calculus $\mathbf{LR2}_{\rightarrow}$, with the same theorems as $\mathbf{LR1}_{\rightarrow}$. This is the weather that we know how to control, along the lines that we set out with Thistlewaite and McRobbie in [TMM88].

$\mathbf{LR2}_{\rightarrow}$ is our second formulation of $\mathbf{LR}_{\rightarrow}$. It will have NO structural rules. Though in order to ensure that $\mathbf{LR2}_{\rightarrow}$ will have the same theorems as $\mathbf{LR1}_{\rightarrow}$ (and so of $\mathbf{R}_{\rightarrow}$, on interpretation), clearly *W will have to be *admissible*. First, we leave the axioms and the rule $\rightarrow^*$ as they were in $\mathbf{LR1}_{\rightarrow}$. In compensation we complicate $*\rightarrow$. The clear invertibility of $\rightarrow^*$ enables us to insist that the *succedent C of the conclusion* of $*\rightarrow$ shall be a *propositional variable p* in $\mathbf{LR2}_{\rightarrow}$.[9] Then we state the rule for building in contraction with a care that we hope is optimal. A useful theft from [TMM88] is a counting function c, which keeps track of the *number of occurrences* of each formula D in the multiset γ. I.e.,

Unit: c(D;D) = 1.
Zero: If D does not occur in γ, c(D;γ) = 0.

[8] It involves what Meyer dubs "Belnap's Crazy Interpretation Theorem," in order to show that the Gentzen system is in an appropriate sense a subsystem of the Hilbert system for $\mathbf{E}_{\rightarrow}$.

[9] We borrow this trick from Kit Fine, who used in another context with E. P. Martin something similar.

Union: Let γ be the *multiset union* of γ_1 and γ_2. Then c(D;γ) = c(D;γ_1) + c(D;γ_2).

Now we build in *W *very carefully* for maximal efficiency into *$\rightarrow$. So here is the rule for $\mathbf{LR2}_{\rightarrow}$.

$$(*\rightarrow) \quad \alpha \vdash A \text{ and } \beta, B \vdash p \Longrightarrow \gamma, A \rightarrow B \vdash p,$$

where p is a propositional variable and the multiset γ is subject to the following constraints:[10]
We require for each formula D, if D $\neq A \rightarrow B$,

1. c(D;γ) = 0 iff c(D;α) = 0 and c(D;β) = 0
2. if c(D;γ) = 1 then either
   (a) c(D;α) = 0 and c(D;β) = 1, or
   (b) c(D;α) = 1 and c(D;β) = 0, or
   (c) c(D;α) = 1 and c(D:β) = 1
3. if c(D;γ) > 1 then c(D;γ) = c(D;α) + c(D;β)

For the particular formula $A \rightarrow B$, if c(A$\rightarrow$B;γ) = 0 then either

1. c(A$\rightarrow$B;α) = 0 and c(A$\rightarrow$B;β) = 0, or
2. c(A$\rightarrow$B;α) = 1 and c(A$\rightarrow$B;β) = 0, or
3. c(A$\rightarrow$B;α) = 0 and c(A$\rightarrow$B;β) = 1, or
4. c(A$\rightarrow$B;α) = 1 and c(A$\rightarrow$B;β) = 1

To be sure, A$\rightarrow$B may occur PROPERLY in the multiset γ which is part of the CONCLUSION of the revised *$\rightarrow$ rule of $\mathbf{LR2}_{\rightarrow}$. But it is then one of the formulas D already considered above, and is subject to the corresponding constraints there laid down.[11]

2.4 Relevant Hilbert systems.

We are mainly interested in the Relevant system $\mathbf{R}_{\rightarrow}$. But we also mention some other *pure implicational* fragments of Relevant (and related) Logics. These may be axiomatized as follows. (The letter following the name of an axiom is its corresponding combinator.) $\mathbf{T}_{\rightarrow}$: $A1 - A4$; $\mathbf{E}_{\rightarrow}$: $A1 - A5$; $\mathbf{R}_{\rightarrow}$: $A1 - A6$. $\mathbf{J}_{\rightarrow}$: $A1 - A7$. **BCK** and **BCI** (using combinatory axioms B, C, K and I to characterize them)

[10] The purpose of these constraints is to render *W admissible while stoutly resisting *K.

[11] Note that we are here treating *$\rightarrow$ in $\mathbf{LR2}_{\rightarrow}$ EXACTLY ANALOGOUSLY to how Po'' is treated on p. 53ff. of [TMM88]. So the proof that exactly the same consecutions $\alpha \vdash A$ are provable in $\mathbf{LR1}_{\rightarrow}$and $\mathbf{LR2}_{\rightarrow}$ will be analogous as well to Thistlewaite's proof in [TMM88] that the formulations $L1$ and $L5$ of **LR** are deductively equivalent.

are other well known systems.[12]

Axioms.

A1.	$A \to A$	Identity (I)
A2.	$(A \to B) \to ((B \to C) \to (A \to C))$	Suffixing (B')
A3.	$(A \to B) \to ((C \to A) \to (C \to B))$	Prefixing (B)
A4.	$(A \to (A \to B)) \to (A \to B)$	Contraction (W)
A5.	$((A \to A) \to B) \to B$	Specialized assertion (CII)
A6.	$(A \to (B \to C)) \to (B \to (A \to C))$	Permutation (C)
A7.	$A \to (B \to A)$	Positive Paradox (K)

and the deduction rule schema:

R1. $\vdash A \to B$ and $\vdash A \implies \vdash B$ Modus Ponens

The axiomatic and consecution formulations of the system $\mathbf{R}_{\to}$ are equivalent. A formula is provable in the Gentzen system iff it is provable in the axiomatic system.
Note that the consecution formulations of the systems $\mathbf{E}_{\to}$ and $\mathbf{T}_{\to}$ impose some restrictions on the rules.

3 Decidability.

As we have indicated, the decision problem for the implicational fragments of **E** and **R** is difficult.[13] We briefly summarize the main steps of the procedure, which were worked out via a "combinatorial lemma" and that underlay the claims of [Kr59]. See also [BW65], [AB75], [Du86], [TMM88].

Suppose that a given formula A is to be proved. Applying the rules (in reverse), a proof search tree for $\vdash A$ is constructed in $\mathbf{LR2}_{\to}$. The basic idea of this formulation of $\mathbf{LR}_{\to}$ is to restrict the application of the contraction rule.

[12] **BCI** corresponds to the implicational fragment of Linear Logic. And we note for the reader's logical pleasure that he/she/it may achieve fame and fortune by solving the *decision problem* for $\mathbf{T}_{\to}$. Having been around since *circa* 1960, this is the most venerable problem in all of relevant logic. And a clue would appear to lie in the principle of commutation of antecedents. $\mathbf{R}_{\to}$, which admits the principle, is decidable; even $\mathbf{E}_{\to}$ admits *enough* commutativity to be decidable. $\mathbf{T}_{\to}$ would seem to have *some* useful permutation of antecedents; e.g., the B and B' axioms result from each other by permutation. But these facts have not yet proved enough to apply to $\mathbf{T}_{\to}$ the Kripke methods that work for other pure relevant logics.

[13] There are *simpler* related problems. For example, the first-degree entailment fragment in which the formulas are of type $A \to B$, where A, B are of degree zero, has a simple decision procedure: there is a four-valued Smiley matrix which is characteristic for the fragment. The zero-degree fragment of **R** and **E** is strictly equivalent to classical propositional logic with truth-values as characteristic model.

And we say that a multiset β *reduces* to γ if γ can be obtained from β by a series of applications of the rule of contraction. Similarly we may say that a consecution $\beta \vdash B$ reduces to one $\gamma \vdash B$ iff β reduces to γ. We have so formulated $\mathbf{LR2}_{\rightarrow}$ that the following lemma of [Cu50] is provable:

Lemma 1 (Curry, 1950). *Suppose a consecution $\beta \vdash C$ reduces to $\gamma \vdash C$, where the former is provable in n steps. Then there is a number $m \leq n$ such that the latter is provable in m steps.*

Let us now follow [BW65] (who were adapting Kripke) in calling a derivation of a consecution $\beta \vdash B$ *irredundant* if no earlier step in its proof tree reduces to any later step. We then have

Theorem 1. *Every provable consecution of* $\mathbf{LR2}_{\rightarrow}$ *has an irredundant derivation.*

Proof. Proof by Curry's lemma.

This clears the ground for Kripke's beautiful combinatorial lemma, which he worked out as a lad. Call two multisets *cognate* if the same *wffs* occur in both of them. And call the *cognation class* of a multiset the class of multisets that are cognate to it. By the subformula property, any formula occurring in the derivation of a given consecution is a subformula of some formula in the conclusion of that derivation. Upshot: only finitely many cognation classes have members in a proof search tree for an $\mathbf{LR}_{\rightarrow}$ consecution. Further upshot: every irredundant $\mathbf{LR2}_{\rightarrow}$ proof search is *finite*. For, by Kripke's lemma (K)

Lemma 2 (Kripke 1959). *Every irredundant sequence of cognate consecutions is finite.*

Proof. Proof by transposition. Suppose that there exists an infinite sequence $\alpha_i \vdash B$ of cognate consecutions. Show by induction on the number of formulas in the (common) cognation class of the α_i that the sequence is redundant; whence, contraposing, any cognate irredundant sequence must be finite.

Meyer had a bit of a problem at this point. He *knew* that the conclusion was true, because Belnap had shown his fellow students and him Kripke's sound argument. But he *did not believe* it. Visions of irredundant infinite cognate sequences fluttered through his dreams. And so he wanted an argument that he did believe.

More of that below! Meanwhile let us draw the immediate conclusion.

Theorem 2 (Kripke 1959). $\mathbf{R}_{\rightarrow}$ *is decidable. So is* $\mathbf{E}_{\rightarrow}$.

Proof. We omit all arguments for $\mathbf{E}_{\rightarrow}$. The interested reader may look up valid ones, e.g., in [AB75]).[14] For $\mathbf{R}_{\rightarrow}$ we argue as follows: construct a proof search tree

[14] It seems that Kripke wrote up his original argument re $\mathbf{E}_{\rightarrow}$ at most in private communications to Anderson and Belnap. In addition to Kripke's Lemma, he had an *interpretation* of his Gentzen system for $\mathbf{E}_{\rightarrow}$ in the axiomatic formulation which, he has written to us, "is quite straightforward". He told us that Dunn has gone through something similar. And he pronounced it "marginally simpler than Belnap's approach, though his is not too complicated."

for the theorem candidate $\vdash A$ in $\mathbf{LR2}_{\rightarrow}$. Only the finitely many subformulas of A are relevant to the (cut-free) proof search; and, by Curry, we may keep that search irredundant. Suppose (curse our luck!) that the proof search tree is infinite. Since the tree has the finite fork property (inasmuch as there are only finitely many things to try to deduce a candidate consecution at any point in the search tree), it follows by König's lemma that there must be some infinite branch. But there are only finitely many cognation classes whose representatives may appear on the bad branch. Accordingly there is an infinite irredundant subbranch of *cognate* $\alpha_i \vdash B$. This contradicts Kripke's lemma, and refutes the hypothesis that we must curse our luck. For there are but finitely many nodes in our proof search tree for the theorem candidate. End of proof.

3.1 Relevant Divisibility.

Decidability is guaranteed by Kripke's lemma, but this lemma itself is not immediately obvious. [Me73,Me98] summarize the main results obtained in trying to fully understand S. Kripke's insights. One of them is an equivalent formulation of Kripke's lemma in terms of a *relevant divisibility principle*.

The "use" criterion for relevance in Relevant Logics says that in a deduction the premisses must be effectively used in the derivation of a conclusion. Let us now consider ordinary divisibility. *Obviously* it involves fallacies of relevance. For consider the suspicious claim that 2 divides 12. There is a factor of 12, namely 3, which is *not used* in performing the division. We rest for now with this observation that ordinary divisibility has a *less than upright character*, whence we slantily write "/" for it thus:

$$m/n \text{ iff } m \text{ divides } n \tag{2}$$

Relevant divisibility | will by contrast have an *upright* character.[15] We define first a function *primeset*, whose application to any positive integer n yields the *set* of the prime divisors of n.
For example, $primeset(2) = \{2\}$ but $primeset(6) = primeset(12) = \{2, 3\}$. Then our definition of | will go like this:

$$m|n \text{ iff } (i)\ m/n \text{ and } (ii)\ primeset(m) = primeset(n) \tag{3}$$

We adapt Kripke's terminology by introducing the following

Definition. We call m and n *cognate* iff $primeset(m) = primeset(n)$.

And it is now clear that ordinary and relevant division coincide on cognate pairs of numbers. Moreover, since a division performed with unused factors is as reprehensible as an argument with unused premisses, we trust that all good relevantists and true will stick henceforth to *vertebrate, relevant* division.

[15] We confess that we have *reversed* the conventions of [Me98], on which | was ordinary and / was relevant divisibility.

4 Kripke, Dickson and Meyer Lemmas.

Let $\mathbb{N}_n$ be the free commutative monoid generated by the first n primes. Then, **Kripke's lemma** (K) can be formulated in the following way:
Let a_i be any sequence of members of $\mathbb{N}_n$ and suppose that for all i, j, if $i < j$ then $a_i \not| a_j$. Then a_i is finite.

We prove (K) in the form of the **Infinite Division Principle** (IDP):

Lemma 3 (Meyer). *Let A_n be any infinite subset of $\mathbb{N}_n$. Then there is an infinite subset A'_n of A_n and a member a of A'_n s.t. for all $b \in A'_n$, $a|b$.*

Proof. We call the element a which divides infinitely many members of A an *infinite divisor for* A. We prove that every infinite A has an infinite divisor by induction on n.

Base case: If $n = 1$, we have an isomorphic copy of $\mathbb{N}$ with the natural order. Every non-empty subset $A \subseteq \mathbb{N}$ has a least element m, which (reversing the isomorphism) will divide all other elements of A.

Inductive case: Suppose then that the lemma holds for all $m < n$. We show that it holds for n.
Let A_n be an infinite subset of $\mathbb{N}_n$. Partition A_n into *cognation classes* by setting
(i) $c \sim d$ iff $primeset(c) = primeset(d)$. There are only finitely many equivalence classes under (i) since there is exactly one such class for each subset of the set of the first n primes. A_n being infinite, one of the classes is infinite. Call it B_n.

Subcase 1: B_n is *bounded by* x on some coordinate m.[16] Let m be the coordinate in question, and let x be the least bound. W.l.o.g. we may take m to be n. Then B_n is the *union* of the *finitely many* cognation classes $B^0_n, \ldots, B^x_n$, whose values on the n^{th} coordinate are respectively $0, \ldots, x$. Since the infinite set B_n is the union of finitely many sets, one of these sets B^i_n is itself infinite. On inductive hypothesis this set, and hence B_n, and hence A_n has an infinite divisor.

Subcase 2: B_n is unbounded on every coordinate m. Let $h = \mathbb{N}_n \longrightarrow \mathbb{N}_{n-1}$ be the *truncation* function defined on $\mathbb{N}_n$ with values in $\mathbb{N}_{n-1}$. In particular, let $h(A_n) = A_{n-1}$. A_{n-1} is either finite or infinite. But since $n > 1$ it cannot be finite (or else we would be in Subcase 1). So A_{n-1} is infinite. Apply the inductive hypothesis. There is an infinite subset $B_{n-1} \subseteq A_{n-1}$ and an $a' \in B_{n-1}$ such that $a'|b', \forall b' \in B_{n-1}$.
Let $B_n = \{c : c \in A_n \text{ and } h(c) \in B_{n-1}\}$.
B_n is infinite since all elements in B_{n-1} are images under h of at least one member of A_n. In particular, the infinite divisor a' is such an image. Pick

[16] Where $C \subseteq A_n$ we say that x bounds C on m iff for all $c \in C$ the m^{th} coordinate $c_m \leq x$. Otherwise we say that C is *unbounded* on m.

the member a'' of B_n that is least on coordinate n among the members of B_n such that $h(a) = a'$.
Is a'' an infinite divisor for B_n? If the answer is "Yes," we are through. So suppose that it is "No." At any rate we know that $a' = h(a'')$ is an infinite divisor for B_{n-1}. So since a'' has *failed* infinitely often to divide members of B_n, on our remaining hypothesis, the reason must be that its n^{th} coordinate is too great. But now we simply fall back on the strategy that has several times already proved successful, setting $C_n = \{c \in B_n : a'|h(c) \text{ and not } a''|c\}$. But the infinite set C_n is built from elements all of which are bounded by a'' on the n^{th} coordinate. This means again that there will be an infinite C_n^i for a particular value i on the last coordinate, whence the inductive hypothesis delivers the theorem in this case also, ending the proof of the IDP.

It can be shown moreover that the IDP is equivalent to K and to **Dickson's lemma** (D) stated in the following form:

Lemma 4. *Let $A_n \subseteq \mathbb{N}_n$ and suppose that for all $a, b \in A_n$, if $a|b$ then $a = b$. Then A_n is finite.*

The proof is left as an exercise.

5 Finitizing the Commutative Monoid Semantics for $\mathbf{R}_{\rightarrow}$.

Building on ideas of Urquhart, Routley, Dunn and Fine, [Me73,MO94,Me98] develop an operational semantics for $\mathbf{R}_{\rightarrow}$ in terms of *commutative monoids*. Kripke's lemma is then applied to deliver an *effective* finite model property for $\mathbf{R}_{\rightarrow}$ and some of its supersystems on that semantics. Referring readers to the cited papers for details, we summarize some main points.

Let S be the sentential variables of $\mathbf{R}_{\rightarrow}$ and F the set of formulas built from S in the usual way. Let $\mathbf{M}$ be an arbitrary partially ordered square-increasing commutative monoid.[17] A *possible interpretation* I in such an $\mathbf{M}$ is a function $I : F \times \mathbf{M} \longrightarrow \{T, F\}$ that satisfies the following truth condition: for all $A, B \in F$, and for all $a \in \mathbf{M}$, $I(A \rightarrow B, a) = T$ iff for all $x \in \mathbf{M}$, if $I(A, x) = T$ then $I(B, ax) = T$. And A is *verified* on I iff $I(A, 1) = T$.

A problem with $\mathbf{R}_{\rightarrow}$ is that not all of its theorems are verified on arbitrary interpretations. In particular, the Contraction axiom A4 above is not verified. To solve the problem, the set of possible interpretations is restricted by imposing a hereditary condition: an interpretation is *hereditary* if for all $\mathbf{M}$, for all $A \in F$ and for all $a, b \in \mathbf{M}$, if $a \leq b$ and $I(A, b) = T$ then $I(A, a) = T$.

It is shown in [Me98] that the only model you'll ever need for $\mathbf{R}_{\rightarrow}$ is the *free* one. Specifically, take this to be $\mathbb{N}_+$, the positive integers, with *primes* as free generators, 1 as monoid identity, multiplication as monoid operation and relevant division $|$ as the square-increasing ordering relation. For the completeness proof for $\mathbf{R}_{\rightarrow}$ assures that every non-theorem is refuted on an appropriate hereditary

[17] I.e., satisfying conditions (i), (ii) and (iv) that we set out in [AB75], p. 376.

interpretation I at 1 in some appropriate **M**. And there is then an equivalent I', i.e. an interpretation verifying exactly the same formulas, in the monoid $\mathbb{N}_+$, of which **M** is a homomorphic image.

But if you want the finite model property, you had better be more careful. First, to refute a given non-theorem A, freely generate a commutative monoid **M** from the subformulas of A. W.l.o.g. we may take this **M** to be $\mathbb{N}_n$, where n is the number of subformulas of A. This technique thus transforms infinitely long vectors with no finite bound into vectors of uniform length n. Hence, the property transforms the cardinality of the primes required for a refutation of a non-theorem A.

Kripke's lemma and the finite generator property suffice for decidability. But we can go further and prove the Finite Model Property.

5.1 The Finite Model Property.

Taking the monoid generated by the primes as characteristic presents another difficulty because arbitrarily high exponents are allowed on any particular prime. This is solved by placing bounds on the exponents which are relevant to a refutation of a formula A. I.e. we *shrink* $\langle \mathbb{N}^k, +, 0\rangle$ to $\mathbf{i}^k = \langle i^k, \oplus, 0\rangle$ where $i, k \in \mathbb{N}_+$. Our additive commutative monoid will be $\langle i, \oplus, 0\rangle$, where $i = \{n : 0 \leq n < i\}$ and $\oplus$ is defined as follows: for $0 \leq m, n < i$, if $m + n \geq i$ then $m \oplus n = i - 1$, otherwise, $m \oplus n = m + n$. That is, **i** is the 1-generator commutative monoid $\langle \mathbb{N}, +, 0\rangle$ bounded at $i - 1$, and the elements of i^k are the k-place sequences of natural numbers $< i$ on every coordinate. Shrinking $\mathbb{N}^k$ to $\mathbf{i}^k$ is guaranteed by the natural homomorphism $h : \mathbb{N}^k \to \mathbf{i}^k$ whose effect on the jth coordinate of each element $a \in \mathbb{N}^k$ is s.t. if $a_j \geq i, (h(a_j)) = i - 1$, else $(h(a_j)) = a_j$. In this way, the coordinates of elements that are greater than $i - 1$ are all reduced to $i - 1$.

Henceforth, by the *index* of a formula A we mean simply its number k of subformulas. And for every non-theorem A of $\mathbf{R}_\to$ there is a procedure that yields a refutation of A in $\mathbb{N}^k = \langle \mathbb{N}^k, +, 0\rangle$ or in the isomorphic $\langle \mathbb{N}_k, \cdot, 1\rangle$. Suppose now that we have a refuting interpretation I for such an A. We single out some elements $b \in \mathbb{N}^k$ as *critical* for A. Specifically, b is critical if it is a "first falsehood" for some subformula B of A. Even more specifically, suppose I fixed.

Then b is critical iff there is a subformula B of A s.t.

1. $I(B, b) = F$
2. for all $c \in \mathbb{N}^k$, if $c|b$, then $I(B, c) = T$.

That is, each subformula B of A gives rise to a family of critical elements. And b is critical iff it is *minimal* in the ordering induced by $|$ in the subset of elements of $\mathbb{N}^k$ at which B is false.

Every formula A has finitely many subformulas B. And we can then apply Dickson's lemma to assure that the non-theorem A gives rise, at most, to finitely many critical elements. And we then apply the shrinking operation to transform a refutation of A in $\mathbb{N}^k$ into a refutation of A in $\mathbf{i}^k$. So

Theorem 3 (Meyer 1973). $\mathbf{R}_{\rightarrow}$ *has the* FMP.

Proof. Let A be a non-theorem of $\mathbf{R}_{\rightarrow}$ of index k. A is refutable in $\langle \mathbb{N}^k, +, 0\rangle$ on a hereditary interpretation I. It suffices to find an interpretation I' in $\mathbf{i}^k$ refuting A and such that I' is hereditary. Apply the procedure just sketched. $\mathbf{i}^k$ being finite, A is refuted in a finite model. End of proof.

6 Partial orders.

Kripke's lemma in its number-theoretic form is about order induced by divisibility on the positive integers. The theory of partial order not only illuminates its significance, but it also guarantees the truth of the lemma and of its equivalent formulations for the free commutative monoid $\langle \mathbb{N}_k, \cdot, 1\rangle$.

Let $\langle X, \leq\rangle$ be a *partially ordered* set; i.e. the ordering relation $\leq$ is reflexive, transitive and antisymmetric.
Then, X satisfies the IDP if for all infinite $A \subseteq X$, there is a $a \in A' \subseteq A \subseteq X$ s.t. A' is infinite and for all $a' \in A'$, $a \leq a'$.

Let $A \subseteq X$. Then, A_{min}, the set of A-*minimal* elements is the set of $a \in A$ s.t. for all $y \in A$, $y \not< a$.
Then, X satisfies the *finite minimum condition* (FMC) iff, for all $A \subseteq X$, A_{min} is finite. Then, D says that, for $n \in \mathbb{N}_+$, $\langle \mathbb{N}_n, |\rangle$ and $\langle \mathbb{N}_n, /\rangle$ satisfy the FMC.

As in [Bi67] X satisfies the *Ascending Chain Condition* (ACC) iff all strictly ascending chains are finite. Dually, it satisfies the DCC iff all strictly descending chains are finite.

An alternative way of showing that $IDP \equiv D \equiv K$ and that all of these hold in the free commutative monoid $\mathbb{N}^k$ partially ordered by divisibility can be based on these definitions.[18] We first note that for *any* partially-ordered set A, the conditions "*A satisfies the IDP*" and "*A satisfies both the DCC and the FMC*" are equivalent.

7 More about Dickson's Lemma.

In addition to the Birkhoff exercise cited above, we also discovered that Higman's lemma [Hi52] states and proves some equivalent formulations of *K, D, IDP*. The property we are interested in is that of a *well-partial-ordering*.

7.1 Higman's Lemma.

There are various ways to characterise a well-partial-order or a well-quasi-order. We concentrate here on the former. Most often the associated proof techniques rely on the notion of *minimal bad sequences*.
Definition. Let $\mathbf{a} = a_1, a_2, \ldots, a_n, \ldots$ be an infinite sequence of elements of a partially ordered set A. Then, $\mathbf{a}$ is called *good* if there exist positive integers i, j such that $i < j$ and $a_i \leq a_j$. Otherwise, the sequence $\mathbf{a}$ is called *bad*.

[18] In fact, these equivalences were suggested by an exercise in Birkhoff [Bi67].

Then,

Definition. Let A be a *partially ordered* set. Then A is *well-partially-ordered*, if every infinite sequence of elements of A is good or, equivalently, if there are no infinite bad sequences. Also equivalently, A is *well-partially-ordered* if it does not contain an infinite descending chain (i.e. $a_0 > a_1 > \cdots > \cdots$), nor an infinite anti-chain (i.e. a set of pairwise incomparable elements).

To be well-partially-ordered is also equivalent to having a finite basis:

Definition. Let A be a *partially ordered* set and $B \subset A$. The *closure* of B, $Cl(B) = \{a \in A | \exists b \in B, b \leq a\}$. And B is closed iff $B = Cl(B)$. A has the *finite basis property* if every subset of A is the closure of a finite set.

All the properties expressed in the preceding definitions are proved equivalent to the well-partial-ordering or to the finite basis property in **Higman's lemma** [Hi52] :

Let A be a *partially ordered* set. Then the following conditions on A are equivalent:

1. every closed subset of A is the closure of a finite subset;
2. the ascending chain condition holds for the closed subsets of A;
3. if B is any subset of A, there is a finite set B_0 such that $B_0 \subset B \subset Cl(B_0)$
4. every infinite sequence of elements of A has an infinite ascending subsequence;
5. if $a_1, a_2, \ldots$ is an infinite sequence of elements of A, there exist integers i, j such that $i < j$ and $a_i \leq a_j$;
6. there exists neither an infinite strictly descending sequence in A (well-founded-ness), nor an infinite antichain, that is, an infinity of mutually incomparable elements of A.

7.2 Hilbert's Finite Basis and Gordan's lemma.

In [Di13], Dickson remarks that his lemma can be obtained from Hilbert's theorem [Hi90].

In its modern reading, Hilbert's theorem says that *if a ring R is Noetherian, then the polynomial ring in n commuting indeterminates $R[X_1, \ldots, X_n]$ is Noetherian*[19].

And a ring R is Noetherian if one the following conditions holds:

(i) every ideal in R is finitely generated,

(ii) any ascending chain of ideals is finite,

(iii) every ideal in R has a maximal element.

Historically, Hilbert's theorem originated in the old and now forgotten invariant theory. A short note on this theorem is not without interest since it throws some light on the ancestry of our D, K and IDP principles.

[19] A *ring* R is an additive commutative group together with an associative and distributive multiplication operation. If I is an additive subgroup of R such that for all $a \in I$, for all $r \in R$, $ar, ra \in I$, then I is an *ideal*

In the middle of the nineteenth century, Cayley had studied "quantics", i.e. homogeneous polynomials with arbitrary constant coefficients of degree n in m independent variables. Gordan proved constructively the existence of finite fundamental invariants and covariants for systems of binary quantics (see [Be45]). His theorem shows that *the number of irreducible solutions in positive integers of a system of homogeneous linear equations is finite.*

Hilbert [Hi90] gave a more general proof of the theorem which applies to any number of variables. But, unlike Gordan's proof, Hilbert's proof gives no indication as to the actual determination of the finite system. It is a mere existence proof, provoking the Gordan quotation which we have adopted as our slogan for the paper—"This is not mathematics, it is theology".

Interestingly, Gordan later found a much simplified proof of Hilbert's theorem. The proof is reproduced in Grace and Young [GY03]. We consider only **Gordan's lemma**:

Let S be a set of products of the form $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ such that the α_i satisfy some condition. Although the number of products that satisfy these conditions may be infinite, a finite number can be chosen so that every other is divisible by one at least of this finite number.

This lemma is often used in proofs of Dickson's lemma, it is even sometimes called Dickson's lemma (see [CLO92]). This is not surprising when it is compared to the original formulation of **Dickson's lemma**:

Any set S of functions of type (1), $F = a_1^{\alpha_1} a_2^{\alpha_2} \cdots a_r^{\alpha_r}$ contains a finite number of functions $F_1, \ldots, F_k$ such that each function F of S can be expressed as a product $F_i f$ where f is of the form (1) but is not necessarily in the set S.

Finally, let us note that König [Kö36](XI, §3, α) uses a lemma that expresses one of the equivalent properties of Higman's lemma: *if two elements of a set M of ordered sequences of positive integers are incomparable, then M is finite.* And he remarks in a note that Gordan's theorem follows from this lemma.

8 Complexity.

With the Dickson-Kripke lemma we have a finiteness or termination condition. But finite can be arbitrarily large. We turn to complexity results for relevant logics, which are all due to A.Urquhart [Ur90,Ur97]. He shows that the decision problem for the conjunction-implication fragment of the logics between T and R is $EXPSPACE$-hard under log-lin reducibility. This result relies on the $EXPSPACE$-completeness of the word problem for commutative semigroups.

Urquhart also shows that Kripke's decision procedure for $R_{\rightarrow}$ is, as an *upper* bound, *primitive recursive in the Ackermann function.* His result is based on the study of decision procedures for Petri Nets and vector addition systems where a lemma essentially equivalent to Kripke's lemma is used to show that the reachability tree of a vector addition system is finite. For each k in a k-dimensional vector addition system, or a k-place Petri Net, the k-finite containment problem has a primitive recursive decision procedure. And in the unbounded case, the procedure is *primitive recursive in the Ackermann function.*

Urquhart sharpens his result in [Ur97] by showing that for the decidable extension of $\mathbf{R}_{\rightarrow}$ called **LR** in [TMM88], the lower and upper bounds converge. This result suggests that the logics decided by a procedure based on *K, D, IDP* are primitive recursive in the Ackermann function.[20]

S. Kripke had conjectured that the proof that $\mathbf{R}_{\rightarrow}$ is decidable might itself be undemonstrable in elementary recursive arithmetic [TMM88]. That is, Kripke conjectured that powerful arithmetical principles were required for his demonstration that proof search trees for $\mathbf{R}_{\rightarrow}$ and its kin are invariably finite. Results originating in the program of 'Reverse Mathematics' [Si88] confirm that conjecture for *primitive* recursive arithmetic. They show that none of the properties expressed by Dickson's lemma, Higman's lemma or Hilbert's theorem are provable in primitive recursive arithmetics.[21]

9 Conclusion.

The idea of the surprising property expressed by Dickson's lemma was in the air at the turn of the century due to Gordan and Hilbert. The property reappeared from time to time. In Logic, well-quasi-orders were used by Y. Gurevich in his early work on Hilbert's *Entscheidungsproblem* to show that some classes of first-order formulae are decidable [BGG97]. In Relevant Logics, it reappeared in Kripke and in Meyer. Its interpretation in terms of infinite divisibility, its proof in the *IDP* form and its application in the decision procedure that are reported here allow one to see similarities to the theory of Gröbner bases, especially as it has been worked out via the Buchberger algorithm.

Now, Dickson's lemma and its mates may appear quite obvious to some. To us, they did not. Is the proof presented here *yet another* proof of Dickson's lemma? It is the proof that satisfies us. And not all purported proofs of the principle do so.[22] In a concluding conclusion, we return to our slogan. For after Gordan had

[20] This "suggestion" is not yet a proof. And Urquhart calls our attention in correspondence to the fact that "actually, I proved a non-primitive-recursive lower bound for all implication-conjunction logics between T and R. So for implication and conjunction, the later results are a strict improvement on the old. The only result in the old paper that is not improved in the later is that $\mathbf{R}_{\rightarrow}$ is $EXSPACE$-hard. The true complexity of $\mathbf{R}_{\rightarrow}$ is a very interesting open problem, and might be worth mentioning."

[21] It is interesting to note that primitive recursive arithmetic, a finitary and constructive system, corresponds to Hilbert's notion of finitism.

[22] For example, consider the proof of T. Becker *et al.,* in [BWK93] which tries to avoid the use of the (questionable to some) axiom of choice (AC). There, the proof of Proposition 4.49 (Dickson's lemma) does not convince us. They claim, independently of AC, that if $\langle M, \leq \rangle$ is a well-quasi-ordered set, then so also is the direct product $\langle M \times N, \leq' \rangle$, where N is the natural numbers and $\leq'$ is the product ordering. To show this, they assume that S is a non-empty subset of $\langle M \times N, \leq' \rangle$. They let, for each $n \in N$, $M_n = \{a \in M \mid \langle a, n \rangle \in S\}$. Because M is well-quasi-ordered, they deduce correctly that each M_n has a finite basis B_n. They then form $\bigcup_{n \in N} B_n$, and

adapted [Hi90] for his own purposes, he conceded that "I have convinced myself that theology also has its merits."

References

AB75. Anderson, A. R. and Belnap, N. D., *Entailment I. The Logic of Relevance and Necessity.* Princeton, Princeton Un. Press, 1975. 224, 225, 225, 229, 230, 233

ABD92. Anderson, A. R., Belnap, N. D. and Dunn, J. M., *Entailment II. The Logic of Relevance and Necessity.* Princeton, Princeton Un. Press, 1992. 224

BWK93. Becker, T., Weispfenning, V. and Kredel, H., *Gröbner Bases. A Computational Approach to Commutative Algebra,* New York, Springer-Verlage, 1993. 238

Be45. Bell, E, T., *The Development of Mathematics*, New York, McGraw-Hill, 1945. 237

BW65. N.D. Belnap, N.D. and Wallace, J.R., A Decision Procedure for the System E_I of Entailment with Negation. *Zeitschr. f. math. Logik und Grundlagen d. Math.* 11 (1965), 277-289. 227, 227, 229, 230

Bi67. Birkhoff, G., *Lattice Theory*, 3rd edition. Providence, AMS, 1967. 235, 235

BGG97. Börger, E., Grädel, E. and Gurevich, Y., *The Classical Decision Problem*, New York, Springer Verlag, 1997. 238

CLO92. Cox, D., Little, J. and O'Sheen, D., *Ideals, Varieties, and Algorithms.* New York, Springer Verlag, 1992. 237

Ch51. Church, A., The Weak Theory of Implication, In: *Kontrolliertes Denken, Untersuchungen zum Logikkalkül und zur Logik der Einzelwissenschaften*, ed., A. Menne et al., Karl Albert Verlag, 1951, pp. 22-37. 224, 225, 226, 239

Cu50. Curry, H.B., *A Theory of Formal Deducibility*, Ann Arbor, 1950. 230

Cu63. Curry, H.B., *Foundations of Mathematical Logic*, New York, 1963. 225, 225

CuC53. Curry, H.B. and Craig, W., Review of [Ch51], *J. Symb. Logic* 18 (1953) pp. 177-8. 225

Di13. Dickson, L.E., Finiteness of the Odd Perfect Primitive Abundant Numbers with n Distinct Prime Factors. *Am. J. Math.* 35 (1913) 413-422.

Do92. Došen, K., The First Axiomatization of Relevant Logic *J. Phil. Log.* 21 (1992), 339-56.

Du86. Dunn, J. M., Relevance Logic and Entailment. In: *Handbook of Philosophical Logic*, Vol.III, D. Gabbay and F. Guenthner (eds.), Dordrecht, D. Reidel, 1986, pp. 117-224. 224 229

GY03. Grace, J.H. and Young, A., *The Algebra of Invariants.* Cambridge University Press, 1903. (Repub. New York, Chelsea Pub. Comp.). 237

Hi52. Higman, G., Ordering by Divisibility in Abstract Algebras. *Proc. London. Math. Soc.* 2, (1952) 326-336. 235, 236

Hi90. Hilbert, D., Über die Theorie des algebraischen Formen. *Math. Annalen* 36 (1890) 473-534. 236, 239

again deduce correctly that there is a finite basis $C \subseteq \bigcup_{n\in N} B_n$. They let $r \in N$ be s.t. $C \subseteq \bigcup_{i=1}^{r} B_i$, and set $B = \{\langle a, i\rangle \in M \times N \mid 1 \leq i \leq r, a \in B_i\}$. They claim that B is a finite basis for S. But, according to us, they *overlook* that not all elements $\langle a, i\rangle$, $i \leq r$, need belong to S. So while B is certainly finite, it is *not necessarily* a basis for S.

Kö36. König, D., *Theorie der endlichen und unendlichen Graphen,* 1936. (Repub. New York, Chelsea Pub. Comp.) 237

Kr59. Kripke, S., The Problem of Entailment, (Abstract). *J. Symb. Logic* 24 (1959) 324. 227, 229

Me73. Meyer, R. K., Improved Decision Procedures for Pure Relevant Logics. Unpublished, 1973. 231, 233

Me98. Meyer, R. K., Improved Decision Procedures for Pure Relevant Logics. t.a. In: *A. Church's Festschrift.* 231, 231, 233, 233

MO94. Meyer, R. K and Ono, H., The Finite Model Property for BCK and BCIW. *Studia Logica*, 53, 1 (1994),107-118. 233

MM82. Meyer, R. K. and McRobbie, M. A., Multisets and Relevant Implication I and II. *Australasian J. of Philosophy* 60 (1982), 107-139 and 265-281. 225

Or28. Orlov, I. E., The Calculus of Compatibility of Propositions, (in Russian). *Matematicheskiĭ sbornik* 35 (1928), 263-286. 224

Si88. Simpson, S.G., Ordinal Numbers and the Hilbert Basis Theorem. *J. Symb. Logic* 53, 3 (1988) 961-974. 238

TMM88. Thistlewaite, P., McRobbie, M. and Meyer, R.K., *Automated Theorem-Proving in Non-Classical Logics,* John Wiley, New York, 1988. 225, 226, 227, 227, 228, 228, 229, 238, 238

Ur84. Urquhart, A., The Undecidability of Entailment and Relevant Implication, *J. Symb. Logic*, 40 (1984), pp. 1059-1073. 226

Ur90. Urquhart, A., The Complexity of Decision Procedures in Relevance Logic. In: J.M. Dunn and A. Gupta (eds), *Truth or Consequences, Essays in Honor of Nuel Belnap.*, Dordrecht, Kluwer Academic Publishers, 1990, pp. 61-76. 237

Ur97. Urquhart, A., The Complexity of Decision Procedures in Relevance Logic II. (t.a.) 237, 238

Existence and Uniqueness of Normal Forms in Pure Type Systems with $\beta\eta$-conversion

Gilles Barthe[1,2,⋆]

[1] Institutionen för Datavetenskap, Chalmers Tekniska Högskola, Göteborg, Sweden
gillesb@cs.chalmers.se
[2] Departamento de Informática, Universidade do Minho, Braga, Portugal
gilles@di.uminho.pt

1 Introduction

Pure Type Systems (PTS_βs) provide a parametric framework for typed λ-calculi à la Church [1,2,10,11]. One important aspect of PTS_βs is to feature a definitional equality based on β-conversion. In some instances however, one desires a stronger definitional equality based on $\beta\eta$-conversion. The need for such a strengthened definitional equality arises for example when using type theory as a logical framework or in categorical type theory.

Pure Type Systems with $\beta\eta$-conversion ($\mathsf{PTS}_{\beta\eta}$s) [9,10,17] form a variant of PTS_βs in which definitional equality is understood as $\beta\eta$-conversion. However, the meta-theory of $\mathsf{PTS}_{\beta\eta}$s is significantly more complex than the one of PTS_βs, see *loc. cit.*, because $\beta\eta$-reduction is not confluent on the set of untyped terms [16]. In fact, Geuvers and Werner [12] have shown that such proofs ought to rely on normalization and cannot be achieved by combinatorial means.

η-expansion $\rightarrow_{\overline{\eta}}$ is an alternative computational interpretation of η-conversion, with numerous applications in categorical rewriting, unification and partial evaluation, see [7] for a survey and references. However, the theory of η-expansion and of the associated notion of $\beta\eta$-long normal form remains largely unexplored for dependent types, despite preliminary investigations in [8,13].

This paper is concerned with the Existence and Uniqueness of Normal Forms and confluence for $\rightarrow_{\beta\eta}$ and $\rightarrow_{\beta\overline{\eta}}$. Our work builds upon, clarifies and generalizes previous work in the area, especially [8,9,10,13,17]. Our motivations are threefold:

1. existing confluence proofs both for $\rightarrow_{\beta\eta}$ and $\rightarrow_{\beta\overline{\eta}}$ seem unnecessarily complex. In particular, they do not offer a clear separation between combinatorial and non-combinatorial arguments;
2. the theory of η-expansion and $\beta\eta$-long normal forms has only been studied for the systems of Barendregt's λ-cube [1,2], whereas implementations of type theory such as Coq [3] or Lego [15] are based on substantially more complex systems;

⋆ Research supported by TMR Fellowship FMBICT972065.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 241–259, 1999.

3. the proof in [8] of the existence of $\beta\eta$-long normal forms for the systems of Barendregt's λ-cube relies on a non-standard induction principle whose exact proof-theoretical strength is unclear, and whose proof of well-foundedness does not scale up easily to more complex calculi [6].

Our main contributions are:

1. a criterion for Uniqueness of Normal Forms, from which we derive that every $\mathsf{PTS}_{\beta\eta}$ has unique normal forms w.r.t. $\to_{\beta\eta}$ and $\to_{\beta\overline{\eta}}$. The criterion, which is proved by purely combinatorial means, isolates Uniqueness of Normal Forms as the combinatorial element of confluence proofs. As a result, we claim that our confluence proofs (both for $\to_{\beta\eta}$ and $\to_{\beta\overline{\eta}}$) are conceptually clearer than the ones appearing in the literature;
2. a proof of η-Subject Reduction for normalizing $\mathsf{PTS}_{\beta\eta}$s. Using this result and (1), we prove confluence of $\to_{\beta\eta}$ for the class of normalizing $\mathsf{PTS}_{\beta\eta}$s, thereby strengthening earlier results by Geuvers [9,10] and Salvesen [17] who proved Church-Rosser property of functional, normalizing $\mathsf{PTS}_{\beta\eta}$s. In particular, our result answers in the affirmative a question by Geuvers;
3. a proof that $\beta\eta$-long normal forms exist and that $\to_{\beta\overline{\eta}}$ is weakly normalizing for most normalizing $\mathsf{PTS}_{\beta\eta}$s. Together with (1), these results answer in the affirmative a conjecture by Ghani [13], who proves confluence and weak normalization of $\to_{\beta\overline{\eta}}$ for the Calculus of Constructions, and conjectures that his results can be generalized to Pure Type Systems.

In a companion paper [4], we prove strong normalization of $\to_{\beta\overline{\eta}}$ for the Calculus of Constructions, thereby answering in the positive another conjecture by Di Cosmo and Ghani.

A technical novelty of the present paper is the notion of affiliated pseudo-term—intuitively, a pseudo-term which carries its own type by being of the form $(\lambda x : A.\ x)\ M$. The notion of affiliated pseudo-term provides an appealing alternative to marked terms and leads to important simplifications in the proof of the existence of $\beta\eta$-long normal forms. As a further application of affiliated pseudo-terms, we generalize the induction principle of [8] to a large class of $\mathsf{PTS}_{\beta\eta}$s.

Organization of the paper The paper is organized as follows: Section 2 provides a brief introduction to Pure Type Systems. Section 3 gives a criterion for Uniqueness Normal Forms and for confluence. These criteria are applied to η-reduction and η-expansion in Section 4 and 5 respectively. The existence of $\beta\eta$-long normal forms is also studied in Section 5. Section 6 generalizes the induction principle of [8] to a large class of $\mathsf{PTS}_{\beta\eta}$s. We conclude in Section 7.

Notation We use standard notation and terminology from Abstract Rewriting Systems [14]. In particular, $\to_{ij}$ denotes the union of two relations $\to_i$ and $\to_j$, $\twoheadrightarrow_i^+$ denotes the transitive closure of $\to_i$, $\twoheadrightarrow_i$ denotes the reflexive-transitive closure of $\to_i$ and $=_i$ denotes the reflexive-symmetric-transitive closure of $\to_i$.

Finally, the relation $\downarrow_i$ is defined by $a \downarrow_i b$ if there exists c such that $a \twoheadrightarrow_i c$ and $b \twoheadrightarrow_i c$.

An object a is a *i-normal form* if there is no b such that $a \to_i b$; the set of i-normal forms is denoted by $\mathsf{NF}(i)$. An object a is *i-normalizing* if there is some $b \in \mathsf{NF}(i)$ such that $a \twoheadrightarrow_i b$; the set of i-normalizing objects is denoted by $\mathsf{WN}(i)$.

2 Pure Type Systems

Basics Pure Type Systems provide a parametric framework for typed λ-calculi à la Church. The parametricity of PTSs is achieved through the notion of specification.

Definition 1 (Specification). *A* specification *is a triple* $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ *where* $\mathcal{S}$ *is a set of* sorts, $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ *is a set of* axioms, $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ *is a set of* rules.

Every specification $\mathbf{S}$ yields a Pure Type System $\lambda\mathbf{S}$ as specified below. Throughout this paper, $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is a fixed specification.

Definition 2 (Pure Type Systems).

1. *The set* $\mathcal{T}$ *of* pseudo-terms *is given by the abstract syntax*

$$\mathcal{T} = V \mid \mathcal{S} \mid \mathcal{T}\mathcal{T} \mid \lambda V : \mathcal{T}.\mathcal{T} \mid \Pi V : \mathcal{T}.\mathcal{T}$$

   where V *is a fixed countably infinite set of* variables.
2. β-reduction $\to_\beta$ *is defined as the compatible closure of the contraction*

$$(\lambda x : A.\, M)\ N \to_\beta M\{x := N\}$$

   where $\bullet\{\bullet := \bullet\}$ *is the standard* substitution *operator.*
3. η-reduction $\to_\eta$ *is defined as the compatible closure of the contraction*

$$\lambda x : A.\, (M\ x) \to_\eta M$$

   provided $x \notin \mathrm{FV}(M)$ *where* $\mathrm{FV}(M)$ *is the standard set of* free variables *of* M.
4. *The set* $\mathcal{G}$ *of* pseudo-contexts *is given by the abstract syntax*

$$\mathcal{G} = \langle\rangle \mid \mathcal{G}, V : \mathcal{T}$$

   The domain of a context Γ *is* $dom(\Gamma) = \{x \mid \exists t \in \mathcal{T}.\ x : t \in \Gamma\}$.
5. *A* judgment *is a triple* $\Gamma \vdash M : A$ *where* $\Gamma \in \mathcal{G}$ *and* $M, A \in \mathcal{T}$.
6. *The* derivability *relation* $\vdash$ *is given by the rules of Figure 1. If* $\Gamma \vdash M : A$ *is derivable, then* Γ, M *and* A *are* legal. *The set of legal contexts is denoted by* $\mathcal{H}$.

In the sequel we shall often use the following set of pseudo-terms.

Definition 3. *The set* $\mathcal{C}$ *is defined by the abstract syntax* $\mathcal{C} = V \mid \mathcal{C}\mathcal{T}$.

$$\text{(axiom)} \quad \langle\rangle \vdash s_1 : s_2 \qquad \text{if } (s_1, s_2) \in \mathcal{A}$$

$$\text{(start)} \quad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \qquad \text{if } x \in V \setminus \mathrm{dom}(\Gamma)$$

$$\text{(weakening)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B} \qquad \text{if } x \in V \setminus \mathrm{dom}(\Gamma) \text{ and } A \in V \cup \mathcal{S}$$

$$\text{(product)} \quad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x : A.\, B) : s_3} \qquad \text{if } (s_1, s_2, s_3) \in \mathcal{R}$$

$$\text{(application)} \quad \frac{\Gamma \vdash F : (\Pi x : A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B\{x := a\}}$$

$$\text{(abstraction)} \quad \frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x : A.\, B) : s}{\Gamma \vdash \lambda x : A.\, b : \Pi x : A.\, B}$$

$$\text{(conversion)} \quad \frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'} \qquad \text{if } B =_{\beta\eta} B'$$

Fig. 1. Rules for Pure Type Systems

The next few results summarize some properties of PTSs that are used crucially throughout the paper. Other properties can be found in e.g. [10]. The first lemma provides a precise analysis of the possible ways in which a legal judgment is derived.

Lemma 1 (Generation).

1. $\Gamma \vdash s : C \Rightarrow \exists (s, s') \in \mathcal{A}.\ C =_{\beta\eta} s'$
2. $\Gamma \vdash x : C \Rightarrow \exists s \in \mathcal{S},\ D \in \mathcal{T}.\ C =_{\beta\eta} D \ \wedge\ (x : D) \in \Gamma \ \wedge\ \Gamma \vdash D : s$
3. $\Gamma \vdash \lambda x : A.\, b : C \Rightarrow \exists s \in \mathcal{S},\ B \in \mathcal{T}.\ C =_{\beta\eta} \Pi x : A.\, B \ \wedge\ \Gamma, x : A \vdash b : B \ \wedge\ \Gamma \vdash \Pi x : A.\, B : s$
4. $\Gamma \vdash \Pi x : A.\, B : C \Rightarrow \exists (s_1, s_2, s_3) \in \mathcal{R}.\ C =_{\beta\eta} s_3 \ \wedge\ \Gamma \vdash A : s_1 \ \wedge\ \Gamma, x : A \vdash B : s_2$
5. $\Gamma \vdash F\, a : C \Rightarrow \exists x \in V,\ A, B \in \mathcal{T}.\ C =_{\beta\eta} B\{x := a\} \ \wedge\ \Gamma \vdash F : \Pi x : A.\, B \ \wedge\ \Gamma \vdash a : A$

The second lemma collects a number of closure results.

Lemma 2 (Closure properties).

1. Thinning. *If* $\Gamma \vdash A : B$ *and*[1] $\Gamma \subseteq \Delta$ *with* Δ *legal then* $\Delta \vdash A : B$.
2. Substitution. *If* $\Gamma, x : A, \Delta \vdash B : C$ *and* $\Gamma \vdash a : A$, *then*[2] $\Gamma, \Delta\{x := a\} \vdash A\{x := a\} : B\{x := a\}$.
3. Correctness of Types. *If* $\Gamma \vdash A : B$ *then either* $B \in \mathcal{S}$ *or* $\exists s \in \mathcal{S}.\ \Gamma \vdash B : s$. *Moreover* $\exists s \in \mathcal{S}.\ \Gamma \vdash B : s$ *if* A *is a variable, a* λ*-abstraction or an application.*

The third lemma ensures that types are closed under β-reduction. The proof of this lemma uses the Key Lemma below.

[1] $\Gamma \subseteq \Delta$ if for every $(x : A) \in \Gamma$, also $(x : A) \in \Delta$.
[2] Substitution is extended from pseudo-terms to pseudo-contexts in the usual way.

Lemma 3 (Subject and Predicate Reduction).

1. *β-Subject Reduction:* $\Gamma \vdash M : A \quad \wedge \quad M \twoheadrightarrow_\beta N \quad \Rightarrow \quad \Gamma \vdash N : A$
2. *β-Predicate Reduction:* $\Gamma \vdash M : A \quad \wedge \quad A \twoheadrightarrow_\beta A' \quad \Rightarrow \quad \Gamma \vdash M : A'$

The fourth lemma ensures that β-convertible legal contexts may be interchanged without affecting the set of derivable judgments.

Lemma 4 (Context conversion). *Assume that $\Gamma \in \mathcal{G}$ is legal, $\Delta \vdash M : A$ and $\Gamma =_{\beta\eta} \Delta$. Then $\Gamma \vdash M : A$.*

Proof. By induction on the length of Γ.

A non-result: confluence It is folklore that $\to_{\beta\eta}$-reduction is not confluent on pseudo-terms. Nederpelt's counter-example [16] provides an easy proof of this fact: for every pairwise disjoint $A, B, x, y \in V$, we have

$$\lambda x : A.\, x \quad {}_\beta\!\leftarrow \quad \lambda x : A.\, (\lambda y : B.\, y)\, x \quad \to_\eta \quad \lambda y : B.\, y$$

(Note that the above example also shows the failure of Uniqueness of Normal Forms and of local confluence.) The failure of confluence makes it difficult to prove basic properties such as Subject Reduction. In order to overcome the problem, we observe that one can complete $\beta\eta$-reduction into a confluent relation. (The following is an inessential variant of Geuvers' approach, chosen so as to avoid the introduction of erased pseudo-terms.)

Definition 4. *Let $s_0 \in \mathcal{S}$ be a fixed sort. κ-reduction is defined as the compatible closure of the rule*

$$\lambda x : A.\, M \to_\kappa \lambda x : s_0.\, M \quad (A \neq s_0)$$

The unique κ-normal form of a pseudo-term M is denoted by M^κ.

$\beta\eta\kappa$-reduction is well-behaved and its associated equality is suitably related to $\beta\eta$-equality.

Proposition 1. *$\to_{\beta\eta\kappa}$ is confluent. Moreover, for every $A, B \in \mathcal{T}$,*

$$A =_{\beta\eta} B \Leftrightarrow A =_{\beta\eta\kappa} B$$

Proof. As in [10].

An important consequence of the above proposition is the so-called Key Lemma.

Lemma 5 (Key Lemma, [10]).

1. *If $\Pi x : A.\, B =_{\beta\eta} \Pi x : A'.\, B'$ then $A =_{\beta\eta} A'$ and $B =_{\beta\eta} B'$.*
2. *If $M, M' \in \mathcal{C}$ and $M =_{\beta\eta} M'$ then one of the two conditions below must hold:*
   (a) $M, M' \in V$ and $M = M'$;
   (b) $M = M_1\, M_2$ and $M' = M_1'\, M_2'$ with $M_1 =_{\beta\eta} M_1'$ and $M_2 =_{\beta\eta} M_2'$.

Proof. Using Proposition 1.

The next lemma is concerned with Uniqueness of Types for elements of $\mathcal{C}$. It is needed to prove both Uniqueness of Normal Forms and Strengthening—hence the somewhat unusual form.

Lemma 6 (Restricted Uniqueness of Types). *Let $M, M' \in \mathcal{C}$. Then*

$$(\Gamma \vdash M : B \quad \wedge \quad \Gamma' \vdash M' : B' \quad \wedge \quad \Gamma \subseteq \Gamma' \quad \wedge \quad M =_{\beta\eta} M') \quad \Rightarrow \quad B =_{\beta\eta} B'$$

Proof. By induction on the structure of $M \in \mathcal{C}$, invoking the Key Lemma to determine the shape of M'.

An inductive definition of β-normal forms Legal β-normal forms are a central theme in the paper. In order to reason about them, it is convenient to use the following characterization.

Definition 5 (β-normal forms). *The set $\mathcal{N}$ is defined by the syntax*

$$\mathcal{N} = \mathcal{N}_\lambda \mid \mathcal{N}_e \quad \mathcal{N}_\lambda = \lambda V : \mathcal{N}_e . \mathcal{N} \quad \mathcal{B} = V \mid \mathcal{B}\mathcal{N} \quad \mathcal{N}_e = \mathcal{B} \mid \mathcal{S} \mid \Pi V : \mathcal{N}_e . \mathcal{N}_e$$

Although not every β-normal form belongs to $\mathcal{N}$, $\mathsf{NF}(\beta)$ and $\mathcal{N}$ coincide on the set of legal terms.

Lemma 7. *Let $M \in \mathcal{T}$ be legal. Then $M \in \mathcal{N}$ iff $M \in \mathsf{NF}(\beta)$.*

Proof. The left-to-right implication is proved by induction on the structure of $M \in \mathcal{N}$. The right-to-left implication is proved by induction on the structure of $M \in \mathcal{T}$.

η-Subject Reduction In [9,10], Geuvers proves that η-Subject Reduction holds for all functional normalizing Pure Type Systems and suggests that the restriction to functional specifications could be dropped. In this subsection, we validate Geuvers's suggestion. The idea is to modify the notion of preservation of sorts [10, Definition 5.2.7, page 121] slightly and to show that normalizing $\mathsf{PTS}_{\beta\eta}$s satisfy this modified notion of preservation of sorts. Proofs remain essentially unchanged.

Proposition 2 (Strengthening and Subject Reduction). *If $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$ then:*

1. *Preservation of sorts: for every $s, s' \in \mathcal{S}$ and $M, M' \in \mathcal{N}_e$,*
   $(\Gamma \vdash M : s \ \wedge \ \Gamma' \vdash M' : s' \ \wedge \ \Gamma \subseteq \Gamma' \ \wedge \ M =_{\beta\eta} M') \ \Rightarrow \ \Gamma \vdash M : s'$
2. *Strengthening:* [3] $(\Gamma_1, y : C, \Gamma_2 \vdash b : B \ \wedge \ x \notin \mathrm{FV}(\Gamma_2, b, B)) \Rightarrow \Gamma_1, \Gamma_2 \vdash b : B$
3. *η-Subject Reduction: i.e.* $(\Gamma \vdash M : A \ \wedge \ M \to_\eta N) \Rightarrow \Gamma \vdash N : A$.

[3] We write $\mathrm{FV}(A_1, A_2)$ instead of $\mathrm{FV}(A_1) \cup \mathrm{FV}(A_2)$ and similarly for pseudo-contexts.

Proof. (1) By induction on the structure of $M \in \mathcal{N}_e$, invoking the Key Lemma to determine the shape of M'. If $M \in \mathcal{B}$, then one uses Lemma 6.

(1') Using (1), β-Subject Reduction and $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$, one can derive for every $s, s' \in \mathcal{S}$, $M \in \mathcal{N}_e$ and $M' \in \mathcal{T}$,
$(\Gamma \vdash M : s \ \wedge \ \Gamma' \vdash M' : s' \ \wedge \ \Gamma \subseteq \Gamma' \ \wedge \ M =_{\beta\eta} M') \ \Rightarrow \ \Gamma \vdash M : s'$

(2) Exactly as the proof of [10, Lemma 5.2.10, page 122], using (1'). First we prove weak strengthening, i.e.

$$\Gamma_1, y : C, \Gamma_2 \vdash b : B \ \wedge \ x \notin \mathrm{FV}(\Gamma_2, b) \ \Rightarrow \ \exists B' \in \mathcal{T}.\, B =_{\beta\eta} B' \ \wedge \ \Gamma_1, \Gamma_2 \vdash b : B'$$

Then we derive strengthening by a case analysis on the correctness of types for B.

(3) Prove by simultaneous induction on the derivation of $\Gamma \vdash M : A$ that

1. $\Gamma \vdash M : A \ \wedge \ A \to_\eta A' \ \Rightarrow \ \Gamma \vdash M' : A$
2. if $\Gamma \vdash M : A \ \wedge \ \Gamma \to_\eta \Gamma' \ \Rightarrow \ \Gamma' \vdash M : A$

3 A criterion for Uniqueness of Normal Forms

Uniqueness of Normal Forms (UN) is an important step towards confluence. The purpose of this section is to give a general criterion for UN. More precisely, we give a sufficient condition for a family $(\mathcal{P}_\Gamma)_{\Gamma \in \mathcal{H}}$ with $\mathcal{P}_\Gamma \subseteq \mathsf{NF}(\beta)$ for every $\Gamma \in \mathcal{H}$ to be such that $(*)$ *for every derivable judgment* $\Gamma \vdash M : A$ *there exists at most one* $M' \in \mathcal{P}_\Gamma$ *such that* $M =_{\beta\eta} M'$ *and* $\Gamma \vdash M' : A$.

A few words are in order to justify our more abstract view of UN:

1. we abstract away from the notion of reduction and focus on normal forms because (UN) itself does not appeal to any notion of reduction—it appeals to conversion only—and because the notion of $\beta\eta$-long normal form is clearly defined, while at the time of writing there is no definite answer as to what is a good notion of η-expansion—this issue is discussed further in Section 5;
2. we index sets of normal forms by legal contexts in order to be able to apply the criterion to $\beta\eta$-long normal forms, which are defined relatively to a context.

We turn to the formulation of the criterion. The next definition introduces normal form candidates. Later we shall show that normal form candidates enjoy $(*)$.

Definition 6 (Normal Form Candidate). *Let* $\mathcal{P} = (\mathcal{P}_\Gamma)_{\Gamma \in \mathcal{H}}$ *such that* $\mathcal{P}_\Gamma \subseteq \mathcal{N}$ *for every* $\Gamma \in \mathcal{H}$.

1. $\mathcal{P}$ *is* subterm-closed *if:*
   (a) $\mathcal{S} \subseteq \mathcal{P}_\Gamma$;
   (b) *if* $y \ P_1 \ \ldots \ P_n \in \mathcal{P}_\Gamma$ *then* $P_1, \ldots, P_n \in \mathcal{P}_\Gamma$;
   (c) *if* $\lambda x : A.\, M \in \mathcal{P}_\Gamma$ *then* $A \in \mathcal{P}_\Gamma$ *and* $M \in \mathcal{P}_{\Gamma, x:A}$;
   (d) *if* $\Pi x : A.\, B \in \mathcal{P}_\Gamma$ *then* $A \in \mathcal{P}_\Gamma$ *and* $B \in \mathcal{P}_{\Gamma, x:A}$.

2. $\mathcal{P}$ *is* λ-compatible *if for every* $M, M' \in \mathcal{P}_\Gamma$,

$$(\Gamma \vdash M : A \ \wedge \ \Gamma \vdash M : A \ \wedge \ M =_{\beta\eta} M' \ \wedge \ M \in \mathcal{N}_\lambda) \ \Rightarrow \ M' \in \mathcal{N}_\lambda$$

3. $\mathcal{P}$ *is a* normal form candidate *if it is subterm-closed and* λ*-compatible.*

Note that, in order to apply our criterion to $\beta\eta$-long normal forms, we do not make any requirement on variables in the definition of subterm-closed. The notion of λ-compatibility provides a useful reasoning principle.

Proposition 3. *Let* $\mathcal{P} = (\mathcal{P}_\Gamma)_{\Gamma \in \mathcal{H}}$ *be* λ*-compatible, let* $\Gamma \in \mathcal{H}$ *and* $M, M' \in \mathcal{P}_\Gamma$. *If*

$$\Gamma \vdash M : A \ \wedge \ \Gamma \vdash M' : A' \ \wedge \ M =_{\beta\eta} M' \ \wedge \ [(A =_{\beta\eta} A') \vee (A, A' \in \mathcal{S})]$$

then one of the following conditions holds:

1. $M, M' \in \mathcal{S}$ *and* $M = M'$;
2. $M, M' \in V$ *and* $M = M'$;
3. $M = M_1\ M_2$ *and* $M' = M_1'\ M_2'$ *with* $M_1 =_{\beta\eta} M_1'$ *and* $M_2 =_{\beta\eta} M_2'$;
4. $M = \lambda x : B.\, P$ *and* $M' = \lambda x : B'.\, P'$ *with* $B =_{\beta\eta} B'$ *and* $P =_{\beta\eta} P'$;
5. $M = \Pi x : B.\, C$ *and* $M' = \Pi x : B'.\, C'$ *with* $B =_{\beta\eta} B'$ *and* $C =_{\beta\eta} C'$.

Proof. By a case analysis on the structure of M. If $M \in \mathcal{N}_e$, we use Proposition 1, the Key Lemma and the fact that $M' \in \mathcal{N}_e$. If $M = \lambda x : B.\, N$ then necessarily $A \notin \mathcal{S}$. If $A \neq_{\beta\eta} A'$ then we are done. So assume $A =_{\beta\eta} A'$. Hence $\Gamma \vdash M' : A$ by conversion and hence by λ-compatibility $M' = \lambda x : B'.\, N'$. Moreover $N =_{\beta\eta} N'$ since

$$N =_{\beta\eta} (\lambda x : B.\, N)\, x = M\, x =_{\beta\eta} M'\, x = (\lambda x : B'.\, N')\, x =_{\beta\eta} N'$$

To conclude that $B =_{\beta\eta} B'$, observe that by generation, there exists $C, C' \in \mathcal{T}$ such that $A =_{\beta\eta} \Pi x : B.\, C$ and $A' =_{\beta\eta} \Pi x : B'.\, C'$. By assumption $A =_{\beta\eta} A'$ and hence $\Pi x : B.\, C =_{\beta\eta} \Pi x : B'.\, C'$. By the Key Lemma, $B =_{\beta\eta} B'$.

Uniqueness of Normal Forms is proved by induction on $\mathcal{N}$.

Proposition 4 (Uniqueness of Normal Forms). *Let* $\mathcal{P} = (\mathcal{P}_\Gamma)_{\Gamma \in \mathcal{H}}$ *be a normal form candidate. Assume* $\Gamma \in \mathcal{H}$ *and* $M, M' \in \mathcal{P}_\Gamma$. *Then*

$$(\Gamma \vdash M : A \wedge \Gamma \vdash M' : A' \wedge M =_{\beta\eta} M' \wedge [(A =_{\beta\eta} A') \vee (A, A' \in \mathcal{S})]) \Rightarrow M = M'$$

Proof. By induction on the structure of M, using Proposition 3 to determine the shape of M'. We treat two cases:

1. If $M = \lambda x : B.\, N$ necessarily $M' = \lambda x : B'.\, N'$ with $B =_{\beta\eta} B'$ and $N =_{\beta\eta} N'$. By generation, there exists $s, s' \in \mathcal{S}$ and $C, C' \in \mathcal{T}$ such that

$$\begin{array}{ccc} \Gamma \vdash B : s & \Gamma, x : B \vdash N : C & \Pi x : B.\, C =_{\beta\eta} A \\ \Gamma \vdash B' : s' & \Gamma, x : B' \vdash N' : C' & \Pi x : B'.\, C' =_{\beta\eta} A' \end{array}$$

   By the Key Lemma, $C =_{\beta\eta} C'$. By induction hypothesis, $B = B'$ and $N = N'$.

2. If $M = M_1\ M_2$ necessarily $M' = M_1'\ M_2'$ with $M_i =_{\beta\eta} M_i'$ for $i = 1, 2$. By generation, there exists $\Pi x : C.\ D, \Pi x : C'.\ D' \in \mathcal{T}$ such that

$$\begin{array}{lll} \Gamma \vdash M_1 : \Pi x : C.\ D & \Gamma \vdash M_2 : C & D\{x := M_2\} =_{\beta\eta} A \\ \Gamma \vdash M_1' : \Pi x : C'.\ D' & \Gamma \vdash M_2' : C' & D'\{x := M_2'\} =_{\beta\eta} A' \end{array}$$

Besides $M_1, M_1' \in \mathcal{B}$ so $\Pi x : C.\ D =_{\beta\eta} \Pi x : C'.\ D'$ by Lemma 6. By the Key Lemma, $C =_{\beta\eta} C'$. By induction hypothesis, $M_1 = M_1'$ and $M_2 = M_2'$. Hence $M = M'$.

Confluence is an easy consequence of Uniqueness of Normal Forms and weak normalization.

Corollary 1 (Confluence). *Let* $\rightarrow_{\phi_\Gamma} \subseteq \mathcal{T} \times \mathcal{T}$ *for every* $\Gamma \in \mathcal{H}$. *Then*

$$(\Gamma \vdash M : A \wedge \Gamma \vdash M' : A' \wedge M =_{\beta\eta} M' \wedge [(A =_{\beta\eta} A') \vee (A, A' \in \mathcal{S})]) \Rightarrow M \downarrow_{\phi_\Gamma} M'$$

provided:

1. $\mathcal{P} = (\mathcal{P}_\Gamma)_{\Gamma \in \mathcal{H}}$ *is a normal form candidate where* $\mathcal{P}_\Gamma$ *denotes the set of legal* ϕ_Γ*-normal forms;*
2. $\lambda\mathbf{S} \models \mathsf{WN}(\phi)$, *i.e.* $\Gamma \vdash M : A \ \Rightarrow\ \exists N \in \mathcal{P}_\Gamma.\ M \twoheadrightarrow_{\phi_\Gamma} N$;
3. $\lambda\mathbf{S} \models \mathsf{SR}(\phi)$, *i.e.* $\Gamma \vdash M : A \ \wedge\ M \rightarrow_{\phi_\Gamma} N \ \Rightarrow\ \Gamma \vdash N : A$.

Proof. By assumption (2), there exists $N, N' \in \mathcal{P}_\Gamma$ such that $M \twoheadrightarrow_{\phi_\Gamma} N$ and $M' \twoheadrightarrow_{\phi_\Gamma} N'$. By assumption (3), $\Gamma \vdash N : A$ and $\Gamma' \vdash N' : A'$. By assumption (1) and Proposition 4, $N = N'$ so $M \downarrow_{\phi_\Gamma} M'$.

4 η-reduction

In this section, we apply Proposition 4 and Corollary 1 to $\beta\eta$-reduction.

Definition 7. *The set* $\mathcal{M}$ *is defined as* $\mathcal{N} \cap \mathsf{NF}(\eta)$.

$\mathcal{M}$ coincides with the set of $\beta\eta$-normal forms on the set of legal terms.

Lemma 8. *Let* $M \in \mathcal{T}$ *be legal. Then* $M \in \mathcal{M}$ *iff* $M \in \mathsf{NF}(\beta\eta)$.

Proof. $M \in \mathsf{NF}(\beta\eta)$ iff $M \in \mathsf{NF}(\beta) \cap \mathsf{NF}(\eta)$. If M is legal, then $M \in \mathsf{NF}(\beta)$ iff $M \in \mathcal{N}$. Hence, if M is legal, then $M \in \mathsf{NF}(\beta\eta)$ iff $M \in \mathcal{N} \cap \mathsf{NF}(\eta)$. Now $\mathcal{M} = \mathcal{N} \cap \mathsf{NF}(\eta)$ so we are done.

Lemma 9 ([8]). *If* $M \in \mathsf{NF}(\beta\eta)$ *is legal then* $M^\kappa \in \mathsf{NF}(\beta\eta\kappa)$.[4]

Proof. Similar to the proof of [8, Proposition 11].

In the sequel, we write $\mathcal{M}$ for $(\mathcal{M})_{\Gamma \in \mathcal{H}}$.

[4] Dowek, Huet and Werner [8, Proposition 11] use a slightly different but equivalent statement.

Proposition 5. *$\mathcal{M}$ is a normal form candidate.*

Proof. $\mathcal{M}$ is a subterm-closed: by inspection on the definition of $\mathcal{M}$. $\mathcal{M}$ is λ-compatible: assume $M_1, M_2 \in \mathcal{M}$ such that

$$\Gamma \vdash M_1 : A \ \wedge \ \Gamma \vdash M_2 : A \ \wedge \ M_1 =_{\beta\eta} M_2 \ \wedge \ M_1 \in \mathcal{N}_\lambda$$

By Proposition 1, $M_1 \downarrow_{\beta\eta\kappa} M_2$. By confluence of $\beta\eta\kappa$-reduction and Lemma 9, we have $M_1^\kappa = M_2^\kappa$. Necessarily $M_2^\kappa \in \mathcal{N}_\lambda$ and thus $M_2 \in \mathcal{N}_\lambda$.

Corollary 2.

1. *$\beta\eta$-reduction has unique normal forms—in the sense of Proposition 4.*
2. *$\beta\eta$-reduction is confluent—in the sense of Proposition 1—provided $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$.*

Proof. (1) By Propositions 4 and 5. (2) By Corollary 1, using Proposition 5, Proposition 2 and the fact that $\lambda\mathbf{S} \models \mathsf{WN}(\beta) \Rightarrow \lambda\mathbf{S} \models \mathsf{WN}(\beta\eta)$, see [10].

5 η-expansion

In this section, we prove the uniqueness of $\beta\eta$-long normal forms for all $\mathsf{PTS}_{\beta\eta}$s and the existence of $\beta\eta$-long normal forms for a large class of $\mathsf{PTS}_{\beta\eta}$s. Somewhat unusually, we do not define $\beta\eta$-long normal forms as the normal forms of some rewrite system—this is explained later in the section. Rather, we give an inductive definition of $\beta\eta$-long normal forms. It is then easy to show UN. Proving the existence of $\beta\eta$-long normal forms is harder and is addressed later in the section.

Definition 8. *Let $\Gamma \in \mathcal{H}$. A term $M \in \mathcal{N}$ is a $\beta\eta$-long normal form in context Γ, written $\xi_\Gamma(M)$, if M is legal in Γ and one of the following conditions holds:*

1. *$M \in \mathcal{S}$;*
2. *$M = \Pi x : B.\, C$, $\xi_\Gamma(B)$ and $\xi_{\Gamma,x:B}(C)$;*
3. *$M = \lambda x : B.\, N$, $\xi_\Gamma(B)$ and $\xi_{\Gamma,x:B}(N)$;*
4. *$M = x\ P_1\ \ldots\ P_n$, $\Gamma \vdash M : A$ for some $A \in \mathcal{C} \cup \mathcal{S}$ and $\xi_\Gamma(P_i)$ for $i = 1, \ldots, n$.*

The set of $\beta\eta$-long normal form in context Γ is denoted Ξ_Γ.

Proposition 6. *$\Xi = (\Xi_\Gamma)_{\Gamma \in \mathcal{H}}$ is a normal form candidate.*

Proof. Ξ is subterm-closed: by inspection on the definition of $\beta\eta$-long normal forms. Ξ is λ-compatible: assume $M, M' \in \xi_\Gamma$ such that

$$\Gamma \vdash M : A \ \wedge \ \Gamma \vdash M' : A \ \wedge \ M =_{\beta\eta} M' \ \wedge \ M \in \mathcal{N}_\lambda$$

Necessarily $A =_{\beta\eta} \Pi x : B.\, C$ for some $B, C \in \mathcal{T}$ and hence M' cannot be a sort or a product for typability reasons. Therefore we must have $M' \in \mathcal{N}_\lambda$ or $M \in \mathcal{B}$. The second case is impossible by definition of ξ_Γ so we must have $M' \in \mathcal{N}_\lambda$.

Corollary 3. *$\beta\eta$-long normal forms are unique—in the sense of Proposition 4.*

In order to extend the result to confluence, we introduce the notion of η-expansion. As for $\beta\eta$-long normal forms, η-expansion is defined relatively to a context.

Definition 9 (η-expansion).

1. *η-expansion $\to_{\overline{\eta}(\Gamma)}$ is defined by the rules*[5]

$$
\begin{array}{rcll}
M \to_{\overline{\eta}(\Gamma)} N & \Rightarrow & M\ P \to_{\overline{\eta}(\Gamma)} N\ P & \\
 & & & \text{if } N \neq \lambda x : B.\ (M\ x) \\
M \to_{\overline{\eta}(\Gamma)} N & \Rightarrow & P\ M \to_{\overline{\eta}(\Gamma)} P\ N & \\
M \to_{\overline{\eta}(\Gamma)} N & \Rightarrow & \lambda x : M.\ P \to_{\overline{\eta}(\Gamma)} \lambda x : N.\ P & \\
M \to_{\overline{\eta}(\Gamma)} N & \Rightarrow & \Pi x : M.\ P \to_{\overline{\eta}(\Gamma)} \Pi x : N.\ P & \\
M \to_{\overline{\eta}(\Gamma, x:A)} N & \Rightarrow & \lambda x : P.\ M \to_{\overline{\eta}(\Gamma)} \lambda x : P.\ N & \\
M \to_{\overline{\eta}(\Gamma, x:A)} N & \Rightarrow & \Pi x : P.\ M \to_{\overline{\eta}(\Gamma)} \Pi x : P.\ N & \\
M \to_{\overline{\eta}(\Gamma)} \lambda x : A.\ M\ x & \text{provided} & \Gamma \vdash M : \Pi x : A.\ B \ \wedge\ M & \\
 & & & \text{not a } \lambda\text{-abstraction}
\end{array}
$$

2. Restricted η_r-expansion $\to_{\overline{\eta_r}(\Gamma)}$ *is defined as above, except for the extra proviso $\xi_\Gamma(A)$ in the last rule.*
3. Another restricted η_a-expansion $\to_{\overline{\eta_a}(\Gamma)}$ *is defined as above, except for the extra proviso $A \in \mathcal{N}$ in the last rule.*

A few words are in order to explain the notions of η-expansion: in [13], Ghani observes that $\to_{\beta\overline{\eta}}$ is not strongly normalizing on legal terms of the Calculus of Constructions because η-expansions may create arbitrary β-redexes. Motivated by this negative result, Ghani suggests η_r-expansion as a more appropriate notion and conjectures that $\to_{\beta\overline{\eta}_r}$ is strongly normalizing on legal terms of the Calculus of Constructions. The notion of η_a-expansion provides another (less severe) restriction of η-expansion to which Ghani's counterexample does not apply. Its advantage over η_r-expansion is that its definition does not rely on the notion of $\beta\eta$-long normal form. Note that in [4], we prove that $\to_{\beta\overline{\eta}_a}$ is strongly normalizing on legal terms of the Calculus of Constructions.

Corollary 4. *Let $\to_{\phi_\Gamma} \subseteq \to_{\beta\overline{\eta}(\Gamma)}$ for every $\Gamma \in \mathcal{H}$. Then $\to_{\phi_\Gamma}$-is confluent—in the sense of Corollary 1—provided $\lambda\mathbf{S} \models \mathsf{WN}(\phi)$ and Ξ_Γ is the set of $\beta\phi_\Gamma$-normal forms legal in context Γ.*

Proof. By Proposition 6, Corollary 1 and the fact that $\to_{\beta\overline{\eta}(\Gamma)}$, and hence every subrelation of it, enjoys Subject Reduction.

The next few results establish weak normalization and under certain conditions confluence of $\to_{\beta\overline{\eta}_r(\Gamma)}$. Similar results for $\to_{\beta\overline{\eta}_a(\Gamma)}$ and $\to_{\beta\overline{\eta}(\Gamma)}$ are omitted.

[5] It is folklore that one cannot take the compatible closure of the last rule below because η-expanding the first argument of an application would create loops, e.g. $M\ N \to_{\overline{\eta}(\Gamma)} (\lambda x : A.\ M\ x)\ N \to_\beta M\ N$.

Lemma 10.

1. $\to_{\overline{\eta}_r(\Gamma)}$ *is strongly normalizing on* Γ*-legal terms.*
2. $\to_{\overline{\eta}_r(\Gamma)}$ *preserves* Γ*-legal* β*-normal forms.*

Proof. (1) Using Klop-Nederpelt's Lemma $\mathsf{WN}+\mathsf{WCR}+\mathsf{INC} \Rightarrow \mathsf{SN}+\mathsf{CR}$ [14], it is enough to show that $\to_{\overline{\eta}_r(\Gamma)}$ is weakly normalizing (observe that the innermost reduction strategy is normalizing), weak Church-Rosser (follows from uniqueness of $\beta\eta$-long normal forms and weak normalization) and increasing ($\to_{\overline{\eta}_r(\Gamma)}$ increases the length of terms). (2) By an induction on the structure of $M \in \mathcal{N}$.

Corollary 5. $\lambda\mathbf{S} \models \mathsf{WN}(\beta\overline{\eta}_r)$ *provided* $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$.

Proof. Assume $\Gamma \vdash M : A$. By assumption, $M \twoheadrightarrow_\beta N$ for some $N \in \mathsf{NF}(\beta)$. By Subject Reduction, $\Gamma \vdash N : A$. By Lemma 10.(1), $N \twoheadrightarrow_{\overline{\eta}_r(\Gamma)} N'$ for some $N' \in \mathsf{NF}(\overline{\eta}_r(\Gamma))$. By Lemma 10.(2), $N' \in \mathsf{NF}(\beta)$ and hence $N' \in \mathsf{NF}(\beta\overline{\eta}_r(\Gamma))$.

To derive confluence, we need to show that the set of legal $\beta\overline{\eta}_r(\Gamma)$-normal forms legal in context Γ coincides with Ξ_Γ. Surprinsingly, this is not so obvious since, given a judgement $x : A \to A \vdash x : A \to A$, we need to know that A is $\beta\eta$-convertible to an $\beta\eta$-long normal form to conclude that x is *not* in $\beta\overline{\eta}_r(\Gamma)$-normal form!

Lemma 11.

1. *If* $\Gamma \vdash M : A$ *and* $M \in \Xi_\Gamma$ *then* $M \in \mathsf{NF}(\beta\overline{\eta}_r(\Gamma))$.
2. *If* $\Gamma \vdash M : A$ *and* $M \in \mathsf{NF}(\beta\overline{\eta}(\Gamma))$ *then* $M \in \Xi_\Gamma$ *provided for every* $\Gamma \vdash B : s$ *there exists* $C \in \Xi_\Gamma$ *such that* $\Gamma \vdash C : s$ *and* $B =_{\beta\eta} C$.

Proof. By induction on the structure of M. We treat (2) in case $M = x\ P_1\ \ldots\ P_n$. Then A cannot be convertible to a product since $M \in \mathsf{NF}(\beta\overline{\eta}(\Gamma))$, and if $A =_{\beta\eta} \Pi y : A_1.\ A_2$ then $M \to_{\overline{\eta}_r(\Gamma)} \lambda y : A_1'.\ M\ y$ where $\Gamma \vdash A_1' : s$ and $A_1' =_{\beta\eta} A_1$. Moreover, $P_i \in \mathsf{NF}(\beta\overline{\eta}(\Gamma))$ for $i = 1, \ldots, n$ so by induction hypothesis $P_i \in \Xi_\Gamma$ for $i = 1, \ldots, n$. Hence $M \in \Xi_\Gamma$.

Corollary 6. $\to_{\beta\overline{\eta}_r}$ *is confluent—in the sense of Corollary 1—provided* $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$ *and for every* $\Gamma \vdash B : s$ *there exists* $C \in \Xi_\Gamma$ *such that* $\Gamma \vdash C : s$ *and* $B =_{\beta\eta} C$.

Proof. Follows from Corollaries 4 and 5 and Lemma 11.

So we are naturally led to consider the existence of $\beta\eta$-long normal forms. We start with some preliminary definitions that introduce the notion of affiliated pseudo-term. Intuititely, affiliated pseudo-terms are pseudo-terms that carry their own type by being of the form $(\lambda x : A.\ x)\ M$. For technical reasons, it is both preferable and sufficient to affiliate variables and applications.

Definition 10 (Affiliated pseudo-terms).

1. The set $\mathcal{U}$ of *affiliated pseudo-terms* is given by the abstract syntax

$$\mathcal{U} = \mathsf{I}_{\mathcal{U}}\, V \mid \mathcal{S} \mid \mathsf{I}_{\mathcal{U}}\, (\mathcal{U}\, \mathcal{U}) \mid \lambda V : \mathcal{U}.\, \mathcal{U} \mid \Pi V : \mathcal{U}.\, \mathcal{U}$$

   where $\mathsf{I}_A = \lambda x : A.\, x$.
2. The *erasure map* $|.| : \mathcal{U} \to \mathcal{T}$ is defined inductively as follows:

$$\begin{aligned} |\mathsf{I}_A\, y| &= y \\ |s| &= s \\ |\mathsf{I}_A\, (u\, v)| &= |u|\, |v| \\ |\lambda y : B.\, M| &= \lambda y : |B|.\, |M| \\ |\Pi y : A.\, B| &= \Pi y : |A|.\, |B| \end{aligned}$$

3. The *affiliated substitution* $M[x := N] \in \mathcal{U}$ (where $M, N \in \mathcal{U}$ and $x \in V$) is defined inductively as follows:

$$\begin{aligned} (\mathsf{I}_A\, y)[y := N] &= N \\ (\mathsf{I}_A\, z)[y := N] &= (\mathsf{I}_{A[y:=N]})\, z \qquad \text{if } z \neq y \\ s[y := N] &= s \\ (\mathsf{I}_A\, (u\, v))[y := N] &= \mathsf{I}_{(A[y:=N])}\, ((u[y := N])\, (v[y := N])) \\ (\lambda z : B.\, c)[y := N] &= \lambda z : (B[y := N]).\, (c[y := N]) \\ (\Pi x : A.\, B)[y := N] &= \Pi x : (A[y := N]).\, (B[y := N]) \end{aligned}$$

4. *β_e-reduction* $\to_{\beta_e}$ is defined on $\mathcal{U} \times \mathcal{U}$ as the compatible closure of the contraction

$$\mathsf{I}_A\, ((\lambda z : C.\, M)\, N) \to_{\beta_e} M[z := N]$$

5. *η_e-reduction* $\to_{\eta_e}$ is defined on $\mathcal{U} \times \mathcal{U}$ as the compatible closure of the contraction

$$\lambda x : C.\, \mathsf{I}_A\, (M\, (\mathsf{I}_B\, x)) \to_{\eta_e} M$$

   provided $x \notin \mathrm{FV}(M)$.
6. The set $\mathcal{W}$ is defined as $\mathcal{U} \cap \mathsf{NF}(\beta_e \eta_e)$.

Using the technique of Section 3, one proves uniqueness of $\beta_e\eta_e$-normal forms.

Proposition 7. *Let $M, N \in \mathcal{W}$.*

$$(\Gamma \vdash M : A \wedge \Gamma \vdash M' : A' \wedge M =_{\beta\eta} M' \wedge [(A =_{\beta\eta} A') \vee (A, A' \in \mathcal{S})]) \Rightarrow M = M'$$

We exploit the uniqueness of $\beta_e\eta_e$-normal forms to define a function $\mu : \mathcal{G} \times \mathcal{T} \to \mathcal{W}$. Ideally, one would like to have $\Gamma \vdash \mu(\Gamma \vdash M) : A$ whenever $\Gamma \vdash M : A$. However, this is not possible in general since the specification may not have enough rules. This motivates the following definition.

Definition 11. *Let $\mathcal{S}_\bullet = \{s \in \mathcal{S} \mid \forall s' \in \mathcal{S}.\, (s, s, s') \notin \mathcal{R}\}$. The specification $\mathbf{S}^\Delta$ is defined by*

$$(\mathcal{S} \cup \{\bullet_s \mid s \in \mathcal{S}_\bullet\}, \mathcal{A}, \mathcal{R} \cup \{(s, s, \bullet_s) \mid s \in \mathcal{S}_\bullet\})$$

Below we use $\vdash^{\Delta}$ to denote derivability in $\lambda\mathbf{S}^{\Delta}$.

Proposition 8. *For every derivable judgment $\Gamma \vdash M : A$ there exists a derivable judgment $\Gamma \vdash^{\Delta} M' : A$ such that $M' \in \mathcal{U}$ and $|M'| = M$.*

Proof. By induction on the structure of derivations using Context Conversion and the fact that $P \twoheadrightarrow_{\beta} |P|$ for every $P \in \mathcal{U}$.

Corollary 7. *Assume $\lambda\mathbf{S}^{\Delta} \models \mathsf{WN}(\beta_e)$. For every $(\Gamma, M) \in \mathcal{G} \times \mathcal{N}$ there exists $M' \in \mathcal{W}$ such that $M' =_{\beta\eta} M$ and for every $A \in \mathcal{T}$,*

$$\Gamma \vdash M : A \;\Rightarrow\; \Gamma \vdash^{\Delta} M' : A$$

Moreover M' is unique if $\Gamma \vdash M : A$ for some $A \in \mathcal{T}$.

Proof. By induction on the structure of $M \in \mathcal{N}$, using Proposition 8, Subject Reduction, restricted Uniqueness of Types and uniqueness of $\beta_e\eta_e$-normal forms (note that Proposition 7 does not make any assumption on the specification hence $\mathbf{S}^{\Delta}$ also has unique $\beta_e\eta_e$-normal forms).

The above corollary justifies the next definition.

Definition 12. *Assume $\lambda\mathbf{S}^{\Delta} \models \mathsf{WN}(\beta_e)$ and assume $\Gamma \vdash M : A$ with $M \in \mathcal{N}$. The pseudo-term $\mu(\Gamma \vdash M)$ is defined as the unique M' given in Proposition 8.*

Before exploiting μ to prove the existence of $\beta\eta$-long normal forms, we give a condition under which the hypothesis of Definition 12 holds.

Proposition 9. *$\lambda\mathbf{S}^{\Delta} \models \mathsf{WN}(\beta_e)$ provided there exists a morphism of specifications[6] $H : \mathbf{S} \to C^{\infty}$ where the specification $C^{\infty} = (\mathcal{S}^{\infty}, \mathcal{A}^{\infty}, \mathcal{R}^{\infty})$ is given by:*

Sorts	$*_i$	$i \in \mathbf{N}$
Axioms	$(*_i, *_j)$	$i, j \in \mathbf{N}$ & $i < j$
Rules	$(*_i, *_j, *_k)$	$i, j, k \in \mathbf{N}$ & $(i, j \leq k)$
	$(*_i, *_0, *_k)$	$i, k \in \mathbf{N}$

Proof. Observe that $\to_{\beta_e} \subseteq \twoheadrightarrow_{\beta}^{+}$, that $\lambda C^{\infty} \models \mathsf{SN}(\beta)$, that morphisms of specifications preserve strong normalization and that H may be extended to a morphism of specifications $H^{\Delta} : \mathbf{S}^{\Delta} \to C^{\infty}$ by setting $H^{\Delta}(\bullet_s) = H(s)$ for every $s \in \mathcal{S}_{\bullet}$.

We now turn to the existence of $\beta\eta$-long normal forms.

Definition 13.

1. *The map $\mathsf{Inf} : \mathcal{W} \to \mathcal{T}$ is defined by induction on the length—and* not *the structure—of terms as in Figure 2.*
2. *The $\beta\eta$-long normal form of a term M in context Γ, written $\mathsf{LNF}_{\Gamma}(M)$, is defined as $\mathsf{Inf}(\mu(\Gamma \vdash M))$.*

$$
\begin{array}{ll}
\mathsf{Inf}(s) & = s \\
\mathsf{Inf}(\Pi y : B.\, C) & = \Pi y : \mathsf{Inf}(B).\, \mathsf{Inf}(C) \\
\mathsf{Inf}(\lambda y : B.\, M) & = \lambda y : \mathsf{Inf}(B).\, \mathsf{Inf}(M) \\
\mathsf{Inf}(R) & = \lambda z_1 : \mathsf{Inf}(D_1).\ \ldots\ \lambda z_n : \mathsf{Inf}(D_n).\, y\ \mathsf{Inf}(Q_1)\ \ldots\ \mathsf{Inf}(Q_m)\ z'_1\ \ldots\ z'_n
\end{array}
$$

where in the last clause:

- $R = \mathsf{I}_{A_m}\ (\ldots(\mathsf{I}_{A_1}\ (\mathsf{I}_{A_0}\ y)\ Q_1)\ \ldots\ Q_m)$;
- $A_m = \Pi z_1 : D_1.\ \ldots\ \Pi z_n : D_n.\, E$ with E not a product;
- $z'_i = \mathsf{Inf}(\mathsf{I}_{D_n}\ z_i)$ for $i = 1, \ldots, n$.

Fig. 2. $\beta\eta$-LONG NORMAL FORMS

The next result shows the existence of $\beta\eta$-long normal forms.

Proposition 10. *Assume* $\lambda\mathbf{S}^{\Delta} \models \mathsf{WN}(\beta_e)$. *If* $\Gamma \vdash M : A$ *then*

1. $\Gamma \vdash \mathsf{LNF}_\Gamma(M) : A$.
2. $\mathsf{LNF}_\Gamma(M) \in \Xi_\Gamma$.
3. $\mathsf{LNF}_\Gamma(M) =_{\beta\eta} M$.

Proof. It is enough to show that if $\Gamma \vdash^{\Delta} M' : A$, $M \in \mathcal{W}$ and $\Gamma \vdash |M'| : A$ then (1) $\Gamma \vdash \mathsf{Inf}(M') : A$ (2) $\mathsf{Inf}(M') \in \Xi_\Gamma$ (3) $\mathsf{Inf}(M') =_{\beta\eta} M'$.

The proof is by induction on the length of $M' \in \mathcal{W}$. (3) can be proved directly, without any reference to derivations. We treat (1) and (2) in the case where $M' = \mathsf{I}_{A_m}\ (\ldots(\mathsf{I}_{A_1}\ (\mathsf{I}_{A_0}\ y)\ Q'_1)\ \ldots\ Q'_m)$ and E not a product.

1. Then
   $\mathsf{Inf}(M') = \lambda z_1 : \mathsf{Inf}(D_1).\ \ldots\ \lambda z_n : \mathsf{Inf}(D_n).\, y\ \mathsf{Inf}(Q'_1)\ \ldots\ \mathsf{Inf}(Q'_m)\ z'_1\ \ldots\ z'_n$
   where $z'_i = \mathsf{Inf}(\mathsf{I}_{D_n}\ z_i)$ for $i = 1, \ldots, n$. By induction hypothesis and some elementary reasoning on derivations, one derives $\Gamma \vdash y\ \mathsf{Inf}(Q'_1)\ \ldots\ \mathsf{Inf}(Q'_m) : A$ and (these derivations are needed to form the λ-abstractions below)

$$
\begin{array}{l}
\Gamma \vdash \Pi z_1 : \mathsf{Inf}(D_1).\ \ldots\ \Pi z_n : \mathsf{Inf}(D_n).\, E : s_1 \\
\Gamma, z_1 : \mathsf{Inf}(D_1) \vdash \Pi z_2 : \mathsf{Inf}(D_2).\ \ldots\ \Pi z_n : \mathsf{Inf}(D_n).\, E : s_2 \\
\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
\Gamma, z_1 : \mathsf{Inf}(D_1),\ \ldots\ , z_{n-1} : \mathsf{Inf}(D_{n-1}) \vdash \Pi z_n : \mathsf{Inf}(D_n).\, E : s_n
\end{array}
$$

   with $s_1, \ldots, s_n \in \mathcal{S}$.
   By Thinning,

$$\Gamma, z_1 : \mathsf{Inf}(D_1),\ \ldots\ , z_n : \mathsf{Inf}(D_n) \vdash y\ \mathsf{Inf}(Q'_1)\ \ldots\ \mathsf{Inf}(Q'_m) : A$$

[6] I.e. $H : \mathcal{S} \to \mathcal{S}^\infty$ is a map such that:

1. for every $(s, s') \in \mathcal{A}$, $(H\ s, H\ s') \in \mathcal{A}^\infty$;
2. for every $(s, s', s'') \in \mathcal{R}$, $(H\ s, H\ s', H\ s'') \in \mathcal{R}^\infty$.

By induction hypothesis and (conversion) since $D_i =_{\beta\eta} \mathsf{Inf}(D_i)$,

$$\Gamma, z_1 : \mathsf{Inf}(D_1),\ \ldots\ , z_n : \mathsf{Inf}(D_n) \vdash z_i' : D_i$$

for $i = 1, \ldots, n$.
By repeated uses of (application),

$$\Gamma, z_1 : \mathsf{Inf}(D_1),\ \ldots\ , z_n : \mathsf{Inf}(D_n) \vdash y\ \mathsf{Inf}(Q_1')\ \ldots\ \mathsf{Inf}(Q_m')\ z_1'\ \ldots\ z_n' : E$$

By repeated uses of (abstraction),

$$\Gamma \vdash \lambda z_1 : \mathsf{Inf}(D_1).\ \ldots\ \lambda z_n : \mathsf{Inf}(D_n).\, y\ \mathsf{Inf}(Q_1')\ \ldots\ \mathsf{Inf}(Q_m')\ z_1'\ \ldots\ z_n' : A$$

2. First note that if $M \in \Xi_\Gamma$ and Γ, Δ is legal then $M \in \Xi_{\Gamma,\Delta}$. Thus by induction hypothesis,
   - $\mathsf{Inf}(Q_i') \in \Xi_{\Gamma, z_1:\mathsf{Inf}(D_1),\ \ldots\ , z_n:\mathsf{Inf}(D_n)}$ for $i = 1, \ldots, m$;
   - $z_j',\ \mathsf{Inf}(D_j) \in \Xi_{\Gamma, z_1:\mathsf{Inf}(D_1),\ \ldots\ , z_{j-1}:\mathsf{Inf}(D_{j-1})}$ for $j = 1, \ldots, n$.

   Since E is not a product,

$$y\ \mathsf{Inf}(Q_1')\ \ldots\ \mathsf{Inf}(Q_m')\ z_1'\ \ldots\ z_n' \in \Xi_{\Gamma, z_1:\mathsf{Inf}(D_1),\ \ldots\ , z_n:\mathsf{Inf}(D_n)}$$

   and thus

$$\lambda z_1 : \mathsf{Inf}(D_1).\ \ldots\ \lambda z_n : \mathsf{Inf}(D_n).\, y\ \mathsf{Inf}(Q_1')\ \ldots\ \mathsf{Inf}(Q_m')\ z_1'\ \ldots\ z_n' \in \Xi_\Gamma$$

6 The induction principle

As mentioned in the introduction, Dowek, Huet and Werner [8] define $\beta\eta$-long normal forms for the systems of Barendregt's λ-cube by appealing to a non-standard induction principle. As a further application of affiliated terms, we generalize their induction principle to a large class of normalizing PTSs.

Definition 14.

1. *A* term in context *is a pair $\Gamma \vdash M$ such that $\Gamma \in \mathcal{G}$, $M \in \mathsf{NF}(\beta\eta)$ and $\Gamma \vdash M : A$ is derivable for some $A \in \mathcal{T}$. The set of terms in contexts is denoted by $\mathcal{D}$.*
2. *The* immediate subterm relation $\lhd \subseteq \mathcal{D} \times \mathcal{D}$ *is defined by the clauses:*

$$\begin{array}{llll} \Gamma \vdash M & \lhd_1 & \Gamma \vdash M\ N & \Gamma \vdash N \lhd_2 \quad \Gamma \vdash M\ N \\ \Gamma, x : A \vdash M & \lhd_3\ \Gamma \vdash \lambda x : A.\, M & \Gamma \vdash A & \lhd_4\ \Gamma \vdash \lambda x : A.\, M \\ \Gamma, x : A \vdash B & \lhd_5\ \Gamma \vdash \Pi x : A.\, B & \Gamma \vdash A & \lhd_6\ \Gamma \vdash \Pi x : A.\, B \end{array}$$

3. *The* normal type relation $\blacktriangleleft \subseteq \mathcal{D} \times \mathcal{D}$ *is defined by*

$$(\Gamma \vdash M) \blacktriangleleft (\Gamma \vdash N) \ \Leftrightarrow\ [\ \Gamma \vdash N : M \wedge M \in \mathsf{NF}(\beta\eta) \wedge N \in \mathcal{B}\ ]$$

4. *The* λ-normal type relation $\blacktriangleleft_\lambda \subseteq \mathcal{D} \times \mathcal{D}$ *is defined by*

$$(\Gamma \vdash M) \blacktriangleleft_\lambda (\Gamma \vdash N) \ \Leftrightarrow\ [\ \Gamma \vdash N : M \wedge M \in \mathsf{NF}(\beta\eta) \wedge N \in \mathcal{N}_\lambda\ ]$$

5. *The relation* $\prec$ *is defined as* $\lhd \cup \blacktriangleleft$.
6. *The relation* $\prec_\lambda$ *is defined as* $\lhd \cup \blacktriangleleft \cup \blacktriangleleft_\lambda$.

The relation $\prec_\lambda$ is the one considered by Dowek, Huet and Werner [8]. It is however sufficient to reason about $\prec$.

Lemma 12. *$\prec_\lambda$ is well-founded iff $\prec$ is.*

Proof. Only the right-to-left implication is interesting. It follows from the observation that if $(\Gamma' \vdash M') \prec (\Gamma \vdash M) \blacktriangleleft_\lambda (\Gamma \vdash \lambda x : A.\, N)$ then $M = \Pi x : A.\, B$ and $(\Gamma' \vdash M') = (\Gamma \vdash A)$ or $(\Gamma' \vdash M') = (\Gamma, x : A \vdash B)$. In the first case, $(\Gamma' \vdash M') \prec (\Gamma \vdash \lambda x : A.\, N)$. In the second case, we proceed by a case analysis on N.

1. If $N \in \mathcal{B}$ then $(\Gamma, x : A \vdash B) \blacktriangleleft (\Gamma, x : A \vdash N) \prec (\Gamma \vdash \lambda x : A.\, N)$.
2. If $N \in \mathcal{N}_\lambda$ then $(\Gamma, x : A \vdash B) \blacktriangleleft_\lambda (\Gamma, x : A \vdash N) \prec (\Gamma \vdash \lambda x : A.\, N)$.
3. If $N \in \mathcal{N}_e \setminus \mathcal{B}$ then $B \in \mathcal{S}$ and there is no $(\Gamma'' \vdash M'')$ such that $(\Gamma'' \vdash M'') \prec_\lambda (\Gamma' \vdash M')$.

It follows that every infinite $\prec_\lambda$-chain either contains finitely many $\prec$-steps and hence an infinite $\blacktriangleleft_\lambda$-chain or yields an infinite $\prec$-chain. The first possibility is impossible because there cannot be two consecutive $\blacktriangleleft_\lambda$-steps and the second is impossible since it contradicts our assumption.

The induction principle is stated as follows.

Theorem 1 (Induction Principle).

$\prec$ is well-founded provided $\lambda\mathbf{S}^\Delta \models \mathsf{WN}(\beta_e)$.

Proof. It suffices to show that for every $(\Gamma \vdash M), (\Gamma \vdash N) \in \mathcal{D}$,

$$(\Gamma \vdash M) \prec (\Gamma' \vdash N) \;\Rightarrow\; \mu(\Gamma \vdash M) < \mu(\Gamma' \vdash N) \;(\&)$$

where $<$ is the strict subterm relation on $\mathcal{T}$. We treat some of the interesting cases:

– $(\Gamma \vdash M) \lhd_1 (\Gamma' \vdash N)$. Then $M \in \mathcal{B}$, $\Gamma = \Gamma'$ and $N = M\ P$. Necessarily $\mu(\Gamma \vdash N)$ will be of the form $\mathsf{I}_A\ (M'\ P')$ with $M' \in \mathcal{W}$, $|M'| = M$ and so $M' =_\beta M$. Now $M =_{\beta\eta} \mu(\Gamma \vdash M)$ and hence $M' =_{\beta\eta} \mu(\Gamma \vdash M)$. Necessarily $|\mu(\Gamma \vdash M)| \in \mathcal{B}$. Moreover, there exist $C, C' \in \mathcal{T}$ such that

$$\Gamma \vdash^\Delta \mu(\Gamma \vdash M) : C \qquad \Gamma \vdash^\Delta M' : C'$$

By Lemma 6, $C =_\beta C'$. By uniqueness of $\beta_e\eta_e$-normal forms, $\mu(\Gamma \vdash M) = M'$ and hence

$$\mu(\Gamma \vdash M) = M' < \mathsf{I}_A\ (M'\ P') = \mu(\Gamma \vdash N)$$

– $(\Gamma \vdash M) \blacktriangleleft (\Gamma' \vdash N)$. Then $\Gamma = \Gamma'$ and by Correctness of Types there exists $s \in \mathcal{S}$ such that $\Gamma \vdash M : s$. Hence $\Gamma \vdash^{\Delta} \mu(\Gamma \vdash M) : s$.
Necessarily $N \in \mathcal{B}$ and hence $|\mu(\Gamma \vdash N)| \in \mathcal{B}$ so $\mu(\Gamma \vdash N)$ will be of the form $\mathsf{I}_A\, N'$ with $\Gamma \vdash^{\Delta} \mathsf{I}_A\, N' : M$ and $A \in \mathcal{W}$. By generation, $A =_{\beta\eta} M$. We also have $M =_{\beta\eta} \mu(\Gamma \vdash M)$ and hence $A =_{\beta\eta} \mu(\Gamma \vdash M)$. In addition, there exists $s' \in \mathcal{S}$ such that $\Gamma \vdash A : s'$. By uniqueness of $\beta_e\eta_e$-normal forms, $A = \mu(\Gamma \vdash M)$ and hence $\mu(\Gamma \vdash M) < \mu(\Gamma \vdash N)$.

Elsewhere [5], J. Hatcliff, M.H.B. Sørensen and the author strengthen this induction principle to define CPS translations and to prove Expansion Postponement for Pure Type Systems.

7 Conclusion

Building up on previous work in the area, we have derived new results on the Existence and Uniqueness of Normal Forms for $\mathsf{PTS}_{\beta\eta}$s. Our main contributions are a criterion for the Uniqueness of Normal Forms for all $\mathsf{PTS}_{\beta\eta}$s and a proof of the existence of $\beta\eta$-long normal forms for most normalizing $\mathsf{PTS}_{\beta\eta}$s. Our results are of practical importance because many implementations of type theories such as Coq [3] and Lego [15] rely on Pure Type Systems that fall beyond Barendregt's λ-cube. Typically, these systems rely on a Calculus of Constructions with universes, for which our results apply. We are currently investigating whether our method extends to richer type theories, e.g. with inductive types and subtyping.

While the results of these paper apply to most Pure Type Systems of interest, Proposition 10 and Theorem 1 require the specification $\mathbf{S}$ to verify $\lambda\mathbf{S}^{\Delta} \models \mathsf{WN}(\beta_e)$ which is unsatisfactory. We conjecture that both results remain true under the weaker assumption $\lambda\mathbf{S} \models \mathsf{WN}(\beta)$.

References

1. H. Barendregt. Introduction to Generalised Type Systems. *Journal of Functional Programming*, 1(2):125–154, April 1991. 241, 241
2. H. Barendregt. Lambda calculi with types. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford Science Publications, 1992. Volume 2. 241, 241
3. B. Barras *et. al.* *The Coq Proof Assistant User's Guide. Version 6.2*, May 1998. 241, 258
4. G. Barthe. Expanding the cube. In W. Thomas, editor, *Proceedings of FOSSACS'99*, volume 1xxx of *Lecture Notes in Computer Science*. Springer-Verlag, 1999. To appear. 242, 251
5. G. Barthe, J. Hatcliff, and M.H.B. Sørensen. An induction principle for Pure Type Systems. Manuscript, 1998. 258
6. C. Cornes. *Conception d'un langage de haut niveau de representation de preuves: Récurrence par filtrage de motifs; Unification en présence de types inductifs primitifs; Synthèse de lemmes d'inversion.* PhD thesis, Université de Paris 7, 1997. 242

7. R. Di Cosmo. A brief history of rewriting with extensionality. Presented at the International Summer School on Type Theory and Term Rewriting, Glasgow, September 1996. 241
8. G. Dowek, G. Huet, and B. Werner. On the existence of long $\beta\eta$-normal forms in the cube. In H. Geuvers, editor, *Informal Proceedings of TYPES'93*, pages 115–130, 1993. 241, 241, 242, 242, 242, 249, 249, 249, 256, 257
9. H. Geuvers. The Church-Rosser property for $\beta\eta$-reduction in typed λ-calculi. In *Proceedings of LICS'92*, pages 453–460. IEEE Computer Society Press, 1992. 241, 241, 242, 246
10. H. Geuvers. *Logics and type systems*. PhD thesis, University of Nijmegen, 1993. 241, 241, 241, 242, 244, 245, 245, 246, 246, 247, 250
11. H. Geuvers and M.J. Nederhof. A modular proof of strong normalisation for the Calculus of Constructions. *Journal of Functional Programming*, 1(2):155–189, April 1991. 241
12. H. Geuvers and B. Werner. On the Church-Rosser property for expressive type systems and its consequence for their metatheoretic study. In *Proceedings of LICS'94*, pages 320–329. IEEE Computer Society Press, 1994. 241
13. N. Ghani. Eta-expansions in dependent type theory—the calculus of constructions. In P. de Groote and J. Hindley, editors, *Proceedings of TLCA'97*, volume 1210 of *Lecture Notes in Computer Science*, pages 164–180. Springer-Verlag, 1997. 241, 241, 242, 251
14. J.W. Klop. Term-rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–116. Oxford Science Publications, 1992. Volume 2. 242, 252
15. Z. Luo and R. Pollack. LEGO proof development system: User's manual. Technical Report ECS-LFCS-92-211, LFCS, University of Edinburgh, May 1992. 241, 258
16. R. Nederpelt. *Strong normalisation in a typed lambda calculus with lambda structured types*. PhD thesis, Technical University of Eindhoven, 1973. 241, 245
17. A. Salvesen. The Church-Rosser property for $\beta\eta$-reduction. Manuscript, 1991. 241, 241, 242

Normalization of Typable Terms by Superdevelopments

Zurab Khasidashvili[1, *] and Adolfo Piperno[2,**]

[1] Department of Mathematics and Computer Science, Ben-Gurion University of the Negev
Beer-Sheva 84105, Israel
zurab@cs.bgu.ac.il

[2] Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
piperno@dsi.uniroma1.it

Abstract. We define a class of *hyperbalanced* λ-terms by imposing syntactic constraints on the construction of λ-terms, and show that such terms are strongly normalizable. Furthermore, we show that for any hyperbalanced term, the total number of superdevelopments needed to compute its normal form can be statically determined at the beginning of reduction. To obtain the latter result, we develop an algorithm that, in a hyperbalanced term M, statically detects all inessential (or unneeded) subterms which can be replaced by fresh variables without effecting the normal form of M; that is, full garbage collection can be performed before starting the reduction. Finally, we show that, modulo a restricted η-expansion, all simply typable λ-terms are hyperbalanced, implying importance of the class of hyperbalanced terms.

1 Introduction

The termination of β-reduction for typed terms is one of the most studied topics in λ-calculus. After classical proofs of Tait [21] and Girard [8], many interesting proofs have been proposed in the literature, which can be classified according to different aspects of the problem. A first classification divides normalization proofs, which show the existence of a terminating reduction path, from strong normalization proofs, which are concerned with termination of every reduction path. On the other hand, normalization and strong normalization proofs for typed λ-calculi usually depend on the choice of a particular type system and on the choice of explicit or implicit typing. Finally, the relationships between normalization and strong normalization, possibly involving additional notions of

* This work was done while Zurab Khasidashvili was working at NTT Basic Research Laboratories, Atsugi, Japan

** Partially supported by MURST grants

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 260–283, 1999.

reduction, are analyzed in the literature to obtain strong normalization results from normalization proofs.

In this paper, new normalization and strong normalization proofs are proposed for typed λ-calculi in Curry style. Due to space limitations, we will not be able to review other strong normalization proofs in the literature. Anyway, these proofs, although often very different in nature, are all based on *types*, while the proofs presented in this paper are, instead, based on *reduction*. Our motivation is to give a better understanding of the reasons *why* a calculus terminates, rather than proving *that* it terminates.

To this aim, and to provide a technique to be possibly used for different type systems, the use of information derived from types is delayed as much as possible so that a strong normalization proof for a typed calculus comes out to have the following pattern: first, a class of untyped λ-terms, the *hyperbalanced* terms, is isolated and proved to have the strong normalization property; later, all typable terms are proved to be in that class. Such a 'program' is carried out in this paper for simply typed λ-calculus and is discussed for a calculus with intersection types.

Our strong normalization proof in obtained by a deep analysis of the structure of hyperbalanced terms. We introduce an *arrow-decoration* for hyperbalanced terms allowing us to predict created redexes along β-reductions. Arrows coming out of abstractions point to head-variable occurrences which these abstractions will substitute to form β-redexes; arrows are composed into *substitution paths*, and the lengths of (prefixes of) substitution paths indicate the 'creation degrees' of corresponding redexes. (In Section 7 we will briefly remark on the relationship between substitution paths and other concepts of paths in the λ-calculus [1].) By using the so called 'decreasing redex labelling lemma [16,15], strong normalization of hyperbalanced terms can be deduced from the fact that any β-step decreases the lengths of paths relative to the contracted redex and does not increase the lengths of other paths. A strong normalization proof for simply typed λ-calculus using a similar idea (but completely different technically) appears in [2].

Arrow-decoration of hyperbalanced terms is useful for other purposes too. For example, it allows us to develop a static garbage-collection algorithm; that is, we can detect all *inessential* [11] subterms in a hyperbalanced term M without any transformation of M, and inessential subterms can be replaced by fresh variables without effecting the normal form of M. Once garbage collection is done before starting to reduce M, there will be no need for run-time garbage collection any more. Furthermore, we use our static garbage-collection algorithm also in our normalization theorem for hyperbalanced terms, to compute (exact) lengths of normalizing reductions by superdevelopments [20,14]. That is, we show that the amount of superdevelopments needed to reduce a hyperbalanced term M to its normal form can be determined at the beginning of reduction just by analyzing the structure of M. Due to the parallel nature of superdevelopments, this result also gives a lower bound on the steps taken to normalize by a machine which is able to contract all redexes present in a superdevelopment simultaneously.

The paper is organized as follows. In Section 2 we introduce the class of hyperbalanced terms, and we prove a strong normalization theorem for such terms in Section 3. In Section 4 we develop a static garbage collection algorithm for hyperbalanced terms, and using it in Section 5 we show how to compute the lengths of normalizing reductions whose steps are superdevelopments (of hyperbalanced terms). The strong normalization theorem for simply typed terms is given in Section 6. The method has also been applied to the case of intersection types, but this is only sketched in the concluding Section 7.

2 Hyperbalanced λ-terms

The set Λ of λ-terms is generated using the grammar

$$\Lambda ::= x \mid (\lambda x.\Lambda) \mid (\Lambda_1\Lambda_2),$$

where x belongs to a countable set *Var* of term variables. Terms are considered modulo renaming of bound variables, and the *variable convention* [4] is assumed; namely, bound variables are always chosen to be different from free ones.

Conventions on the elimination of redundant parentheses in applications, as well as on grouping of consecutive λ-abstractions, are usually assumed on terms. It comes out that, for example, $\lambda xy.M$ is taken as an abbreviation for $\lambda x(\lambda y.M)$ and $M_1M_2M_3$ as that for $((M_1M_2)M_3)$. To express these conventions directly into the syntax of λ-terms, the following grammar is used with startsymbol Λ:

$$\begin{array}{lcl} \Lambda & ::= & S \mid \lambda x_1 \dots x_n.S \quad (n > 0); \\ S & ::= & x \mid (R_i) \quad (i > 0) \mid (Q_{i,j}) \quad (i, j > 0); \\ R_i & ::= & x\Lambda_1 \dots \Lambda_i; \\ Q_{i,j} & ::= & (\lambda x_1 \dots x_j.S)\Lambda_1 \dots \Lambda_i. \end{array} \tag{1}$$

Notation 1 Following (1), $M \propto N$ says that M is a syntactic component of N. Terms of the shape $Q_{i,j}$, respectively $\lambda x_1 \dots x_n.S$, respectively x or R_i, will be called *application terms*, *abstraction terms,* and *head variable terms*. In particular, in an application term $(\lambda x_1 \dots x_j.L)L_1 \dots L_i$, the functional part $\lambda x_1 \dots x_j.L$ will be called the *operator*, L will be called the *body*, $L_1, \dots, L_i$ will be called the *arguments*, and the variables $x_1, \dots, x_j$ will be referred to as the *binding variables*. Similarly, in a head variable term $xN_1 \dots N_j$, $N_1 \dots N_j$ will be called the *arguments* and x will be called the *head-variable*. Note that, if a term M contains an application subterm N, the operator of N is not a syntactic component of M, while arguments in N are syntactic components of N itself, hence of M.

Consider the set of λ-terms whose generic element is such that every application subterm occurring in it has as many binding variables as arguments.

Definition 1 (BALANCED TERMS) Define the set $\widehat{\Lambda} \subset \Lambda$ of *balanced* terms as the set of terms generated using the grammar:

$$\begin{array}{lcl} \widehat{\Lambda} & ::= & S \mid \lambda x_1 \ldots x_n.S \quad (n > 0); \\ S & ::= & x \mid (R_i) \quad (i > 0) \mid (Q_i) \quad (i > 0); \\ R_i & ::= & x\widehat{\Lambda}_1 \ldots \widehat{\Lambda}_i; \\ Q_i & ::= & (\lambda x_1 \ldots x_i.S)\widehat{\Lambda}_1 \ldots \widehat{\Lambda}_i. \end{array}$$

The main definition of this section follows. It asks a term to preserve the property of being balanced during reduction. The definition requires an additional condition on the shape of application subterms.

Definition 2 *(i)* An application subterm $N \equiv (\lambda x_1 \ldots x_n.P)N_1 \ldots N_n$ of a balanced term $M \in \widehat{\Lambda}$ is *semi-hyperbalanced* iff whenever N_1 has k (≥ 0) initial abstractions, every free occurrence of x_1 in P has exactly k arguments.

(ii) (Hyperbalanced terms) Define the set $\widehat{\widehat{\Lambda}} \subset \widehat{\Lambda}$ of *hyperbalanced* terms as the greatest set of balanced terms such that

1. $\widehat{\widehat{\Lambda}}$ is closed with respect to β-reduction;
2. for any $M \in \widehat{\widehat{\Lambda}}$, every application subterm in M is semi- hyperbalanced.

Example 1 The term $\lambda x_0((\lambda x_1 x_2.(x_1(x_1 x_2)))\ (\lambda x_3.x_3)\ x_0)$ is hyperbalanced. On the other hand, $\lambda x_0((\lambda x_1 x_2.(x_1(x_2 x_0)))\ (\lambda x_3.x_3)\ x_0)$ is not hyperbalanced since it reduces to a term,

$$\lambda x_0(\underline{(\lambda x_2.((\lambda x_3.x_3)\ (x_2 x_0)))\ x_0}),$$

whose underlined redex is not semi-hyperbalanced. The term $\Omega = (\lambda x.xx)(\lambda x.xx)$ is an example of a (non-normalizable!) balanced term whose reducts all remain balanced but which is not hyperbalanced.

Notation 2 We mainly follow [4,5]. We write $M \xrightarrow{u} N$ or $M \to N$ if N is obtained from N by contracting a β-redex u in M. And $N \subseteq M$ denotes that N is a subterm occurrence in M. We use w, v to denote application subterms, as well as redexes.

In the next section, a strong normalization theorem for hyperbalanced terms will be established.

3 Strong normalization

In order to statically detect whether a λ-occurrence in a hyperbalanced term will be consumed in a β-reduction, we need to anticipate the 'potential locations' where it may be consumed, i.e., to find head-variables which it may substitute to form β-redexes. For that sake, we now introduce an *arrow-decoration* of hyperbalanced terms.

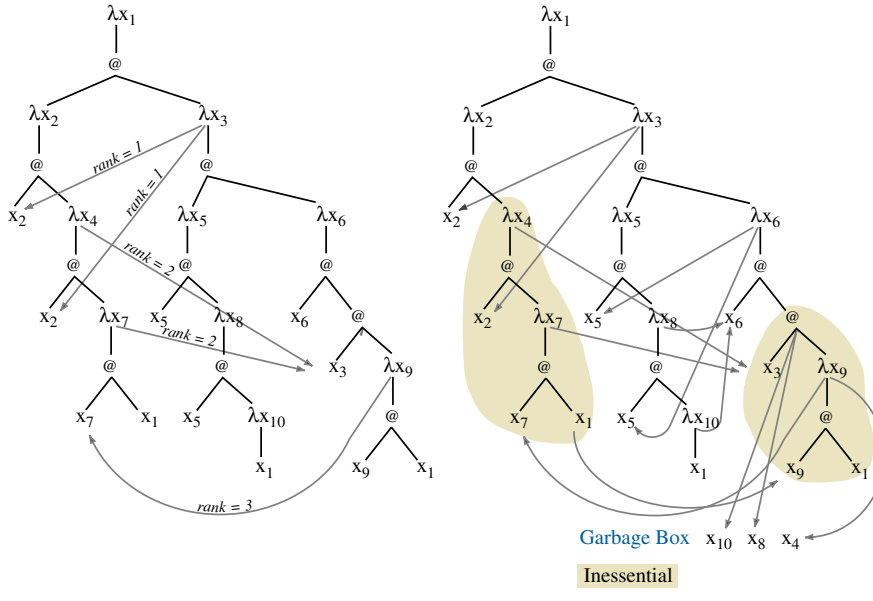


Fig. 1. Arrow-decoration and static garbage collection.

Arrow-Decoration Algorithm: Let $M \in \widehat{\widehat{\Lambda}}$.

Step 1: For any application subterm $w \equiv (\lambda y_1 \dots y_l.N)M_1 \dots M_l$ in M, draw arrows connecting every M_i with all free occurrences of y_i in N (if any), $i = 1, \dots, l$. The *rank* of all such arrows is 1, and the arrows are *relative* to w. If u is the β-redex *corresponding* to w (i.e., $u \equiv (\lambda y_1.\lambda y_2 \dots y_l.N)M_1$), arrows coming out of M_1 are also called *relative* to u.

Further, for any $k \geq 2$, do (until applicable):

Step k: For any head-variable subterm $zN_1 \dots N_m$ such that z has received an arrow κ of rank $k-1$ from an abstraction $L \equiv \lambda z_1 \dots z_m.P$ in the $(k-1)$th step, draw arrows connecting every N_j with all free occurrences of z_j in P, $j = 1, \dots, m$. The rank of any such arrow ι is k. Call ι a *successor* of κ, and call κ the *predecessor* of ι. If κ is relative to an application subterm $w \subseteq M$ (respectively to a β-redex u), then so is ι.

Example 2 The arrow-decoration algorithm is illustrated on Figure 1. In the left figure, only arrows relative to the topmost application subterm (which coincides with its corresponding β-redex) are drawn, while the right figure shows full arrow-decoration of the same term (with some extra information which will be explained later).

Call the arguments of application subterms in M, as well as the arguments of head-variables that receive arrows from λ-abstractions, *mobile subterms*; furthermore, call any mobile subterm from which no arrows are coming out *fictitious*.

For any non-fictitious mobile subterm N, call the target variable x of an arrow ι coming out of N a *matching* variable for N, written as $N \mapsto x$. We call a maximal sequence of arrows $\iota_1, \ldots, \iota_r$ a *substitution path* in M if ι_{i+1} is a successor of ι_i. The rank of an arrow ι will be denoted by $rank(\iota)$.

Remark 1 Let us note immediately that an arrow $\iota : N \mapsto x$ in a term $M \in \widehat{\widehat{\Lambda}}$, relative to an application subterm $w \subseteq M$, is determined solely by some of the nodes of w: the nodes of M that are outside w, and the nodes of w that are in the components not containing N or x, or are in the component containing N but are inside N, do not matter (thus ι does *not* depend on what the actual subterm N is). Indeed, if $rank(\iota) = 1$, then this is immediate, and if ι is a successor of $\kappa : E \mapsto y$, the only information needed to draw ι is to know that E is an abstraction which binds x, and y is the head-variable occurrence for which N is an argument. Furthermore, note here that if ι has a successor $\kappa' : E' \mapsto y'$, then E' is a proper subterm of E, which implies that all paths are finite and that the arrow-decoration algorithm terminates. Finally, note that for any arrow $\iota : N \mapsto x$ relative to w, either N or x is in the body of w, while the other end of ι is in an argument of w.

In [11] a concept of *descendant* is introduced to trace subterms along β-reductions. In a β-step $M \xrightarrow{u} M'$, where $u \equiv (\lambda x.L)N$, the contracted redex u does not have a residual, but it has a descendant: the descendant of u, as well as of its operator $\lambda x.L$ and its body L, is the contractum of u. The descendants of free occurrences of x in L are the substituted occurrences of N. The descendants of other subterms of M are intuitively clear. This definition immediately extends to (many-step) β-reductions. We refer to [13] for a more precise definition of descendants for *Expression Reduction Systems* (ERSs) in general. In the next section, we will use descendants to develop our static garbage-collection algorithm for hyperbalanced terms.

Definition 3 Let $M \xrightarrow{u} M'$, let $u \equiv (\lambda y.P_0)P$, and let $\iota : N \mapsto x$ be an arrow in M. We define the *u-residuals* of ι in M' by a case analysis:

(a) If ι is relative to u and $rank(\iota) = 1$ (i.e, if $N = P$ and $x = y$), then ι *collapses*, and it does not have u-residuals.

(b) If both N and x are in P, then for any pair of u-descendants N' and x' of N and x that are in the same u-descendant P' of P, $\iota' : N' \mapsto x'$ is a u-residual of ι in M'. (Thus in this case ι has as many u-residuals as the number of free occurrences of y in P_0.)

(c) If ι is relative to u and $n = rank(\iota) \geq 2$, then ι is a successor of an arrow $\kappa : L \mapsto z$ of rank $n - 1$, where x is bound by the top λ of $L \subseteq M$ and z is the head-variable of the application subterm for which N is an argument. In this case, the unique u-residual ι' of ι is defined by induction on n (as an arrow $N' \mapsto x'$ where N' and x' are some u-descendants of N and x) as follows:

(c_1) $n = 2$. Then $L = P$, and z is a free occurrence of y in P_0 (see Figure 6 in the appendix); and if x' is the descendant of x that is in the copy of P that substitutes z during contraction of u and N' is the only u-descendant of N, then $\iota' : N' \mapsto x'$ is by definition the only u-residual of ι in M' (even if x may have u- descendants other than x').[3]

(c_2) $n = 3, 5, \ldots$. Then x is in P_0 and $N \subseteq P$ (see Figure 7 in the Appendix). If $\kappa' : L' \mapsto z'$ is the only u-residual of κ in M' (where L' and z' are u-descendants of L and z), then N has a u-descendant N' that is an argument of z' (N may have other u-descendants too), x has exactly one u-descendant x' (which is bound by the top λ of L'), and $\iota' : N' \mapsto x'$ is by definition the unique u-residual of ι.

(c_3) $n = 4, 6, \ldots$. Then x is in P and $N \subseteq P_0$ (see Figure 8 in the Appendix). If $\kappa' : L' \mapsto z'$ is the only u-residual of κ in M' (where L' and z' are u-descendants of L and z), then N has exactly one u-descendant N' (which is an argument of z'), x has a u-descendant x' that is bound by the top λ of L' (x may have other u-descendants too), and $\iota' : N' \mapsto x'$ is by definition the unique u-residual of ι.

(d) If N or x has no u-descendants, then we say that u *discards* ι, and ι does not have a u-residual in M'.

(e) Otherwise (if the situations in (a)-(d) do not apply), for any pair of u-descendants N' and x' of N and x, $\iota' : N' \mapsto x'$ is a u-residual of ι in M'.

The *ancestor* relation on arrows is the converse of the residual relation.

Lemma 1 Let $M \in \widehat{\widehat{\Lambda}}$ and let $M \xrightarrow{u} M'$. Then the residuals of arrows in M are arrows in M' (thus Definition 3 is correct), and the latter form all arrows in M' (i.e., no arrows are 'created').

Proof By a very tedious case analysis, in each case using Definition 3, the arrow-decoration algorithm, and Remark 1. For the details, see the Appendix.

The following corollary follows immediately from Lemma 1 (and its proof).

Corollary 1 Let $M \in \widehat{\widehat{\Lambda}}$ and $M \xrightarrow{u} M'$. Then:

1. If u discards an arrow κ in M, then it discards all of its successors as well.
2. If ι' is a successor of κ' in M', then the ancestor of κ' is the predecessor of the ancestor of ι' in M.
3. If ι is a successor of κ in M and κ collapses during u (thus $rank(\kappa) = 1$), then ι has a residual arrow ι' such that $rank(\iota') = 1$.

[3] In all pictures below, descendants of a subterm E will be denoted by E', E'', etc. (thus, for example, an occurrence of λx binds all free occurrences of $x', x'', \ldots$ in its body).

4. Any u-descendant of a fictitious mobile subterm in M remains fictitious in M'.

Now we can define a concept of residual for paths as follows: Let $M \xrightarrow{u} M'$, let $\pi : \iota_0, \iota_1, \ldots, \iota_k$ be a path in M, and depending on whether ι_0 is relative to u or not, let $\pi' : \iota'_1, \ldots, \iota'_m$ or $\pi' : \iota'_0, \iota'_1, \ldots, \iota'_m$ be a path in M' such that ι'_i is a u-residual of ι_i. Then we call π' a residual of π. It follows from Lemma 1 and Corollary 1 that any path in M' is a residual of a path in M. Furthermore, u-residuals of all paths relative to u are shorter than the ancestor paths in M, and the residuals of other paths are not longer than the ancestor paths in M. This is the reason why all hyperbalanced terms are strongly normalizable. To make a formal proof, we can, for example, use a well-known method for proving termination of rewriting from [16]; the formulation is closer to the one in [15].

Lemma 2 (Decreasing redex labelling [16,15]) Let A be a set of λ-terms, and let any β-redex in a term in A be assigned an element of a well-founded partial order so that, in any β-reduction step $M \xrightarrow{u} M'$ in A (i.e, $M, M' \in A$), any created redex in M' is assigned a strictly smaller element than that assigned to u, and the residuals of redexes in M are assigned elements equal or smaller than the elements assigned to their ancestors. Then there is no infinite reduction in A.

Let $M \xrightarrow{u} M'$ be a β-reduction step, where $u \equiv (\lambda x.P)Q$. As pointed out in [16], new β-redexes can be created in M' in three substantially different ways:

(1) *upwards*: when P is an abstraction and it becomes the operator of a new redex;
(2) *downwards*: when $P \not\equiv x$ and Q becomes the functional part of a new redex;
(3) *upwards*: when $u \equiv (\lambda x.x)Q$ (i.e., $P \equiv x$), $(\lambda x.x)QQ' \subseteq M$, and QQ' is a redex.

Note that the third type of creation is never the case in the setting of hyperbalanced terms.

Theorem 1 β-reduction is strongly normalizing on hyperbalanced terms.

Proof Let us assign to every β-redex $u \subseteq M \in \widehat{\widehat{\Lambda}}$ a *degree*, $deg(u) = (l, h)$, where l is the length of a longest substitution path relative to the application subterm w corresponding to u, and h is the number of symbols (or lambdas) in the pattern of w. Then a β-step $M \xrightarrow{u} M'$ can create:

(a) a redex v' of type (2) with $deg(v') = (l', h')$, and in this case $l' < l$;
(b) a redex v'' of type (1) with $deg(v'') = (l'', h'')$, in which case $l'' \leq l$ and $h'' < h$.

Thus in any case every created redex has a strictly smaller degree w.r.t. the lexicographic ordering on pairs (l, h). Further, it is easy to see that the degrees of residuals are not greater than the degrees of their ancestor redexes, and the theorem follows from Lemma 2.

4 Static garbage collection

In [6], a theory of *needed* reduction is developed for the λ-calculus, extending a similar theory of normalization by neededness for orthogonal Term Rewriting Systems (OTRSs) in [10]. The main result of the theory is that every term not in normal form has a *needed* redex – one whose residual is contracted in every normalizing reduction of that term – and that repeated contraction of needed redexes normalizes every term having a normal form. In the λ-calculus, the leftmost-outermost redex is needed in any term, while finding a needed redex in a term in an OTRS is a difficult task, and much work has been done to improve and strengthen the theory of finding needed redexes initiated in [10].

An alternative concept of neededness, that of *essentiality*, was independently developed in [11] for the λ-calculus and later extended to orthogonal ERSs (OERSs) (see e.g., [9,12]). Essential are subterms that have descendants under any reduction of the given term, and in particular all essential redexes are needed. The main difference is that essentiality (but not neededness) makes sense for all subterms, e.g., for bound occurrences of variables, and we will take advantage of this fact in our algorithm for finding (all, not just one) essential subterms in hyperbalanced terms.

Definition 4 A subterm $N \subseteq M$ is called *essential* if it has a descendant under any reduction starting from M, and is called *inessential* otherwise [11].

The following lemma from [9] shows that inessential subterms do not contribute to the normal form of a term in an OERS, and they can be replaced with arbitrary terms without effecting the normal form. Inessential subterms of a term form 'garbage', and the lemma states that the garbage can safely be collected, i.e., replaced say by fresh variables.

Lemma 3 Let $N_1, \ldots, N_n$ be non-overlapping inessential subterms in N, in an OERS, and let M be obtained from N by replacing $N_1 \ldots N_n$ with $M_1 \ldots M_n$. Then $M_1, \ldots, M_n$ are inessential in M. Furthermore, M has a normal form iff N does, and the normal forms coincide.

Below, $\|M\|_{lo}$ will denote the length of the normalizing leftmost-outermost reduction starting from $M \in \widehat{\widehat{\Lambda}}$.

Lemma 4 Let $M \in \widehat{\widehat{\Lambda}}$ and let $N \subseteq M$ be a mobile subterm.

(a) If N is fictitious, then N does not have a descendant in the normal form M'' of M (i.e., N is inessential).
(b) Otherwise, any descendant N'' of N in M'' (if any) coincides with a descendant of a matching head-variable for N in M''.

Proof By induction on $\|M\|_{lo}$. We can assume that M is not a normal form, and let $M \xrightarrow{u} M' \twoheadrightarrow M''$ be a normalizing leftmost-outermost reduction.

(a) If N is fictitious, so is any of its u-descendants by Corollary 1. Hence all u-descendants of N are inessential in M' by the induction assumption, implying that N is inessential in M (by Definition 4).

(b) Let N' be a u-descendant of N that has a descendant N'' in M''. Then, by (a), N' must have a matching variable x', and x' must be a u-descendant of a matching variable x for N by Lemma 1. By the induction assumption, N'' coincides with a descendant of x', hence with a descendant of x, and we are done.

Definition 5 Call a hyperbalanced term M *good* iff all fictitious mobile subterms in M, and only such subterms, are occurrences of a special variable which we denote by $\circ$ (thus $\circ$ cannot appear as a bound variable, and a redex in M can only erase an occurrence of $\circ$).

Lemma 5 The set of good terms is closed under β-reduction.
Proof Let M be good, and let $M \xrightarrow{u} M'$. In M', all occurrences of $\circ$ are descendants of occurrences of $\circ$ in M. Hence by Corollary 1, all occurrences of $\circ$ in M' are fictitious. For the converse, let N' be a fictitious mobile subterm in M'. It remains to be shown that N' is an occurrence of $\circ$. Suppose not. Since N' is fictitious, there is an arrow $\kappa' : E' \mapsto x'$ in M' such that N' is an argument of x', and there is no arrow coming out of N'. By Lemma 1, κ' is a u-residual of an arrow $\kappa : E \mapsto x$ in M, where x' is a u-descendant of x; and the corresponding (to N') argument N of x is not an occurrence of $\circ$. Since M is good, κ has a successor $\iota : N \mapsto y$ in M, and y has a u-descendant in M' (since $y \not\equiv \circ$); thus, by Definition 3 and Lemma 1, there is an arrow $\iota' : N' \mapsto y'$ in N' – a contradiction (since N' is fictitious).

Lemma 6 Let M be a good term. Then any inessential subterm $N \subseteq M$ is an occurrence of $\circ$. (The converse is also true, by Lemma 4.)
Proof By induction on $\|M\|_{lo}$. Let $M \xrightarrow{u} M'$ be a leftmost-outermost reduction step, and suppose on the contrary that N is not an occurrence of $\circ$. Since M is good, N has a u-descendant (by Definition 5). But none of the u-descendants of N are occurrences of $\circ$, hence they are essential in M' by the induction assumption (M' is good by Lemma 5), implying by Definition 4 that N is essential too – a contradiction.

By Lemmas 6 and 5, β-reduction does not introduce garbage in a good term; that is, (full) static garbage collection in a hyperbalanced term is enough, and there is no need for run-time garbage collection any more. This is not surprising since β-reduction does not introduce new symbols. The next theorem gives a full garbage-collection algorithm for hyperbalanced terms.

Theorem 2 Let M be a hyperbalanced term. Then its inessential subterms can be found by the following algorithm:

Step 1**:** Find in M all outermost fictitious mobile subterms $L^1_1, \ldots, L^1_{k_1}$ ($L^1_1, \ldots, L^1_{k_1}$ are disjoint). If $k_1 = 0$, then M does not have inessential subterms and the algorithm stops. Otherwise, continue with the second step.

Step $n\,(>1)$**:** Let

$$L^1_1, \ldots, L^1_{k_1}, \ldots, L^{n-1}_1, \ldots, L^{n-1}_{k_{n-1}}$$

be all inessential mobile subterms found during the first $n-1$ steps. Let $L^n_1, \ldots, L^n_{k_n}$ be all the outermost subterms among mobile subterms in M that are different from $L^1_1, \ldots, L^{n-1}_{k_{n-1}}$ and that do not have matching head-variables outside the subterms $L^1_1, \ldots, L^{n-1}_{k_{n-1}}$. If $k_n = 0$ then the algorithm terminates. Otherwise, continue with the next step.

Call all subterms L^i_j $(1 \le i \le m, 1 \le j \le k_i)$, where m is the number of steps of the algorithm on the input M, *computably inessential*. Then a subterm in M is inessential iff it is in a computably inessential subterm.

Before presenting a formal proof, let us illustrate the algorithm on an example. Consider the same hyperbalanced term as in Example 2 (see Figure 1, right). We observe that the rightmost shaded subterm, call it L^1_1, is (the only) fictitious mobile subterm (both 'imaginary arrows' coming out of it point to variables in the 'garbage box', which consists of all 'fictitious' binding variables). Thus the first step of the algorithm yields a computably inessential subterm L^1_1. Next we observe that the leftmost shaded subterm, L^2_1, is the outermost among mobile subterms whose matching variables are inside L^1_1. Thus the second step of the algorithm yields L^2_1. It remains to notice that the algorithm terminates with the third step as M does not contain a mobile subterm outside L^1_1, L^2_1 whose matching variables are inside L^1_1 or L^2_1. We conclude by Theorem 2 that a subterm $N \subseteq M$ is inessential in M iff it is a subterm of L^1_1 or L^2_1.

Proof (of Theorem 2) Since M has only a finite number of mobile subterms, the algorithm clearly terminates; i.e, m is finite. We first prove by induction on n that $L^n_1, \ldots, L^n_{k_n}$ are inessential for any $n \le m$.

Case $n = 1$**:** It is clear from Lemma 4 that $L^1_1, \ldots, L^1_{k_1}$ are inessential.

Case $n > 1$**:** Assume $L^1_1, \ldots, L^{n-1}_{k_{n-1}}$ are all inessential. Then the matching variables for $L^n_1, \ldots, L^n_{k_n}$ are inessential, since subterms of inessential subterms are inessential [9], and again by Lemma 4 $L^n_1, \ldots, L^n_{k_n}$ are inessential.

Now it remains to prove that for any subterm in M that is not inside a computably inessential subterm is essential. Let $M_\circ$ be obtained from M by replacing all computably inessential subterms by $\circ$. Since no occurrence of $\circ$ in $M_\circ$ can receive an arrow (because occurrences of $\circ$ are free variables), and since no mobile subterm (and in particular, no computably inessential subterm) in M receives an arrow, it is immediate from Remark 1 that for any occurrence N of $\circ$ in $M_\circ$, there is at least one path in $M_\circ$ ending at the head-variable for N, and there is no arrow going out of N. This is because any arrow going out of the corresponding mobile subterm in M points to a variable inside a computably

inessential subterm, by the definition of the algorithm, and such a variable does not have a corresponding variable in $M_\circ$. That is, all occurrences of $\circ$ in $M_\circ$ are fictitious, and the converse is also immediate from Remark 1. Thus $M_\circ$ is good, and by Lemma 6 all subterms different from occurrences of $\circ$ are essential in $M_\circ$. Therefore, by Lemma 3, all subterms in M outside the replaced (i.e., computably inessential) subterms are essential in M.

5 Computing the lengths of normalizing 1-reductions

A *(complete) superdevelopment* (see [20,14]) of M is a reduction sequence in which all redexes initially present in M and all upwards created redexes are contracted.

In this section it will be proved that, whenever $M \in \widehat{\widehat{\Lambda}}$, it is possible to predict the (exact) number of superdevelopments needed to normalize it.

To give a formal treatment of superdevelopments, the following notion of reduction is introduced: this is an adaptation of the classical notion of *1-Reduction* appearing in the Tait and Martin-Löf proof of the Church-Rosser theorem for β-reduction (see [4], §3.2.3).

Definition 6 (1-Reduction) Define the relation $\xrightarrow{1} \subseteq \widehat{\widehat{\Lambda}}^2$ as the least relation satisfying

(i) $M \xrightarrow{1} M$, if M is a β-normal form;
(ii) $T \xrightarrow{1} T' \Rightarrow \lambda x_1 \dots x_n.T \xrightarrow{1} \lambda x_1 \dots x_n.T'$;
(iii) $T_1 \xrightarrow{1} T_1', \dots, T_i \xrightarrow{1} T_i' \Rightarrow xT_1 \dots T_i \xrightarrow{1} xT_1' \dots T_i'$;
(iv) $S \xrightarrow{1} S', T_1 \xrightarrow{1} T_1', \dots, T_i \xrightarrow{1} T_i' \Rightarrow (\lambda x_1 \dots x_i.S)T_1 \dots T_i \xrightarrow{1} S'[T_1'/x_1, \dots, T_i'/x_i]$.

It is easy to see that, on hyperbalanced terms, superdevelopments 'coincide' with complete β_m*-developments*, where the β_m*-reduction* [15] is defined by the following rule (or rather, a rule-schema over i):

$$\beta_m : \ (\lambda x_1 \dots x_i.S)T_1 \dots T_i \to S[T_1/x_1, \dots, T_i/x_i]).$$

More precisely, we have the following lemma.

Lemma 7 Let $M \in \widehat{\widehat{\Lambda}}$. Then $M \xrightarrow{1} M'$ iff M' is obtained from M by a complete β_m-development of the set of β_m-redexes corresponding to the application subterms in M.

Below, for any $M \in \widehat{\widehat{\Lambda}}$, $M_\circ$ will denote the corresponding good term (obtained from M by replacing all outermost inessential subterms with occurrences of $\circ$). Further, the *norm* of $M \in \widehat{\widehat{\Lambda}}$ is the length of a longest substitution path in $M_\circ$. By Lemma 6, $(M_\circ)_\circ \equiv M_\circ$; thus the norms of M and $M_\circ$ coincide. (By Theorem 2, the norm of M can be computed statically.)

Lemma 8 Let $M \in \widehat{\widehat{\Lambda}}$. Then M is in β-normal form iff $M_\circ$ is, and $M \xrightarrow{1} M'$ implies $M_\circ \xrightarrow{1} M'_\circ$.

Proof Easy induction on the definition of $\xrightarrow{1}$.

Theorem 3 Let $M \in \widehat{\widehat{\Lambda}}$, and let n be its norm. Then exactly n 1-steps normalize M.

Proof By Lemma 8, it is enough to show that the length of the normalizing 1-reduction starting from $M_\circ$ is n. The latter follows immediately from Lemma 5 and from the fact that, by Lemma 7, every 1-reduction step decreases the norm of a good term exactly by 1 (β-steps in good terms do not discard any arrows).

It is shown in [12] that a shortest development of a set of redexes U, in a term in an OERS, can be constructed by reducing only 'locally essential' redexes [11] in U in an inside-out order, and local essentiality is statically decidable. Hence this algorithm can be used to construct 1-reductions with the minimal number of β-reduction steps.[4]

6 Simple Types

We will now prove that any λ-term typable in Curry type assignment system has a hyperbalanced η-expansion. Using the strong normalization theorem 1, a strong normalization result for typable terms will immediately follow by η-postponement [4].

In this section, the final letters of the greek alphabet $\sigma, \tau, \ldots$ will range over *simple (or Curry) types*, which are generated using the following grammar:

$$\sigma ::= \alpha \mid (\sigma \to \sigma), \tag{2}$$

where α ranges over a countable set of type variables. We call $Type_{\to}$ the set of types resulting from (2). As usual, $\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \tau$ is an abbreviation for $\sigma_1 \to (\sigma_2 \to (\cdots (\sigma_n \to \tau) \cdots))$. Note that a type σ always has the shape

$$\sigma_1 \to \sigma_2 \to \cdots \to \sigma_n \to \alpha,$$

for some type variable α and $n \geq 0$: in this case we say that the *functionality* of σ is n (notation: $f(\sigma) = n$). If $n > 0$, the type will be called an *arrow-type*. So a type is either a type variable or an arrow-type.

[4] In [12], an algorithm is obtained for constructing the shortest normalizing reductions in *Persistent* OERSs, which are OERSs where only a restricted form of redex-creation is possible, and developments in OERSs can be encoded as reductions in *non-creating* OERSs (where no redex-creation is possible). It is relevant to remark here that, because redex-creation is restricted in persistent OERSs, weak and strong normalization become decidable [12].

Definition 7 The *Curry type assignment system* ($\mathsf{TA}_{\to}$) proves statements (*typings*) of the kind $A \vdash_{\to} M:\sigma$, where A is a *basis* (a partial function from term variables to types), $M \in \Lambda$ is the *subject* and $\sigma \in \mathit{Type}_{\to}$ is the *predicate* of the typing. The system $\mathsf{TA}_{\to}$ consists of the following rules:

$$(\mathrm{Var})_{\to} \quad \frac{A(x) = \sigma}{A \vdash_{\to} x:\sigma}$$

$$(\to\mathrm{I})_{\to} \quad \frac{A\,[x:\varphi] \vdash_{\to} M:\psi}{A \vdash_{\to} \lambda x.M:\varphi \to \psi}$$

$$(\to\mathrm{E})_{\to} \quad \frac{A \vdash_{\to} M:\varphi \to \psi \quad A \vdash_{\to} N:\varphi}{A \vdash_{\to} MN:\psi}$$

M has a typing, or, equivalently, M is typable, iff there exists a basis A and a type σ such that $A \vdash_{\to} M:\sigma$. The set of terms typable in $\mathsf{TA}_{\to}$ will be denoted by $\Lambda_{\mathsf{TA}_{\to}}$. Moreover, $\mathcal{D}: A \vdash_{\to} M:\sigma$ denotes a derivation in $\mathsf{TA}_{\to}$ proving the typing $A \vdash_{\to} M:\sigma$.

Definition 8 (Extensional derivation [18,19]) A derivation $\mathcal{D}{:}A \vdash_{\to} M:\sigma$ is *extensional* if and only if every head-variable or application term appearing in $\mathcal{D}$ as the subject of a rule is not assigned an arrow type. We write $A \vdash^{\mathrm{E}}_{\to} M:\sigma$ if there exists an extensional derivation for $A \vdash_{\to} M:\sigma$.

Note that, whenever in an extensional derivation for a term M a syntactical component N of it has been assigned an arrow type of functionality k, then N *must* have exactly k initial abstractions.

Using a restricted notion of η-expansion, called *proper η-expansion*, we can modify the shape of a term without affecting its computational behaviour.

Definition 9 The relation $\xrightarrow{\hat{\eta}} \subseteq \Lambda^2$ of *one-step proper η-expansion* is the least contextually closed relation satisfying:

(i) $xT_1 \dots T_i \xrightarrow{\hat{\eta}} \lambda z.xT_1 \dots T_i z$, if $z \notin FV(xT_1 \dots T_i)$;
(ii) $(\lambda x_1 \dots x_j.S)T_1 \dots T_i \xrightarrow{\hat{\eta}} \lambda z.(\lambda x_1 \dots x_j.S)T_1 \dots T_i z$, if $z \notin FV(\lambda x_1 \dots x_j.S)T_1 \dots T_i$.

$\xrightarrow{\hat{\eta}}{\!\!\!\!\to}$ is the reflexive transitive closure of $\xrightarrow{\hat{\eta}}$.

Remark 2 Note that the previous definition does not allow abstraction terms to be expanded; it comes out that, as in [18], a proper η-expansion never introduces new β-redexes.

The following theorem has been proved in [18]. It shows that any typing in Curry type assignment system can be transformed into an extensional one.

Theorem 4 (EXTENSIONALITY [18]) If $A \vdash_{\rightarrow} M{:}\sigma$, then there exists a proper η-expansion M_σ of M such that $A \vdash^{\mathrm{E}}_{\rightarrow} M_\sigma{:}\sigma$.

In the previous theorem, a typing derivation was turned into an extensional one by means of proper η-expansions. We will now prove that every application subterm appearing in an extensional derivation is semi-hyperbalanced.

Lemma 9 Let $\mathcal{D}{:}A \vdash^{\mathrm{E}}_{\rightarrow} M{:}\sigma$ and let $N \equiv (\lambda x_1 \ldots x_n.P)Q_1 \ldots Q_n$ be an application subterm of M; then for $i = 1, \ldots, n$ if the functionality of the type assigned in $\mathcal{D}$ to Q_i is equal to k, each occurrence of the variable x_i appears in P followed by exactly k arguments.
Proof It is an easy induction on the structure of the extensional derivation $\mathcal{D}$.

Further, to be extensionally typed in Curry type assignment system is preserved by β-reduction: if $A \vdash^{\mathrm{E}}_{\rightarrow} M{:}\sigma$ and $M \xrightarrow{\beta} N$, then $A \vdash^{\mathrm{E}}_{\rightarrow} N{:}\sigma$ [18]. Thus we have immediately from Lemma 9 and Definition 2 that:

Corollary 2 $\Lambda^{\mathrm{E}}_{\mathsf{TA}_{\rightarrow}} \subseteq \widehat{\widehat{\Lambda}}$.

We are now able to prove the main results of this section.

Theorem 5 Every λ-term typable in Curry type assignment system strongly normalizes.
Proof Let $M \in \Lambda_{\mathsf{TA}_{\rightarrow}}$. Then, by Theorem 4 and Corollary 2, there exists $M' \in \widehat{\widehat{\Lambda}}$ such that $M \xrightarrow{\hat{\eta}}\!\!\!\!\rightarrow M'$. In addition, M' strongly normalizes by Theorem 1. The theorem follows by η-postponement (see [4], p. 386).

7 Conclusion

A new method for proving normalization of typed terms has been presented. It does not use type information directly but, rather, studies properties of untyped λ-terms which appear to be relevant in normalizing reductions. This immediately hints to the possible applicability of the technique to different type systems.

In effect, the method can be successfully applied to prove normalization of terms typable in the intersection discipline, but, due to lack of space, this result cannot be presented here. More precisely, it has been applied to *strict intersection* types (see [7,3]) using the assignment system described in [19], in which a notion of extensional derivation can be introduced. In such a scenario, in a way similar to Curry types, the number of superdevelopments needed to compute the normal form of a term M can be statically determined. It has to be noted that a typing derivation in the intersection type discipline may contain several subderivations of typings of the same term, namely, when the argument part of an application is required to be assigned an intersection of types. The definition of extensional derivation can be adapted to this case by allowing different expansions, when needed, for each copy of the derivation of a subterm. The arrow decoration

procedure is modified accordingly, working this time on derivations rather than on terms.

Note that, although developed for the purpose of finding lengths of normalizing 1-reductions, the static garbage-collection algorithm is important in its own right. For example, applying it at the beginning of a call-by-value reduction can eliminate the main cause for the inefficiency of call-by-value computation. Of course, the complexity of the algorithm w.r.t. β-reduction must be analyzed first.

Finally, we remark that there is a similarity between the substitution paths introduced in this paper and other concepts of paths in the λ-calculus [1], and it is worth investigating the precise relationship. In particular, we expect that every arrow of the form $\iota : \lambda x_1 \ldots x_m.s \mapsto x$, in a hyperbalanced term M, determines exactly m redex-families [17], where m is the *weight* of ι. Thus the length of the normalizing leftmost-outermost complete-family reduction starting from M coincides with the sum of weights of arrows in $M_\circ$ augmented by the number of redexes in $M_\circ$, and therefore it can be computed statically. The simplicity of paths in the case of hyperbalanced terms may even lead to a 'virtual β-normalization' techniques for hyperbalanced terms.

Acknowledgements We thank an anonymous referee for careful reading.

References

1. Asperti, A., Danos, V., Laneve, C., Regnier, L., Paths in the lambda-calculus. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, 1994.
2. Asperti, A., Mairson H.G., Parallel beta reduction is not elementary recursive. In Proc. of ACM Symposium on Principles of Programming Languages (POPL), 1998.
3. van Bakel, S., Intersection Type Disciplines in Lambda Calculus and Applicative Term Rewriting Systems. PhD thesis, Matematisch Centrum Amsterdam, 1993.
4. Barendregt, H., *The Lambda Calculus. Its syntax and Semantics (revised edition)*. North Holland, 1984.
5. Barendregt, H., *Lambda Calculi with Types*. in Handbook of Logic in Computer Science, Vol. 2, Oxford University Press, 1992.
6. Barendregt H.P., Kennaway J.R., Klop J.W., Sleep M.R., Needed Reduction and spine strategies for the lambda calculus. Information and Computation, 75(3):191-231, 1987.
7. Coppo, M., Dezani-Ciancaglini, M., An Extension of the Basic Functionality Theory for the λ-Calculus, *Notre Dame J. of Formal Logic*, 21(4):685-693, 1980.
8. Girard, J.-Y., Une extension de l'interprétation de Gödel à l'analyse, et son application à l'elimination des coupures dans l'analyse et la thèorie des types. In *Procs of 2nd Scandinavian Logic Symposium*, North-Holland, 1971.
9. Glauert, J.R.W., Khasidashvili, Z., Relative Normalization in Orthogonal Expression Reduction Systems. In: Proc. of the 4^{th} International workshop on Condi-

tional (and Typed) Term Rewriting Systems, CTRS'94, Springer LNCS, 968:144-165, 1994.
10. Huet G., Lévy J.-J.. Computations in Orthogonal Rewriting Systems. In: Computational Logic, Essays in Honor of Alan Robinson, J.-L. Lassez and G. Plotkin, eds. MIT Press, 1991.
11. Khasidashvili, Z., β-reductions and β-developments of λ-terms with the least number of steps. In: Proc. of the International Conference on Computer Logic, COLOG'88, Springer LNCS, 417:105-111, 1990.
12. Khasidashvili Z., On higher order recursive program schemes. In: Proc. of the 19^{th} International Colloquium on Trees in Algebra and Programming, CAAP'94, Springer LNCS, 787:172-186, 1994.
13. Khasidashvili, Z., On Longest Perpetual Reductions in Orthogonal Expression Reduction Systems. Submitted.
14. Klop, J.W., van Oostrom, V., van Raamsdonk, F., Combinatory reduction systems: introduction and survey. *Theoretical Computer Science*, 121:279-308, 1993.
15. Klop, J.W., Combinatory Reduction Systems. PhD thesis, Matematisch Centrum Amsterdam, 1980.
16. Lévy, J.-J., An algebraic interpretation of the $\lambda\beta\kappa$-calculus and a labelled λ-calculus. *Theoretical Computer Science*, 2:97-114, 1976.
17. Lévy, J.-J., Réductions correctes et optimales dans le λ-calcul. PhD thesis, Univerité Paris 7, 1978.
18. Piperno, A., Ronchi della Rocca, S., Type inference and extensionality. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, 1994.
19. Piperno, A., Normalization and extensionality. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society Press, 1995.
20. van Raamsdonk, F., Confluence and superdevelopments. In: Proc. of the 5^{th} International Conference on Rewrite Techniques and Applications, RTA'93, Springer LNCS, 690:168-182, 1993.
21. Tait, W.W., Intensional interpretation of functionals of finite type I. *J. Symbolic Logic*, 32:198-212, 1967.

8 Appendix: Proof of Lemma 1

In the pictures used to illustrate the proof, the application subterm w may have more than one argument; and in Figures 9, 10, 11, and 12 one of the two displayed components of w (not necessarily the left one) is its body.

Proof Let $u = (\lambda y.P_0)P$, and let $v \subseteq M$ be the application subterm corresponding to u.

(1) We first show that any u-residual $\iota' : N' \mapsto x'$ of an arrow $\iota : N \mapsto x$ in M relative to an application subterm $w \subseteq M$ is an arrow in M', by induction on $n = rank(\iota)$.

Case $n = 1$: We consider two subcases:

(i) $v = w$. Then $N \neq P$ (otherwise ι collapses by Definition 3.(a)), w has a unique u-descendant w' (see Figure 2), ι' is the unique u-residual of ι (Definition 3.(e)), and ι' is clearly an arrow of rank 1 relative to w' (by the arrow-decoration algorithm).

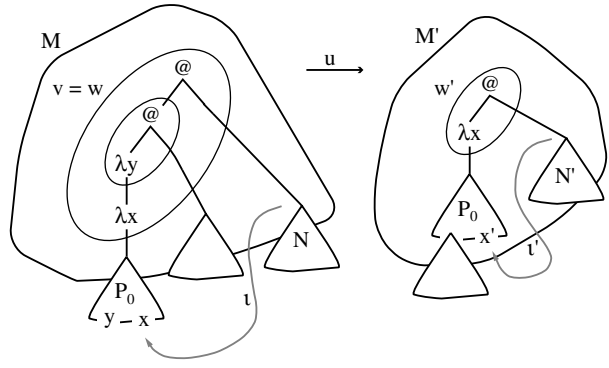


Fig. 2.

(ii) $v \neq w$. If $u \cap w = \emptyset$ or $w \subseteq P$, then w coincides with its u-descendant(s) w' (see Figure 3), and in this case the (corresponding) u-residual ι' of ι (cases (e) or (b) of Definition 3, respectively) is clearly an arrow of rank 1 relative to w', in M'. Further, if $w \subseteq P_0$, then w has exactly one u-descendant $w' = (P/y)w$ (again Figure 3), and the only residual ι' of ι (Definition 3.(e)) is an arrow of rank 1 relative to w' by Remark 1. Otherwise, u is in a component (i.e., an argument or the body) of w and there are two subcases:

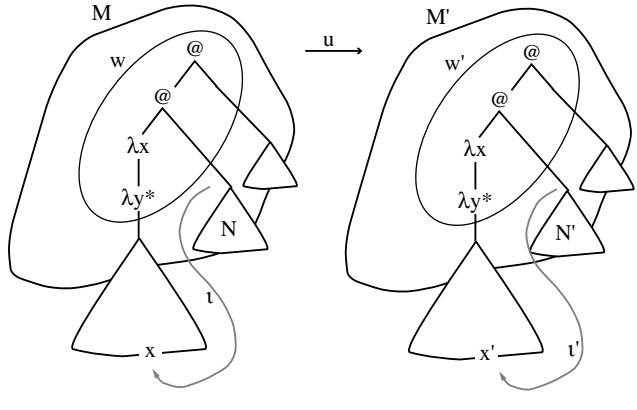


Fig. 3.

1. u is in an argument of w, in which case (Figure 3) the only residual ι' of ι (Definition 3.(e)) is an arrow of rank 1 relative to w' by Remark 1.
2. u is in the body of w, in which case (see Figure 4) ι' is an arrow of rank 1 relative to the unique u-descendant w' of w by the arrow-decoration algorithm (ι may have more than one u-residuals, Definition 3.(e)).

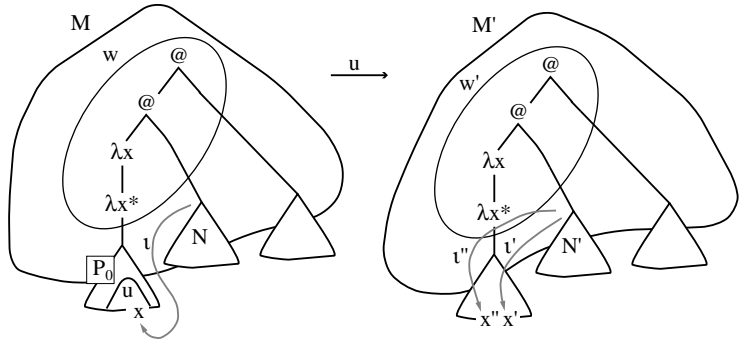


Fig. 4.

Case $n > 1$: In this case x is bound by the top λ of an abstraction $L \subseteq M$, and there is an arrow $\kappa : L \mapsto z$ of rank $n-1$ in M such that z is the head-variable of the head-variable subterm with N as an argument. We consider subcases depending on relative positions of w and u.

(i) $w = v$. If ι is not relative to u, the situation is as in Figure 5 (the arrow ι may as well have a reverse direction, i.e., start in the body of w and point to a variable in an argument, depending on whether $rank(\iota)$ is odd or even), and the only u- residual ι' of ι (Definition 3.(e)) is an arrow in M' by the arrow-decoration algorithm. Otherwise (i.e., if ι is relative u), we distinguish further subcases depending on $rank(\iota)$:

1. If $rank(\iota) = 2$, then we have the situation as in Figure 6: the predecessor κ of ι collapses (Definition 3.(a)), and it is immediate from the arrow-decoration algorithm that the only u-residual ι' of ι (Definition 3.(c_1)) is an arrow of rank 1 in M'.
2. If $rank(\iota) = 3, 5, \ldots$, then the situation is as in Figure 7: by the induction assumption, the unique u-residual κ' of κ (cases (c_1) or (c_3) of Definition 3) is an arrow in M', and the unique u-residual ι' of ι (Definition 3.(c_2)) is a successor of κ' by the arrow-decoration algorithm, hence ι' is an arrow in M'.
3. If $rank(\iota) = 4, 6, \ldots$, then the situation is as in Figure 8: by the induction assumption, the unique u-residual κ' of κ (Definition 3.(c_2)) is an arrow in M', and the unique u-residual ι' of ι (Definition 3.(c_3)) is a successor of κ' by the arrow-decoration algorithm, hence ι' is an arrow in M'.

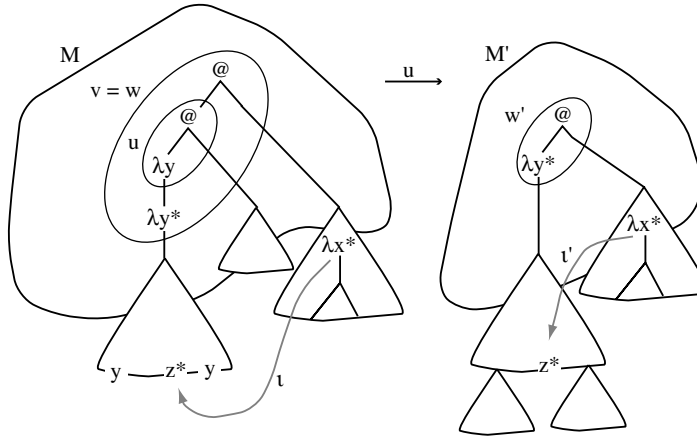


Fig. 5.

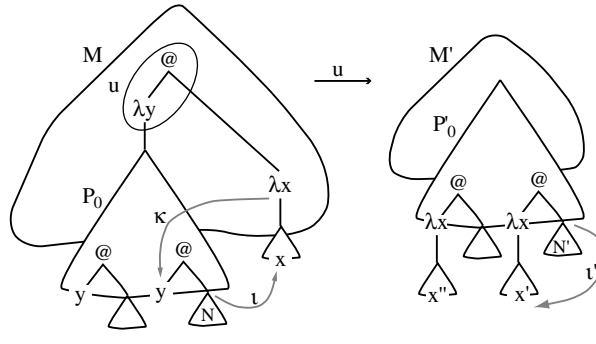


Fig. 6.

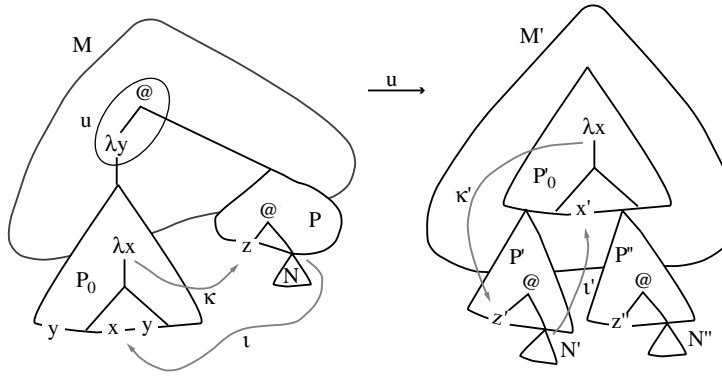


Fig. 7.

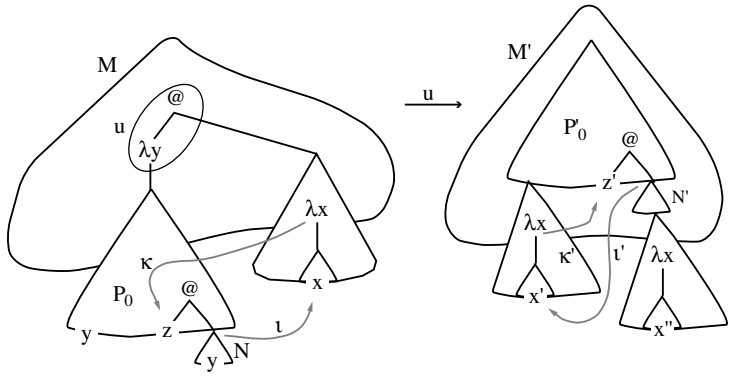


Fig. 8.

(ii) $w \neq v$. If $u \cap w = \emptyset$ or if $w \subseteq P$ then w coincides with its u-descendant(s) w'; further, ι has exactly one u-residual ι' in (each) w' (cases (e) or (b) of Definition 3, respectively), and ι' is an arrow relative to w' by Remark 1 (see Figure 9). If $w \subseteq P_0$, then w has exactly one u-descendant $w' = (P/y)w$, and the only residual ι' of ι (Definition 3.(e)) is an arrow relative to w' again by Remark 1 (see again Figure 9). Otherwise, u is in a component of w, and we consider three subcases:

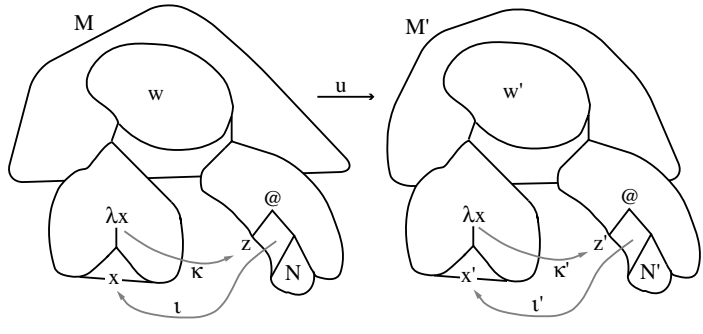


Fig. 9.

1. u is in the same component as L. If $u \cap L = \emptyset$ the situation is trivial (by Remark 1) and corresponds to Figure 9. If $L \subseteq P_0$, then the situation is again as in Figure 9: by the induction assumption, the only residual κ' of κ (Definition 3.(e)) is an arrow in M' which implies that the only residual ι' of ι (Definition 3.(e)) is an arrow too, in M'. If $L \subseteq P$, then u may duplicate L, as in Figure 10: then the u-residuals of κ (Definition 3.(b)) are arrows in M' by the induction assumption, implying that the u-residuals of ι (Definition 3.(b)) in M' are arrows too, by the arrow-decoration algorithm.

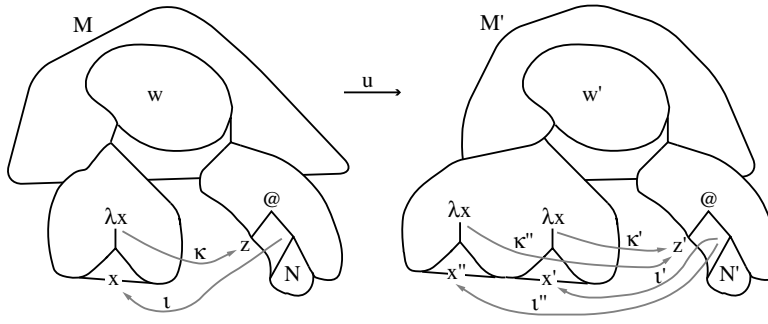


Fig. 10.

It remains to consider the case when $u \subseteq L$. If u does not duplicate x then the situation is as in Figure 9. Otherwise, the situation is as on Figure 11: κ has exactly one u-residual κ' (Definition 3.(e)) which is an arrow in M' by the induction assumption, and all u-residuals of ι (Definition 3.(e)) are successors of κ' by the arrow-decoration algorithm, hence they are arrows in M'.

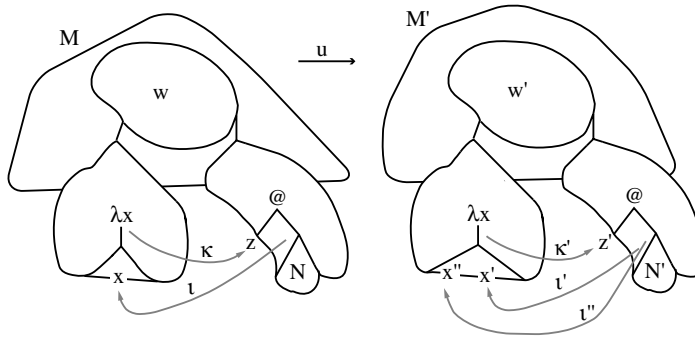


Fig. 11.

2. u is in the same component of w as N (or equivalently, z). If u does not duplicate the head-variable subterm corresponding to z then the situation is again as in Figure 9. Otherwise the situation is as in Figure 12: κ has as many u-residuals $(\kappa', \kappa'', \ldots)$ as the number of u-descendants of z (Definition 3.(e)), and each residual is an arrow in M' by the induction assumption; further, any u-residual $(\iota', \iota'', \ldots)$ of ι (Definition 3.(e)) is a successor of some u-residual of κ by the arrow-decoration algorithm, hence it is an arrow in M'.
3. u is in a component of w that does not contain neither L nor N. Then the situation is again as in Figure 9, and we use Remark 1.

(2) Now we show by induction on the rank $n = rank(\iota')$ of an arrow $\iota' : N' \mapsto x'$ in M' that $\iota : N \mapsto x$ is an arrow in M whose u-residual is ι', where x is the

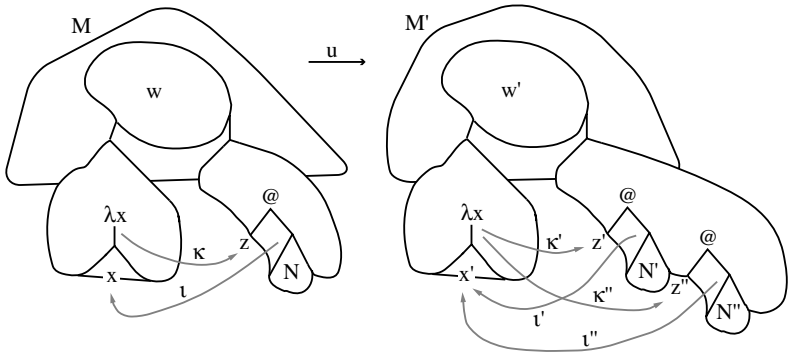


Fig. 12.

only u-ancestor of x' and N is the only u-ancestor of N' that is an abstraction (but is not the operator of a redex).[5] Let ι' be relative to an application subterm w'.

Case $n = 1$: We need to consider the following three subcases:

(i) w' is a u-descendant of an application subterm w such that $v \not\equiv w$. Then the situation is an in Figure 3 or Figure 4: ι' is a u-residual of ι (cases (b) or (e) of Definition 3), and ι is clearly an arrow in M (by the arrow-decoration algorithm).

(ii) u is a proper subterm of v and w' is the u-descendant of v. Then the situation is as in Figure 2: clearly ι is an arrow in M relative to v, and ι' is its u-residual by Definition 3.(e).

(iii) w' is 'created' by u. Then the situation is as on Figure 6: ι is a successor of κ in M (thus an arrow in M) and it is the u-ancestor of ι' by Definition 3.(c_1).

Case $n > 1$: Then ι' is a successor of an arrow $\kappa' : L' \mapsto z'$ in M', where N' is an argument of z' and the top λ of L' binds x'. By the induction assumption, κ' is a u-residual of an arrow $\kappa : L \mapsto z$ in M. Let κ be relative to w. We distinguish the following two subcases:

(i) $v \neq w$. Then depending on relative position of u w.r.t. w, L and z, the situation can be as in Figures 9, 10, 11 or 12. In each case, we already know that ι is a successor of κ (since ι' cannot have more than one u-ancestors); thus it is an arrow in M.

(ii) $v = w$. Then the situation can be as in Figures 8, 7 or 5, and again in each case we already know that ι is a successor of κ. This completes the proof.

[5] The only two different abstractions in M that may have the same descendant abstraction in M' are the operator $\lambda y.P_0$ and the body P_0 of the contracted redex u (when P_0 is an abstraction), and from the proof below it will be clear that there is no arrow $\iota' : N' \mapsto x'$ in M' such that N' is the u-descendant of P_0.

Subtyping Functional+Nonempty Record Types

Sergei Vorobyov

Max-Planck Institut für Informatik
Im Stadtwald, D-66123 Saarbrücken, Germany
sv@mpi-sb.mpg.de

Abstract. *Solving systems of subtype constraints* (or *subtype inequalities*) is in the core of efficient *type reconstruction* in modern object-oriented languages with subtyping and inheritance, two problems known *polynomial time equivalent*. It is important to know how different combinations of type constructors influence the complexity of the problem. We show the *NP-hardness* of the satisfiability problem for subtype inequalities between object types built by using simultaneously both the functional and the nonempty record type constructors, but without any atomic types and atomic subtyping.
The class of constraints we address is intermediate with respect to known classes. For pure functional types with atomic subtyping of a special non-lattice (*crown*) form solving subtype constraints is PSPACE-complete (Tiuryn 1992, Frey 1997). On the other hand, if there are no atomic types and subtyping on them, but the largest $\top$ type is included, then both pure functional and pure record (separately) subtype constraints are *polynomial time solvable* (Kozen, Palsberg & Schwartzbach 1994, Palsberg 1995), which is mainly due to the lattice type structure. We show that combining the functional and nonempty record constructors yields NP-hardness *without any atomic subtyping*, and the same is true for just a single type constant with a nonempty record constructor.

1 Introduction

Subtyping is a fundamental feature of the contemporary object-oriented languages. In this paper we address the inherent computational complexity of the *subtyping constraints satisfaction problem* for object types built by using simultaneously the functional and the nonempty record type constructors, but without any atomic types and subtyping relation on them. The motivation for subtyping record+functional types comes from type reconstruction and type-checking in Object-Oriented Programming, where an object (record) can be emulated by another object that has more refined methods (functions). To handle this phenomenon, the so-called *subsumption* rule with record+functional types is introduced. The primary importance of the problem stems from the fact that satisfiability of systems of subtype inequalities is known to be *polynomial time equivalent* to the *type reconstruction* problem for lambda and object terms (Kozen et al. 1994, Hoang & Mitchell 1995, Palsberg 1995).

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 283–297, 1999.

The *Satisfiability of Subtype Constraints Problem* (further, the SSCP for short) is defined as follows (J. Tiuryn calls it SSI, *Satisfiability of Subtype Inequalities*, (Tiuryn 1992, Tiuryn 1997)).

SSCP: *Given a finite set of subtype inequalities* $\left\{\sigma_i \leq \tau_i\right\}_{i=1}^{n}$ *between type expressions* σ_i, τ_i *containing free type variables, does there exist a substitution of type expressions for these variables making all inequalities true?* □

The SSCP is obviously parameterized by:

- the choice of type constructors involved in type expressions,
- the choice of subtyping rules,
- presence/absence of atomic types with a subtype relation on them.

The most common features involved in type construction considered in the literature are as follows:

1. the functional type constructor $\sigma \to \tau$,
2. the record type constructor $[l_1{:}\tau_1 \ldots, l_n{:}\tau]$,
3. atomic types with a subtype relation on them, e.g., $int \leq real$,
4. the largest type $\top$ (also denoted Ω [1]), and the least type $\bot$.

In Section 2 we survey known cases of SSCP for different types built by combinations of the type constructors above, well investigated in the literature. These include *partial types*, *object types*, *functional types with atomic subtyping* and different assumptions on atomic type orderings.

> *In this paper we consider the SSCP for types built by using simultaneously the functional and the* nonempty *record type constructors, but without atomic types and atomic subtyping.*

Remark 1. We *explicitly exclude the empty record* (playing the role of the $\top$ type for record types) from consideration for the following reasons.

Different combinations of type constructors and solvability of corresponding systems of subtype inequalities are investigated in the literature (Aiken, Wimmers & Lakshman 1994, Palsberg & O'Keefe 1995, Palsberg & Smith 1996, Trifonov & Smith 1996, Brandt & Henglein 1997). However, usually the $\top$ (largest, or Ω) and $\bot$ (least) types are introduced in the system from the very beginning. This immediately transforms the type structure into a lattice and makes it too coarse by introducing too many undesired, non-informative, or even 'senseless' solutions, like $\bot \to \top$. On the positive side of introducing $\top$, $\bot$ is the efficiency of the typechecking algorithms (usually polynomial[2]). But once we are satisfied with efficiency, the next step is to consider a more refined question: "we know this term is typable by *some type*, but is it typable by some '*meaningful*' type?"

[1] We keep both notations $\top$ and Ω to be consistent with cited papers.

[2] (Tiuryn 1992, Benke 1993, Pratt & Tiuryn 96, Tiuryn 1997) show that the lattice structure is helpful in getting polynomial algorithms.

At this stage one might wish to get rid of $\top$, $\bot$ types and all the types they generate. Another consequence of introducing $\top$, $\bot$ is that they make the system quite insensitive to combinations of different type constructors and make it quite impossible to analyze the relative costs of different features. Note that the introduction of $\top$, $\bot$ (turning types into lattices) is orthogonal to the whole line of research on non-lattice base types subtyping (Tiuryn 1992, Lincoln & Mitchell 1992, Hoang & Mitchell 1995, Pratt & Tiuryn 96, Tiuryn 1997). □

2 Related Work and Contribution of the Paper

We briefly survey the known results on complexity of the SSCP for different type systems in more detail.

Partial Types. (Kozen et al. 1994) consider the so-called *partial types* introduced by (Thatte 1988), built from a single top type Ω by using the single functional type constructor $\rightarrow$, i.e., defined by the grammar

$$\tau ::= \Omega \mid \tau_1 \rightarrow \tau_2$$

and the subtype relation $\leq$ defined by the rules

$$\begin{aligned} \tau &\leq \Omega \qquad \text{for any type } \tau, \\ \sigma \rightarrow \tau &\leq \sigma' \rightarrow \tau' \quad \text{iff } \sigma' \leq \sigma \text{ and } \tau \leq \tau'. \end{aligned} \tag{1}$$

(The latter is the standard subtyping rule for functional types.)

By using a nice automata-theoretic technique (Kozen et al. 1994) prove that the SSCP for these partial types is solvable in *deterministic time* $O(n^3)$. (O'Keefe & Wand 1992) were the first to show decidability of the problem by giving an exponential algorithm.

Object Types. (Palsberg 1995) considers the so-called *object types* built by using the single record type constructor, i.e., defined by the grammar

$$\tau \ ::= \ [l_1 : \tau_1, \ldots, l_n : \tau_n] \quad (\text{for } n \geq 0),$$

where the field labels l_i's are drawn from an infinite set. (Note that in case $n = 0$ we get the *empty record* type $[\,]$. Obviously, $\tau \leq [\,]$ for each type τ. Thus the empty record $[\,]$ plays the role of the largest type Ω or $\top$).

The subtype relation on these object types is defined by

$$\sigma \leq \tau \text{ iff } \Big(\tau \text{ has field } l : \rho \Rightarrow \sigma \text{ has field } l : \rho\Big). \tag{2}$$

Note that this subtyping rule is different from a better known rule

$$\sigma \leq \tau \text{ iff } \Big(\tau \text{ has field } l : \tau' \ \Rightarrow \ \sigma \text{ has field } l : \sigma' \text{ such that } \sigma' \leq \tau'\Big). \tag{3}$$

(Palsberg 1995) shows, also by using the automata-theoretic techniques similar to (Kozen et al. 1994), that the SSCP for the object types is decidable in deterministic time $O(n^3)$, and proves that the problem is PTIME-complete[3].

Functional Types with Atomic Subtyping. The SSCP for functional types defined starting from a set of *atomic types* with a *given subtype relation* on them extended to the set of all functional types by the standard subtyping rule (1) attracted most attention in the literature (Lincoln & Mitchell 1992, Tiuryn 1992, Pratt & Tiuryn 96, Benke 1993, Hoang & Mitchell 1995, Tiuryn 1997, Frey 1997). When the subtyping relation on atomic types is identity, the whole subtype relation is also identity, and the SSCP becomes the *unification problem for simple types*, known to be PTIME-complete (Dwork, Kanellakis & Mitchell 1984). (Lincoln & Mitchell 1992), improving (Wand & O'Keefe 1989), demonstrated that for some fixed ordering of atomic types the SSCP is NP-hard. (Tiuryn 1992, Pratt & Tiuryn 96) improved it further to PSPACE-hardness for some simple fixed orderings of atomic types called *crowns*. (Lincoln & Mitchell 1992, Tiuryn 1992) gave the NEXPTIME upper bound for the problem. Recently (Frey 1997) improved it to PSPACE. Thus the SSCP for functional types with atomic types is PSPACE-complete, in general. When the subtype relation on atomic types is a disjoint union of lattices (Tiuryn 1992) or a tree-like partial order (Benke 1993), then the SSCP is in PTIME, i.e., becomes tractable (Tiuryn 1992, Pratt & Tiuryn 96).

It is interesting to note that the partial types subtyping of (Kozen et al. 1994) and the object types subtyping of (Palsberg 1995) form lattices[4]. But the precise relation between the PTIME results of (Tiuryn 1992, Benke 1993, Pratt & Tiuryn 96, Tiuryn 1997) on the one hand, and the results of (Kozen et al. 1994, Palsberg 1995) on the other, seemingly, remains unexplored (do they 'imply' each other?).

Contribution of This Paper. The complexity of the SSCP for types built by using *simultaneously* the functional and the nonempty record type constructors (both in presence or absence of atomic types) remained unknown, although type systems combining functions and records are quite natural in object-oriented programming. The papers cited above concentrate either solely on functional or only on record types, and/or immediately introduce the $\top$ and $\bot$ type. The main result of this paper is that

> *Even without any atomic types the SSCP for functional+nonempty record types is NP-hard. The same holds for the types formed by the nonempty record type constructor+a single atomic type constant (or any other type constructor).* □

Thus, the absence of subtyping on atomic types does not lead to tractability, as contrasted to PSPACE/PTIME complexity for functional types with/without atomic subtyping, recall (Tiuryn 1992, Lincoln & Mitchell 1992, Tiuryn 1997).

[3] Reportedly, F. Henglein improved it to $O(n^2)$ by exploiting non-contravariance of record subtyping (J. Palsberg, private communication).

[4] With $glb(\sigma_1 \to \sigma_2, \tau_1 \to \tau_2) = lub(\sigma_1, \tau_1) \to glb(\sigma_2, \tau_2)$, etc.

Moreover, functions+records without atomic types, seemingly, do not allow to model the *crown structures* of (Tiuryn 1992, Pratt & Tiuryn 96), proved to be a useful uniform tool in showing NP and PSPACE lower bounds. Our proofs are not done by reduction from results of (Tiuryn 1992, Pratt & Tiuryn 96), we use a different method.

Our result shows that the automata-based deterministic polynomial time algorithms of (Kozen et al. 1994, Palsberg 1995) for subtyping partial types and object types (separately) cannot be combined to yield a polynomial deterministic algorithm for functional+nonempty record types. The intrinsic interplay between the nonempty record and the functional (or another) constructor adds to the computational complexity of the SSCP (even without atomic subtyping).

The remainder of the paper is organized as follows. In Section 3 we formally introduce the type system, the problem, and state the results. In Sections 4 – 7 we prove the Main Theorem and give some generalizations. Section 8–9 are devoted to discussion, related results, conclusions.

3 Functional and Record Types

Let $V = \{\alpha, \beta, \gamma, \ldots\}$ be an infinite set of *type variables* and $L = \{l_1, \ldots, l_n, \ldots\}$ be a finite or infinite set of *field labels*.

In this paper we consider the following types.

Definition 1 (Types). *Define:*

- ***Functional+Nonempty Record Types*** *by the grammar:*

$$\tau \ ::= \ V \mid \tau' \to \tau'' \mid [l_1 \colon \tau_1, \ldots, l_n \colon \tau_n], \tag{4}$$

  where $n > 0$ and l_i's are different labels from L.
- ***Pure Nonempty Record Types*** *by the grammar:*

$$\tau \ ::= \ V \mid [l_1 \colon \tau_1, \ldots, l_n \colon \tau_n], \tag{5}$$

  where $n > 0$ and l_i's are different labels from L. □

That is, types are constructed inductively, starting from type variables, by using the functional $\to$ and the nonempty record $[\ldots]$ type constructors. The subexpressions $l_i \colon \tau_i$ in the record construction are called *record fields*. Note that in the record construction the set of fields is *always nonempty*, i.e., we *explicitly exclude the empty record*.

Notation. Types will be denoted by Greek letters τ, σ, ρ, etc. To stress the outermost type constructor of a type we will sometimes superscript a type by the corresponding outermost constructor, like $\sigma^{\to}$ or $\tau^{[\,]}$. A substitution θ of types for variables is defined as usual and denoted by $\theta(\tau)$.

Definition 2. Subtype relation $\leq$ *is defined by the following standard rules:*

Reflexivity: $\tau \leq \tau$,
Transitivity: $\sigma \leq \tau$ *and* $\tau \leq \rho$ *imply* $\sigma \leq \rho$,
Functional: $\sigma \to \tau \leq \sigma' \to \tau'$ *iff* $\sigma' \leq \sigma$ *and* $\tau \leq \tau'$,
Record: $\sigma^{[\,]} \leq \tau^{[\,]}$ *iff for every field* $l\colon\tau'$ *in* $\tau^{[\,]}$ *there is a field* $l\colon\sigma'$ *in* $\sigma^{[\,]}$ *such that* $\sigma' \leq \tau'$.

A subtype judgment $\sigma \leq \tau$ *is* true *iff it is derivable by these rules.* □

Remark 1 *Note that a functional type is never a subtype of a record type, nor vice versa. All provable subtyping judgments are either* $\alpha \leq \alpha$ *for a type variable* α*, or of the form* $\sigma^{\to} \leq \tau^{\to}$*, or of the form* $\sigma^{[\,]} \leq \tau^{[\,]}$*, i.e., the subtyping relation is* strictly structural.

In fact, our lower complexity bound results are independent of the functional subtyping rule, which we adopt only as a standard one. The two essential things, our results really depend on, are: nonemptiness *of the record type constructor and the* strict structuredness *of the subtyping relation mentioned above.* □

Here is the problem we are interested in, both for functional+nonempty record, pure nonempty record types, nonempty object types.

Satisfiability of Subtype Constraints Problem (SSCP).

Given a finite system of subtype inequalities $\left\{\ \sigma_i \leq \tau_i\ \right\}_{i=1}^{n}$*, does there exist a substitution of types for variables occurring in* σ_i*'s,* τ_i*'s making all inequalities simultaneously true?* □

Our main result is as follows.

Main Theorem. *The SSCP is NP-hard for the following types:*

1. *nonempty record types with subtyping rule (3) and at least some other type constructor,*
2. *nonempty record types with subtyping rule (2) and possibly other type constructors,*

provided that:

- *record types are comparable with record types only,*
- *non-record types are comparable with non-record types only,*
- *there are* ≥ 2 *field labels.*

In particular, the SSCP is NP-hard for the following types:

1. *functional + nonempty record types,*
2. *nonempty record constructor + a single type constant,*
3. *nonempty record constructor + another type constructor,*
4. *Palsberg's system of object types with subtyping rule (2) without empty record.*

4 Proof of the Main Theorem

We first prove the claim for functional+nonempty record types, then in Section 5 sketch the modifications necessary for the case of pure nonempty record types with a constant and subtyping rule (2), and in Section 6 give the general pattern of the proof (due to an anonymous referee).

The proof is by reduction from the well-known NP-complete problem:

SATISFIABILITY. *Given a Boolean formula in Conjunctive Normal Form (CNF), does there exist an assignment of truth values to variables making the formula true?* □

We therefore proceed to representing truth values, clauses, and satisfiability in terms of types and subtyping.

4.1 Truth Values

Fix a type variable γ. Define the *truth values* $\mathbf{t}$ (true) and $\mathbf{f}$ (false) as types

$$\mathbf{t} \equiv_{df} [1{:}\,\gamma], \quad \mathbf{f} \equiv_{df} \gamma \to \gamma, \tag{6}$$

where 1 is an arbitrary label from L.

Clearly, neither $\mathbf{f} \leq \mathbf{t}$, nor $\mathbf{t} \leq \mathbf{f}$, because one is functional and the other is record (recall that the subtyping is strictly structural; see Remark 1). Note also that $\mathbf{t}$, $\mathbf{f}$ do not have common supertypes. This is because the empty record $[\,]$ is excluded from consideration by Definition 1. It seems tempting to get rid of the functional types altogether by defining $\mathbf{f} \equiv_{df} [2{:}\,\gamma]$, which seems as good as $\mathbf{f} \equiv_{df} \gamma \to \gamma$, however, in Section 6 we show that that this does not work.

4.2 Representing Clauses

A *clause* is a disjunction of literals. A *literal* is either a propositional variable, or a negation of a propositional variable. Without loss of generality we assume that a clause never contains complementary pairs of literals. Propositional variables $A_1, \ldots, A_k$ are represented by labels $1, \ldots, k$. A clause

$$C \equiv A_{i_1} \vee \ldots \vee A_{i_m} \vee \neg A_{j_1} \vee \ldots \vee \neg A_{j_n},$$

where $\{i_1, \ldots, i_m\} \cap \{j_1, \ldots, j_n\} = \emptyset$ and $\{i_1, \ldots, i_m\} \cup \{j_1, \ldots, j_n\} \subseteq \{1, \ldots, k\}$, is *represented as a type* (it is essential here that a clause does not contain complementary literals)

$$C^* \equiv_{df} [\, i_1{:}\,\mathbf{t}, \ldots, i_m{:}\,\mathbf{t},\ j_1{:}\,\mathbf{f}, \ldots, j_n{:}\,\mathbf{f}\,].$$

The following proposition relates satisfiability of clauses and subtype judgments.

Proposition 1. *A truth assignment* $\nu : \{A_1, \ldots, A_k\} \longrightarrow \{\boldsymbol{t}, \boldsymbol{f}\}$ *satisfies a clause* C *if and only if* $i{:}\,\nu(A_i)$ *occurs in* C^* *for some* $i \in \{1, \ldots, k\}$.

Proof. ν satisfies C iff for some propositional variable A_i one has:

- either $\nu(A_i) = \mathbf{t}$ and $C = \ldots \vee A_i \vee \ldots$; by definition of C^*, $C = \ldots \vee A_i \vee \ldots$ iff $i{:}\,\mathbf{t}$ occurs in C^*,
- or $\nu(A_i) = \mathbf{f}$ and $C = \ldots \vee \neg A_i \vee \ldots$; by definition of C^*, $C = \ldots \vee \neg A_i \vee \ldots$ iff $i{:}\,\mathbf{f}$ occurs in C^*. □

4.3 Satisfiability of a Propositional Formula

Definition 3. *The* translation *of a propositional formula in CNF, i.e., a conjunction of clauses (containing no complementary pairs of literals)*

$$\Phi \equiv \bigwedge_{i=1}^{l} C_i$$

is defined as a set of subtyping judgments

$$\Phi^* \equiv_{df} \left\{ C_i^* \leq \beta_i,\ \alpha \leq \beta_i \right\}_{i=1}^{l}, \tag{7}$$

where α, β_i *are fresh pairwise distinct type variables.*

A solution *to* Φ^* *is a substitution of types for free type variables making all subtyping judgments in* Φ^* *true.* □

The main technical result we need to prove the Main Theorem is as follows.

Lemma 1. *Φ is satisfiable if and only if Φ^* has a solution.*

Proof of ($\Rightarrow$)**.** Let an assignment $\nu(A_j) = v_j$ of truth values $v_j \in \{\mathbf{t}, \mathbf{f}\}$ ($1 \leq j \leq k$) satisfy Φ. Then, by record subtyping rule (3), the substitution

$$\alpha \leftarrow [1{:}\,v_1, \ldots, j{:}\,v_j, \ldots, k{:}\,v_k],$$
$$\beta_i \leftarrow [j{:}\,v_j \mid \nu(A_j) = v_j \text{ and } j{:}\,v_j \in C_i^*]$$

is a solution to Φ^*. Note that β_i's are substituted by *nonempty* records. □

Proof of ($\Leftarrow$)**.** Suppose, a type substitution θ is a solution to Φ^*. Construct the truth assignment ν by defining for every $j = 1, \ldots, k$

- $\nu(A_j) = \mathbf{t}$ if $j{:}\,\psi$ occurs in $\theta(\alpha)$ for some *record* type ψ, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

We claim that this ν satisfies $\Phi \equiv \wedge_{i=1}^{l} C_i$, i.e., ν satisfies the clause C_i for every $i = 1, \ldots, l$. Since θ is a solution to Φ^*, we have $\theta(C_i^*) \leq \theta(\beta_i) \geq \theta(\alpha)$.

By definition, C_i^* is a record type, hence $\theta(C_i^*)$ and $\theta(\beta_i)$ are also record types. Let $\theta(\beta_i) = [\ldots, j{:}\,\tau, \ldots]$ (nonempty!), where the type τ is:

1. either a record type (this happens when $C_i^* = [\ldots, j\colon \mathbf{t}, \ldots]$, consequently, when $C_i = \ldots \vee A_j \vee \ldots$),
2. or a functional type (this happens when $C_i^* = [\ldots, j\colon \mathbf{f}, \ldots]$, consequently, when $C_i = \ldots \vee \neg A_j \vee \ldots$).

Since $\theta(\beta_i) \geq \theta(\alpha)$, we have $\theta(\alpha) = [\ldots, j\colon \sigma \ldots]$, where the type σ is, respectively (corresponding to the two cases above):

1. a record type; in this case, by definition, $\nu(A_j) = \mathbf{t}$, consequently, $C_i = \ldots \vee A_j \vee \ldots$ is true in the truth assignment ν, or
2. a functional type; in this case, by definition, $\nu(A_j) = \mathbf{f}$, consequently, $C_i = \ldots \vee \neg A_j \vee \ldots$ is true in the truth assignment ν.

Thus, the assignment ν satisfies C_i for all $1 \leq i \leq l$, and the proof is finished. □

Since the solvability of a system of subtyping judgments is equivalent to the solvability of a single judgment (by using records), and the translation * of propositional formulas into systems of subtype constraints (7) is obviously polynomial time computable, we thus proved the first claim of the Main Theorem.

Remark 2. The presented proof makes transparent the following distinction between solving subtype inequalities *with and without* the empty record. If it is allowed, we may always guess this empty record as a supertype of any record type (leads to PTIME). If it is forbidden, we should make a nondeterministic choice between all possible nonempty subrecords (leads to NP).

Remark 3 ((Just Two Labels Suffice)). The proof above uses the number of record field labels equal to the number of propositions in an input propositional formula. In fact, just two labels, say 0 and 1, suffice. Instead of flat records $[1.v_1, \ldots, k.v_k]$ used in the proof we could have used *nested records* like

$$[0\colon [0\colon [0\colon u_{000}, 1\colon u_{001}], 1\colon [0\colon u_{010}, 1\colon u_{011}]], 1\colon [0\colon [0\colon u_{100}, 1\colon u_{101}], 1\colon [0\colon u_{110}, 1\colon u_{111}]]],$$
$$[0\colon [0\colon [0\colon v_{000}, 1\colon v_{001}], 1\colon [0\colon v_{010}, 1\colon v_{011}]], 1\colon [0\colon [0\colon v_{100}, 1\colon v_{101}], 1\colon [0\colon v_{110}, 1\colon v_{111}]]].$$

It is clear that two these record types are in the subtype relation iff for all $i, j, k \in \{0, 1\}$ one has $u_{ijk} \leq v_{ijk}$.

Thus the NP-hardness result holds already in the case of a two-label set. In contrast, for just one label the SSCP is deterministic polynomial (even linear) time decidable. This follows from the linearity of typability of an untyped term by simple types. □

Remark 4 ((Narrow Records Suffice)). Another generalization comes from the well-known fact that the particular case of *SATISFIABILITY*, *3-SATISFIABILITY*, restricted to formulas with at most three literals per clause is also NP-complete. It follows that the SSCP remains NP-hard for systems of constraints containing at most three fields per record. □

Remark 5 ((On Functional Constructor)). Our proof uses no assumptions about the $\rightarrow$ constructor, except for its difference from the record constructor; see Remark 1. Consequently, the functional subtyping rule (1) may be changed, e.g., made domain-covariant. This should be contrasted to the results of (Tiuryn 1992), which exploit the domain-contravariance of $\rightarrow$.

5 Satisfiability for Palsberg's Object Types without Empty Record is NP-Hard

If the empty record type [ ] is excluded from the type system of (Palsberg 1995) (by adding either a type constant or type variables to make the set of types nonempty), there is a jump in complexity (as compared with deterministic $O(n^3)$):

Lemma 2. *The SSCP for the pure nonempty record types (5) with the record subtyping rule (2) is NP-hard.*

Proof Sketch. The proof is similar to the proof of Lemma 1. It suffices to represent the truth values without functional type constructor as

$$\mathbf{t} \equiv_{df} [1\colon[1\colon\gamma]] \text{ and } \mathbf{f} \equiv_{df} [1\colon[2\colon\gamma]]. \tag{8}$$

Without the empty record type and with the subtyping rule (2), one has:

1. For every substitution θ, every supertype of $\theta([\ldots, j\colon\mathbf{t}, \ldots])$ (respectively, of $\theta([\ldots, j\colon\mathbf{f}, \ldots])$) is of the form $[\ldots, j\colon[1\colon[1\colon\tau]], \ldots]$ (resp., $[\ldots, j\colon[1\colon[2\colon\tau]], \ldots]$).
2. The types of the form $[\ldots, j\colon[1\colon[1\colon\sigma]], \ldots]$, $[\ldots, j\colon[1\colon[2\colon\tau]], \ldots]$ cannot have common subtypes.

Now modify the proof of ($\Leftarrow$) in Lemma 1 by defining for every $j = 1, \ldots, k$:

- $\nu(A_j) = \mathbf{t}$ if some $\theta(\beta_i)$ has a field $j\colon[1\colon[1\colon\tau]]$ for some type τ, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

The subsequent case analyses depend on whether a type has form $[1\colon[1\colon\tau]]$ or $[1\colon[2\colon\tau]]$. The remainder of the proof works without any substantial changes. □

It follows that with subtyping rule (2) deriving pure nonempty types is *more complicated* than deriving object types (unless $P = NP$). Both kinds of typability differ, and once we know whether a term has an object type, the next question to ask is whether a term has a nonempty record type. Both represent important interesting problems worth consideration.

6 General Proof Pattern

The proof of Section 4 for functional+nonempty record types with subtyping rule (3) is based on the simultaneous use of both functional and record type constructors in the definitions (6) of the truth value types **t** and **f**. The argument in Section 5, for nonempty object types with subtyping rule (2), suggests that the use of functional types *might also be avoided* with subtyping rule (3).

The first idea to avoid using functional types in the proof of Section 4, is to define truth values **t** and **f** as in (8). However, with this choice Lemma 1 *fails*, because for every conjunction of clauses Φ its translation Φ^* to the set of subtyping judgments defined by (7) *is satisfiable*. Indeed, let $\beta_i = C_i^*$ and

$\alpha = \left[\, i{:}\,[1{:}\,[1{:}\,\gamma, 2{:}\,\gamma]]\,\right]_{i=1}^{k}$ (where $A_1, \ldots, A_k$ are all variables in Φ). We see that with the choice (8) the proof of ($\Leftarrow$) in Lemma 1 breaks in the case analysis, where we construct the truth assignment by selecting *true* or *false* depending on whether a type has one of the two mutually exclusive structures.

To better explain this phenomenon, one of the anonymous referees suggested the following '*General pattern*' of the proofs of ($\Leftarrow$) in Lemmas 1 and 2. Both proofs rely on the existence of a property T of types (depending on the codings of $\mathbf{t}$, $\mathbf{f}$) such that for each $\rho \in \{\mathbf{t}, \mathbf{f}\}$, all substitutions θ, all types σ, τ with $\theta(\rho) \leq \tau \geq \sigma$ the following properties are satisfied:

$$\begin{cases} T(\tau) \;\Rightarrow (\rho = \mathbf{t} \wedge\; T(\sigma)), \\ \neg T(\tau) \Rightarrow (\rho = \mathbf{f} \wedge \neg T(\sigma)). \end{cases} \tag{9}$$

(Intuitively, $T(\xi)$ and $\neg T(\xi)$ stand for 'ξ represents *true* or *false*', respectively.)

When θ solves $\Phi^* = \{C_i^* \leq \beta_i \geq \alpha\}_{i=1}^{l}$, we define the truth assignment ν by:

- $\nu(A_j) = \mathbf{t}$ if $\theta(\alpha)$ contains a field $j{:}\,\sigma$ such that $T(\sigma)$, and
- $\nu(A_j) = \mathbf{f}$ otherwise.

This assignment ν satisfies $\bigwedge_{i=1}^{l} C_i$.

In the proof ($\Leftarrow$) of Lemma 1 we selected $T(x) =$ 'x *is a record type*', appropriate for the truth values encodings (6) and subtyping rule (3). Clearly, this choice satisfies (9).

In the proof ($\Leftarrow$) of Lemma 2 we selected $T(x) =$ 'x *is a subtype of* $[1{:}\,[1{:}\,\xi]]$ (for some type ξ)', appropriate for the truth values encodings (8) and subtyping rule (2). Clearly, this choice satisfies (9), because the empty record is excluded, and the rule (2) guarantees that the types $[1{:}\,[1{:}\,\phi]]$ and $[1{:}\,[2{:}\,\psi]]$ do not have a common subtype. However, as we saw above, these types *do have a common subtype* with subtyping rule (3).

It is clear that no choice of pure record types for $\mathbf{t}$, $\mathbf{f}$ in the presence of subtyping rule (3) satisfies (9). This can be proved by recursively constructing the lower bound for any two pure record types. Thus, *the use of nonempty record types jointly with functional types*[5] *is essential* in the proof of the Main Theorem in Section 4. This does not imply, however, that the problem is in PTIME, nor does it prevent that the problem (for pure record types) is also NP-hard (by a different proof). But to the author's knowledge, until now this problem is open.

7 Pure Nonempty Records with a Single Atomic Constant

If we allow a *single type constant*[6] c, such that $c \leq \tau$ and $\sigma \leq c$ are only possible for $\sigma = \tau = c$, then the proof of our main result from Section 4 works (slightly

[5] or any other type constant or constructor; see Section 7 below.

[6] Before we had only type variables and two type constructors $\rightarrow$, $[\,]$.

modified according to the 'General pattern' of Section 6) for the truth values defined by

$$\mathbf{t} \equiv_{df} c, \quad \mathbf{f} \equiv_{df} [1\colon c].$$

Indeed, these types satisfy all the needed properties (9) if we let $T(\xi) \equiv_{df}$ 'ξ equals c'. Exactly the same argument works for nonempty records + another type constructor (provided that types with different type constructors are incomparable).

It is interesting to compare this result with the results on complexity of functional types subtyping with subtyping on atomic types.

1. If the atomic subtyping is a lattice (which is the case of a single type constant) then the satisfiability of functional type constraints is PTIME decidable. It becomes PSPACE-complete for n-crowns ($n \geq 2$), (Tiuryn 1992).
2. In the pure nonempty record subtyping it is NP-hard already in the case of a single (atomic) type constant (which is clearly a lattice), in contrast to (Tiuryn 1992, Tiuryn 1997).

What are the precise lower and upper bounds for the SSCP in the case of pure record types in presence of a nontrivial atomic subtyping, lattice and non-lattice? A classification similar to (Tiuryn 1992, Benke 1993, Tiuryn 1997) would be interesting.

8 Upper Bound

(Kozen et al. 1994, Hoang & Mitchell 1995, Palsberg 1995) show that satisfiability of systems of subtype inequalities (for partial types or record types) is polynomial time equivalent to the type reconstruction problem (i.e., given a term can it be assigned some type?). By similar arguments we can prove that in any reasonable type system based on functional+record types with subtyping as defined in Definitions 1, 2 the type reconstruction problem is polynomial time equivalent to the SSCP, hence also is NP-hard.

Therefore, unlike the results of (Kozen et al. 1994, Palsberg 1995), which show deterministic cubic time tractability of the SSCP for either functional or record types (separately), our Main Theorem shows that subtyping and type reconstruction in presence of *both functional and nonempty record* types is NP-hard, i.e., presumably intractable. It follows that the automata-theoretic decidability techniques of (Kozen et al. 1994, Palsberg 1995) (for the functional and record subtyping, separately) do not carry over straightforwardly to the combined case of functional+nonempty record types. Extra effort is necessary even to establish decidability of the SSCP for functional+record types. Here we just claim without a proof the following complexity result

Theorem 2. *The SSCP for functional+nonempty record types of Definitions 1, 2 is in NEXPTIME.* □

Recall that NEXPTIME is the class of problems solvable by nondeterministic Turing machines in time $O(2^{n^k})$ for some fixed k, where n denotes the length of input. The proof of Theorem 2 is by a tedious pumping argument (outside the scope of this paper) showing that whenever an instance of SSCP of size n has a solution, then it necessarily has a solution of depth polynomial in n. Thus, given an SSCP instance, it suffices to nondeterministically guess a polynomially deep solution tree (forest), with the resulting tree size $O(2^{poly(n)})$, and to check that it is indeed a solution in deterministic exponential time. This proves the NEXPTIME membership. Of course, this is not a very efficient algorithm. It remains an open problem whether the SSCP for functional+record types is in PSPACE or NP. We believe that more sophisticated data structures, like DAGs and automata on DAGs, may lead to improvement of the above NEXPTIME upper bound to PSPACE, or even NP. It is also possible that the sophisticated techniques of (Tiuryn 1992) may raise the lower bound to PSPACE. This remains an intriguing subject for further investigations, we will report on elsewhere.

9 Conclusions

We presented the NP-lower bound for the *Satisfiability of Subtype Constraints Problem* (SSCP) for the types built by using simultaneously the functional $\rightarrow$ and the nonempty record $[\ldots]$ type constructors. Earlier research concentrated exclusively either on the SSCP for pure functional, or for pure record types, but not for both simultaneously, or else immediately included $\top$ and $\bot$ types turning the type structure into a lattice. Both in the case of the pure functional types (Lincoln & Mitchell 1992, Tiuryn 1992, Kozen et al. 1994) and pure record types (Palsberg 1995), deterministic polynomial time algorithms are possible. In the case of the pure functional types constructed from atomic types with nontrivial subtyping relation on them the SSCP, in general, is NP-hard (Lincoln & Mitchell 1992) and even PSPACE-hard (Tiuryn 1992) (and, in fact, PSPACE-complete (Frey 1997)). In contrast, our result shows that even without any atomic types, the SSCP for functional+nonempty record types is NP-hard. We give the NEXPTIME upper bound for the problem, but conjecture that by using more sophisticated data structures and more subtle arguments this upper bound can be improved to PSPACE (or even NP). It is also quite possible that the techniques of (Tiuryn 1992) may lead to the PSPACE lower bound.

All these topics constitute an interesting and promising direction for future research, together with the investigation of the related partial type systems with possible simplification of the SSCP, as suggested by results of (Kozen et al. 1994, Palsberg 1995).

We conclude by summarizing several earlier mentioned technical problems remaining open:

1. Improve the NP lower and/or the NEXPTIME upper bounds for the SSCP for functional+nonempty record types with subtyping rule (3).

2. We gave the NP lower bound for pure nonempty record types with a single atomic constant. Does the same hold without the constant, or does the complexity drop to PTIME?
3. The SSCP for Palsberg's object types without empty record is NP-hard. What is the upper bound?
4. Pure functional+nonempty record types seemingly do not allow for modeling crowns. Give a strict proof, so as to demonstrate that our results are independent of (Tiuryn 1992, Pratt & Tiuryn 96).
5. Results of (Tiuryn 1992, Tiuryn 1997) and (Kozen et al. 1994, Palsberg 1995, Brandt & Henglein 1997) on PTIME decidability of constraints of different kinds heavily exploit the fact that the type structure is a lattice. Could one construct a uniform polynomial time algorithm, which works for an arbitrary combination of type features, once types form a lattice?
6. What are the precise lower and upper bounds for the SSCP in the case of pure record types in the presence of a nontrivial atomic subtyping, lattice and non-lattice? A classification similar to (Tiuryn 1992, Benke 1993, Tiuryn 1997) would be interesting.
7. Does there exist a functional+record system of partial types, which: 1) has the PTIME decidable SSCP and type reconstruction problem, 2) types all normal forms, 3) all typable terms are SN, 4) a typable term never goes wrong (appropriately defined)? Could PTIME decidability of such a system be obtained by a generalization of the automata-based decision procedures of (Kozen et al. 1994, Palsberg 1995)?

Acknowledgments. Thanks to David McAllester for motivating this research, for fruitful discussions, careful proofreading, thoughtful and illuminating remarks. Thanks to the anonymous referees for reporting gaps in the proofs, careful reading, numerous constructive suggestions for improvements, sharp remarks. One of the referees suggested substantial generalizations of the main theorem and the general proof pattern of Section 6.

References

Aiken, A., Wimmers, E. & Lakshman, T. (1994), Soft typing with conditional types, *in* '21st ACM Symp. on Principles of Programming Languages (POPL'94)', pp. 163–173.

Benke, M. (1993), Efficient type reconstruction in presence of inheritance, *in* 'Mathematical Foundations of Computer Science'93', Vol. 711 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 272–280.

Brandt, M. & Henglein, F. (1997), Coinductive axiomatization of recursive type equality and subtyping, *in* 'Typed Lambda Calculi and Applications, TLCA'97, Nancy, France', Vol. 1210 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 63–81.

Dwork, C., Kanellakis, P. & Mitchell, J. (1984), 'On the sequential nature of unification', *J. Logic Programming* **1**, 35–50.

Frey, A. (1997), Satisfying subtype inequalities in polynomial space, *in* 'Static Analysis (SAS'97)', Vol. 1302 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 265–277.

Hoang, M. & Mitchell, J. C. (1995), Lower bounds on type inference with subtypes, *in* '22nd ACM Symp. on Principles of Programming Languages (POPL'95)', pp. 176–185.

Kozen, D., Palsberg, J. & Schwartzbach, M. I. (1994), 'Efficient inference of partial types', *J. Comput. Syst. Sci.* **49**, 306–324.

Lincoln, P. & Mitchell, J. C. (1992), Algorithmic aspects of type inference with subtypes, *in* '19th ACM Symp. on Principles of Programming Languages (POPL'92)', pp. 293–304.

O'Keefe, P. M. & Wand, M. (1992), Type inference for partial types is decidable, *in* 'European Symposium on Programming (ESOP'92)', Vol. 582 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 408–417.

Palsberg, J. (1995), 'Efficient inference of object types', *Information and Computation* **123**, 198–209. Preliminary version in LICS'94.

Palsberg, J. & O'Keefe, P. (1995), 'A type system equivalent to flow analysis', *ACM Trans. Progr. Lang. Sys.* **17**(4), 576–599. Preliminary version in POPL'95.

Palsberg, J. & Smith, J. (1996), 'Constrained types and their expressiveness', *ACM Trans. Progr. Lang. Sys.* **18**(5), 519–527.

Pratt, V. & Tiuryn, J. (96), 'Satisfiability of inequalities in a poset', *Fundamenta Informaticæ* **28**, 165–182.

Thatte, S. (1988), Type inference with partial types, *in* 'International Colloquium on Automata, Languages, and Programming (ICALP'88)', Vol. 317 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 615–629.

Tiuryn, J. (1992), Subtype inequalities, *in* '7th Annual IEEE Symp. on Logic in Computer Science (LICS'92)', pp. 308–315.

Tiuryn, J. (1997), Subtyping over a lattice (abstract), *in* '5th Kurt Gödel Colloquium (KGC'97)', Vol. 1289 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 84–88.

Trifonov, V. & Smith, S. (1996), Subtyping constrained types, *in* '3rd Static Analysis Symposium (SAS'96)', Vol. 1145 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 349–365.

Wand, M. & O'Keefe, P. (1989), On the complexity of type inference with coercion, *in* 'Functional Programming Languages and Computer Architecture'89', pp. 293–298.

Monotone Fixed-Point Types and Strong Normalization

Ralph Matthes*

Institut für Informatik
Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany
matthes@informatik.uni-muenchen.de

Abstract. Several systems of fixed-point types (also called retract types or recursive types with explicit isomorphisms) extending system F are considered. The seemingly strongest systems have monotonicity witnesses and use them in the definition of beta reduction for those types. A more naïve approach leads to non-normalizing terms. All the other systems are strongly normalizing because they embed in a reduction-preserving way into the system of non-interleaved positive fixed-point types which can be shown to be strongly normalizing by an easy extension of some proof of strong normalization for system F.

1 Introduction

It is well-known that inductive data types may be encoded impredicatively. Unfortunately the encoding in system F only models iteration. In [5] it is shown for the example Int of natural numbers that full primitive recursion is only achieved for numerals instead of arbitrary terms of type Int. We read (on p. 91): "We make no secret of the fact that this is a defect of system F." The second problem (also shown in [5]) with this encoding is the lack of efficiency: The predecessor function (which only works on numerals) needs $O(n)$ steps to compute the predecessor of the successor of numeral n.

It is now tempting to study extensions of system F by inductive types with full primitive recursion overcoming those defects ([4], [9]). However, led by the fact that those anomalies of the encoding already show up for the predecessor function, we study extensions by fixed-point types which amounts to postulating the existence of efficient[1] predecessors for the data types under consideration.

In [4] such an extension (by "retract types") is studied for "positive type schemes". In the present paper we free ourselves from positivity restrictions and replace them by monotonicity witnesses which are terms in the system. Even then a naive definition leads to non-normalizing terms as shown by Lemma 1.

* I am thankful for support by the Volkswagen-Stiftung.

[1] An efficient predecessor computes the predecessor of a successor term in a constant number of steps.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 298–312, 1999.

This problem is overcome by a beta rule with a somewhat artificial reference to the monotonicity witness (Theorem 1).

In sections 3 and 4 we study positivity restrictions (slightly more general than in [4]) and verify that they give rise to monotonicity witnesses which are well-behaved in the sense that the above artificial reference is compensated with. The key property is the first functor law stating that the identity is mapped to the identity (the proof of which requires some extensionality expressed by eta rules). Therefore we may say that the new beta rule for μ helps to separate concerns: Strong normalization is guaranteed for any monotone fixed-point type and an efficient predecessor is achieved if we find a monotonicity witness obeying the first functor law.

In section 5 a streamlined proof of strong normalization following the saturated sets approach is outlined. It turns out that most of the work is done for system F and that the extra clauses for non-interleaving positive $\mu\alpha\rho$ are dealt with very easily.

Strong normalization of the systems of monotone fixed-point types now follows from the embeddings shown in section 6.

2 Definition of Systems of Monotone Fixed-Point Types

We consider extensions of system F (see e. g. [5]) by fixed-point types, i. e. we have infinitely many type variables (denoted by α, β, ...) and with types ρ and σ we also have the product type $\rho \times \sigma$ and the function type $\rho \to \sigma$. Moreover, given a variable α and a type ρ we form the universal type $\forall\alpha\rho$ and the fixed-point type $\mu\alpha\rho$. (The μ shall not indicate that we model a least fixed point. For ease of presentation we do not use another symbol.) The $\forall$ and μ bind α in ρ. The renaming convention for bound variables is adopted. Let $\mathsf{FV}(\rho)$ be the set of type variables occurring free in ρ.

The terms of F are presented as in [5], i. e. without contexts and with fixed types (see [3] p. 159 for comments on this original typing à la Church). We have infinitely many term variables with types (denoted e. g. by x^ρ), pairing $\langle r^\rho, s^\sigma\rangle^{\rho\times\sigma}$, projections $(r^{\rho\times\sigma}\mathsf{L})^\rho$ and $(r^{\rho\times\sigma}\mathsf{R})^\sigma$, lambda abstraction $(\lambda x^\rho r^\sigma)^{\rho\to\sigma}$ for terms, term application $(r^{\rho\to\sigma}s^\rho)^\sigma$, lambda abstraction $(\Lambda\alpha r^\rho)^{\forall\alpha\rho}$ for types (under the usual proviso) and type application $(r^{\forall\alpha\rho}\sigma)^{\rho[\alpha:=\sigma]}$. We freely use the renaming convention for bound term and type variables of terms.

Beta reduction $\mapsto$ for system F is as usual given by

$$\begin{array}{ll} (\beta_\times) & \langle r, s\rangle\mathsf{L} \mapsto r \\ & \langle r, s\rangle\mathsf{R} \mapsto s \\ (\beta_\to) & (\lambda x^\rho r)s \mapsto r[x^\rho := s] \\ (\beta_\forall) & (\Lambda\alpha r)\sigma \mapsto r[\alpha := \sigma] \ . \end{array}$$

The reduction relation $\to$ is defined as the term closure of $\mapsto$. The main theorem on F (due to Girard) is that $\to$ is strongly normalizing, i. e. there are no infinite reduction sequences $r_1 \to r_2 \to r_3 \to \cdots$ (In [9] all the definitions and lemmas

are given quite carefully so as to make sure that the main theorem can also be proved in this presentation without contexts.)

We now want to extend the term system of F by rules pertaining to the type construct $\mu\alpha\rho$ and also extend the notion of beta reduction without giving up strong normalization.

In presentations not concerned about normalization as e. g. [1] (the authors call the fixed-point types "recursive types with explicit isomorphism") there is a term construction which "folds" $\rho[\alpha := \mu\alpha\rho]$ into $\mu\alpha\rho$ and one which "unfolds" terms of type $\mu\alpha\rho$ to terms of type $\rho[\alpha := \mu\alpha\rho]$. This would give the rules:

(μ-I) If $t^{\rho[\alpha:=\mu\alpha\rho]}$ is a term, then $(\mathrm{C}_{\mu\alpha\rho}t)^{\mu\alpha\rho}$ is a term.
(μ-E) If $r^{\mu\alpha\rho}$ is a term, then $(r\mathrm{E}_\mu)^{\rho[\alpha:=\mu\alpha\rho]}$ is a term.

The obvious beta rule for μ would be

$$(\beta_\mu) \qquad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu \mapsto t \ .$$

[The presentation follows the pattern of natural deduction very closely: There is an introduction rule (μ-I) by which terms of the new type are introduced ("constructed"—hence the symbol $\mathrm{C}_{\mu\alpha\rho}$) and an elimination rule (μ-E) by which terms of the new type are used. A beta rule gives a reduct for a term in which an elimination follows an introduction.]

Because of this rule and its apparent similarity to retracts in domain theory the fixed-point types get the name retract types in [4].

Unfortunately this allows for non-normalizing terms which will be shown even for a restricted system in Lemma 1.

Having a normalization proof by saturated sets (some variant of Girard's candidates) in mind, and also Tarski's theorem that every monotone endofunction on a complete lattice has a fixed point, and that saturated sets form a complete lattice with set inclusion as partial ordering, the following restriction of the use of $\mu\alpha\rho$ comes very naturally: We may only use $\mu\alpha\rho$ if there is a term witnessing the monotonicity of $\lambda\alpha\rho$, i. e. if there is a term m of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ (with $\beta \notin \{\alpha\} \cup \mathsf{FV}(\rho)$). We may enforce this restriction in two ways: Either by assuming the term m in the introduction rule (which will give the system $\mathsf{EMF}_{\mathsf{naive}}$ of elimination-based monotone fixed-point types) or by assuming it in the elimination rule (thus defining the system $\mathsf{IMF}_{\mathsf{naive}}$ of introduction-based monotone fixed-point types).

Hence, the terms of $\mathsf{EMF}_{\mathsf{naive}}$ are defined by extending the definition of terms of system F by

(μ-I) If $t^{\rho[\alpha:=\mu\alpha\rho]}$ is a term and $m^{\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]}$ is a term, then $(\mathrm{C}_{\mu\alpha\rho}mt)^{\mu\alpha\rho}$ is a term.
(μ-E) If $r^{\mu\alpha\rho}$ is a term, then $(r\mathrm{E}_\mu)^{\rho[\alpha:=\mu\alpha\rho]}$ is a term.

The relation $\mapsto$ is extended for $\mathsf{EMF}_{\mathsf{naive}}$ by

$$(\beta_\mu) \qquad (\mathrm{C}_{\mu\alpha\rho}mt)\mathrm{E}_\mu \mapsto t \ .$$

Similarly, the terms of $\mathsf{IMF}_{\mathsf{naive}}$ are defined by extending the definition of terms of system F by

(μ-I) If $t^{\rho[\alpha:=\mu\alpha\rho]}$ is a term, then $(\mathrm{C}_{\mu\alpha\rho}t)^{\mu\alpha\rho}$ is a term.
(μ-E) If $r^{\mu\alpha\rho}$ and $m^{\forall\alpha\forall\beta.(\alpha\to\beta)\to\rho\to\rho[\alpha:=\beta]}$ are terms, then $(r\mathrm{E}_\mu m)^{\rho[\alpha:=\mu\alpha\rho]}$ is a term.

The relation $\mapsto$ of system F is extended for $\mathsf{IMF}_{\mathsf{naive}}$ by

$$(\beta_\mu) \qquad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu m \mapsto t \ .$$

Lemma 1. $\mathsf{EMF}_{\mathsf{naive}}$ *and* $\mathsf{IMF}_{\mathsf{naive}}$ *are not normalizing.*

Proof. We only show it for $\mathsf{EMF}_{\mathsf{naive}}$ because the proof for $\mathsf{IMF}_{\mathsf{naive}}$ is essentially the same. Set $1 := \forall\alpha.\alpha \to \alpha$, $\mathsf{IN1} := \Lambda\alpha\lambda x^\alpha.x$ (of type 1) and $\mathsf{p} := \mu\alpha.\alpha \to 1$. We get the trivial closed monotonicity witness $m := \Lambda\alpha\Lambda\beta\lambda f^{\alpha\to\beta}\lambda x^{\alpha\to 1}\lambda y^\beta.\mathsf{IN1}$ for $\lambda\alpha.\alpha \to 1$. Define $\omega := \lambda x^{\mathsf{p}}.(x\mathrm{E}_\mu)x$ of type $\mathsf{p} \to 1$. Then $C_{\mathsf{p}}m\omega : \mathsf{p}$ and therefore $\Omega := \omega(C_{\mathsf{p}}m\omega) : 1$. We calculate

$$\Omega \mapsto_{\beta_\to} (\mathrm{C}_{\mathsf{p}}m\omega)\mathrm{E}_\mu(\mathrm{C}_{\mathsf{p}}m\omega) \to_{\beta_\mu} \omega(\mathrm{C}_{\mathsf{p}}m\omega) = \Omega \ .$$

Because there are no other possibilities of reduction the term Ω is not even weakly normalizing. □

We observe that the monotonicity witness does not enter the definition of beta reduction and therefore cannot contribute to normalization in case of trivially monotone fixed-point types (another example being $\mu\alpha.\alpha \to \alpha$ with essentially the same term as above).

Note that with recursive types in type assignment systems it would not be possible to carry monotonicity witnesses around, and since [10] it is known that non-positivity always leads to the typability of non-normalizing terms. The proof above exhibits that the generic non-normalizing term $(\lambda x.xx)(\lambda x.xx)$ of untyped lambda calculus is typable with (monotone) recursive types.

In order to solve the normalization problem in our setting of monotonicity witnesses we define a variant EMF of $\mathsf{EMF}_{\mathsf{naive}}$ by changing the definition of reduction to

$$(\beta_\mu)' \qquad (\mathrm{C}_{\mu\alpha\rho}mt)\mathrm{E}_\mu \mapsto m(\mu\alpha\rho)(\mu\alpha\rho)(\lambda y^{\mu\alpha\rho}y)t \ .$$

Similarly, we define a variant IMF of $\mathsf{IMF}_{\mathsf{naive}}$ by replacing (β_μ) by

$$(\beta_\mu)' \qquad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu m \mapsto m(\mu\alpha\rho)(\mu\alpha\rho)(\lambda y^{\mu\alpha\rho}y)t \ .$$

[The rules at least make sense in that they do not alter the types and do not introduce new free variables. It may be instructive to compare those rules with the beta rules of iteration on inductive types where the monotonicity witnesses are needed to lift the whole function being defined by iteration such that the recursion argument may be inserted and the result be given as an argument to the step term.]

In EMF and IMF the term in the proof of the above lemma does not pose a problem (we show it for EMF):

$$\Omega \mapsto_{\beta_\to} (\mathrm{C}_{\mathsf{p}}m\omega)\mathrm{E}_\mu(\mathrm{C}_{\mathsf{p}}m\omega) \to_{\beta_\mu{}'} m\mathsf{pp}(\lambda y^{\mathsf{p}}y)\omega(\mathrm{C}_{\mathsf{p}}m\omega) \to^2_{\beta_\forall}\to^3_{\beta_\to} \mathsf{IN1}$$

and there are no other reduction possibilities.

The same difference occurs with the fixed-point combinator for types of the form $\rho \to \rho$ (see [6], p.241) which makes use of the type $\mu\alpha.\alpha \to \rho \to \rho$ with $\alpha \notin \mathsf{FV}(\rho)$ which again has a trivial monotonicity witness.

Theorem 1. *The systems* EMF *and* IMF *are strongly normalizing.*

The proof will be done by embedding the systems into the system NPF of non-interleaved positive fixed-point types which will be defined in the next section.

But let us first comment on the theorem: From Lemma 1 we know that a predecessor for types such as p is incompatible with normalization. The above theorem tells us that strong normalization is gained by changing the beta rule in the described way. Therefore we do not have to impose a positivity restriction but will get a "harmless" term in degenerate cases instead of the predecessor. And the "harmlessness" is simply enforced by the type of m which has to express monotonicity! Section 4 reveals that there are enough non-degenerate cases.

3 Non-Interleaved Positive Fixed-Point Types

The system NPF will have the standard (β_μ)-rule $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu \mapsto t$ but is strongly normalizing due to a restriction imposed on the formation of types of the form $\mu\alpha\rho$. We only allow $\mu\alpha\rho$ as a type of the system if all free occurrences of α in ρ are at positive positions and not in some subexpression of the form $\mu\beta\sigma$. The formal definition also has to deal with negative positions because we do not restrict ourselves to strict positivity.

Inductively define the set NPTypes of non-interleaved positive types and simultaneously for every $\rho \in \mathsf{NPTypes}$ the set $\mathsf{NPos}(\rho)$ of type variables which only occur free at positive positions in ρ and are not in the scope of an application of the μ-rule and the set $\mathsf{NNeg}(\rho)$ of type variables which only occur free at negative positions in ρ and are not in the scope of an application of the μ-rule as follows:

(V) $\alpha \in \mathsf{NPTypes}$ and $\mathsf{NPos}(\alpha) := \mathsf{Typevars}$ and $\mathsf{NNeg}(\alpha) := \mathsf{Typevars} \setminus \{\alpha\}$.

($\times$) If $\rho \in \mathsf{NPTypes}$ and $\sigma \in \mathsf{NPTypes}$, then $\rho \times \sigma \in \mathsf{NPTypes}$.
$\mathsf{NPos}(\rho \times \sigma) := \mathsf{NPos}(\rho) \cap \mathsf{NPos}(\sigma)$ and $\mathsf{NNeg}(\rho \times \sigma) := \mathsf{NNeg}(\rho) \cap \mathsf{NNeg}(\sigma)$.

($\to$) If $\rho \in \mathsf{NPTypes}$ and $\sigma \in \mathsf{NPTypes}$, then $\rho \to \sigma \in \mathsf{NPTypes}$.
$\mathsf{NPos}(\rho \to \sigma) := \mathsf{NNeg}(\rho) \cap \mathsf{NPos}(\sigma)$.
$\mathsf{NNeg}(\rho \to \sigma) := \mathsf{NPos}(\rho) \cap \mathsf{NNeg}(\sigma)$.

($\forall$) If $\rho \in \mathsf{NPTypes}$, then $\forall\alpha\rho \in \mathsf{NPTypes}$.
$\mathsf{NPos}(\forall\alpha\rho) := \mathsf{NPos}(\rho) \cup \{\alpha\}$ and $\mathsf{NNeg}(\forall\alpha\rho) := \mathsf{NNeg}(\rho) \cup \{\alpha\}$.

(μ) If $\rho \in \mathsf{NPTypes}$ and $\alpha \in \mathsf{NPos}(\rho)$, then $\mu\alpha\rho \in \mathsf{NPTypes}$.
$\mathsf{NPos}(\mu\alpha\rho) := \mathsf{NPos}(\rho) \setminus \mathsf{FV}(\mu\alpha\rho)$.
$\mathsf{NNeg}(\mu\alpha\rho) := (\mathsf{NNeg}(\rho) \setminus \mathsf{FV}(\rho)) \cup \{\alpha\}$.

Let NPF be the extension of system F to the types in NPTypes and with the standard rules (μ-I), (μ-E) and (β_μ) mentioned first. (The definitions for

system F are still correct because NPTypes is closed under substitution which is somewhat awkward to prove because one needs additional statements[2] on NPos and NNeg to be proved simultaneously.)

Interleaving μ-types are ruled out in NPF but not nesting! Define $\mathsf{nat} := \mu\alpha.1+\alpha$ (for the definition of 1 see the proof of Lemma 1, $\rho+\sigma := \forall\alpha.(\rho \to \alpha) \to (\sigma \to \alpha) \to \alpha$ with $\alpha \notin \mathsf{FV}(\rho) \cup \mathsf{FV}(\sigma)$) and then define $\mathsf{tree} := \mu\alpha.1 + (\mathsf{nat} \to \alpha)$ which is clearly nested. The following type (due to Ulrich Berger) is not allowed although it is positive:

$$\mathsf{Tree} := \mu\alpha.1 + ((\mu\beta.1 + (\alpha \to \beta)) \to \alpha)$$

NPF is a slight generalization (due to nesting) of F_{ret} in [4] which is shown to be strongly normalizing by an embedding into Mendler's system [10] via a system of non-interleaved positive inductive types with primitive recursion (in [4]).

NPF is a subsystem of the system M2Lμ in [7] (p. 308) where it is argued (p. 310) that this system may be proved strongly normalizing by adopting the same method as for Mendler's system.

In [9] a direct proof is given of strong normalization of the variant of Mendler's system which simply leaves out the positivity requirement in [10] (which is crucial to the normalization proof presented there). This more liberal system also embeds in a reduction-preserving way into NPF which will be sketched in the final section. Therefore the author thinks that a direct proof of strong normalization for NPF gives an easier access to the above-mentioned normalization results.

Before turning to normalization proofs we try to see in which way NPF embeds into the systems of monotone fixed-point types.

4 Embedding NPF into Monotone Fixed-Point Types

Clearly, we expect non-interleaved positive fixed-point types to be monotone fixed-point types. This will be studied more carefully in this section. It turns out that in presence of eta rules we indeed get an embedding.

4.1 Definition of Monotonicity Witnesses for NPF

We want to define closed terms $\mathsf{map}_{\lambda\alpha\rho}$ of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ in NPF for every $\mu\alpha\rho \in \mathsf{NPTypes}$, i. e. $\rho \in \mathsf{NPTypes}$ and $\alpha \in \mathsf{NPos}(\rho)$. Because positivity and negativity are defined simultaneously, we have to define auxiliary terms $\mathsf{comap}_{\lambda\alpha\rho}$ for $\alpha \in \mathsf{NNeg}(\rho)$ simultaneously (which will show up in the case of arrow types):

Define for every type $\rho \in \mathsf{NPTypes}$ and every $\alpha \in \mathsf{NPos}(\rho)$ (and $\beta \notin \{\alpha\} \cup \mathsf{FV}(\rho)$) a closed term

$$\mathsf{map}_{\lambda\alpha\rho} : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$$

[2] One has to prove: Let $\rho, \boldsymbol{\sigma} \in \mathsf{NPTypes}$. Then $\rho[\boldsymbol{\alpha} := \boldsymbol{\sigma}] \in \mathsf{NPTypes}$ and $\mathsf{NPos}(\rho) \setminus \mathsf{FV}(\boldsymbol{\sigma}) \subseteq \mathsf{NPos}(\rho[\boldsymbol{\alpha} := \boldsymbol{\sigma}])$ and $\mathsf{NNeg}(\rho) \setminus \mathsf{FV}(\boldsymbol{\sigma}) \subseteq \mathsf{NNeg}(\rho[\boldsymbol{\alpha} := \boldsymbol{\sigma}])$. Moreover, if $\beta \notin \mathsf{FV}(\rho)$ then $\alpha \in \mathsf{NPos}(\rho)$ implies $\beta \in \mathsf{NPos}(\rho[\alpha := \beta])$ and $\alpha \in \mathsf{NNeg}(\rho)$ implies $\beta \in \mathsf{NNeg}(\rho[\alpha := \beta])$.

and for every $\alpha \in \mathsf{NNeg}(\rho)$ (and $\beta \notin \{\alpha\} \cup \mathsf{FV}(\rho)$) a closed term

$$\mathsf{comap}_{\lambda\alpha\rho} : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho[\alpha := \beta] \to \rho$$

by recursion on ρ.

A definition which yields normal terms is shown in [9] pp. 73–75. Because we do not allow for interleaving one only has to omit the case (μ). The reader also finds a definition in [7] p. 311. Again one has to omit the μ-clause and should make a case distinction whether α occurs free in ρ or not. Another place is [4] p. 206 which essentially contains the definitions. Because we have no interleaving it does not matter that those definitions are all made for systems of *inductive* types (i. e. $\mu\alpha\rho$ denotes a least fixed-point), and moreover, the definition is quite straightforward and by structural recursion on ρ.

4.2 The Embeddings

An embedding $-'$ of NPF in $\mathsf{EMF}_{\mathsf{naive}}$ is now immediately defined by induction on the terms of NPF setting

$$(\mu\text{-I}) \quad (\mathrm{C}_{\mu\alpha\rho}t)' := \mathrm{C}_{\mu\alpha\rho}\mathsf{map}_{\lambda\alpha\rho}t'.$$

and homomorphic rules for all the other term formers. This gives terms in $\mathsf{EMF}_{\mathsf{naive}}$ because $\mathsf{map}_{\lambda\alpha\rho}$ is built without (μ-I) and hence already is a term of $\mathsf{EMF}_{\mathsf{naive}}$.

It is easy to see that if $r \to s$ in NPF, then $r' \to s'$ in $\mathsf{EMF}_{\mathsf{naive}}$. (For (β_μ) we use that $\mathsf{map}_{\lambda\alpha\rho}$ is a closed term, for $(\beta_\forall)$ we have to prove that type substitution commutes with forming $\mathsf{map}_{\lambda\alpha\rho}$.)

It is clear how to change the embedding to an embedding of NPF into $\mathsf{IMF}_{\mathsf{naive}}$.

Unfortunately, the two embeddings do not preserve (β_μ) in EMF and IMF, respectively. An easy solution to this problem is the addition of η-rules for $\times$, $\to$ and $\forall$ as usual:

$$\begin{array}{lll} (\eta_\times) & \langle r\mathsf{L}, r\mathsf{R}\rangle \mapsto r & \\ (\eta_\to) & \lambda x^\rho.rx \mapsto r & \text{if } x \notin \mathsf{FV}(r) \\ (\eta_\forall) & \Lambda\alpha.r\alpha \mapsto r & \text{if } \alpha \notin \mathsf{FTV}(r) \end{array}$$

(where $\mathsf{FTV}(r)$ is the set of free type variables of r).

Indicate the presence of these additional rules by the index eta in the names of the systems.

Now it is easy to see that in $\mathsf{NPF}_{\mathsf{eta}}$ (and trivially also in $\mathsf{EMF}_{\mathsf{eta}}$ and $\mathsf{IMF}_{\mathsf{eta}}$) we have that map and comap satisfy the first functor law, more precisely

$$\mathsf{map}_{\lambda\alpha\rho}\sigma\sigma(\lambda x^\sigma x)t^{\rho[\alpha:=\sigma]} \to^* t \text{ and } \mathsf{comap}_{\lambda\alpha\rho}\sigma\sigma(\lambda x^\sigma x)t^{\rho[\alpha:=\sigma]} \to^* t$$

(where $\to^*$ denotes the reflexive and transitive closure of $\to$).

Therefore we get that whenever $r \to s$ in NPF, then $r' \to^+ s'$ in the systems $\mathsf{EMF}_{\mathsf{eta}}$ and $\mathsf{IMF}_{\mathsf{eta}}$, respectively (where $\to^+$ denotes the transitive closure of $\to$).

This entails that from strong normalization of $\mathsf{EMF}_{\mathsf{eta}}$ or $\mathsf{IMF}_{\mathsf{eta}}$, we can infer strong normalization of NPF.

But now we show directly that even $\mathsf{NPF}_{\mathsf{eta}}$ is strongly normalizing and derive strong normalization of $\mathsf{EMF}_{\mathsf{eta}}$ and $\mathsf{IMF}_{\mathsf{eta}}$ from this fact via an embedding.

We show even more:

Let $\mathsf{NPF}_{\mathsf{eta}+}$ be the system $\mathsf{NPF}_{\mathsf{eta}}$ extended by the η-rule for fixed-point types:

$$(\eta_\mu) \qquad \mathrm{C}_{\mu\alpha\rho}(r^{\mu\alpha\rho}\mathrm{E}_\mu) \mapsto r$$

This rule justifies the name "fixed-point types" instead of retract types.

Theorem 2. *The system* $\mathsf{NPF}_{\mathsf{eta}+}$ *is strongly normalizing.*

5 A Direct Proof of Strong Normalization for $\mathsf{NPF}_{\mathsf{eta}+}$

Every term and type in this section is taken from $\mathsf{NPF}_{\mathsf{eta}+}$.

5.1 Head Forms, the Set SN and Saturated Sets

Lemma 2 (On head forms). *Every term has exactly one of the following forms:*

(V) $x^\rho \boldsymbol{s}$
(×-I) $\langle r, s\rangle$
(×-R) $\langle r, s\rangle \mathsf{L}\boldsymbol{s}$, $\langle r, s\rangle \mathsf{R}\boldsymbol{s}$
(→-I) $\lambda x^\rho r$
(→-R) $(\lambda x^\rho r)s\boldsymbol{s}$
(∀-I) $\Lambda\alpha r$
(∀-R) $(\Lambda\alpha r)\sigma\boldsymbol{s}$
(μ-I) $\mathrm{C}_{\mu\alpha\rho}t$
(μ-R) $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu\boldsymbol{s}$

The vector $\boldsymbol{s}$ denotes a list made up of the symbols L, R and E_μ and of terms and types such that e. g. $x^\rho\boldsymbol{s}$ is a well-typed term. Hence, the vector notation is a means of expressing multiple eliminations.

Proof. Very easy induction on terms. □

Define the set SN of terms by (strictly positive) induction as follows:

(V) If $x^\rho\boldsymbol{s}$ is a term and the terms in $\boldsymbol{s}$ are in SN, then $x^\rho\boldsymbol{s} \in \mathsf{SN}$.
(×-I) If $r \in \mathsf{SN}$ and $s \in \mathsf{SN}$, then $\langle r, s\rangle \in \mathsf{SN}$.
($\beta_\times$) If $r\boldsymbol{s} \in \mathsf{SN}$ and $s \in \mathsf{SN}$, then $\langle r, s\rangle\mathsf{L}\boldsymbol{s} \in \mathsf{SN}$.
If $s\boldsymbol{s} \in \mathsf{SN}$ and $r \in \mathsf{SN}$, then $\langle r, s\rangle\mathsf{R}\boldsymbol{s} \in \mathsf{SN}$.
(→-I) If $r \in \mathsf{SN}$, then $\lambda x^\rho r \in \mathsf{SN}$.
($\beta_\rightarrow$) If $(\lambda x^\rho r)s\boldsymbol{s}$ is a term, $s \in \mathsf{SN}$ and $r[x^\rho := s]\boldsymbol{s} \in \mathsf{SN}$, then $(\lambda x^\rho r)s\boldsymbol{s} \in \mathsf{SN}$.
(∀-I) If $\Lambda\alpha r$ is a term and $r \in \mathsf{SN}$, then $\Lambda\alpha r \in \mathsf{SN}$.

($\beta_\forall$) If $(\Lambda\alpha r)\sigma\boldsymbol{s}$ is a term and $r[\alpha := \sigma]\boldsymbol{s} \in \mathsf{SN}$, then $(\Lambda\alpha r)\sigma\boldsymbol{s} \in \mathsf{SN}$.
(μ-I) If $t \in \mathsf{SN}$, then $\mathrm{C}_{\mu\alpha\rho}t \in \mathsf{SN}$.
(μ-R) If $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu\boldsymbol{s}$ is a term and $t\boldsymbol{s} \in \mathsf{SN}$, then $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu\boldsymbol{s} \in \mathsf{SN}$.

Lemma 3. SN *is the set* sn *of strongly normalizing terms.*

Proof. $\mathsf{sn} \subseteq \mathsf{SN}$ is shown by main induction on the height of the reduction tree and side induction on the term height. (This part of the lemma is not needed for the proof of strong normalization.) $\mathsf{SN} \subseteq \mathsf{sn}$ is proved by showing that the defining clauses of SN are closure properties of the set sn. To my knowledge it is unavoidable to check those closure properties if one wants to prove strong normalization. See e. g. [9] for an exposition. □

The idea of using such a characterization of the set of strongly normalizing terms (at least for the untyped λ-calculus) was brought to my attention by [11] and [8].

I want to stress that it is invisible in this approach that we deal with $\mathsf{NPF}_{\mathsf{eta}+}$ instead of NPF—at least it does not enter the definitions. Only when showing $\mathsf{SN} \subseteq \mathsf{sn}$ one has to consider additional cases which are trivial to handle.[3]

Let SN_τ be the set of terms of type τ in SN. A set $\mathcal{M}$ of terms is called τ-saturated, if the following conditions are met.

(SN) If $r \in \mathcal{M}$, then $r \in \mathsf{SN}_\tau$.
(V) If $x^\rho\boldsymbol{s}$ is a term of type τ and the terms in $\boldsymbol{s}$ are in SN, then $x^\rho\boldsymbol{s} \in \mathcal{M}$.
($\beta_\times$) If $r\boldsymbol{s} \in \mathcal{M}$ and $s \in \mathsf{SN}$, then $\langle r, s\rangle\mathsf{L}\boldsymbol{s} \in \mathcal{M}$. If $s\boldsymbol{s} \in \mathcal{M}$ and $r \in \mathsf{SN}$, then $\langle r, s\rangle\mathsf{R}\boldsymbol{s} \in \mathcal{M}$.
($\beta_\to$) If $(\lambda x^\rho r)s\boldsymbol{s}$ is a term, $s \in \mathsf{SN}$ and $r[x^\rho := s]\boldsymbol{s} \in \mathcal{M}$, then $(\lambda x^\rho r)s\boldsymbol{s} \in \mathcal{M}$.
($\beta_\forall$) If $(\Lambda\alpha r)\sigma\boldsymbol{s}$ is a term and $r[\alpha := \sigma]\boldsymbol{s} \in \mathcal{M}$, then $(\Lambda\alpha r)\sigma\boldsymbol{s} \in \mathcal{M}$.
(β_μ) If $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu\boldsymbol{s}$ is a term and $t\boldsymbol{s} \in \mathcal{M}$, then $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu\boldsymbol{s} \in \mathcal{M}$.

Let SAT_τ be the set of all τ-saturated sets. Obviously, $\mathsf{SN}_\tau \in \mathsf{SAT}_\tau$.

Let M be a set of terms. The τ-saturated closure $\mathsf{cl}_\tau(M)$ of M is defined to be the smallest τ-saturated set containing $M \cap \mathsf{SN}_\tau$. (This may be defined by a strictly positive inductive definition.)

The candidate method goes as follows: By means of saturated sets we define (by recursion on types) predicates of strong computability with respect to an assignment of saturated sets for type variables (a candidate assignment) and finally show (by induction on terms) that every term is strongly computable under substitution. Hence every term is contained in a saturated set (the computability predicate) which only consists of strongly normalizing terms (due to the preceding lemma).

This all works if there are constructions for saturated sets corresponding to the type constructs of the system for which the introduction rules and the elimination rules are sound. (This is presented at length in [9].)

[3] The reason is: All of the systems are weakly orthogonal which amounts to checking that the critical pairs between beta and eta rules are trivial.

5.2 Calculating with Saturated Sets

Here we concentrate on the constructions of saturated sets corresponding to the type constructs.

Let $\mathcal{M} \in \mathsf{SAT}_\rho$ and $\mathcal{N} \in \mathsf{SAT}_\sigma$. Define

$$\begin{aligned} M_I &:= \{r^{\rho\times\sigma} | \exists s \in \mathcal{M} \exists t \in \mathcal{N}. r = \langle s, t\rangle\} \\ M_E &:= \{r^{\rho\times\sigma} | r\mathsf{L} \in \mathcal{M} \wedge r\mathsf{R} \in \mathcal{N}\} \end{aligned}$$

and set $\mathcal{M} \times_I \mathcal{N} := \mathsf{cl}_{\rho\times\sigma}(M_I)$ and $\mathcal{M} \times_E \mathcal{N} := \mathsf{cl}_{\rho\times\sigma}(M_E)$. Hence, $\mathcal{M} \times_I \mathcal{N}$ and $\mathcal{M} \times_E \mathcal{N}$ are $(\rho \times \sigma)$-saturated sets and the construction is isotone in the arguments $\mathcal{M}$ and $\mathcal{N}$.

Writing $\mathcal{M} \times \mathcal{N}$ for $\mathcal{M} \times_I \mathcal{N}$ or $\mathcal{M} \times_E \mathcal{N}$ (for both of them) we get (after some reasoning)

($\times$-I) If $r \in \mathcal{M}$ and $s \in \mathcal{N}$, then $\langle r, s\rangle \in \mathcal{M} \times \mathcal{N}$.
($\times$-E) If $r \in \mathcal{M} \times \mathcal{N}$, then $r\mathsf{L} \in \mathcal{M}$ and $r\mathsf{R} \in \mathcal{N}$.

Let $\mathcal{M} \in \mathsf{SAT}_\rho$ and $\mathcal{N} \in \mathsf{SAT}_\sigma$. Define

$$\begin{aligned} M_I &:= \{r^{\rho\to\sigma} | \exists x^\rho \exists t^\sigma r = \lambda x^\rho t \wedge \forall s \in \mathcal{M}\, t[x^\rho := s] \in \mathcal{N}\} \\ M_E &:= \{r^{\rho\to\sigma} | \forall s \in \mathcal{M}\, rs \in \mathcal{N}\} \end{aligned}$$

and set $\mathcal{M} \to_I \mathcal{N} := \mathsf{cl}_{\rho\to\sigma}(M_I)$ and $\mathcal{M} \to_E \mathcal{N} := \mathsf{cl}_{\rho\to\sigma}(M_E)$. Hence, $\mathcal{M} \to_I \mathcal{N}$ and $\mathcal{M} \to_E \mathcal{N}$ are $(\rho \to \sigma)$-saturated sets and the construction is isotone in the argument $\mathcal{N}$ and antitone in the argument $\mathcal{M}$.

Writing $\mathcal{M} \to \mathcal{N}$ for $\mathcal{M} \to_I \mathcal{N}$ or $\mathcal{M} \to_E \mathcal{N}$ (for both of them) we get (after some reasoning)

($\to$-I) If $t[x^\rho := s] \in \mathcal{N}$ for all $s \in \mathcal{M}$, then $\lambda x^\rho t \in \mathcal{M} \to \mathcal{N}$.
($\to$-E) If $r \in \mathcal{M} \to \mathcal{N}$ and $s \in \mathcal{M}$, then $rs \in \mathcal{N}$.

Fix $\lambda\alpha\rho$. Let $\Phi := (\Phi_\tau)_\tau$ be a family of mappings $\Phi_\tau : \mathsf{SAT}_\tau \to \mathsf{SAT}_{\rho[\alpha:=\tau]}$. Define

$$\begin{aligned} M_I &:= \{r^{\forall\alpha\rho} | \exists\alpha \exists t^\rho. r = \Lambda\alpha t \wedge \forall\tau\forall\mathcal{P} \in \mathsf{SAT}_\tau\, t[\alpha := \tau] \in \Phi_\tau(\mathcal{P})\} \\ M_E &:= \{r^{\forall\alpha\rho} | \forall\tau\forall\mathcal{P} \in \mathsf{SAT}_\tau\, r\tau \in \Phi_\tau(\mathcal{P})\} \end{aligned}$$

and set $\forall_I(\Phi) := \mathsf{cl}_{\forall\alpha\rho}(M_I)$ and $\forall_E(\Phi) := \mathsf{cl}_{\forall\alpha\rho}(M_E)$. Hence, $\forall_I(\Phi)$ and $\forall_E(\Phi)$ are $\forall\alpha\rho$-saturated sets and the construction is (pointwise) isotone in the argument Φ.

Writing $\forall(\Phi)$ for $\forall_I(\Phi)$ or $\forall_E(\Phi)$ (for both of them) we get (after some reasoning)

($\forall$-I) If $\forall\tau\forall\mathcal{P} \in \mathsf{SAT}_\tau\, t[\alpha := \tau] \in \Phi_\tau(\mathcal{P})$, then $\Lambda\alpha t \in \forall(\Phi)$.
($\forall$-E) If $r \in \forall(\Phi)$, then $\forall\tau\forall\mathcal{P} \in \mathsf{SAT}_\tau\, r\tau \in \Phi_\tau(\mathcal{P})$.

We now come to the interesting case of fixed-point types. Fix $\lambda\alpha\rho$. Let ϕ be an isotone mapping from $\mathsf{SAT}_{\mu\alpha\rho}$ to $\mathsf{SAT}_{\rho[\alpha:=\mu\alpha\rho]}$. Define for $\mathcal{M} \in \mathsf{SAT}_{\mu\alpha\rho}$

$$\begin{aligned} M_I(\mathcal{M}) &:= \{r^{\mu\alpha\rho} | \exists t \in \phi(\mathcal{M}).r = \mathrm{C}_{\mu\alpha\rho}t\} \\ M_E(\mathcal{M}) &:= \{r^{\mu\alpha\rho} | r\mathrm{E}_\mu \in \phi(\mathcal{M})\} \end{aligned}$$

and $\Psi_I : \mathsf{SAT}_{\mu\alpha\rho} \to \mathsf{SAT}_{\mu\alpha\rho}$ and $\Psi_E : \mathsf{SAT}_{\mu\alpha\rho} \to \mathsf{SAT}_{\mu\alpha\rho}$ by setting $\Psi_I(\mathcal{M}) := \mathsf{cl}_{\mu\alpha\rho}(M_I(\mathcal{M}))$ and $\Psi_E(\mathcal{M}) := \mathsf{cl}_{\mu\alpha\rho}(M_E(\mathcal{M}))$ and set

$$\mu_I(\phi) := \text{ any fixed point of } \Psi_I \quad \text{and} \quad \mu_E(\phi) := \text{ any fixed point of } \Psi_E.$$

If there are several fixed-points, then μ_I and μ_E need not be isotone in ϕ. Of course, there is always a fixed-point because Ψ_I and Ψ_E are monotone endofunctions on the complete lattice $\mathsf{SAT}_{\mu\alpha\rho}$. This time the work behind "after some reasoning" is spelled out precisely. It is the suite of lemmas which may nearly be written down mechanically.

Lemma 4. *$M_I(\mathcal{M}) \subseteq \mathsf{SN}$ and $M_E(\mathcal{M}) \cap \mathsf{SN} \in \mathsf{SAT}_{\mu\alpha\rho}$.*

Proof. First part obvious. Second part: One only has to append E_μ to $\boldsymbol{s}$. □

Lemma 5. *$\mathcal{M} \in \mathsf{SAT}_{\mu\alpha\rho}$ is a pre-fixed point of Ψ_I iff $\forall t \in \phi(\mathcal{M}).C_{\mu\alpha\rho}t \in \mathcal{M}$. $\mathcal{M}$ is a post-fixed point of Ψ_E iff $\forall r \in \mathcal{M}.rE_\mu \in \phi(\mathcal{M})$.*

Proof. The preceding lemma yields $M_I(\mathcal{M}) \subseteq \Psi_I(\mathcal{M})$ and $\Psi_E(\mathcal{M}) \subseteq M_E(\mathcal{M})$. Moreover, for $\mathcal{N} \in \mathsf{SAT}_{\mu\alpha\rho}$ we have that $M_I(\mathcal{M}) \subseteq \mathcal{N}$ implies $\Psi_I(\mathcal{M}) \subseteq \mathcal{N}$ and $\mathcal{N} \subseteq M_E(\mathcal{M})$ implies $\mathcal{N} \subseteq \Psi_E(\mathcal{M})$. □

Lemma 6. *$M_I(\mathcal{M}) \subseteq M_E(\mathcal{M})$ for every $\mathcal{M} \in \mathsf{SAT}_{\mu\alpha\rho}$.*

Proof. Let $t \in \phi(\mathcal{M})$. We have to show that $\mathrm{C}_{\mu\alpha\rho}t \in M_E(\mathcal{M})$. Hence, we have to show that $(\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu \in \phi(\mathcal{M})$. Because $\phi(\mathcal{M})$ is saturated it suffices to show $t \in \phi(\mathcal{M})$. (Note that this is the only explicit reference to the clause (β_μ) in the definition of saturated sets.) □

Lemma 7. *$\mu_E(\phi)$ is a pre-fixed point of Ψ_I and $\mu_I(\phi)$ is a post-fixed point of Ψ_E.*

Proof. Show that $\Psi_I(\mu_E(\phi)) \subseteq \mu_E(\phi)$: Because $\mu_E(\phi)$ is a pre-fixed point of Ψ_E it suffices to show that $\Psi_I(\mu_E(\phi)) \subseteq \Psi_E(\mu_E(\phi))$. Because $\mathsf{cl}_{\mu\alpha\rho}$ is monotone, it suffices to show $M_I(\mu_E(\phi)) \subseteq M_E(\mu_E(\phi))$ which is an instance of the preceding lemma. $\mu_I(\phi) \subseteq \Psi_E(\mu_I(\phi))$ is established similarly. □

Writing $\mu(\phi)$ for $\mu_I(\phi)$ or $\mu_E(\phi)$ (for both of them) we finally get

(μ-I) $\forall t \in \phi(\mu(\phi)).\mathrm{C}_{\mu\alpha\rho}t \in \mu(\phi)$.
(μ-E) $\forall r \in \mu(\phi).r\mathrm{E}_\mu \in \phi(\mu(\phi))$.

5.3 Computability Predicates

Now the saturated sets are used to define the computability predicates which may be seen as a semantics of the types.

Define the $(\rho[\boldsymbol{\alpha} := \boldsymbol{\sigma}])$-saturated set $\mathsf{SC}_\rho[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}]$ of strongly computable terms w.r.t. the type ρ, the finite list of type variables $\boldsymbol{\alpha}$ and the (equally long) list of saturated sets $\boldsymbol{\mathcal{P}}$, where σ_i is the unique type such that $\mathcal{P}_i$ is σ_i-saturated, by recursion on ρ and simultaneously prove that $\mathsf{SC}_\rho[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}]$ is an isotone function of $\mathcal{P}_i$ for $\alpha_i \in \mathsf{NPos}(\rho)$ and an antitone function of $\mathcal{P}_i$ for $\alpha_i \in \mathsf{NNeg}(\rho)$ and does not depend on $\mathcal{P}_i$ if $\alpha \notin \mathsf{FV}(\rho)$ as follows.

(V) $\mathsf{SC}_\alpha[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \mathsf{cl}_\alpha(\mathit{\Lambda})$, if α does not occur in the list $\boldsymbol{\alpha}$. (Note that $\mathsf{cl}_\alpha(\mathit{\Lambda}) = \mathsf{SN}_\alpha$.)
$\mathsf{SC}_\alpha[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \mathcal{P}_i$, if i is the smallest index such that $\alpha_i = \alpha$.

$(\times)$ $\mathsf{SC}_{\rho\times\sigma}[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \mathsf{SC}_\rho[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] \times \mathsf{SC}_\sigma[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}]$.

$(\to)$ $\mathsf{SC}_{\rho\to\sigma}[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \mathsf{SC}_\rho[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] \to \mathsf{SC}_\sigma[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}]$.

$(\forall)$ $\mathsf{SC}_{\forall\alpha\rho}[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \forall((\Phi_\sigma)_\sigma)$, where $\Phi_\sigma : \mathsf{SAT}_\sigma \to \mathsf{SAT}_{\rho[\boldsymbol{\alpha},\alpha:=\boldsymbol{\sigma},\sigma]}$ is defined by $\Phi_\sigma(\mathcal{P}) := \mathsf{SC}_\rho[\boldsymbol{\alpha}, \alpha := \boldsymbol{\mathcal{P}}, \mathcal{P}]$ (We assume that $\alpha \notin \boldsymbol{\alpha} \cup \mathsf{FV}(\boldsymbol{\sigma})$).

(μ) $\mathsf{SC}_{\mu\alpha\rho}[\boldsymbol{\alpha} := \boldsymbol{\mathcal{P}}] := \mu(\phi)$, where $\phi : \mathsf{SAT}_{\mu\alpha\rho} \to \mathsf{SAT}_{\rho[\boldsymbol{\alpha},\alpha:=\boldsymbol{\sigma},\mu\alpha\rho]}$ is defined by $\phi(\mathcal{P}) := \mathsf{SC}_\rho[\boldsymbol{\alpha}, \alpha := \boldsymbol{\mathcal{P}}, \mathcal{P}]$ (with the same assumption as for $(\forall)$). In this case it is important to have the simultaneously proved statements at hand. Thanks to the restriction to non-interleaved fixed-point types they can be derived from the induction hypothesis.

It is now standard technique first to prove a substitution lemma for computability predicates (by induction on types) and second to prove that every term is strongly computable under substitution (by induction on terms) which gives the theorem to be proved. (For system F and extensions thereof by inductive types this is done in all details in [9].)

6 Embedding EMF and IMF into NPF

Define for every type ρ of EMF a type ρ' of NPF with the only non-homomorphic rule:

$$(\mu\alpha\rho)' := \mu\alpha\forall\beta.(\alpha \to \beta) \to \rho'[\alpha := \beta] \ .$$

It is easy to see that we indeed get non-interleaving (non-strictly) positive fixed-point types and that ρ and ρ' have the same free type variables.

Lemma 8. $(\rho[\boldsymbol{\alpha} := \boldsymbol{\sigma}])' = \rho'[\boldsymbol{\alpha} := \boldsymbol{\sigma}']$.

Proof. Induction on ρ. □

Define for every term r^ρ of EMF a term r' of type ρ' of NPF as follows (the other clauses are homomorphic):

$$(\mu\text{-I}) \quad (\mathrm{C}_{\mu\alpha\rho} m t)' := \mathrm{C}_{(\mu\alpha\rho)'}\Big(\mathit{\Lambda}\beta\lambda y^{(\mu\alpha\rho)'\to\beta}.m'(\mu\alpha\rho)'\beta y t'\Big)$$

(μ-E) $(r^{\mu\alpha\rho}\mathrm{E}_\mu)' := r'\mathrm{E}_\mu(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)$.

[More precisely, we simultaneously have to prove that r' has the same type variables as r and that x^τ is a free variable of r' iff $\tau = \sigma'$ and x^σ is free in r.]

Lemma 9. *$(r[\boldsymbol{x}^{\boldsymbol{\rho}} := \boldsymbol{s}^{\boldsymbol{\rho}}])' = r'[\boldsymbol{x}^{\boldsymbol{\rho}'} := \boldsymbol{s}']$ and $(r[\boldsymbol{\alpha} := \boldsymbol{\sigma}])' = r'[\boldsymbol{\alpha} := \boldsymbol{\sigma}']$.*

Proof. Induction on r. □

Lemma 10. *If $r \to \hat{r}$, then $r' \to^+ \hat{r}'$.*

Proof. Induction on $r \to \hat{r}$: The cases ($\beta_\to$) and ($\beta_\forall$) follow immediately from the preceding lemma. ($\beta_\times$) is trivial. (β_μ):

$$\begin{aligned}((\mathrm{C}_{\mu\alpha\rho}mt)\mathrm{E}_\mu)' &= \mathrm{C}_{(\mu\alpha\rho)'}\Big(\Lambda\beta\lambda y^{(\mu\alpha\rho)'\to\beta}.m'(\mu\alpha\rho)'\beta yt'\Big)\mathrm{E}_\mu(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)\\ &\to \Big(\Lambda\beta\lambda y^{(\mu\alpha\rho)'\to\beta}.m'(\mu\alpha\rho)'\beta yt'\Big)(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)\\ &\to \Big(\lambda y^{(\mu\alpha\rho)'\to(\mu\alpha\rho)'}.m'(\mu\alpha\rho)'(\mu\alpha\rho)'yt'\Big)(\lambda x^{(\mu\alpha\rho)'}x)\\ &\mapsto m'(\mu\alpha\rho)'(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)t'\\ &= \Big(m(\mu\alpha\rho)(\mu\alpha\rho)(\lambda x^{\mu\alpha\rho}x)t\Big)'\end{aligned}$$

The term closure clauses are easily proved by using the induction hypothesis. □

In order to see how the embedding of IMF works it is convenient to introduce existential types into NPF which can later be eliminated in a reduction-preserving way.[4] Hence, we embed IMF into $\mathsf{NPF}_{\mathsf{ex}}$ which is NPF extended by types of the form $\exists\alpha\rho$ (more formally we have to extend the definition of NPTypes), by the term rules

($\exists$-I) If $t^{\rho[\alpha:=\tau]}$ is a term, then $(\mathrm{C}_{\exists\alpha\rho,\tau}t)^{\exists\alpha\rho}$ is a term.
($\exists$-E) If $r^{\exists\alpha\rho}$ is a term, $s^{\forall\alpha.\rho\to\sigma}$ is a term and $\alpha \notin \mathsf{FV}(\sigma)$, then $(r\mathrm{E}_\exists s)^\sigma$ is a term.

and by the reduction rule

$$(\beta_\exists) \qquad (C_{\exists\alpha\rho,\tau}t)\mathrm{E}_\exists s \mapsto s\tau t \ .$$

[These are clearly the rules pertaining to the second-order existential quantifier in systems of natural deduction together with a sound transformation rule.]

Define for every type ρ of IMF a type ρ' of $\mathsf{NPF}_{\mathsf{ex}}$ with the only non-trivial rule:

$$(\mu\alpha\rho)' := \mu\alpha\exists\beta.(\beta \to \alpha) \times \rho'[\alpha := \beta] \ .$$

We even get strictly-positive non-interleaving fixed-point types (if strict positivity is defined in a way which allows existential quantifiers).

Lemma 8 also holds for this definition.

Define for every term r^ρ of IMF a term r' of type ρ' of $\mathsf{NPF}_{\mathsf{ex}}$ as follows (the other clauses are homomorphic):

[4] The well-known impredicative coding to be found e. g. in [5] p. 86 also preserves reduction.

$$(\mu\text{-I})\quad (\mathrm{C}_{\mu\alpha\rho}t)' := \mathrm{C}_{(\mu\alpha\rho)'}\mathrm{C}_{\exists\beta.(\beta\to(\mu\alpha\rho)')\times\rho'[\alpha:=\beta],(\mu\alpha\rho)'}\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle$$
$$(\mu\text{-E})\quad (r^{\mu\alpha\rho}\mathrm{E}_\mu m)' := r'\mathrm{E}_\mu\mathrm{E}_\exists\Big(\Lambda\beta\lambda z^{(\beta\to(\mu\alpha\rho)')\times\rho'[\alpha:=\beta]}.m'\beta(\mu\alpha\rho)'(z\mathsf{L})(z\mathsf{R})\Big).$$

Lemma 9 also holds for this definition.

The proof that we indeed get an embedding is changed only for (β_μ): Set $s := \Lambda\beta\lambda z^{(\beta\to(\mu\alpha\rho)')\times\rho'[\alpha:=\beta]}.m'\beta(\mu\alpha\rho)'(z\mathsf{L})(z\mathsf{R})$.

$$\begin{aligned}((\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu m)' &= \mathrm{C}_{(\mu\alpha\rho)'}\mathrm{C}_{\exists\beta.(\beta\to(\mu\alpha\rho)')\times\rho'[\alpha:=\beta],(\mu\alpha\rho)'}\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\mathrm{E}_\mu\mathrm{E}_\exists s\\
&\to \mathrm{C}_{\exists\beta.(\beta\to(\mu\alpha\rho)')\times\rho'[\alpha:=\beta],(\mu\alpha\rho)'}\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\mathrm{E}_\exists s\\
&\mapsto s(\mu\alpha\rho)'\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\\
&\to \Big(\lambda z^{\cdots}.m'(\mu\alpha\rho)'(\mu\alpha\rho)'(z\mathsf{L})(z\mathsf{R})\Big)\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\\
&\mapsto m'(\mu\alpha\rho)'(\mu\alpha\rho)'(\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\mathsf{L})(\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\mathsf{R})\\
&\to m'(\mu\alpha\rho)'(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)(\langle\lambda x^{(\mu\alpha\rho)'}x,t'\rangle\mathsf{R})\\
&\to m'(\mu\alpha\rho)'(\mu\alpha\rho)'(\lambda x^{(\mu\alpha\rho)'}x)t'\\
&= \Big(m(\mu\alpha\rho)(\mu\alpha\rho)(\lambda x^{\mu\alpha\rho}x)t\Big)'\end{aligned}$$

It is clear that the embeddings are also embeddings of $\mathsf{EMF}_{\mathsf{eta}}$ and $\mathsf{IMF}_{\mathsf{eta}}$ into $\mathsf{NPF}_{\mathsf{eta}}$.

7 Other Results

It is quite easy to extend the method in [12] to prove confluence of all of the systems considered in this paper. (For the systems of inductive types see [9].)

One may also prove strong normalization of PF which is NPF with interleaving allowed. One simply always has to take least or greatest fixed-points of the functions Ψ_I or Ψ_E. I believe that one may construct an embedding out of this proof which finally embeds PF into NPF.

Keeping the promise at the end of section 3 we finish by discussing Mendler's [10] system. It is an extension of system F by some kind of inductive and coinductive types. We shall concentrate on the inductive types: $\mu\alpha\rho$ is in the system if α occurs only positively in ρ. Interleaving is allowed. The term definition is extended by the usual rule $(\mu\text{-I})$ and some eliminiation rule:

$(\mu\text{-E})$ If $r^{\mu\alpha\rho}$ is a term and $s^{\forall\alpha.(\alpha\to\mu\alpha\rho)\to(\alpha\to\sigma)\to\rho\to\sigma}$ is a term, then $(r\mathrm{E}_\mu s)^\sigma$ is a term.

(β_μ) is replaced by some kind of recursion rule

$$(\beta_\mu)''\qquad (\mathrm{C}_{\mu\alpha\rho}t)\mathrm{E}_\mu s \mapsto s(\mu\alpha\rho)(\lambda y^{\mu\alpha\rho}y)(\lambda x^{\mu\alpha\rho}.x\mathrm{E}_\mu s)t\ .$$

Mendler does not give any explanation of this rule but in [13] one can find one and also much more discussion in [9]. It turns out that strong normalization does not need the restriction to positivity which is however exploited in the proof in [10]. Let us sketch an embedding of the system without positivity requirement into NPF (whence the system inherits strong normalization from NPF): Set

$$(\mu\alpha\rho)' := \mu\beta\forall\gamma.(\forall\alpha.(\alpha\to\beta)\to(\alpha\to\gamma)\to\rho'\to\gamma)\to\gamma\ ,$$

$$(\mathrm{C}_{\mu\alpha\rho}t)' := \mathrm{C}_{(\mu\alpha\rho)'}\Big(\Lambda\gamma\lambda z^{\tau}.z(\mu\alpha\rho)'(\lambda y^{(\mu\alpha\rho)'}y)(\lambda x^{(\mu\alpha\rho)'}.x\mathrm{E}_{\mu}\gamma z)t'\Big)$$

with $\tau := \forall\alpha.(\alpha \to (\mu\alpha\rho)') \to (\alpha \to \gamma) \to \rho' \to \gamma$ and

$$(r\mathrm{E}_{\mu}s)' := r'\mathrm{E}_{\mu}\sigma's' \ .$$

Following the procedure of section 6 one can indeed show that this gives an embedding.

Acknowledgements to Thorsten Altenkirch for many fruitful discussions and to the referees for their valuable hints.

References

1. Martín Abadi and Marcelo Fiore. Syntactic Considerations on Recursive Types. In *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, pp. 242–252. IEEE Computer Society, July 1996.
2. Thorsten Altenkirch. Strong Normalization for T+. Unpublished note, 1993.
3. Henk Barendregt. Lambda Calculi with Types. In S. Abramsky, D. Gabbay and T. Maibaum, editors. Volume 2 of *Handbook of Logic in Computer Science*, pp. 117–309. Oxford Univ. Press, 1993.
4. Herman Geuvers. Inductive and Coinductive Types with Iteration and Recursion. In B. Nordström, K. Pettersson and G. Plotkin, editors. *Preliminary Proceedings of the Workshop on Types for Proofs and Programs, Båstad, June 1992*, pp. 193–217 (`ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`).
5. Jean-Yves Girard, Yves Lafont and Paul Taylor. Proofs and Types. Cambridge Univ. Press, 1989.
6. Carl A. Gunter. Semantics of Programming Languages: Structures and Techniques. Foundations of Computing Series. The MIT Press, Cambridge, Massachusetts, 1992.
7. Daniel Leivant. Contracting Proofs to Programs. In P. Odifreddi, editor. *Logic in Computer Science*, Vol. 31 APIC Studies in Data Processing, Academic Press, pp. 279–327, 1990.
8. Ralph Loader. Normalisation by Translation. Unpublished note announced on the "types" mailing list on April 6, 1995.
9. Ralph Matthes. Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types. PhD thesis, University of Munich, 1998, to appear (available via `http://www.tcs.informatik.uni-muenchen.de/~matthes/`).
10. Nax P. Mendler. Recursive Types and Type Constraints in Second-Order Lambda Calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science*, pp. 30–36. IEEE Computer Society, June 1987.
11. Femke van Raamsdonk and Paula Severi. On Normalisation. Technical Report CS-R9545, CWI, June 1995.
12. Masako Takahashi. Parallel Reduction in λ-Calculus, *Information and Computation* 118, pp. 120–127, 1995.
13. Tarmo Uustalu and Varmo Vene. A Cube of Proof Systems for the Intuitionistic Predicate μ-, ν-Logic. In M. Haveraaen and O. Owe, editors, *Selected Papers of the 8th Nordic Workshop on Programming Theory (NWPT '96), Oslo, Norway, December 1996*, volume 248 of *Research Reports, Department of Informatics, University of Oslo*, pp. 237–246, May 1997.

Morphisms and Partitions of V-sets

Rick Statman *

Department of Mathematical Sciences
Carnegie Mellon University
Pittsburgh, PA 15213
statman@cs.cmu.edu

1 Introduction

The set theoretic connection between functions and partitions is not worthy of further remark. Nevertheless, this connection turns out to have deep consequences for the theory of the Ershov numbering of lambda terms and thus for the connection between lambda calculus and classical recursion theory. Under the traditional understanding of lambda terms as function definitions, there are morphisms of the Ershov numbering of lambda terms which are not definable. This appears to be a serious incompleteness in the lambda calculus. However, we believe, instead, that this indefinability is a defect in our understanding of the functional nature of lambda terms. Below, for a different notion of lambda definition, we shall prove a representation theorem (completeness theorem) for morphisms. This theorem is based on a construction which realizes certain partitions as collections of fibers of morphisms defined by lambda terms in the classical sense of definition.

Our notion of definition begins with a Curried context F and ends with another G. We think of a combinator M printed in the righthand part of our screen and F printed in the lefthand part

$$FM$$

We convert this screen to others by beta conversion and we ask which screens of the form

$$GN$$

will appear. We also ask whether this relationship between M and N is functional (modulo conversion). When this relationship is functional we obviously have an Ershov morphism. We shall show that every morphism can be represented in this way.

We shall also present a number of related results about morphisms and partitions.

* This research supported in part by the National Science Foundation CCR-9624681

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 313–322, 1999.

2 Preliminaries

For notation and terminology concerning lambda calculus we refer the reader to Barendregt [1]. For terminology and notation concerning recursion theory we refer the reader to Rogers [2] especially for the notions of e-reducibility and e-degree. In addition, for a discussion of DRE sets (Difference of RE sets sets) the reader should see Soare [3].

A combinator is a closed lambda term. A V-set is a set of combinators which is both recursively enumerable and closed under beta conversion. We have elsewhere called these sets "Visseral" [5] and "Varieties" [4] hence the name "V-set". $'n'$ is the numeral for n. $\#$ is the Gödel numbering of combinators from Barendregt [1] page 143. If X is a V-set then $\#X = \{\#M : M \in X\}$. $'M'$ is the numeral for the Gödel number of M. $Mb = (l, Ms)$ is the Ershov numbering introduced by Albert Visser in [7] 1.6. Here Ms is the set of equivalence classes of combinators under the equivalence relation of beta conversion and $l : \text{Nat} \to Ms$ satisfying $l(\#M) = (M/ = beta)$. A partial function $m : Ms \to Ms$ is said to be a partial morphism if there is a partial recursive function f with $\text{dom}(f) = \{\#M : M/ = beta \in \text{dom}(m)\}$ such that the diagram

$$\begin{array}{ccccccc} & m & & l & & f & & l & \\ Ms & \leftarrow & Ms & \leftarrow & \text{Nat} & \to & \text{Nat} & \to & Ms \end{array}$$

commutes. The relation $\sim Mb$ is defined by

$$\#M \sim Mb\ \#N \leftrightarrow M = beta\ N.$$

Given a V-set X a partition P of X is said to be V if the blocks of P are V-sets. Given a V-partition P of X , a set of natural numbers is said to be a (multi-) representation of P if it contains (at least) exactly one RE index for each block of P and nothing else. P is said to be RE if it has an RE multi-representation and P is said to be recursive if it has a recursive representation. The V-partitions of X form a lower semi-lattice whose meet operation is the usual meet in the lattice of partitions. The largest element in the semi-lattice is $\{X\}$ here denoted 1 and the smallest element is $\{M/ = beta : M \in X\}$ here denoted 0. When 0 is the subject of concern it is convenient to talk about systems of distinct representatives (SDR's) instead of representations. Indeed, for each representation R of 0 there is an SDR of $<$ or $=$ e-degree obtained by taking the first member in the domain of $\{e\}$ for each e in R. Similarly for each SDR S of 0 there is a representation of $<$ or $=$ e-degree obtained by taking an index for the function which converges on precisely those N such that $N = beta\ M$, for each $M \in S$. We define the degree of X, $\deg(X)$, to be the infinum of the e-degrees of the SDR's of 0, if this exists.

3 Morphisms

If you read Visser's paper [7] carefully you learn enough to prove the following.

Theorem 1. (Visser fixed point theorem): Suppose that S is an

RE set, X is a V-set not including all combinators,
and G is any combinator. Then there exists
a combinator F such that

$$F'n' = beta \begin{cases} G\ 'F'\ 'n' & \text{if } n\ \in\ S \\ \text{does not belong to } X & \text{otherwise.} \end{cases}$$

Proof: Let E be the Kleene enumerator constructed in [1] page 167. Let f be the usual partial recursive enumeration of the partial recursive functions of one variable so that if g has index j then $f(j,x) = g(x)$. By Visser's theorem [7] 1.6.1.1 there exists a total recursive function h such that whenever $f(x,x)$ converges we have $f(x,x)\ \sim Mb\ h(x)$. Now g be partial recursive with index j. Kleene proved [1] that there is a combinator H such that

$$H\ 'n' \begin{cases} 'g(n)' & \text{if} g(n) \text{ converges} \\ \text{a term with no normal form} & \text{else} \end{cases}$$

Now the map $j \mapsto \#H$ is total recursive and thus has an index i. This i is also mapped to a combinator H here denoted J . Thus for any j and H with $j \mapsto \#H$ we have $E(J\ 'j') = betaH$. Now define the total recursive a by $a(x,y) = \#G'(\lambda\ z.E(E(J\ 'x')z)'\ 'y')$ and x define the partial recursive c by

$$c(x,y,z) = \begin{cases} a(x,y) \text{ if } y\ \in\ S \text{ before } h(z)\ \in\ \#X \\ \qquad \text{in some given enumerations of } S \text{ and } X \\ k \qquad \text{if } h(z) \in \#X \text{ before } y\ \in\ S \end{cases}$$

for k any natural number not in $\#X$. By the $S(m,n)$ theorem ([2]) there exists a total recursive b such that $c(x,y,z) = f(b(x,y),z)$. Let $d(x,y) = h(b(x,y))$. We observe that $y\ \in\ S \Rightarrow c(x,y,b(x,y))$ converges $\Rightarrow d(x,y)\ \sim Mb\ c(x,y,b(x,y))$ $= a(x,y)$. $\sim (y\ \in\ S) \Rightarrow c(x,y,b(x,y))$ diverges $\Rightarrow d(x,y)$ does not belong to $\#X$. Now by the recursion theorem there exists a j such that $d(j,y) = f(j,y)$. We now set $F = \lambda\ z.E(E(J\ 'j')z)$ and we claim that this is the desired combinator. Suppose first that $y\ \in\ S$. We have

$f(j,y) = d(j,y)\ \sim Mb\ a(j,y)$ so

$E\ 'f(j,y)' = beta\ E\ 'a(j,y)'$ and

$E(E(J\ 'j')y) = beta\ G\ '\lambda\ z.E(E(J\ 'j')z)'y,$ thus

$Fy = beta\ G\ 'F'y.$

On the other hand if $\sim (y \in S)$ then $f(j,y) = d(j,y)$ which does not belong to $\#X$, so $F\ 'y' = beta\ E(E(J\ 'e')\ 'y') = beta\ E\ 'f(j,y)'$ which does not belong to X.
This completes the proof.

Theorem 2 (morphism extension): Suppose that m is a partial morphism and X is a V-set not containing all combinators. Then there exists a total morphism n extending m such that if $M/ = beta$ does not belong to dom(m) then $n(M/ = beta)$ is disjoint from X .

Proof: Suppose that m and X are given together with the partial recursive function f which makes the morphism diagram commute. Now we can construct a combinator H such that

$$Hx\ 'M' = beta \begin{cases} Ex\ 'N' & \text{if } N = beta\ M \& \#N < \#M \& N \text{ is the first} \\ & \text{such found in some enumeration of} \\ & \text{the beta converts of } M \\ E\ 'f(\#M)' & \text{if } f(\#M) \text{ converges (i.e., the calculation} \\ & \text{of } f(\#M) \text{ terminates) before such } N \text{ is found.} \end{cases}$$

Where E is the Kleene enumerator constructed in [1] page 167. Thus by the Visser fixed point theorem there exists a combinator F such that

$$F\ 'M' = beta \begin{cases} H\ 'F'\ 'M' & \text{if there exists } N = beta\ M \text{ with} \\ & \#N < \#M \text{ or } f(\#M) \text{ converges} \\ \text{does not belong to } X & \text{otherwise.} \end{cases}$$

Finally, let h be defined by $h(\#M) = \#(F\ 'M')$. Clearly h is a total recursive function. Moreover it is easy to see that

(1) if $M = beta\ N$ then $h(\#M) \sim Mb\ h(\#N)$
(2) if $f(\#M)$ converges then $f(\#M) \sim Mb\ h(\#M)$.
(3) if $f(\#M)$ diverges then $l(h(\#M))$ does not belong to X.

Thus there is a total morphism n with the desired properties which makes the above diagram commute for h. End of proof.

Each combinator M induces a morphism via the recursive map $\#N \mapsto \#MN$. However, there are morphisms not induced by such combinator application. For example, take a left invertible combinator such as $C* = \lambda\ xy.yx$ with left inverse $C * I$. Given X define $X+$ by

$x+ = x$

$(XY)+ = C * I(X+)(Y+)$

$(\lambda\ x.X)+ = C * (\lambda\ x.(X+))$.

Then the map $M \mapsto M+$ induces a morphism which is not induced by combinator application (just assume that $F\ Omega\ =\ beta\ Omega+$, where Omega $= (\lambda x.xx)(\lambda x.xx)$).

Theorem 3: (morphism representation): Suppose that m is a morphism. Then there exist combinators F and G such that for any M and N we have

$$m(M/ = beta) = (N/ = beta) \Leftrightarrow FM = betaGN.$$

Proof: We shall actually prove a stronger result. Suppose that P is an RE V-partition of the V-set X . We shall construct a combinator H such that $HM = beta\ HN \Leftrightarrow (M = beta\ N) \vee (\ M$ and N belong to the same block of P (and thus belong to X)). Given this we get the theorem as follows. Suppose that m is given. Let $X = \{\langle'0', M\rangle :$ all combinators $M\} U \{\langle'1', N\rangle : N/ = beta\ \in\ rng(m)\}$ and let $P = \{\{\langle'0', M\rangle : m(M/ = beta) = (N/ = beta)\} U \{\langle'1', N\rangle\} :$ for all $N/ = beta\ \in\ rng(m)\}$ where $\langle,\rangle$ is the pairing function in [1] pg 133. Then P is RE. Given H as above we set $F = \lambda\ x.H\langle'0', x\rangle$ and $G = \lambda\ x.H\langle'1', x\rangle$. Now $FM = beta\ GN \Leftrightarrow H\langle'0', M\rangle = beta\ H\langle'1', N\rangle \Leftrightarrow \langle'0', M\rangle$ and $\langle'1', N\rangle$ belong to the same block of $P \Leftrightarrow m(M/ = beta) = (N/ = beta)$.

Suppose now that X and P are given. Fix an RE representation of P . By Kleene's enumerator construction ([1] pg 167) there exists a combinator E such that $E\ '(2r+1) * (2^s)' = beta$ the s-th member of the r-th block of P where, of course, the same block of P may occur many times in the enumeration because P is only assumed to be RE. We make the following definitions; let a be a new free variable

$S = \lambda\ xyz.\ y(xyz)$
$T = \lambda\ xyz.\ xy(xyz)$
$Y = (\lambda\ xy.\ y(xxy))(\lambda\ xy.y(xxy))$
$A = \lambda\ fgxyz.\ fx(a(Ex))[f(Sx)y(g(Sx))z]$
$B = \lambda\ fgx.f(Sx)\ (a(E(Tx)))(g(Sx))(gx)$
$G = Y(\lambda\ u.\ B(Y(\lambda\ v.Auv))u)$
$F = Y(\lambda\ u.\ AuG)$
$H = \lambda\ xa.\ F\ '1'(ax)(G\ '1')$.

We now calculate for $k > 0$

$H(E\ 'k') \to \lambda\ a.F\ '1'(a(E\ 'k'))(G\ '1') \twoheadrightarrow$

$\lambda\ a.F'1'(a(E\ '1'))[F\ '2'(a(E\ 'k'))(G\ '2')(G\ '1')] \twoheadrightarrow$

$\lambda\ a.F\ '1'(a(E\ '1'))(F\ '2'(a(E\ '2'))(...F\ 'k-1'(a(E\ 'k-1'))$
$[F\ 'k'(a(E\ 'k'))(G\ 'k')(G\ 'k-1')]...)(G\ '2'))(G\ '1') \twoheadrightarrow$

$\lambda\ a.F\ '1'(a(E\ '1'))\ (F\ '2'(a(E\ '2'))\ (...F\ 'k-1'(a(E\ 'k-1'))\ [F\ 'k'$
$(a(E\ 'k'))[F\ 'k+1'(a(E(T\ 'k'))(G\ 'k+1')(G\ 'k')](G\ 'k-1')]\ ...)$
$(G\ '2'))(G\ '1') \twoheadrightarrow$

$\lambda\ a.F\ '1'(a(E\ '1'))\ (F\ '2'(a(E\ '2'))\ (...F\ 'k-1'(a(E\ 'k-1'))\ [F\ 'k'(a(E\ 'k'))$
$[F\ 'k+1'(a(E('2k'))(G\ 'k+1')(G\ 'k')](G\ 'k-1')]\ ...)\ (G\ '2'))\ (G\ '1')$
$\twoheadleftarrow H(E\ '2k').$

Thus we conclude that if M and N belong to the same block of P then $HM = beta\ HN$. Next we prove that if $HM = betaHN$ then either $M = beta\ N$ or M and N belong to the same block of P . Suppose that $F\ 'k'(aM)(G\ 'k') = beta\ F\ 'k'(aN)(G\ 'k')$. Then by the Church-Rosser and standardization theorems there exists a common reduct Z and standard reductions R1 : $F\ 'k'(aM)(G\ 'k') \twoheadrightarrow Z$ and R2 : $F\ 'k'(aN)(G\ 'k') \twoheadrightarrow Z$. The proof is by induction on the sum of the lengths of these standard reduction sequences. The following facts about the definitions are easy to verify and will be used below.

(1) $Fxyz$ is an order zero unsolvable
(2) $\lambda\ xy.y(xxy) = / = beta\ A$
(3) $Y = / = beta\ \lambda\ u.AuG$
(4) $Y = / = beta\ AF$
(5) $\lambda\ u.AuG = / = beta\ AF.$

Basis: the sum of the lengths of the reductions is 0. In this case it is clear that $M = beta\ N$. Induction step; the sum of the lengths is say $r > 0$. For any term X the head reduction of $F\ 'k'(aX)(G\ 'k')$ cycles through segments which are 8 terms long viz; $\mathrm{nt}(\lambda\ x.x(Yx))(\lambda\ u.B(Y(\lambda\ v.Avu))u)\ 'k' \rightarrow$

$(\lambda\ u.B(Y(\lambda\ v.Avu))u)G\ 'k' \rightarrow$

$BFG\ 'k' \rightarrow$

$(\lambda\ gx.F(Sx)(a(E(Tx)))(g(Sx))(gx))G\ 'k' \rightarrow$

$(\lambda\ x.F(Sx)(a(E(Tx)))(G(Sx))(Gx))\ 'k' \rightarrow$

$F(S\ 'k')(a(E(T\ 'k')))(G(S\ 'k'))(G\ 'k')$

and none of the heads of any of these terms except the last is of order 0. Thus the head reduction part of the standard reduction of $G'k'$ goes at least this far. Indeed, we may assume that $t = 0$ and the induction hypothesis applies to the pair $F(S\ 'k')(aM)(G(S\ 'k'))$ and $F(S\ 'k')(a(E(T\ 'k')))(G(S\ 'k'))$. This completes the proof for case 1.

Case 2: both R1 and R2 complete the full cycle of the first 8 head reductions. W.l.o.g. we may assume that the head reduction part of R2 ends in

$$V((F(S\,'k')(a(E\,'k'))(G(S\,'k'))\,(...(F(S\,'k')(a(E\,'k'))(G(S\,'k')))\\ [F(S\,'k')(aN)(G(S\,'k'))(G\,'k')]...)$$

where $(F(S\,'k')(a(E\,'k'))(G(S\,'k'))$ appears $s <$ or $= t$ times,and $F(S\,'k')$ $(a(E\,'k'))$ head reduces to V in $<$ or $= 7$ steps. We distinguish two subcases.

Subcase 1: $s = t$. In this case, by fact (1), the induction hypothesis applies to the pair $F(S\,'k')(aM)(G(S\,'k'))$ and $F(S\,'k')(aN)(G(S\,'k'))$

Subcase 2: $s < t$. In this case, by fact (1), the induction hypothesis applies to the pair $F(S\,'k')(aN)(G(S\,'k'))$ and $F(S\,'k')(a(E\,'k'))(G(S\,'k'))$. In addition, $G\,'k'$ and $F(S\,'k')(a(E\,'k'))(G(S\,'k'))(...(F(S\,'k')(a(E\,'k'))(G(S\,'k')))[(F(S\,'k')$ $(aM)(G(S\,'k'))(G\,'k')...)$ with $t-s$ occurrences of $F(S\,'k')(a(E\,'k'))(G(S\,'k'))$, have a common reduct by standard reduction whose total length is $< r$. Here the argument of case 1 applies so that we may conclude that there exists an m whose odd part is the same as the odd part of k such that $M = beta\ E\,'k'$. This completes the proof of case 2.

It is interesting to note here that uniformization of the relation defined by the equation $FM = beta\ GN$ cannot always be carried out by a morphism, For example, if $F = I$ and $G = E$ then for each M there exists N such that $FM = beta\ GN$ namely $N = \#M$ but there is no morphism which takes $M/ = beta$ to the equivalence class of one particular $\#N$ for $N = beta\ M$. For, such a morphism would solve the beta conversion problem effectively.

4 Recursive partitions

When it comes to the subsemilattice of recursive V-partitions of X it can be very small including only 1 or very large including also 0. Our study actually began with trying to understand the following theorem. This lead to theorem 6 and then theorem 3.

Theorem 4 (Grzegorczyk-Scott): The only recursive V-partition of the set of all combinators is 1.

We know 4 constructions which yield V-sets whose only recursive V-partition is 1. Of course, if X is such a V-set then the image of X under any morphism also has this property.

Construction 1 (direct construction by the recursion theorem):

It is convenient to formulate this construction in terms of the precompleteness of the Ershov numbering. Given partial recursive functions f and g define another partial recursive function h which matches the members of the domain of f to the members of the domain of g in the order in which they converge. By precompleteness h "extends" to a total recursive d. Moreover, the index of d is a total recursive function $c(x, y)$ of the indicies of f and g. By the Ershov fixed point theorem there exists a $z = b(x, y)$ which is a fixed point of d i.e., $z \sim Mb\ d(z)$. Now if we let X be the smallest V-set such that $\#M \in dom(f) \Rightarrow M \in X$, Y the smallest V-set such that $\#M \in dom(g) \Rightarrow M \in Y$, and $\#M = b(x, y)$, then $M \in Union(X, Y) \Rightarrow M \in Intersection(X, Y)$. In other words b is a productive function for V-sets. In addition there is a well known total recursive function $a(x, y)$ which gives the index of a partial recursive function whose domain is the intersection of the domains of $\{x\}$ and $\{y\}$. Finally let $j(x)$ be the index of the RE set $\{b(a(x, y), a(x, z)) :$ all natural numbers $y, z\}$. By the recursion theorem there exists an index i such that $W(i) = W(j(i))$. It is easy to verify that the beta conversion closure of $\{M : \#M \in W(i)\}$ cannot be partitioned into two RE blocks which respect beta conversion. It is not clear to us exactly when this construction yields the set of all combinators. We leave this question open.

Construction 2 (the Grzegorczyk-Scott construction):

This construction uses the fixed point theorem. We suppose that we have a partition $\{X, Y\}$ with $M \in X$ and $N \in Y$. We define a total recursive f by $L \in X \Rightarrow f(\#L) = \#N$ and $L \in Y \Rightarrow f(\#L) = \#M$. Here we can use the Ershov fixed point theorem, where Scott and Grzegorczyk use Kleene's representation of the recursive functions and the traditional fixed point theorem. There exists L such that $f(\#L) \sim Mb\ \#L$. This contradicts the choice of the definition of f. This type of arguement applies to fragnments of lambda calculus suitable for representing all recursive functions such as the lambda I calculus or the combinators hereditarily of order 2 (HOT[6]).

Construction 3 (Visser's construction):

Visser's construction begins like construction (2) except we do not assume that X and Y are disjoint; we assume only that $M \in X - Y$ and $N \in Y - X$. Let A and B be a pair of recursively inseparable sets and let A and B be the beta conversion closure of the set Church numerals representing the members of A and, resp., of B. By the precompleteness of the Ershov numbering the partial recursive function f defined by $L \in A \Rightarrow f(\#L) = \#M$ and $L \in B \Rightarrow f(\#L) = \#N$ "extends" to a total recursive function g. It is easy to obtain a recursive separation of A and B using g, X, and Y . Indeed Visser's arguement reachees an apparently stronger conclusion; viz, the set of all combinators cannot be covered by two incomparable V-sets

Construction 4 (a simple set construction):

Let $W(i)$ be a simple set and let f be the partial morphism defined by $f(\#M) = \#I$ if there exists a member of $W(i) \sim M\ \#M$. By theorem 2 there exists a total morphism g "extending" f, and there exists a combinator F representing g. Consider the V-set Z which is the beta conversion closure of $\{E(F\ 'M') :$ all combinators $M\}$. Now suppose that X, Y is a partition of Z into V-sets. First note that if $E(F\ 'M') \in X$ then there are infinitely many N beta convertible to M such that $E(F\ 'N')$ also belongs to X and similarly for Y . Now I belongs to one of these sets say X and let B be the set of all k such that $E(F\ 'k') \in Y$. Since B is infinite and RE it must intersect $W(i)$ in, say, the natural number k. But since $k \in W(i)$ we have $E(F\ 'k') = beta\ I \in X$. This is a contradiction.

0 can be recursive or not recursive. An example of the first is $X =$ the beta conversion closure of the set of normal forms. An example of the second is $X =$ the set of all combinators. Indeed, an SDR for this X solves the beta conversion problem.

If 0 is recursive then there is a clear sense in which every recursive partition of natural numbers yields a recursive partition of X. The canonical example is the beta closure of the Church numerals. In [3] Visser observed the following.

Theorem 5 (Visser) : If 0 is recursive then any morphism into X is constant.

The following theorem explains where all the RE V-partitions of a V-set come from. It is a corollary to the proof of theorem 3.

Theorem 6 (partition representation theorem): Suppose that P is an RE V-partition of the V-set X . Then there exists a combinator H such that $HM = beta\ HN \Leftrightarrow M = beta\ N \vee\ M$ and N belong to the same block of P (and thus belong to X).

5 Partitions of higher complexity

It turns out that the most natural notion of the complexity of an SDR for 0 is the notion of e-degree (partial or enumeration degree [2]). The following theorem characterizes those e-degrees.

Theorem 7 (existence of deg(X)): deg(X) exists and is the e-degree of a DRE SDR for 0. Moreover, each DRE e-degree is deg(X) for some V-set X.

Proof: Given a V-set X define $X* = \{\langle M, N\rangle : M = / = beta\ N$ for $M, N \in X\}$. Now let S be any SDR for 0. Define the enumeration operator F by $\{\langle M, N, \langle P, Q\rangle\rangle : P = beta\ M\&Q = beta\ N\&M = / = N\}$. Then $F(S) = X*$. Now fix an enumeration of X . Define a sequence M(1), M(2),..., by $M(m)$ is the

first element in the enumeration of X not $=$ *beta* to one of the $M(1), ..., M(m-1)$. This sequence forms and SDR S for 0. Now define the enumeration operator G by $\{\langle\langle N(1), N(2)\rangle, \langle N(1), N(3)\rangle, ..., \langle N(n-2),\ N(n-1)\rangle, N(n)\rangle : N(1), ..., N(n)$ appear in order in an initial segment of the fixed enumeration of $X\}$ and the other members of the initial segment *beta* convert to one or more of the $N(i)$. Then $G(X*) = S$. Thus the e-degree of $S =$ the e-degree of $X*$ which is less than or equal to the e-degree of any SDR for 0. In addition, S is the difference of RE sets since it is the difference of the members of the enumeration and the members of the enumeration that beta convert to earlier members. This proves the first part of the theorem. To prove the second part let $D = A - B$ be the difference of the RE sets A and B. By the Visser fixed point theorem there exists a combinator F such that

$$Fn \begin{cases} KI & \text{if } n \in B \\ \text{a term of order 0} & \text{otherwise.} \end{cases}$$

Let X be the beta conversion closure of the set $\{F'n''n' : n$ a natural number$\}$. Now for this X clearly 0 has an SDR consisting of $\{I\}$ union with $\{F'n''n' : n \in A\& \sim (n\ \in\ B)\}$, since, by the construction of F, no two of the latter set can beta convert. Thus deg(X) is e-reducible to D. On the other hand, it is clear that D can be enumerated from $X*$ and thus deg$(X) =$ the e-degree of D. This completes the proof.

References

1. Barendregt, H., The Lambda Calculus, North Holland, (1983).
2. Rogers, H., Theory of Recursive Functions McGraw Hill, (1967).
3. Soare, R., Recursively Enumerable Sets and Degrees Springer -Verlag, (1993).
4. Statman, R., On sets of solutions to combinator equations, *TCS*, **66** (1989) pgs. 99-104.
5. Statman, R., The Visser fixed point theorem, unpublished manuscript.
6. Combinators hereditarily of order two, CMU Math. Dept. Research Report 88-33, (1988).
7. Statman, R., Visser, Numerations, lambda calculus, and arithmetic in *To H.B. Curry: Essays On Combinatory Logic, Lambda Calculus, And Formalism*, Hindley and Seldin eds., Academic Press (1972), pgs. 259-284.

Computational Adequacy in an Elementary Topos

Alex K. Simpson

LFCS, Division of Informatics, University of Edinburgh
JCMB, King's Buildings, Edinburgh, EH9 3JZ
Tel. +44 131 650 5113
Fax. +44 131 667 7209
Alex.Simpson@dcs.ed.ac.uk

Abstract. We place simple axioms on an elementary topos which suffice for it to provide a denotational model of call-by-value PCF with sum and product types. The model is synthetic in the sense that types are interpreted by their set-theoretic counterparts within the topos. The main result characterises when the model is computationally adequate with respect to the operational semantics of the programming language. We prove that computational adequacy holds if and only if the topos is 1-consistent (i.e. its internal logic validates only true Σ^0_1-sentences).

1 Introduction

One axiomatic approach to domain theory is based on axiomatizing properties of the category of *predomains* (in which objects need not have a "least" element). Typically, such a category is assumed to be bicartesian closed (although it is not really necessary to require all exponentials) with natural numbers object, allowing the denotations of simple datatypes to be determined by universal properties. It is well known that such a category cannot have a fixed-point operator [9], but crucially predomains have a *lift* monad acting on them which plays a critical role in recovering the more familiar category of *domains* in which the expected fixed-point operator resides. The lift monad also determines the subcategory of strict functions between domains, with respect to which the fixed-point is characterised by the property of *uniformity*. Further, the monad determines a category of partial functions, which is arguably the most suitable category for program semantics. The development of this viewpoint can be found in [23,2,18,29,3].

In recent years it has become apparent that many natural categegories of predomains arise as full subcategories of elementary toposes. For example, the category of ω-complete partial orders and ω-continuous functions is a full reflective subcategory of the Grothendieck topos, $\mathcal{H}$, considered in [6,5]. More generally, models of Fiore and Plotkin's axioms for domain theory also have such embeddings [4]. Other categories of predomains are found as full subcategories of realizability toposes, see [15] for examples and references. Certain such

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 323–342, 1999.

examples have been shown to account for phenomena such as effectivity [21,7] and strong stability [19]. Work in progress by a number of researchers looks likely to establish similar embeddings for categories of games.

The wealth of examples suggests that elementary toposes provide a plausible environment for developing a unified account of the many models of domain theory. However, early axiomatizations of domain theory inside toposes, [10,31,27,24], included axioms that are valid only in restricted classes of models. A general axiomatization, encompassing both parallel and sequential computation, was proposed in [15], where its consequences were worked out in detail in the specific case of realizability toposes. This axiomatization has since proved to be applicable also to: Grothendieck toposes [5,4], models of intuitionistic Zermelo-Frænkel set theory [30], and models of intuitionistic type theory [25].

In this paper we consider the same general axiomatic approach in the setting of an elementary topos. Given an elementary topos with a natural numbers object and a distinguished dominance (determining a lift monad), we isolate a full subcategory of predomains, the *well-complete* objects, as in [15]. Under a first (now ubiquitous) axiom, this category is closed under function spaces, products and the lift functor and supports recursion on appropriate objects. To obtain closure under finite coproducts (in the topos), it is necessary to strengthen the axiom in a simple way. For the natural numbers to be a predomain, yet another simple strengthening is necessary in general, although not in the case that Markov's Principle is valid in the topos (Theorem 1).

The axioms are sufficient to enable any of the many variants of PCF [22] to be modelled in the topos. Moreover, the closure properties of predomains have the important consequence that datatypes are modelled by their set-theoretic counterparts within the internal logic of the topos. Obtaining such a set-theoretic interpretation of type constructors was one of Dana Scott's motivations for proposing synthetic domain theory. He hoped that the set-theoretic viewpoint would lead to simple and intuitive logics for reasoning about programs. Whether or not this proves to be the case, we would like to stress another motivation for the work in this paper, and for the programme of synthetic domain theory as a whole. The axioms of synthetic domain theory allow technically distinct constructions across a wide range of models to be uniformly accounted for as examples of the same set-theoretic construction (when regarded from the viewpoint of the internal logic). Thus synthetic domain theory offers a general and unifying axiomatic account of the constructions of domain theory, and one which applies across a very wide class of models.

The main goal of this paper is to exploit this axiomatic account to understand the induced interpretation of PCF-like langages in the topos. For maximum generality, we consider a call-by-value version with sum and product types. The question we consider is: When is the interpretation of the language in the topos *computationally adequate* with respect to its operational semantics [8,32]? We obtain a complete characterisation in terms of the internal logic of the topos. We prove that computational adequacy holds if and only if the topos is 1-consistent (Theorem 2). (A topos is said to be 1-consistent if it validates only true Σ^0_1-

sentences.) Thus computational adequacy is equivalent to a simple logical property whose statement makes no reference to PCF. It is to be expected that an identical result will hold for other programming languages too.

2 Partiality and Lifting

Throughout this paper we assume that $\mathcal{E}$ is an elementary topos with a natural numbers object $\mathbf{N}$, see e.g. [13,17]. We write $\mathbf{0}$ and $\mathbf{1}$ for chosen initial and terminal objects respectively, $[0, s] : \mathbf{1} + \mathbf{N} \longrightarrow \mathbf{N}$ for the structure map of $\mathbf{N}$ as the initial algebra of the endofunctor $1 + (-)$ on the topos, and *pred* : $\mathbf{N} \longrightarrow \mathbf{1} + \mathbf{N}$ for its inverse. We write $\mathbf{2}$ for the object $\mathbf{1} + \mathbf{1}$, denoting the left and right injections by $\bot, \top : \mathbf{1} \longrightarrow \mathbf{2}$. We consider $\mathbf{2}$ as a subobject of the subobject classifier, Ω, via a distinguished mono taking $\bot, \top : \mathbf{1} \longrightarrow \mathbf{2}$. to the standard $\bot, \top : \mathbf{1} \longrightarrow \Omega$.

We shall require one other piece of primitive data, an object Σ arising as a subobject $\Sigma \rightarrowtail \Omega$ (thus Σ classifies a family of subobjects, the *Σ-subobjects*, in $\mathcal{E}$). Moreover, we require that Σ is a *dominance* in the sense of Rosolini [26,3]. Specifically this means that $\top : \mathbf{1} \longrightarrow \Omega$ factors through the mono $\Sigma \rightarrowtail \Omega$ (i.e. all isomorphisms are Σ-subobjects) and that Σ-subobjects are closed under composition.

Because $\mathbf{2}$ and Σ are subobjects of Ω, we shall often consider them, in the internal logic, as subsets of the set of all propositions. In particular, we sometimes refer to $\mathbf{2}$ as the subset of *logically decidable* propositions, because $\mathbf{2}$ can be defined internally as $\{p \in \Omega \mid p \vee \neg p\}$.

The conditions above imply that the collection of partial maps in $\mathcal{E}$ with Σ-subobjects for domains forms a category under the usual composition of partial maps. We write $X \rightharpoonup Y$ for the object of such *Σ-partial maps* from X to Y (easily defined using the internal logic). For $f \in X \rightharpoonup Y$ and $x \in X$, we shall loosely write $f(x)$ to mean a possibly undefined element of Y. We use equality between such possibly undefined expressions in the strict sense, thus $e = e'$ means that both e and e' are defined and equal. We write $e\!\downarrow$ to mean that e is defined, i.e. the Σ-property $\exists y \in Y.\, e = y$ holds (such a y is necessarily unique). We use Kleene equality, $e \simeq e'$ to mean that e is defined iff e' is, in which case both are equal. The above notation is adopted for its readability. The formally inclined reader will have no problem in translating the expressions we use into an appropriate rigorous logic for partial terms (see e.g. [28]), although our partial terms will always be special ones whose definedness property is a Σ-proposition.

The operation $\mathbf{1} \rightharpoonup (-)$ determines, in the obvious way, an endofunctor on $\mathcal{E}$, the *lift* functor L. Given $e \in LX$, we write $e\!\downarrow$ in place of the more cumbersome $e(*)\!\downarrow$ (where we write $*$ for the unique element of $\mathbf{1}$), and similarly, when $e\!\downarrow$ we write e for the determined value $e(*) \in X$. We write $\chi : LX \longrightarrow \Sigma$ for the morphism mapping any $e \in LX$ to its definedness property $e\!\downarrow$, which is indeed a Σ proposition.

The lift functor carries the structure of a monad (L, η, μ) on $\mathcal{E}$. Its unit $\eta : X \longrightarrow LX$ maps any x to the unique $e \in LX$ such that $e = x$ (thus

$e\downarrow$). The lift monad is a strong commutative monad satisfying some additional well-documented properties, see e.g. [26,3,1]. The Kleisli category, $\mathcal{E}_L$, of the lift monad is isomorphic to the category of Σ-partial maps on $\mathcal{E}$. The object Σ is isomorphic to $L\mathbf{1}$ and $X \rightharpoonup Y$ is isomorphic to the exponential $(LY)^X$. An alternative, but equivalent, development is to take the lift monad as primitive (as in [5]) and to derive the dominance and partial function space using the above isomorphisms

One consequence of $\mathcal{E}$ having a natural numbers object, is that the functor, L, has both an initial algebra $\sigma : L\mathbf{I} \longrightarrow \mathbf{I}$ and a final coalgebra $\tau : \mathbf{F} \longrightarrow L\mathbf{F}$. Moreover, the object $\mathbf{F}$ is a retract of $\Sigma^{\mathbf{N}}$, and the unique algebra homomorphism from σ to τ^{-1} is a mono $\iota : \mathbf{I} \rightarrowtail \mathbf{F}$. These results were proved in 1995 by the author and Mamuka Jibladze independently. For a published account see [11].

The initial algebra of L as a functor, interacts nicely with the monad structure on L. We call a morphism $\alpha : LX \longrightarrow X$ a *monad algebra* if it is an Eilenberg-Moore algebra for the monad (L, η, μ) [16]. The morphism $\mu' = \sigma \circ \mu \circ L\sigma^{-1}$ is a monad algebra $L\mathbf{I} \longrightarrow \mathbf{I}$. We write $up : \mathbf{I} \longrightarrow \mathbf{I}$ for the composite $\sigma \circ \eta$. The result below appears as Theorem A.5 of [12], where it is attributed to Bénabou and Jibladze.

Proposition 1. *For any monad algebra $\alpha : LX \longrightarrow X$ and any morphism $f : X \longrightarrow X$, there exists a unique algebra homomorphism $h : \mu' \longrightarrow \alpha$ such that $f \circ h = h \circ up$ (in $\mathcal{E}$).*

An important example of a monad algebra is $\alpha : L(X \rightharpoonup Y) \longrightarrow (X \rightharpoonup Y)$ defined by $\alpha(e)(x) \simeq e(x)$. When referring to such monad algebras, we shall just refer to the underlying object $X \rightharpoonup Y$, always understanding the structure map to be that given above.

3 Completeness and Fixed-points

The mono $\iota : \mathbf{I} \rightarrowtail \mathbf{F}$ plays a fundamental role in developing a basic notion of "chain completeness" sufficient for establishing fixed-points for endomorphisms on suitable objects. For further motivation see [15]. The results in this section are, by now, standard [15,27,25].

Definition 1. An object X is said to be *complete* if the induced morphism $X^\iota : X^{\mathbf{F}} \longrightarrow X^{\mathbf{I}}$ is an isomorphism.

If X is complete and $\alpha : LX \longrightarrow X$ is a monad algebra, then, for any morphism $f : X \longrightarrow X$ let $h : \mathbf{I} \longrightarrow X$ be the unique algebra homomorphism given by Proposition 1. Because X is complete, $h : \mathbf{I} \longrightarrow X$ extends along ι to a unique morphism $\overline{h} : \mathbf{F} \longrightarrow X$. Let $\infty : \mathbf{1} \longrightarrow \mathbf{F}$ be the unique coalgebra homomorphism from $\eta : \mathbf{1} \longrightarrow L\mathbf{1}$ to $\tau : \mathbf{F} \longrightarrow L\mathbf{F}$. We write $fix_\alpha(f)$ for the point $\overline{h} \circ \infty$.

Proposition 2. *For any monad algebra $\alpha : LX \longrightarrow X$ with X complete, and any morphism $f : X \longrightarrow X$, it holds that $f \circ fix_\alpha(f) = fix_\alpha(f)$. (i.e. fix is a fixed-point operator.)*

Moreover, for any monad algebras $\alpha : LX \longrightarrow X$ and $\beta : LY \longrightarrow Y$ (where X and Y are complete), morphisms $f : X \longrightarrow X$ and $g : Y \longrightarrow Y$, and algebra homomorphism $h : X \longrightarrow Y$, if $h \circ f = g \circ h$ then $fix_\beta(g) = h \circ fix_\alpha(f)$. (i.e. fix is uniform.)

Because the property of Σ being a dominance can be expressed in the internal logic [26], the dominance transfers to any slice topos $\mathcal{E}/Z$. Thus the above proposition holds in any slice. Essentially, this means that the obvious internalization of the above proposition holds in the internal logic of $\mathcal{E}$.

Our main application of uniformity will be in the proof of Lemma 5 in Section 7, where it will be used in the following form. Suppose that X is complete and carries a monad algebra $\alpha : LX \longrightarrow X$. We say that a mono $m : X' \rightarrowtail X$ *carries a subalgebra* of α if there exists $\alpha' : LX' \rightarrowtail X'$ such that $m \circ \beta = \alpha \circ Lm$. One easily shows that such an α' is unique, and further that it is itself a monad algebra. (Indeed, it holds for an arbitrary monad that any subalgebra of a monad algebra is also a monad algebra.) Now suppose that X' is complete and that an endomorphism $f : X \longrightarrow X$ restricts to an endomorphism $f' : X' \longrightarrow X'$ (i.e. that $m \circ f' = f \circ m$). Then it follows from the uniformity of fixed-points that $m \circ fix_{\alpha'}(f') = fix_\alpha(f)$, i.e. that $fix_\alpha(f)$ factors through the subobject $m : X' \rightarrowtail X$.

4 Axioms for Synthetic Domain Theory

The complete objects are not, in general, closed under the lifting functor, L, [20]. As our category of predomains we take the largest full subcategory of complete objects that *is* closed under lifting, following [15].

Definition 2. An object X is said to be *well-complete* if LX is complete.

We write $\mathcal{C}$ for the full subcategory of well-complete objects of $\mathcal{E}$. We shall adopt $\mathcal{C}$ as our category of predomains.

Our axioms will all be assertions that useful objects are well-complete. We begin with the basic *completeness axiom* of [15].

Axiom 1. $\mathbf{1}$ is well-complete.

Henceforth we assume that Axiom 1 holds. As a first consequence, we obtain a helpful reformulation of the notion of well-completeness [5].

Proposition 3. *The following are equivalent.*

1. *X is well-complete.*
2. *$\mathcal{E} \models \forall F' \subseteq_\Sigma \mathbf{F}.\ \forall f \in X^{(\mathbf{I} \cap F')}.\ \exists! \overline{f} \in X^{F'}.\ f = \overline{f} \circ \iota$.*

Here, as is standard, we write $\exists!$ for the unique existence quantifier. We henceforth take the formula in statement 2 above as stating the property of X being well-complete in the internal logic.

Axiom 1 alone implies many important closure properties of well-complete objects. The proposition below is now standard, with parts appearing essentially in [15,5,25]. The easiest proof uses the formulation of well-completeness given by Proposition 3.2.

Proposition 4. *1. Any well-complete object is complete.*
2. For any internal family $\{X_j\}_{j\in J}$ *in* $\mathcal{E}$ *(given by a morphism* $X \longrightarrow I$*),*

$$\mathcal{E} \models (\forall j \in J.\ X_j \text{ well-complete}) \longrightarrow (\textstyle\prod_{j\in J} X_j) \text{ well-complete}$$

3. If X *and* Y *are well-complete then, for any* $f, g : X \longrightarrow Y$*, the equalizing object is well-complete.*
4. If X *is well-complete then so is* LX*.*

Externally, statement 2 above implies that $\mathcal{C}$ is closed under finite products in $\mathcal{E}$ (hence, by statement 3, under finite limits too), and is moreover an exponential ideal of $\mathcal{E}$ (i.e. If Y is well-complete then so is Y^X for any X in $\mathcal{E}$). In particular $\mathcal{C}$ is cartesian closed. Furthermore, if Y is well-complete then so is $X \multimap Y$ (because $X \multimap Y \cong (LY)^X$ and $\mathcal{C}$ is closed under lifting).

At present there is nothing to prevent taking $\Sigma = \mathbf{1}$ (for any topos $\mathcal{E}$ whatsoever), in which case $\mathbf{1}$ is the only (well-)complete object. The remaining axioms will rule out such trivial examples (except in the case of a trivial $\mathcal{E}$), and, more positively, imply that $\mathcal{C}$ is closed under useful coproducts in $\mathcal{E}$. First, we consider the basic implications between the properties we shall later assume as axioms.

Proposition 5. *Consider the statements below.*

1. $\mathbf{0}$ *is well-complete.*
2. $\mathbf{2}$ *is well-complete.*
3. $\mathbf{N}$ *is well-complete.*

Then $3 \Rightarrow 2 \Rightarrow 1$*.*

Proof. $\mathbf{0}$ is an equalizer of $\bot, \top : \mathbf{1} \longrightarrow \mathbf{2}$, and $\mathbf{2}$ is a retract of $\mathbf{N}$. The implications follow by Proposition 4(3). □

Examples in [20] show that neither of the implications of Proposition 5 can be reversed in general.

In the remainder of this section we examine further consequences of and relationships between each of the 3 properties in Proposition 5.

Proposition 6. *The following are equivalent.*

1. $\mathbf{0}$ *is well-complete.*
2. $\mathcal{E} \models \bot \in \Sigma$*.*

Proof.

1 ⟹ 2. Suppose $\mathbf{0}$ is well-complete. Then $L\mathbf{0}$ is complete and carries a monad algebra $\mu : LL\mathbf{0} \longrightarrow L\mathbf{0}$. Therefore the identity $\mathsf{id} : L\mathbf{0} \longrightarrow L\mathbf{0}$ has a fixed-point $\mathit{fix}(\mathsf{id}) \in L\mathbf{0}$. Because $\mathcal{E} \models \forall x \in \mathbf{0}.\ \bot$, we have that $\mathcal{E} \models \neg(\mathit{fix}(\mathsf{id})\downarrow)$, i.e. $\mathcal{E} \models (\mathit{fix}(\mathsf{id})\downarrow) = \bot$. But $\mathcal{E} \models (\mathit{fix}(\mathsf{id})\downarrow) \in \Sigma$ (because $\mathit{fix}(\mathsf{id}) \in L\mathbf{0}$). Thus indeed $\mathcal{E} \models \bot \in \Sigma$.

2 $\Longrightarrow$ **1.** If $\bot \in \Sigma$ then $\mathbf{1} \cong L\mathbf{0}$. Thus $L\mathbf{0}$ is complete, hence $\mathbf{0}$ is well-complete. □

Axiom 0. $\mathbf{0}$ is well-complete.

Henceforth in this section we assume Axiom 0.

A first consequence of Axiom 0 is that, for any object X of $\mathcal{E}$, we have a point $\bot_{LX} \in LX$ defined as the everywhere undefined partial function in $\mathbf{1} \rightharpoonup X$. Given $f : X \longrightarrow Y$, it is clear that $Lf : LX \longrightarrow LY$ maps $\bot_{LX}$ to $\bot_{LY}$. More generally, given any algebra for the lift functor, $\alpha : LX \longrightarrow X$, define $\bot_\alpha \in X$ by $\bot_\alpha = \alpha(\bot_{LX})$. For any $\beta : LY \longrightarrow Y$ and any algebra homomorphism $h : \alpha \longrightarrow \beta$, it is clear that $h(\bot_\alpha) = \bot_\beta$. In the case that there is an identified (usually monad) algebra α on an object X, we often write $\bot_X$ for $\bot_\alpha$, and refer to $\bot_X$ as the *bottom* of X. Thus algebra homomorphisms preserve bottoms.

Another consequence of Axiom 0 is that a morphism $step : \mathbf{N} \longrightarrow \mathbf{I}$ can be defined, using the initial algebra property of $\mathbf{N}$, as the unique map making the diagram below commute.

$$\begin{array}{ccc} \mathbf{1}+\mathbf{N} & \xrightarrow{1+step} & \mathbf{1}+\mathbf{I} \\ {\scriptstyle [0,s]}\downarrow & & \downarrow{\scriptstyle \sigma \circ [\bot_{L\mathbf{I}}, \eta]} \\ \mathbf{N} & \xrightarrow{step} & \mathbf{I} \end{array}$$

The following technical lemma (whose straightforward proof is omitted, see e.g. [20]) will be used in the proof of Theorem 1 below. Recall, e.g. from [17], that a mono $m : X' \longrightarrow X$ is said to be $\neg\neg$-*dense* if

$$\mathcal{E} \models \forall x \in X.\ \neg\neg(\exists x' \in X'.\ m(x') = x).$$

Lemma 1. *The morphism* $step : \mathbf{N} \longrightarrow \mathbf{I}$ *is a* $\neg\neg$*-dense mono.*

Next we examine condition 2 of Proposition 5.

Proposition 7. *The following are equivalent:*

1. $\mathbf{2}$ *is well-complete.*
2. $\mathcal{C}$ *is closed under finite coproducts in* $\mathcal{E}$.

Proof. The proof from [15] transfers to this more general setting. □

Axiom 2. $\mathbf{2}$ is well-complete.

Henceforth we asume Axiom 2. By Proposition 5 we can now drop Axiom 0. In fact, it is straightforward to show that Axiom 2 alone implies Axiom 1, so we can also drop Axiom 1. Although we do not yet know that $\mathbf{N}$ is well-complete, Axiom 2 does allow a number of conditions to be given that are equivalent to, or at least imply, the well-completeness of $\mathbf{N}$.

To state the theorem, we require further terminology. An object X of $\mathcal{E}$ is said to be $\neg\neg$-*separated* if

$$\mathcal{E} \models \forall x, y \in X.\ \neg\neg(x = y) \ \longrightarrow\ x = y.$$

This is a standard concept in topos theory with many equivalent formulations (see e.g. [17]). The following easy result depends only on Σ being a subobject of Ω containing $\top$.

Lemma 2. *The following are equivalent.*

1. *Σ is $\neg\neg$-separated.*
2. *$\mathcal{E} \models \forall p \in \Sigma.\ (\neg\neg p) \longrightarrow p$.*

The property of Σ being $\neg\neg$-separated has been referred to as Markov's Principle in the synthetic domain theory literature, see e.g. [25]. For certain dominances in certain toposes (e.g. the semidecidable subobject classifier in the effective topos [26,21]) this terminology is justified because the $\neg\neg$-separation of Σ is equivalent to the standard logical property known as Markov's Principle (see statement 4 of Theorem 1 below). However, in general, it seems sensible to maintain a distinction between the two non-equivalent properties. The theorem below, which depends heavily on Axiom 2, shows that both notions have a role to play in synthetic domain theory.

Theorem 1. *Consider the following statements.*

1. $\mathbf{N}$ *is well-complete.*
2. *pred* $: \mathbf{N} \longrightarrow \mathbf{1} + \mathbf{N}$ *is the final coalgebra for the functor* $\mathbf{1} + (-)$ *on the category* $\mathcal{E}_L$ *of* Σ*-partial maps.*
3. $\mathcal{E} \models \forall P \in \mathbf{2}^{\mathbf{N}}.\ (\exists n \in \mathbf{N}.\, P(n)) \in \Sigma$.
4. *Markov's Principle holds, i.e.*
   $\mathcal{E} \models \forall P \in \mathbf{2}^{\mathbf{N}}.\ \neg\neg(\exists n \in \mathbf{N}.\, P(n)) \longrightarrow \exists n \in \mathbf{N}.\, P(n)$.

Then $1 \Leftrightarrow 2 \Leftrightarrow 3 \Leftarrow 4$. *Moreover, if* Σ *is* $\neg\neg$*-separated then* $3 \Rightarrow 4$.

Proof.

$\mathbf{1} \Longrightarrow \mathbf{2}$. Suppose that $\mathbf{N}$ is well-complete, and that $h : X \rightharpoonup 1 + X$ is any Σ-partial map. We must show that there is a unique Σ-partial map $g : X \rightharpoonup \mathbf{N}$ such that the diagram below commutes.

$$\begin{array}{ccc} \mathbf{1} + X & \xrightarrow{1+g} & \mathbf{1} + \mathbf{N} \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle [0,s]} \\ X & \xrightarrow{\quad g \quad} & \mathbf{N} \end{array}$$

Define $\phi : (X \rightharpoonup \mathbf{N}) \longrightarrow (X \rightharpoonup \mathbf{N})$ by:

$$\phi(f) \ = \ [0, s] \circ (1 + f) \circ h$$

(the composition is, of course, composition of partial maps). By the well-completeness of $\mathbf{N}$, we have that $X \rightharpoonup \mathbf{N}$ is complete and carries a monad algebra for L. Therefore, by Proposition 2, we can define $g = \mathit{fix}(\phi)$, which, by its very definition, makes the diagram commute.
For uniqueness, suppose that g, g' are two partial maps making the diagram commute. Then, by an internal induction on n,

$$\mathcal{E} \models \forall n \in \mathbf{N}.\, \forall x \in X.\, g(x) = n \text{ iff } g'(x) = n.$$

(Here we are using our conventions about possibly undefined expressions, as $g(x)$ and $g'(x)$ need not be defined.) Thus $g = g'$.

2 $\Longrightarrow$ 3. Suppose that $\mathit{pred} : \mathbf{N} \longrightarrow \mathbf{1} + \mathbf{N}$ is the final coalgebra. Consider the map $d : \mathbf{2}^{\mathbf{N}} \longrightarrow \mathbf{1} + \mathbf{2}^{\mathbf{N}}$ defined, using the internal logic, by:

$$d(P) = \begin{cases} \mathit{inl}(*) & \text{if } P(0) \\ \mathit{inr}(\lambda n.P(n+1)) & \text{if not } P(0) \end{cases}$$

This is a good definition, because $P(0)$ is a logically decidable proposition. Let $h : \mathbf{2}^{\mathbf{N}} \longrightharpoonup \mathbf{N}$ be the unique coalgebra homomorphism from d to pred. Using only the fact that h is a coalgebra homomorphism, one has, by an internal induction on n, that

$$\mathcal{E} \models \forall n \in \mathbf{N}.\, \forall P \in \mathbf{2}^{\mathbf{N}}.\, d(P) = n \text{ iff } (P(n) \wedge \forall m < n.\, \neg P(m)). \quad (1)$$

Note that $d(P)$ need not be defined. Indeed, we claim that

$$\mathcal{E} \models \forall P \in \mathbf{2}^{\mathbf{N}}.\, (\exists n \in \mathbf{N}.\, P(n)) \text{ iff } d(P)\!\downarrow .$$

Statement 3 follows, because $(d(P)\!\downarrow) \in \Sigma$ (as d is a Σ-partial map).
The claim is derived from (1) by the following internal reasoning. For the right-to-left implication, if $d(P)\!\downarrow$ then $d(P) = n$ for some n. But, by (1), $d(P) = n$ implies $P(n)$, thus indeed $\exists n \in \mathbf{N}.\, P(n)$. For the converse, suppose $\exists n \in \mathbf{N}.\, P(n)$. Then, because, for all n, the proposition $P(n)$ is logically decidable, there exists a least n such that $P(n)$. It follows from (1) that $d(P)$ equals this least n. Hence $d(P)\!\downarrow$ as required.

3 $\Longrightarrow$ 1. Consider the subobject $D \longrightarrow \mathbf{2}^{\mathbf{N}}$ defined by

$$D = \{P \in \mathbf{2}^{\mathbf{N}} \mid \forall n \in \mathbf{N}.\, P(n) \longrightarrow \forall m < n.\, \neg P(m)\}.$$

As the proposition $\forall m < n.\, \neg P(m)$ is logically decidable, it is easy to show that the mono $D \longrightarrow \mathbf{2}^{\mathbf{N}}$ is split. Hence D is a retract of $\mathbf{2}^{\mathbf{N}}$, and thus well-complete (because $\mathbf{2}^{\mathbf{N}}$ is well-complete by Axiom 2).
Consider the subobject $C \longrightarrow D$ defined by

$$C = \{P \in D \mid \exists n \in \mathbf{N}.\, P(n)\}.$$

Assume that statement 3 of the theorem holds. Then the inclusion $C \longrightarrow D$ is an equalizer of the maps $\lambda P \in D.\, (\exists n \in \mathbf{N}.\, P(n)) : D \to \Sigma$ (which is a

map to Σ by the assumption) and $\lambda P \in D.\, \top$. Therefore C is well-complete. We show that $\mathbf{N}$ is isomorphic to C, hence $\mathbf{N}$ is indeed well-complete. The isomorphism from $\mathbf{N}$ to C maps n to $\lambda m \in \mathbf{N}.\, (n = m)$. Its inverse maps any $P \in C$ to the unique n such that $P(n)$. It is easily checked that the two maps are indeed mutually inverse.

4 $\Longrightarrow$ 3. First define a function $\phi : \Sigma^{\mathbf{2}^{\mathbf{N}}} \longrightarrow \Sigma^{\mathbf{2}^{\mathbf{N}}}$ by:

$$\phi(f)(P) \;=\; \begin{cases} \top & \text{if } P(0) \\ f(\lambda n.\, P(n+1)) & \text{if not } P(0). \end{cases}$$

Now $\Sigma^{\mathbf{2}^{\mathbf{N}}}$ is complete and carries a monad algebra for L. So, by Proposition 2, we can define a morphism $e : \mathbf{2}^{\mathbf{N}} \longrightarrow \Sigma$ by $e = \mathit{fix}(\phi)$. We claim that Markov's Principle implies:

$$\mathcal{E} \;\models\; \forall P \in \mathbf{2}^{\mathbf{N}}.\, (\exists n \in \mathbf{N}.\, P(n)) \;\mathit{iff}\; e(P)$$

Statement 3 follows because $e(P) \in \Sigma$.

For the left-to-right implication of the claim, one proves (internally) that, for all $n \in \mathbf{N}$, $P(n)$ implies $e(P)$. This is a straightforward induction on n, using only the fixed-point property of e. Markov's Principle is not required.

To prove the right-to-left implication of the claim, let $h : \mathbf{I} \longrightarrow \Sigma^{\mathbf{2}^{\mathbf{N}}}$ be the unique algebra homomorphism (given by Proposition 1) such that $\phi \circ h = h \circ \mathit{up}$. Let $\overline{h} : \mathbf{F} \longrightarrow \Sigma^{\mathbf{2}^{\mathbf{N}}}$ be the unique extension of h. Then, by definition, $e = \overline{h}(\infty)$. Assume $\neg\exists n \in \mathbf{N}.P(n)$. Below we show that, for all $i \in \mathbf{I}$, $h(i)(P) = \bot$. It follows that the unique extension $\lambda j \in \mathbf{F}.\overline{h}(j)(P) : \mathbf{F} \longrightarrow \Sigma$ of $\lambda i \in \mathbf{I}.\, h(i)(P) : \mathbf{I} \longrightarrow \Sigma$ is the constant function $\lambda j \in \mathbf{F}.\, \bot : \mathbf{F} \longrightarrow \Sigma$. Therefore $e(P) = \overline{h}(\infty)(P) = \bot$, i.e. $\neg e(P)$. Thus $\neg\exists n \in \mathbf{N}.\, P(n)$ implies $\neg e(P)$, or equivalently $e(P)$ implies $\neg\neg\exists n \in \mathbf{N}.\, P(n)$. Thus, by Markov's Principle, $e(P)$ implies $\exists n \in \mathbf{N}.\, P(n)$ as required.

It remains to show that $\neg\exists n \in \mathbf{N}.\, P(n)$ implies, for all $i \in \mathbf{I}$, $h(i)(P) = \bot$. Assume $\neg\exists n \in \mathbf{N}.\, P(n)$. The monad algebras on $\mathbf{I}$ and $\Sigma^{\mathbf{2}^{\mathbf{N}}}$ determine their bottoms: $\bot_{\mathbf{I}} = \mathit{step}(0)$ and $\bot_{\Sigma^{\mathbf{2}^{\mathbf{N}}}} = \lambda P.\bot$. As algebra homomorphisms preserve bottoms, $h(\mathit{step}(0))(P) = \bot$. Now, by an easy induction on n, for all $n \in \mathbf{N}$, it holds that $h(\mathit{step}(n))(P) = \bot$ (because we have assumed $\forall n.\neg P(n)$). However, by Lemma 1, we have that, for all $i \in \mathbf{I}$, $\neg\neg(\exists n.\, i = \mathit{step}(n))$. Therefore, for all $i \in \mathbf{I}$, $\neg\neg(h(i)(P) = \bot)$, i.e. $\neg\neg\neg h(i)(P)$, i.e. $\neg h(i)(P)$, i.e. $h(i)(P) = \bot$.

3 $\Longrightarrow$ 4. Immediate from Lemma 2 when Σ is $\neg\neg$-separated.

□

Axiom N. $\mathbf{N}$ is well-complete.

By Proposition 5, Axiom N implies Axiom 2, so in its presence Axiom 2 may be dropped. However, in practice it might be more convenient to establish Axiom 2 directly, and then use one of the conditions of Theorem 1 to derive Axiom N. Markov's Principle is a particularly useful condition because it is independent of the definition of Σ. Indeed the proof in [15] that Axiom 2 implies Axiom N in

realizability models makes implicit use of the validity of Markov's Principle in all realizability toposes. Markov's Principle is also valid in all presheaf toposes, but not in all Grothendieck toposes. The other statements in Theorem 1 also have their uses. For example, in Section 8, we make crucial use of statement 3 as a consequence of Axiom N.

We call an elementary topos $\mathcal{E}$ together with a dominance Σ satisfying Axiom N a *natural model (of synthetic domain theory)*. (The adjective "natural" is to emphasise that the natural numbers object is well-complete.) All the realizability examples considered in [15] provide natural models, as does the the model $\mathcal{H}$ from [6,5]. Throughout the rest of this paper, unless otherwise stated, we assume that $\mathcal{E}$ and Σ together form a natural model.

5 Interpreting a Programming Language

The axioms we have are sufficient for simply-typed programming languages like PCF [22] and its variants to be modelled in $\mathcal{E}$. We exploit all the structure we have identified on well-complete objects by including sums and product types in the language. The call-by-value language we introduce is essentially equivalent to similar languages considered in e.g. [8,32].

We shall use σ, τ, ... to range over *types*, which are given by the grammar:

$$\sigma \ ::= \ \mathbf{1} \mid \mathbf{N} \mid \sigma + \tau \mid \sigma \times \tau \mid \sigma \to \tau.$$

Assuming a countably infinite collection of variable symbols, *contexts* are finite sequences of variable-type assignments, written $x_1 : \sigma_1, \ldots, x_k : \sigma_k$. (We do not assume that all the x_i are distinct.) We use Γ, ... to range over contexts, and we write $x \in \Gamma$ to say that the variable x appears in Γ.

$$\frac{x \notin \Gamma'}{\Gamma, x{:}\sigma, \Gamma' \vdash x : \sigma} \qquad \frac{}{\Gamma \vdash * : \mathbf{1}} \qquad \frac{}{\Gamma \vdash 0 : \mathbf{N}} \qquad \frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash s(M) : \mathbf{N}} \qquad \frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \mathit{pred}(M) : \mathbf{1} + \mathbf{N}}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathit{inl}_\tau(M) : \sigma + \tau} \qquad \frac{\Gamma \vdash M : \sigma_1 + \sigma_2 \quad \Gamma, x_1 : \sigma_1 \vdash N_1 : \tau \quad \Gamma, x_2 : \sigma_2 \vdash N_2 : \tau}{\Gamma \vdash \mathit{case}\ M\ \mathit{of}\ \mathit{inl}(x_1).\, N_1,\ \mathit{inr}(x_2).\, N_2 : \tau}$$

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \mathit{inr}_\sigma(M) : \sigma + \tau} \qquad \frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash (M, N) : \sigma \times \tau} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathit{fst}(M) : \sigma} \qquad \frac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathit{snd}(M) : \tau}$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\sigma.\, M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \tau}{\Gamma \vdash M(N) : \tau} \qquad \frac{\Gamma, f : \sigma \to \tau, x : \sigma \vdash M : \tau}{\Gamma \vdash \mathit{rec}\, f : \sigma \to \tau\, (x).\, M : \sigma \to \tau}$$

Fig. 1. Typing rules

The notation for terms is introduced simultaneously with their typing rules, which are presented in Fig. 1. These rules manipulate judgements $\Gamma \vdash M : \sigma$, which say that the term M has type σ in context Γ. The notions of bound and

free variable occurrences are defined in the standard way. We *do not* identify terms up to alpha equivalence (as we have no reason to do so). (This simplifies the formalization of the programming language in $\mathcal{E}$.)

Observe that M has at most one type in any Γ (as just sufficient type information is included in terms for this to hold). Moreover, any typing judgement $\Gamma \vdash M : \sigma$ has at most one derivation. If $M : \sigma$ is derivable in the empty context then we say that M is a *program* of type σ.

$$\frac{}{* \Longrightarrow *} \qquad \frac{}{0 \Longrightarrow 0} \qquad \frac{M \Longrightarrow V}{s(M) \Longrightarrow s(V)} \qquad \frac{M \Longrightarrow 0}{pred(M) \Longrightarrow inl(*)} \qquad \frac{M \Longrightarrow s(V)}{pred(M) \Longrightarrow inr(V)}$$

$$\frac{M \Longrightarrow V}{inl(M) \Longrightarrow inl(V)} \qquad \frac{M \Longrightarrow inl(U) \quad N_1[U/x_1] \Longrightarrow V}{case\ M\ of\ inl(x_1).\, N_1,\ inr(x_2).\, N_2 \Longrightarrow V}$$

$$\frac{M \Longrightarrow V}{inr(M) \Longrightarrow inr(V)} \qquad \frac{M \Longrightarrow inr(U) \quad N_2[U/x_2] \Longrightarrow V}{case\ M\ of\ inl(x_1).\, N_1,\ inr(x_2).\, N_2 \Longrightarrow V}$$

$$\frac{M \Longrightarrow U \quad N \Longrightarrow V}{(M, N) \Longrightarrow (U, V)} \qquad \frac{(M, N) \Longrightarrow (U, V)}{fst(M) \Longrightarrow U} \qquad \frac{(M, N) \Longrightarrow (U, V)}{snd(M) \Longrightarrow V}$$

$$\frac{}{\lambda x.\, M \Longrightarrow \lambda x.\, M} \qquad \frac{M \Longrightarrow \lambda x.\, L \quad N \Longrightarrow U \quad L[U/x] \Longrightarrow V}{M(N) \Longrightarrow V}$$

$$\frac{}{rec\, f\,(x).\, M \Longrightarrow \lambda x.\, M[\, rec\, f\,(x).\, M\, /\, f\,]}$$

Fig. 2. Operational semantics

The behaviour of programs is specified by the operational semantics presented in Fig. 2. The rules manipulate judgements of the form $M \Longrightarrow V$ where both M and V are programs (to ease readability, we omit type information from the terms). In the rules, we write $N[U/x]$ to mean the term N with program U substituted for all free occurrences of x in N. Because the only entities ever substituted are closed terms, the possibility of variable capture does not arise, therefore a simple textual substitution suffices.

The programs V which appear on the right-hand side of the arrow in Fig. 2 are known as *values*. It is straightforward to show that, for any program M, there exists at most one value V such that $M \Longrightarrow V$. Moreover, for any M, V, there exists at most one derivation of the judgement $M \Longrightarrow V$. These results state that program evaluation is deterministic. It is also consistent with the typing rules: if M is a program of type σ and $M \Longrightarrow V$ then V is also has type σ. (Because we use a a simple textual substitution and do not identify terms up to alpha-equivalence, this last result depends on the possibility, which we do permit, of contexts containing repeated variables.)

We now turn to the denotational semantics. Types σ are interpreted as well-complete objects $[\![\sigma]\!]$, using the full strength of Axiom N to obtain the desired

closure conditions. The definition of $[\![\sigma]\!]$ is by the expected induction on types. From the viewpoint of the internal logic of $\mathcal{E}$, the interpretation gives the full set-theoretic Σ-partial type hierarchy over $\mathbf{N}$ (i.e. $[\![\mathbf{1}]\!] = \mathbf{1}$; $[\![\mathbf{N}]\!] = \mathbf{N}$; $[\![\sigma + \tau]\!] = [\![\sigma]\!] + [\![\tau]\!]$; $[\![\sigma \times \tau]\!] = [\![\sigma]\!] \times [\![\tau]\!]$; $[\![\sigma \to \tau]\!] = [\![\sigma]\!] \rightharpoonup [\![\tau]\!]$).

A context $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ is interpreted as a product $[\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!]$. A term M such that $\Gamma \vdash M : \sigma$ is interpreted as a Σ-partial map $[\![M]\!] : [\![\Gamma]\!] \relbar\!\!\rightharpoonup [\![\sigma]\!]$. This map can be defined by induction on the structure of M, using the external structure of $\mathcal{C}$. However, for later purposes, it is convenient to make an equivalent definition, using the internal logic of $\mathcal{E}$. For any term M such that $\Gamma \vdash M : \sigma$ we define, by induction on the structure of M, a definite description (in the internal logic) of a partial function $(\!|M|\!) \in ([\![\Gamma]\!] \rightharpoonup [\![\sigma]\!])$. Then $(\!|M|\!)$ determines a morphism $\mathbf{1} \longrightarrow ([\![\Gamma]\!] \rightharpoonup [\![\sigma]\!])$ in $\mathcal{E}$, which in turn determines the required Σ-partial map $[\![M]\!] : [\![\Gamma]\!] \relbar\!\!\rightharpoonup [\![\sigma]\!]$.

Most of the definition of $(\!|M|\!)$ is straightforward (indeed set-theoretic). The most interesting clause is the definition of $(\!|\mathit{rec}\, f : \sigma \to \tau\, (x).\, M|\!)$. Suppose that $\Gamma, f : \sigma \to \tau, x : \sigma \vdash M : \tau$. Then we have $(\!|M|\!) \in ([\![\Gamma]\!] \times [\![\sigma \to \tau]\!] \times [\![\sigma]\!]) \rightharpoonup [\![\tau]\!]$. So:

$$\mathcal{E} \models \forall \gamma \in [\![\Gamma]\!].\, (\lambda f \in [\![\sigma]\!] \rightharpoonup [\![\tau]\!].\, (\lambda x \in [\![\sigma]\!].\, (\!|M|\!)(\gamma, f, x))) \in ([\![\sigma]\!] \rightharpoonup [\![\tau]\!])^{([\![\sigma]\!] \rightharpoonup [\![\tau]\!])}.$$

Write $\phi_{M\gamma}$ for $(\lambda f \in [\![\sigma]\!] \rightharpoonup [\![\tau]\!].\, (\lambda x \in [\![\sigma]\!].\, (\!|M|\!)(\gamma, f, x)))$. As $[\![\sigma]\!] \rightharpoonup [\![\tau]\!]$ is well-complete, hence complete, with a monad algebra structure (see Section 2), we use Proposition 2 to define $(\!|\mathit{rec}\, f : \sigma \to \tau\, (x).\, M|\!) \in [\![\Gamma]\!] \rightharpoonup [\![\sigma \to \tau]\!]$ by:

$$(\!|\mathit{rec}\, f : \sigma \to \tau\, (x).\, M|\!)(\gamma) \;=\; \mathit{fix}(\phi_{M\gamma}).$$

Observe that $(\!|\mathit{rec}\, f : \sigma \to \tau\, (x).\, M|\!)$ is a total function from $[\![\Gamma]\!]$ to $[\![\sigma]\!] \rightharpoonup [\![\tau]\!]$.

6 Computational Adequacy

We now come to the main question addressed in this paper, the question of *computational adequacy*, relating the operational behaviour of the programming language and its denotational interpretation.

We say that a program $M : \sigma$ *converges* (notation $M \Downarrow$) if there exists V such that $M \Longrightarrow V$. For the denotational analogue, we write $[\![M]\!]\downarrow$ if the Σ-partial map $[\![M]\!] : \mathbf{1} \relbar\!\!\rightharpoonup [\![\sigma]\!]$ is total. Denotational convergence corresponds to the expected property of $(\!|M|\!)$ in the internal logic of $\mathcal{E}$. We have $(\!|M|\!) \in \mathbf{1} \rightharpoonup [\![\sigma]\!]$, i.e. $(\!|M|\!) \in L[\![\sigma]\!]$, and, using our conventions for elements of lifted objects, $[\![M]\!]\downarrow$ if and only if $\mathcal{E} \models (\!|M|\!)\downarrow$. By an easy induction on operational derivations, one shows that for all programs $M : \sigma$, if $M \Longrightarrow V$ then $[\![M]\!]\downarrow$ and $[\![M]\!] = [\![V]\!]$.

Definition 3. We say that $[\![\cdot]\!]$ is *computationally adequate* if, for all programs $M{:}\ \sigma$, $[\![M]\!]\downarrow$ implies $M \Downarrow$.

As stated in the introduction, we shall obtain a complete characterisation of computational adequacy in terms of a logical property of $\mathcal{E}$. Recall that any semidecidable k-ary relation on $\mathbf{N}$ can be written in the standard form $\exists n_1 \in \mathbf{N}, \ldots, n_l \in \mathbf{N}.\, P(m_1, \ldots, m_k, n_1, \ldots, n_k)$, where P is a $(k + l)$-ary

primitive recursive relation. By a natural encoding of primitive recursive predicates in the internal logic of $\mathcal{E}$ [13], one obtains the analogous formal notion of a Σ^0_1-*formula*. In particular, a Σ^0_1-*sentence* is a sentence of the form $\exists n_1 \in \mathbf{N}, \ldots, n_l \in \mathbf{N}.\, \psi(n_1, \ldots, n_k)$, where ψ is a naturally encoded k-ary primitive recursive predicate. (In fact, because one can define a primitive recursive pairing operation, it is sufficient to consider just unary predicates ψ.) We say that $\mathcal{E}$ is *1-consistent* if, for all Σ^0_1-sentences ϕ, it holds that $\mathcal{E} \models \phi$ implies that ϕ is true in reality. The next theorem is the main result of the paper.

Theorem 2. *The following are equivalent.*

1. $[\![\cdot]\!]$ *is computationally adequate.*
2. $\mathcal{E}$ *is 1-consistent.*

To prove the theorem, we carry out, as far as possible, a proof of computational adequacy within the internal logic of $\mathcal{E}$. In fact $\mathcal{E}$ always believes itself to be computationally adequate. The property of 1-consistency is just what is needed to relate this internal belief with external reality.

First, we need to formalize the syntax of the programming language, and its operational semantics, within the internal logic. This is an exercise in Gödel numbering. One encodes the types, terms and contexts as natural numbers in a standard way, so that all convenient operations on them are primitive recursive. Moreover, the relation $\Gamma \vdash M : \sigma$ is also primitive recursive, as the term M determines the whole derivation tree, allowing a primitive recursive decision procedure. We write $\mathcal{P}_\sigma$ for the formalized set of programs of type σ, which is, via its Gödel numbering, a primitive recursive subset of $\mathbf{N}$.

For the operational semantics, the Gödel numbering is extended to encode *derivations* of evaluation judgements $M \Longrightarrow V$. The relation "π is a derivation of $M \Longrightarrow V$" is a a primitive recursive ternary relation on π, M and V. Thus the binary relation $M \Longrightarrow V$ and the unary predicate $M\Downarrow$ are both Σ^0_1.

The proposition below, is stated using the formalized operational semantics.

Proposition 8. *For all programs* $M{:}\sigma$*, if* $[\![M]\!]\downarrow$ *then* $\mathcal{E} \models M\Downarrow$.

The lengthy proof is left for Sections 7 and 8. Here we apply the result to prove Theorem 2.

The $2 \Longrightarrow 1$ implication of Theorem 2 is now immediate. If $\mathcal{E}$ is 1-consistent then we have that $[\![M]\!]\downarrow$ implies $\mathcal{E} \models M\Downarrow$, by Proposition 8, which in turn implies $M\Downarrow$, by 1-consistency.

For the converse, suppose $[\![\cdot]\!]$ is computationally adequate. We must show 1-consistency. Accordingly, let P be a primitive recursive predicate. Without loss of generality, we can assume P is unary. Using a straightforward encoding of primitive recursive predicates, we can define a program $M : \mathbf{N} \to (\mathbf{1} + \mathbf{1})$ such that, using only the fixed-point property of *fix*,

$$\mathcal{E} \models \forall n \in \mathbf{N}.\, (\!|M|\!)(n) = \mathit{inl}(*) \;\; \mathit{iff} \;\; P(n),$$

and also $M(\overline{n}) \Longrightarrow \mathit{inl}(*)$ if and only if $P(n)$ (where $\overline{n}$ is the value $s^n(0)$). Let $N : \mathbf{1}$ be the following search program.

$$(\, \mathit{rec}\, f{:}\mathbf{N}\to\mathbf{1}\,(n).\, \mathit{case}\; M(n)\; \mathit{of}\; \mathit{inl}(x).*,\; \mathit{inr}(y).\, f(s(n))\,)(0)$$

Then, by an internal induction on n using only the fixed-point property of *fix*,

$$\mathcal{E} \models (\exists n \in \mathbf{N}.\, P(n)) \longrightarrow (\!|N|\!)\downarrow .$$

Also, by an induction on the number of unfoldings of *rec* in the operational semantics, $N\Downarrow$ implies there exists n such that $P(n)$.

Now, to show 1-consistency, suppose that $\mathcal{E} \models \exists n \in \mathbf{N}.\, P(n)$. Then, by the implication derived above, $\mathcal{E} \models (\!|N|\!)\downarrow$, i.e. $[\![N]\!]\downarrow$. It follows, by computational adequacy, that $N\Downarrow$. Thus indeed there exists n such that $P(n)$. This completes the derivation of Theorem 2 from Proposition 8.

We end this section with some applications of Theorem 2. It is easily verified that any non-trivial Grothendieck topos is 1-consistent (indeed, it is a folk theorem that any Grothendieck topos validates all classically true sentences of elementary arithmetic). Also, any non-trivial realizability topos, $\mathcal{E}$, is 1-consistent (because the hom-set $\mathcal{E}(\mathbf{1}, \mathbf{N})$ consists of just the numerals). Therefore, by Theorem 2, any non-trivial natural model furnished by either a Grothendieck or realizability topos is computationally adequate. (A different argument for computational adequacy in the case of realizability toposes appears in [14].)

One may wonder whether in fact any non-trivial model is computationally adequate. As a negative application of Theorem 2, we show that this is not the case. To be precise, we say that a model $(\mathcal{E}, \Sigma)$ is *trivial* if $\mathcal{E}$ is equivalent to a category with a single (necessarily identity) morphism. Non-triviality alone implies many important well-behavedness properties of $\mathcal{E}$, e.g. the two morphisms $\bot, \top : \mathbf{1} \to \mathbf{2}$ are distinct and the numerals $\overline{n} : \mathbf{1} \to \mathbf{N}$ are all distinct. Non-triviality is obviously also a necessary condition for computational adequacy to hold.

Corollary 1. *There exist non-trivial natural models $(\mathcal{E}, \Sigma)$ such that the induced $[\![\cdot]\!]_{(\mathcal{E},\Sigma)}$ is not computationally adequate.*

Proof. Suppose, on the contrary, that any non computationally adequate model is trivial. Let ψ be the (easily constructed) sentence in the internal logic of an elementary topos with natural numbers object and distinguished object Σ saying "Σ is a dominance and Axiom N holds". Thus $(\mathcal{E}, \Sigma) \models \psi$ if and only if $(\mathcal{E}, \Sigma)$ is a natural model of synthetic domain theory. Let ϕ be any false Σ^0_1-sentence. By Theorem 2, any model of $\psi \wedge \phi$ is not computationally adequate and hence, by the assumption, trivial. By the completeness theorem for elementary toposes with respect to their internal logic [13], this means that $\neg(\psi \wedge \phi)$ is a theorem of the internal logic. On the other hand, when ϕ is a true Σ^0_1-sentence, then $\psi \wedge \phi$ is equivalent to ψ, which is consistent as there exist non-trivial natural models. In summary, for Σ^0_1-sentences ϕ, we have that $\neg(\psi \wedge \phi)$ is a theorem in the internal logic of toposes if and only if ϕ is false. But the theorems of the internal logic are recursively enumerable. Therefore, we can recursively enumerate the false Σ^0_1-sentences (i.e. the true Π^0_1-sentences). This is impossible, contradicting the initial assumption.

□

Incidentally, it is not too difficult to give a similar direct proof of Corollary 1 (not relying on Theorem 1) using the halting problem for the programming language in place of the truth of Σ^0_1-sentences.

7 Internal Adequacy

In this section we provide the missing proof of Proposition 8. This is achieved by formalising a standard relational proof of computational adequacy, see e.g. [8,32], internally within $\mathcal{E}$.

For each type σ we define a predicate $\preceq^\sigma \rightarrowtail [\![\sigma]\!] \times \mathcal{P}_\sigma$ in the internal logic of $\mathcal{E}$. The definition proceeds inductively on the structure of σ, so that the relations satisfy the (internal) equivalences below.

$$
\begin{array}{lll}
* \preceq^{\mathbf{1}} M & \textit{iff} & M \Longrightarrow * \\
n \preceq^{\mathbf{N}} M & \textit{iff} & M \Longrightarrow \overline{n} \\
\mathit{inl}(d) \preceq^{\sigma+\tau} M & \textit{iff} & M \Longrightarrow \mathit{inl}(V) \ \textit{where}\ d \preceq^\sigma V \\
\mathit{inr}(d) \preceq^{\sigma+\tau} M & \textit{iff} & M \Longrightarrow \mathit{inr}(V) \ \textit{where}\ d \preceq^\tau V \\
(d, d') \preceq^{\sigma\times\tau} M & \textit{iff} & M \Longrightarrow (U, V) \ \textit{where}\ d \preceq^\sigma U \ \wedge\ d' \preceq^\tau V \\
f \preceq^{\sigma\to\tau} M & \textit{iff} & M \Longrightarrow \lambda x.\, L \ \textit{where}\ \forall d \in [\![\sigma]\!].\, \forall N \in \mathcal{P}_\sigma. \\
 & & \quad (d \preceq^\sigma N \wedge f(d)\!\downarrow) \longrightarrow f(d) \preceq^\tau L[N/x]
\end{array}
$$

These equivalences are easily translated into formal definitions, in the internal logic, of the $\preceq^\sigma$ predicates. It is a straightforward consequence of the definitions that (internally) if $M, M' \in \mathcal{P}_\sigma$ are such that both $M \Longrightarrow V$ and $M' \Longrightarrow V$ (the same V) then, for all $d \in [\![\sigma]\!]$, $d \preceq^\sigma M$ iff $d \preceq^\sigma M'$. This fact will be used in the proof of Lemma 5 below without further comment.

Define $[\![\sigma]\!]_M = \{x \in [\![\sigma]\!] \mid x \preceq^\sigma M\}$. This definition determines, for any type σ, an internal $\mathcal{P}_\sigma$-indexed family, $\{[\![\sigma]\!]_M\}_{M\in\mathcal{P}_\sigma}$, of subobjects of $[\![\sigma]\!]$.

Lemma 3. *For all types σ, τ,*

$$\mathcal{E} \models \forall M \in \mathcal{P}_{\sigma\to\tau}.\ M\!\Downarrow \longrightarrow ([\![\sigma \to \tau]\!]_M \textit{ carries a subalgebra of } [\![\sigma]\!] \rightharpoonup [\![\tau]\!]).$$

Proof. Recall $[\![\sigma]\!] \rightharpoonup [\![\tau]\!]$ has the algebra $\alpha : L([\![\sigma]\!] \rightharpoonup [\![\tau]\!]) \longrightarrow ([\![\sigma]\!] \rightharpoonup [\![\tau]\!])$ defined by $\alpha(e)(x) \simeq e(x)$. Reasoning internally in $\mathcal{E}$, suppose that $M \in \mathcal{P}_{\sigma\to\tau}$ and $M\!\Downarrow$. We show that $e \in L[\![\sigma \to \tau]\!]_M$ implies $\alpha(e) \in [\![\sigma \to \tau]\!]_M$. Suppose $e \in L[\![\sigma \to \tau]\!]_M$. As $M\!\Downarrow$, we have that $M \Longrightarrow \lambda x.\, L$ for some $\lambda x.\, L$. We must show that, for all $d \in [\![\sigma]\!]$ and all $N \in \mathcal{P}_\sigma$, if $d \preceq^\sigma N$ and $\alpha(e)(d)\downarrow$ then $\alpha(e)(d) \preceq^\tau L[N/x]$. But if $\alpha(e)(d)\downarrow$ then $\alpha(e)(d) = e(d)$ and also $e\downarrow$, hence $e \in [\![\sigma \to \tau]\!]_M$. It follows that $d \preceq^\sigma N$ implies $e(d) \preceq^\tau L[N/x]$, i.e. $\alpha(e)(d) \preceq^\tau L[N/x]$ as required. □

Lemma 4. *For all types σ,*

$$\mathcal{E} \models \forall M \in \mathcal{P}_\sigma.\ [\![\sigma]\!]_M \textit{ is well-complete.}$$

The proof of Lemma 4, which is surprisingly involved, is given in Section 8.

Lemma 5. *If* $x_1{:}\sigma_1, \ldots x_k{:}\sigma_k \vdash M{:}\tau$ *then*

$$\begin{aligned}\mathcal{E} \models \forall d_1 \in [\![\sigma_1]\!] \ldots \forall d_k \in [\![\sigma_k]\!].\, \forall N_1 \in \mathcal{P}_{\sigma_1} \ldots \forall N_k \in \mathcal{P}_{\sigma_k}.\\ d_1 \preceq^{\sigma_1} N_1 \wedge \ldots \wedge d_k \preceq^{\sigma_k} N_k \wedge (\!|M|\!)(d_1, \ldots, d_k)\downarrow\\ \longrightarrow (\!|M|\!)(d_1, \ldots, d_k) \preceq^{\tau} M[N_1, \ldots, N_k \,/\, x_1, \ldots, x_k]\end{aligned}$$

Proof. This is proved by (external) induction on the derivation of $\Gamma \vdash M{:}\tau$ (where here and below we write Γ for the context $x_1{:}\sigma_1, \ldots x_k{:}\sigma_k$). In each case, reasoning internally in $\mathcal{E}$, we suppose that, for each j with $1 \leq j \leq k$, we have $d_j \in [\![\sigma_j]\!]$ and $N_j \in \mathcal{P}_{\sigma_j}$ such that $d_k \preceq^{\sigma_k} N_k$. Then we show that $(\!|M|\!)(\overrightarrow{d})\downarrow$ implies $(\!|M|\!)(\overrightarrow{d}) \preceq^{\tau} M[\overrightarrow{N}/\overrightarrow{x}]$ (where, of course, $\overrightarrow{d}$ abbreviates $d_1, \ldots, d_k$, and $\overrightarrow{N}$ abbreviates $N_1, \ldots, N_k$). Here we consider only the critical case in which M is $\mathit{rec}\, f{:}\sigma' \to \tau'\,(y).\, M'$ (and hence τ is $\sigma' \to \tau'$).

In this case, $\Gamma \vdash M : \tau$ is derived from $\Gamma, f : \sigma' \to \tau', y : \sigma' \vdash M' : \tau'$. Given $\overrightarrow{d}$ and $\overrightarrow{N}$ as above, it holds automatically that $(\!|M|\!)(\overrightarrow{d})\downarrow$, so we must show that $(\!|M|\!)(\overrightarrow{d}) \preceq^{\sigma' \to \tau'} M[\overrightarrow{N}/\overrightarrow{x}]$, i.e. that $(\!|M|\!)(\overrightarrow{d}) \in [\![\sigma' \to \tau']\!]_{M[\overrightarrow{N}/\overrightarrow{x}]}$. Recall that $(\!|M|\!)(\overrightarrow{d}) = \mathit{fix}(\phi_{M'\overrightarrow{d}})$, where $\phi_{M'\overrightarrow{d}}$ is the endofunction on $[\![\sigma']\!] \rightharpoonup [\![\tau']\!]$. defined by $\phi_{M'\overrightarrow{d}}(f)(e) \simeq (\!|M'|\!)(\overrightarrow{d}, f, e)$. We claim that $\phi_{M'\overrightarrow{d}}$ restricts to an endofunction on $[\![\sigma' \to \tau']\!]_{M[\overrightarrow{N}/\overrightarrow{x}]}$. By Lemmas 3 and 4, $[\![\sigma' \to \tau']\!]_{M[\overrightarrow{N}/\overrightarrow{x}]}$ carries a monad algebra structure and is (well-)complete. By (the internalized) Proposition 2 (see, in particular, the discussion immediately after), the element $\mathit{fix}(\phi_{M'\overrightarrow{d}}) \in \sigma' \rightharpoonup \tau'$ is contained in the subset $[\![\sigma' \to \tau']\!]_{M[\overrightarrow{N}/\overrightarrow{x}]}$. Thus indeed $(\!|M|\!)(\overrightarrow{d}) \in [\![\sigma' \to \tau']\!]_{M[\overrightarrow{N}/\overrightarrow{x}]}$.

It remains to prove the claim. Suppose then that $f \preceq^{\sigma' \to \tau'} M[\overrightarrow{N}/\overrightarrow{x}]$. We must show that $\phi_{M'\overrightarrow{d}}(f) \preceq^{\sigma' \to \tau'} M[\overrightarrow{N}/\overrightarrow{x}]$. As M is $\mathit{rec}\, f : \sigma' \to \tau'\,(y).\, M'$, we have that $M[\overrightarrow{N}/\overrightarrow{x}] \Longrightarrow \lambda y : \sigma'.\, M'[\overrightarrow{N}, M[\overrightarrow{N}/\overrightarrow{x}] \,/\, \overrightarrow{x}, f]$. Take any $e \in [\![\sigma']\!]$ and $N' \in \mathcal{P}_{\sigma'}$ such that $e \preceq^{\sigma'} N'$ and $\phi_{M'\overrightarrow{d}}(f)(e)\downarrow$. We must show that $\phi_{M'\overrightarrow{d}}(f)(e) \preceq^{\tau'} M'[\overrightarrow{N}, M[\overrightarrow{N}/\overrightarrow{x}], N' \,/\, \overrightarrow{x}, f, y]$. But $\phi_{M'\overrightarrow{d}}(f)(e) = (\!|M'|\!)(\overrightarrow{d}, f, e)$ and indeed $\phi_{M'\overrightarrow{d}}(f)(e)\downarrow \preceq^{\tau'} M'[\overrightarrow{N}, M[\overrightarrow{N}/\overrightarrow{x}], N'/\overrightarrow{x}, f, y]$, by the induction hypothesis on M'. □

Proposition 8 is an easy consequence of Lemma 5. For any program $M{:}\sigma$, if $[\![M]\!]\downarrow$ then, equivalently, $\mathcal{E} \models (\!|M|\!)\downarrow$. So, by Lemma 5, $\mathcal{E} \models (\!|M|\!) \preceq^{\sigma} M$. Hence, by the definition of $\preceq^{\sigma}$, indeed $\mathcal{E} \models M\Downarrow$.

One remark about Lemma 5 is that the quantification over terms is external. This is by necessity, as there are limitations on the extent to which the denotational semantics can be formalized within $\mathcal{E}$. In particular, writing $\mathcal{T}$ for the formalized set of types (as a primitive recursive subset of $\mathbf{N}$), one cannot define within the internal logic a semantic function $[\![\cdot]\!] \in \prod_{\sigma \in \mathcal{T}} \mathcal{P}_{\sigma} \rightharpoonup [\![\sigma]\!]$, because the family $\{[\![\sigma]\!]\}_{\sigma \in \mathcal{T}}$ need not live as an internal family within $\mathcal{E}$ (its usual definition involves the Axiom of Replacement from set theory).

8 The Well-completeness of $[\![\sigma]\!]_M$

It remains only to prove Lemma 4, showing (internally) the well-completeness of the sets $[\![\sigma]\!]_M$. Although the proof of Lemma 5 required only the completeness of these sets, it is necessary to establish the well-completeness in order to have a property that can be proved by induction on types.

We begin with some useful closure properties of well-complete objects. To establish these, it is convenient to work with the formulation of well-completeness given by Proposition 3.2. (We write "X *inhabited*" for "$\exists x \in X.\ \top$".)

Lemma 6.

$$\mathcal{E} \models (\exists p \in \Sigma.\ (X \textit{ inhabited} \longrightarrow p) \wedge (p \longrightarrow X \textit{ well-complete})) \longrightarrow X \textit{ well-complete}$$

Proof. We reason in the internal logic. Let $p \in \Sigma$ satisfy the assumption. Suppose $F' \subseteq_\Sigma \mathbf{F}$. Take any $f \in X^{I'}$, where $I' = \mathbf{I} \cap F'$. We show that F' inhabited implies X well-complete (using the assumed f). First , observe that I' inhabited implies X inhabited (because $i \in I'$ implies $f(i) \in X$), so the two functions $\lambda i.\top \in \Sigma^{I'}$ and $\lambda i.p \in \Sigma^{I'}$ are equal (by the assumption). Therefore $\lambda j.\top \in \Sigma^{F'}$ and $\lambda j.p \in \Sigma^{F'}$ both extend $\lambda i.\top \in \Sigma^{I'}$. By the well-completeness of Σ, there is a unique such extension, thus, for all $j \in F'$, $p = \top$, i.e. F' inhabited implies p. But p implies X is well-complete. Thus indeed F' inhabited implies X is well-complete.

We must show there is a unique function $\overline{f} \in X^{F'}$ such that $f = \overline{f} \circ \iota$. For any $j \in F'$, we have that X is well-complete so there exists a unique $\overline{f_j} \in X^{F'}$ such that $f = \overline{f_j} \circ \iota$ (the notation is to emphasise that $\overline{f_j}$ depends on the assumed element j). Therefore $\overline{f} = \lambda j \in F'.\, \overline{f_j}(j)$ defines a function in $X^{F'}$. It is easily checked that this is the unique function such that $f = \overline{f} \circ \iota$. □

Lemma 7. *If $\{X_j\}_{j \in J}$ is an internal family of subobjects of a well-complete object Y then*

$$\mathcal{E} \models (\forall j \in J.\ X_j \textit{ well-complete}) \longrightarrow (\textstyle\bigcap_{j \in J} X_j) \textit{ well-complete.}$$

We now turn to the proof of Lemma 4. We prove, by an (external) induction on the structure of the type σ, the required property:

$$\mathcal{E} \models \forall M \in \mathcal{P}_\sigma.\ [\![\sigma]\!]_M \text{ is well-complete.}$$

Lemma 6 will be basic to the argument. To apply it, we crucially observe that, for any Σ_1^0-sentence ϕ, it holds that $\phi \in \Sigma$. This is because any primitive-recursive predicate determines a corresponding $P \in \mathbf{2}^{\mathbf{N}}$, so any Σ_1^0-sentence, ϕ, is of the form $\exists n \in \mathbf{N}.\, P(n)$ for some $P \in \mathbf{2}^{\mathbf{N}}$, and therefore $\phi \in \Sigma$, by Theorem 1.3. We consider just one case of the induction.

When σ is $\sigma' \to \tau'$, reasoning internally, take any $M \in \mathcal{P}_{\sigma' \to \tau'}$. If $[\![\sigma' \to \tau']\!]_M$ is inhabited then there exists (a unique) $(\lambda x.\, L) \in \mathcal{P}_{\sigma' \to \tau'}$ such that $M \Longrightarrow \lambda x.\, L$. We claim that $M \Longrightarrow \lambda x.\, L$ implies $[\![\sigma' \to \tau']\!]_M$ is well-complete. From which,

it follows, by Lemma 6, that $[\![\sigma' \to \tau']\!]_M$ is well-complete as required (as the proposition $M \Longrightarrow \lambda x.\, L$ is a Σ-property).

To prove the claim, observe that if $M \Longrightarrow \lambda x.\, L$ then

$$[\![\sigma' \to \tau']\!]_M \;=\; \bigcap_{N \in \mathcal{P}_{\sigma'}} \bigcap_{d \in [\![\sigma']\!]_N} X_{Nd}$$

where

$$X_{Nd} \;=\; \{f \in [\![\sigma' \to \tau']\!] \mid f(d)\!\downarrow \longrightarrow\; f(d) \preceq^{\tau'} L[N/x]\}.$$

As $[\![\sigma' \to \tau']\!]$ is well-complete, it suffices, by Lemma 7, to show the well-completeness of each set X_{Nd}. However, writing i for the inclusion function from $[\![\tau']\!]_{L[N/x]}$ to $[\![\tau']\!]$, and g for the function $\lambda f.\, f(d) : [\![\sigma' \to \tau']\!] \longrightarrow L[\![\tau']\!]$, we have an evident pullback diagram:

$$\begin{array}{ccc} X_{Nd} & \longrightarrow & L[\![\tau']\!]_{L[N/x]} \\ \downarrow & \lrcorner & \downarrow Li \\ [\![\sigma' \to \tau']\!] & \xrightarrow{\;p\;} & L[\![\tau']\!] \end{array}$$

As the three vertices of the diagram being pulled back are well-complete (in particular, $L[\![\tau']\!]_{L[N/x]}$ is well-complete by the induction hypothesis on τ'), so is the pullback X_{Nd} as required. This completes the proof of Lemma 4.

Acknowledgements

This paper was written during a visit to Utrecht University supported by the NWO Pionier Project *The Geometry of Logic* led by Ieke Moerdijk. I thank Jaap van Oosten for many helpful and interesting discussions. Paul Taylor's diagram macros were used.

References

1. A. Bucalo and G. Rosolini. Lifting. In *Category Theory and Computer Science, Proceedings of CTCS '97*. Springer LNCS 1290, 1997.
2. R.L. Crole and A.M. Pitts. New foundations for fixpoint computations: FIX-hyperdoctrines and the FIX-logic. *Inf. and Comp.*, 98:171–210, 1992.
3. M.P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps*. Cambridge University Press, 1996.
4. M.P. Fiore and G.D. Plotkin. An extension of models of axiomatic domain theory to models of synthetic domain theory. In *Proceedings of CSL 96*, pages 129–149. Springer LNCS 1258, 1997.
5. M.P. Fiore and G. Rosolini. The category of cpos from a synthetic viepoint. Presented at *MFPS XIII*, 1997.

6. M.P. Fiore and G. Rosolini. Two models of synthetic domain theory. *Journal of Pure and Applied Algebra*, 116:151–162, 1997.
7. P.J. Freyd, P. Mulry, G. Rosolini, and D.S. Scott. Extensional PERs. In *Proc. of 5th Annual Symposium on Logic in Computer Science*, 1990.
8. C.A. Gunter. *Semantics of Programming Languages*. MIT Press, 1992.
9. H. Huwig and A. Poigné. A note on inconstencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, 73:101–112, 1990.
10. J.M.E. Hyland. First steps in synthetic domain theory. In *Category Theory, Proceedings, Como 1990*. Springer LNM 1488, 1990.
11. M. Jibladze. A presentation of the initial lift algebra. *Journal of Pure and Applied Algebra*, 116:185–198, 1997.
12. A. Joyal and I. Moerdijk. *Algebraic Set Theory*. LMS Lecture Notes, CUP, 1995.
13. J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge studies in Advanced Mathematics. Cambridge University Press, 1986.
14. J.R. Longley. *Realizability Toposes and Language Semantics*. Ph.D. thesis, Department of Computer Science, University of Edinburgh, 1995.
15. J.R. Longley and A.K. Simpson. A uniform account of domain theory in realizability models. *Math. Struct. in Comp. Sci.*, 7:469–505, 1997.
16. S. Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer Verlag, 1971.
17. S. Mac Lane and I. Moerdijk. *Sheaves in Geometry and Logic: a First Introduction to Topos Theory*. Universitext. Springer Verlag, 1992.
18. P. S. Mulry. Categorical fixed-point semantics. *Theoretical Computer Science*, 70:85–97, 1990.
19. J. van Oosten. A combinatory algebra for sequential functionals of higher type. In *Logic Colloquium 1997*. Cambridge University Press. To appear. Currently available as Preprint 996, Dept. of Mathematics, Utrecht University, 1997.
20. J. van Oosten and A.K. Simpson. *Axioms and (Counter)examples in Synthetic Domain Theory*. Preprint 1080, Dept. of Mathematics, Utrecht University, 1998.
21. W.K.-S. Phoa. Effective domains and intrinsic structure. In *Proceedings of 5th Annual Symposium on Logic in Computer Science*, 1990.
22. G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
23. G.D. Plotkin. *Denotational semantics with partial functions*. Lecture notes, CSLI Summer School, 1985.
24. B. Reus. *Program Verification in Synthetic Domain Theory*. PhD thesis, University of Munich, 1995.
25. B. Reus and Th. Streicher. General synthetic domain theory — a logical approach. In *Proceedings of CTCS '97*, pages 293–313. Springer LNCS 1290, 1997.
26. G. Rosolini. *Continuity and Effectivity in Topoi*. PhD thesis, Oxford, 1986.
27. G. Rosolini. *Notes on Synthetic Domain Theory*. Unpublished notes, Available from `ftp://ftp.disi.unige.it/`, 1995.
28. D. S. Scott. Identity and existence in intuitionistic logic. In *Applications of Sheaves*, pages 661–696. Springer LNM 753, 1979.
29. A.K. Simpson. *Recursive types in Kleisli categories*. Unpublished manuscript. Available from `ftp://ftp.dcs.ed.ac.uk/pub/als/Research/`, 1992.
30. A.K. Simpson. *Algebraic Compactness in Intuitionistic Set Theory*. Presented at *PSSL*, Edinburgh, October 1995. In preparation, 1999.
31. P. Taylor. The fixed point property in synthetic domain theory. In *Proc. of 6th Annual Symposium on Logic in Computer Science*, 1991.
32. G. Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.

Logical Relations and Inductive/Coinductive Types

Thorsten Altenkirch

Ludwig-Maximilians-Universität, Oettingenstr. 67, 80538 München, Germany,
alti@informatik.uni-muenchen.de
http://www.tcs.informatik.uni-muenchen.de/~alti/

Abstract. We investigate a λ calculus with positive inductive and coinductive types , which we call $\lambda^{\mu,\nu}$, using logical relations. We show that parametric theories have the strong categorical properties, that the representable functors and natural transformations have the expected properties. Finally we apply the theory to show that terms of functorial type are almost canonical and that monotone inductive definitions can be reduced to positive in some cases.

1 Introduction

We investigate a λ calculus with positive inductive and coinductive types, which we call $\lambda^{\mu,\nu}$, using logical relations. Here μ is used to construct the usual datatypes as initial algebras, where ν-types are used for lazy types as terminal coalgebras.

A calculus related to $\lambda^{\mu,\nu}$ has for example been investigated by Geuvers [Geu92], but he does not consider nested μ-types. Loader introduces the strictly positive fragment (and only μ-types) in [Loa97] and shows that the theory of the PER model is maximal consistent. The restriction to the strict positive fragment gives rise to a predicative theory closely related to the theory $\mathbf{ID}_{<\omega}$ known from proof theory. The precise strength of the system of positive inductive definitions is unknown but can hardly be called predicative. Note that a proof by Buchholz [Buc81] that positive inductive definitions are conservative over strictly positive ones, does not apply to theories with parameters, i.e. to μ-types with free type-variables. The contribution of ν-types to the strength of the system seems peripheral but we shall not be concerned with this issue here. ν-Types certainly have an important role for representing infinite structures in programming.

Logical relations have been well investigated in the context of System F to model Reynold's notion of *parametricity* [Rey83]. In his stimulating paper [Wad89] shows how to obtain *free theorems* and in [AP93] this is made precise by presenting a logic for parametricity. In particular it is shown show that parametricity entails that the 2nd order encodings of μ and ν-types are strong, i.e. have the expected categorical properties. Several authors have presented categorical versions of parametricity [MR91,RR94,Has90]

The system we present here can be encoded in System F. However, we believe that it is of independent interest since inductive and coinductive definitions

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 343–354, 1999.

should be considered as more primitive than 2nd order quantification. In particular it is very straightforward to identify a predicative subsystem by restricting to strictly positive definitions. We present a notion of logical relations for inductive and coinductive types using least and greatest fixpoints on the metalevel, which means that the metatheory for the strict positive fragment itself is predicative and does not require 2nd order quantification.

One question which caused some confusion in the context of System F are the prerequisites for parametricity, i.e. the question whether Wadler's theorems are really free. I.e. the 2nd order encodings are not strong in the initial theory (not even if you restrict yourselves to closed terms), although it seems naively parametric. We say that a theory is parametric iff the identity extension property (not lemma) holds and show that all parametric theories are strong in the categorical sense.

We apply the theory to a question which came up in the work by Ralph Matthes [Mat98] on *monotone* inductive types, which is an apparent generalisation of positive. Matthes conjectured that good monotonicity witnesses can be already recognised by just testing the first functor law. It turns out that we can prove this conjecture rather easily using the theory of logical relations developed here. We also show that in the positive fragment the functions expressible with monotone inductive types are the same as in the standard theory. Everything dualises easily to the coinductive case. Another application of logical relations is the verification of the functor laws for the canonical map-terms. It relies on the fact that all terms with the type of a natural transformation are indeed natural which again can be easily verified using logical relations.

2 The calculus $\boldsymbol{\lambda^{\mu,\nu}}$

Our starting point is simply typed λ-calculus over an infinite set of type variables V^{Ty}.

Let $S = \{+, -\}$ with the operation $- \in S \to S$ satisfying $-(+) = -, -(-) = +$. We simultaneously define the set of types Ty and the relation $\mathbf{OCC} \subseteq S \times \mathrm{V}^{\mathrm{Ty}} \times \mathrm{Ty}$ of positive and negative occurrence:

$$\frac{X \in \mathrm{V}^{\mathrm{Ty}}}{X \in \mathrm{Ty}} \qquad \frac{\sigma, \tau \in \mathrm{Ty}}{\sigma \to \tau, \sigma \times \tau, \sigma + \tau \in \mathrm{Ty}} \qquad \frac{X \mathbf{OCC}^+ \sigma}{\mu X.\sigma, \nu X.\sigma \in \mathrm{Ty}}$$

$$\frac{}{X \mathbf{OCC}^+ X} \qquad \frac{X \neq Y}{X \mathbf{OCC}^s Y}$$

$$\frac{X \mathbf{OCC}^{-s} \sigma \qquad X \mathbf{OCC}^s \tau}{X \mathbf{OCC}^s \sigma \to \tau} \qquad \frac{X \mathbf{OCC}^s \sigma \qquad X \mathbf{OCC}^s \tau}{X \mathbf{OCC}^s \sigma \times \tau \qquad X \mathbf{OCC}^s \sigma + \tau}$$

$$\frac{X \mathbf{OCC}^s \sigma \qquad Y \mathbf{OCC}^+ \sigma}{X \mathbf{OCC}^s \mu Y.\sigma \qquad X \mathbf{OCC}^s \nu Y.\sigma}$$

We write $\boldsymbol{X} \mathbf{OCC}^s \sigma$ if $\boldsymbol{X} = X_1, \ldots, X_n$ s.t. for all $1 \leq i \leq n$: $X_i \mathbf{OCC}^s \sigma$.

The strictly positive system can be obtained by defining $-(-) = -$ instead, or alternatively by changing the rule for $\to$ to

$$\frac{X \notin \mathrm{FV}\sigma \qquad X\mathbf{OCC}^+\tau}{X\mathbf{OCC}^+\sigma \to \tau}$$

The definition of $\mathbf{OCC}^-$ can then be omitted.

Given a type σ we say $\sigma \in \mathrm{Ty}(\boldsymbol{X})$ if all free type variables in σ are included in the finite sequence $\boldsymbol{X}$. We write $\mathrm{Tm}(\sigma)$ for the set of closed terms of type σ.

Inspired by categorical notation we define

$$1_\sigma \equiv \lambda x : \sigma.x$$

and given $f : \rho \to \tau, g : \sigma \to \rho$

$$f \circ g \equiv \lambda x : \sigma.f(gx)$$

We may drop type annotations whenever they are uniquely determined from the context.

We write $\sigma(X)$ for a type in which the variable X appears freely and then in the same context $\sigma(\tau)$ for the substitution $\sigma[X := \tau]$. This is extended to sequences, i.e. $\sigma(\boldsymbol{X})$ and $\sigma(\boldsymbol{\tau}) \equiv \sigma[\boldsymbol{X} := \boldsymbol{\tau}] \equiv \sigma[X_1 := \tau_1, \ldots, X_n := \tau_n]$ where we silently assume that the sequences involved have the right length.

Given $\sigma, \tau, \theta \in \mathrm{Ty}(\boldsymbol{X})$ and $\rho(X) \in \mathrm{Ty}(\boldsymbol{X}, X)$ with $X\mathbf{OCC}^+\rho(X)$ we introduce the following (families of) constants:

$$\begin{aligned}
\pi_1^{\sigma,\tau} &: (\sigma \times \tau) \to \sigma \\
\pi_2^{\sigma,\tau} &: (\sigma \times \tau) \to \tau \\
\mathrm{pair}^{\sigma,\tau} &: \sigma \to \tau \to (\sigma \times \tau) \\
\mathrm{i}_1^{\sigma,\tau} &: \sigma \to (\sigma + \tau) \\
\mathrm{i}_2^{\sigma,\tau} &: \tau \to (\sigma + \tau) \\
\mathrm{case}^{\sigma,\tau,\theta} &: (\sigma \to \theta) \to (\tau \to \theta) \to (\sigma + \tau) \to \theta \\
\mathrm{C}_{X.\rho(X)} &: \rho(\mu X.\rho) \to \mu X.\rho \\
\mathrm{It}^{\sigma}_{X.\rho} &: (\rho(\sigma) \to \sigma) \to (\mu X.\rho) \to \sigma \\
\mathrm{D}_{X.\rho(X)} &: \nu X.\rho \to \rho(\nu X.\rho) \\
\mathrm{Co}^{\sigma}_{X.\rho(X)} &: (\sigma \to \rho(\sigma)) \to \sigma \to (\nu X.\rho)
\end{aligned}$$

We define the empty type $\emptyset \equiv \mu X.X$ and the unit type $\mathbf{1} \equiv \nu X.X$ and $* = \mathrm{Co}_{X.X}\ 1_{\emptyset\to\emptyset}1_\emptyset : \mathbf{1}$ as the canonical inhabitant of $\mathbf{1}$. We write $\mathbf{2} = \mathbf{1} + \mathbf{1}$ for the type of booleans and $\mathrm{true} = \mathrm{i}_1\ *, \mathrm{false} = \mathrm{i}_2\ *$.

We are now going to define internal functors (or map terms) for positive and negative type abstractions. Given $\boldsymbol{X}\mathbf{OCC}^+\rho(\boldsymbol{X};\boldsymbol{Y}), \boldsymbol{Y}\mathbf{OCC}^-\rho(\boldsymbol{X},\boldsymbol{Y})$, for $1 \le i \le |\boldsymbol{X}|$: $f_i : \sigma_i \to \sigma_i'$ and for $1 \le j \le |\boldsymbol{Y}|$: $g_j : \tau_j \to \tau_j'$ we define

$$\rho(\boldsymbol{f};\boldsymbol{g}) : \rho(\boldsymbol{\sigma};\boldsymbol{\tau'}) \to \rho(\boldsymbol{\sigma'};\boldsymbol{\tau})$$

by induction over the structure of ρ:

$$\begin{aligned}
X_i(\boldsymbol{f};\boldsymbol{g}) &= f_i \\
Z(\boldsymbol{f};\boldsymbol{g}) &= 1_Z \\
(\rho \to \rho')(\boldsymbol{f};\boldsymbol{g}) &= \lambda g.\rho'(\boldsymbol{f};\boldsymbol{g}) \circ g \circ \rho(\boldsymbol{g};\boldsymbol{f}) \\
(\rho \times \rho')(\boldsymbol{f};\boldsymbol{g}) &= \lambda p.\mathrm{pair}(\rho(\boldsymbol{f};\boldsymbol{g})(\pi_1\ p))(\rho'(\boldsymbol{f};\boldsymbol{g})(\pi_2\ p)) \\
(\rho + \rho')(\boldsymbol{f};\boldsymbol{g}) &= \lambda s.\mathrm{case}\ (\mathrm{i}_1 \circ \rho(\boldsymbol{f};\boldsymbol{g}))(\mathrm{i}_2 \circ \rho'(\boldsymbol{f};\boldsymbol{g})) \\
(\mu X.\rho)(\boldsymbol{f};\boldsymbol{g}) &= \mathrm{It}_{X.\rho}\ (\mathrm{C} \circ \rho(1, \boldsymbol{f};\boldsymbol{g})) \\
(\nu X.\rho)(\boldsymbol{f};\boldsymbol{g}) &= \mathrm{Co}_{X.\rho}(\rho(1, \boldsymbol{f};\boldsymbol{g}) \circ \mathrm{D})
\end{aligned}$$

A theory $\sim$ is a family of equivalence relations $\sim_\sigma \subseteq \mathrm{Tm}(\sigma) \times \mathrm{Tm}(\sigma)$ which is a congruence for application and closed under β[1]: $(\lambda x : \sigma.t)u \sim t[x := u]$. Additionally we assume the following equivalences:

$$\begin{aligned}
\pi_1^{\sigma,\tau}(\mathrm{pair}^{\sigma,\tau}\ t\ t') &\sim t \\
\pi_2^{\sigma,\tau}(\mathrm{pair}^{\sigma,\tau}\ t\ t') &\sim t' \\
(\mathrm{case}\ t\ u) \circ \mathrm{i}_1 &\sim t \\
(\mathrm{case}\ t\ u) \circ \mathrm{i}_2 &\sim u \\
(\mathrm{It}_{X.\rho} t) \circ \mathrm{C}_{X.\rho} &\sim t \circ \rho(\mathrm{It}_{X.\rho} t) \\
\mathrm{D}_{X.\rho} \circ (\mathrm{Co}_{X.\rho} t) &\sim \rho(\mathrm{Co}_{X.\rho} t) \circ t
\end{aligned}$$

In general we assume that a theory is consistent, i.e. not all well typed equations on closed terms hold. The initial or least theory is denoted by $\sim^\beta$.

For $\sim^\beta$ a strongly normalising and confluent term rewriting system can be found. These properties can be verified by developing the notion of logical predicates corresponding to the logical relations presented below, see [Mat98]. The main corollary of this is:

Proposition 1. *$\sim^\beta$ is decidable for typed terms.*

2.1 Examples

We can embed System T by defining $\mathrm{Nat} = \mu X.\mathbf{1} + X$ with $0 = \mathrm{C}_{X.\mathbf{1}+X}(\mathrm{i}_1 *), \mathrm{s} = \mathrm{C}_{X.\mathbf{1}+X} \circ \mathrm{i}_2$. From It we can derive the standard iterator for Nat. Also a recursor $\mathrm{R} : \sigma \to (\mathrm{Nat} \times \sigma \to \sigma) \to \mathrm{Nat} \to \sigma$ is definable using the iterator. R does not satisfy the usual equations for a recursor. However, it can be shown that this encoding is correct in extensional theories, to be defined below.

Using ν-types we can define the *conatural* numbers $\mathrm{Nat}' = \nu X.\mathbf{1} + X$, which intuitively are the natural numbers extended by $\omega = \mathrm{Co}^{X.\mathbf{1}+X,\mathbf{1}}\mathrm{i}_2 *$.

Other examples are: lists over σ:

$$\mathrm{L}^\sigma = \mu X.\mathbf{1} + \sigma \times X,$$

[1] Note that we do not include ξ or η, i.e. our starting point is an extension of typed combinatory logic. Later we will introduce extensional theories which are closed under ξ and η for all type formers.

infinite streams over σ:

$$\mathrm{S}^{\sigma} = \nu X.\sigma \times X,$$

finitely branching trees:

$$\mathrm{F} = \mu X.\mathrm{L}^{X}.$$

The last definition is interesting because it uses a μ-type with a free type variable. We call this an inductive definition with parameters[2]. It seems that the counterpart of inductive definition with parameters is (intentionally ?) excluded from the standard definitions of ID, e.g.[Buc81].

We define the type of trees branching over σ:

$$\mathrm{T}^{\sigma} = \mu X.1 + \sigma \to X,$$

e.g. $\mathrm{T}^{\mathbf{2}}$ are binary trees. Using T we can define a hierarchy of tree types:

$$\mathrm{T}_0 = \emptyset$$
$$\mathrm{T}_{n+1} = \mathrm{T}^{\mathrm{T}_n}$$

The hierarchy of trees exhausts the proof-theoretic strength of $\mathrm{ID}_{<\omega}$, however in the presence of non-strict positive definitions with parameters, we can define

$$\mathrm{T}^{\infty} = \mu X.\mathrm{T}^{\mathrm{T}^{X}}$$

This type may raise some doubts in whether this system is predicative (i.e. proof theoretically conservative over the strict positive system).

3 Logical Relations

We consider here n-ary relations on closed terms which have to be closed under $\sim^{\beta}$. Given $\boldsymbol{\sigma} = \sigma_1, \ldots, \sigma_n$ a sequence of closed types, we write $R \in \mathrm{Rel}(\boldsymbol{\sigma})$ for $R \subseteq \mathrm{Tm}(\sigma_1) \times \ldots \times \mathrm{Tm}(\sigma_n)$. We define an operator $\tau^{\#}(-)$ which to any sequence $\boldsymbol{R} = R_1, \ldots, R_m$ with $R_i \in \mathrm{Rel}(\boldsymbol{\sigma_i})$ assigns

$$\tau^{\#}(\boldsymbol{R}) \in \mathrm{Rel}(\tau(\boldsymbol{\sigma_1}), \ldots, \tau(\boldsymbol{\sigma_m}))$$

At the same time we have to verify that if $X_i\mathbf{OCC}^{+}(\mathbf{OCC}^{-})\rho$ then $\rho^{\#}(-)$ is monotone(antitone) in its ith argument.

We extend application to tuples:

$$\boldsymbol{t}\ \boldsymbol{u} = t_1\ u_1, \ldots, t_n\ u_n$$

and

$$c\ \boldsymbol{u} = c\ u_1, \ldots, c\ u_n$$

where c is a constant.

[2] [Mat98] calls this *interleaving inductive definitions.*

Projection

$$X_i^\#(\boldsymbol{R})\ \boldsymbol{t} :\Longleftrightarrow\ R_i\ \boldsymbol{t}$$

$\to$

$$(\sigma \to \tau)^\#(\boldsymbol{R})\ \boldsymbol{t} :\Longleftrightarrow\ \forall \boldsymbol{u}.\sigma^\#(\boldsymbol{R})\boldsymbol{u} \to \tau^\#(\boldsymbol{R})(\boldsymbol{t}\ \boldsymbol{u})$$

$\times$

$$(\sigma \times \tau)^\#(\boldsymbol{R})\ \boldsymbol{t} :\Longleftrightarrow\ \sigma^\#(\boldsymbol{R})(\pi_1\boldsymbol{t}) \wedge \tau^\#(\boldsymbol{R})(\pi_2\boldsymbol{t})$$

$+$ $(\sigma_1 + \sigma_2)^\#(\boldsymbol{R})$ is the least relation generated by

$$\frac{\sigma_k^\#(\boldsymbol{R})\boldsymbol{u}}{(\sigma_1 + \sigma_2)^\#(\boldsymbol{R})(\mathrm{i}_k\ \boldsymbol{u})}\ +^\#$$

μ Given $\mu X.\rho$ we know that $X\mathbf{OCC}^+\rho$ and hence $\rho^\#(\boldsymbol{R}, -)$ is monotone. We define $(\mu X.\rho)^\#$ as the least relation closed under

$$\frac{\rho^\#(\boldsymbol{R}, (\mu X.\rho)^\#)\ \boldsymbol{t}}{(\mu X.\rho)^\#(\mathrm{C}_{X.\rho}\ \boldsymbol{t})}\ \mu^\#$$

ν Given $\nu X.\rho$ we know that $X\mathbf{OCC}^+\rho$ and hence $\rho^\#(\boldsymbol{R}, -)$ is monotone. We define $(\nu X.\rho)^\#$ as the greatest relation closed under

$$\frac{(\nu X.\rho)^\#\ \boldsymbol{t}}{\rho^\#(\boldsymbol{R}, (\nu X.\rho)^\#)\ (\mathrm{D}_{X.\rho}\ \boldsymbol{t})}\ \nu^\#$$

Proposition 2 (Fundamental property of logical relations). *Given* $\sigma(\boldsymbol{X}) \in \mathrm{Ty}(\boldsymbol{X})$, $t(\boldsymbol{X}) : \sigma(\boldsymbol{X})$ *and* $\boldsymbol{R} : \boldsymbol{\tau_1} \dots \boldsymbol{\tau_m}$ *we have that*

$$\sigma^\#(\boldsymbol{R})\ (t(\boldsymbol{\tau_1}), \dots, t(\boldsymbol{\tau_m}))$$

Proof. To simplify notation we consider only the binary case here (and use infix notation for the relation). The proof proceeds by induction on (the derivation of) $t(\boldsymbol{X}) : \sigma(\boldsymbol{X})$. Since the proof for simply typed λ terms is standard (we only have to check combinatory logic), we concentrate on the new constants:

$\times$**,**$+$ We just do $+$: The case for i_k follows directly from $+^\#$. Assume $f_k(\sigma_k \to \theta)^\#(\boldsymbol{R})f'_k$ and $t(\sigma_1 + \sigma_2)^\#(\boldsymbol{R})t'$. By the definition of $(\sigma_1 + \sigma_2)^\#(\boldsymbol{R})$ we have that $t \sim \mathrm{i}_k\ u$, $t' \sim \mathrm{i}_k\ u'$ and $u\sigma_k^\#(\boldsymbol{R})u'$. Using case $f_1f_2(\mathrm{i}_k u) \sim f_k\ u$ (and analogously for f'_k, u') we get case $f_1f_2(\mathrm{i}_k u)\theta^\#(\boldsymbol{R})$case $f'_1f'_2(\mathrm{i}_k u')$ and hence the property for case.

μ**,**ν The case for C follows directly from $\mu^\#$. Assume $f(\rho(\sigma) \to \sigma)^\#(\boldsymbol{R})f'$ define $R \in \mathrm{Rel}(\mu X.\rho(X, \tau), \mu X.\rho(X, \tau'))$ by $tRt' :\Longleftrightarrow$ It $f\ t\sigma^\#(\boldsymbol{R})$It $f'\ t'$. Using the β-equality for μ it is easy to see that R is closed under $\mu^\#$. Hence by minimality we have that $t(\mu X.\rho)^\#(\boldsymbol{R})t'$ implies tRt' which directly entails the correctness for It. The case for ν is an easy dualisation.

□

4 Extensional Theories

Using logical relations we show properties of extensional theories, that is theories in which semantically equivalent terms are identified. We show that parametric theories have the universal categorical properties.

Definition 1 (Strong Theories). *We call a theory* strong *iff it satisfies the usual universal properties of function spaces, products, coproducts, initial algebras and terminal coalgebras. Equivalently the theory $\sim$ has to be closed under the usual $\xi_{\to}$ and $\to_\eta$ rules and:*

$$\frac{}{\text{pair } (\pi_1\ t)\ (\pi_2 t) \sim t}\ \eta_\times$$

Note that there is no $\xi_\times$ because the appropriate equality follows from the fact that application is a congruence.

$$\frac{}{(\text{case } \mathrm{i}_1\ \mathrm{i}_2) \sim 1}\ \eta_+ \qquad \frac{}{u \circ (\text{case } t_1\ t_2) \sim \text{case } (u \circ t_1)\ (u \circ t_2)}\ \xi_+$$

$$\frac{}{\mathrm{It}^{X.\rho}\mathrm{C}_{X.\rho} \sim 1}\ \eta_\mu \qquad \frac{\begin{array}{c} f : \rho(\tau) \to \tau \\ g : \rho(\sigma) \to \sigma \\ h : \tau \to \sigma \end{array} \quad h \circ f \sim g \circ \rho(h)}{h \circ (\mathrm{It}\ f) \sim \mathrm{It}\ g}\ \xi_\mu$$

$$\frac{}{\mathrm{Co}_{X.\rho}\mathrm{D}_{X.\rho} \sim 1}\ \eta_\nu \qquad \frac{\begin{array}{c} f : \tau \to \rho(\tau) \\ g : \sigma \to \rho(\sigma) \\ h : \sigma \to \tau \end{array} \quad f \circ h \sim \rho(h) \circ g}{(\mathrm{Co}\ f) \circ h \sim \mathrm{Co}\ g}\ \xi_\nu$$

We here define parametric theories using n-ary relations.

Definition 2 (Parametric theories). *We define the n-diagonal as*

$$\Delta^n_\sigma = \{(t_1, \ldots, t_n) \mid \forall 1 \le i, j \le n. t_i \sim t_j\} \in \mathrm{Rel}(\sigma^n).$$

We call a theory parametric iff it satisfies the identity extension property, that is for any $\tau(\boldsymbol{X}) \in \mathrm{Ty}(\boldsymbol{X})$ we have:

$$\tau^\#(\Delta^n_{\sigma_1}, \ldots, \Delta^n_{\sigma_m}) = \Delta^n_{\tau(\boldsymbol{\sigma})}$$

In the case $n = 2$ this is just

$$t \tau^\#(\sim_{\boldsymbol{\sigma}}) t' \iff t \sim_{\sigma(\boldsymbol{\tau})} t'$$

Definition 3 (Observational equivalence). *We define the observational equivalence by*

$$t \sim^{\mathrm{obs}}_\sigma u \iff \forall p : \sigma \to \mathbf{2}. p\ t \sim^\beta p\ u$$

The following proposition is folklore and goes back to Statman for simply type λ calculus and Moggi [Mog] in the case of System F. In fact it can be extended to all theories which have a type of booleans with a polymorphic if.

Proposition 3. *The observational congruence is the greatest consistent theory, that is if $t \sim u$ in any (consistent) theory then $t \sim^{\mathrm{obs}} u$*

Proof. To see that obs is actually a theory note that $t \sim^{\mathrm{obs}} t' \to tu \sim^{\mathrm{obs}} t'u$ and symmetrically. It is easy to see that obs is closed by the defining equations for $\sim^{\beta}$. To see that $\sim^{\mathrm{obs}}$ is consistent use $1_{\mathbf{2}}$ to see that true $\not\sim^{\mathrm{obs}}$ false.

Given any consistent theory with $t \sim_{\sigma} t'$. Assume that there is a $p : \sigma \to \mathbf{2}$ s.t. $p\ t \not\sim^{\beta} p\ t'$. W.l.o.g assume $p\ t \sim^{\beta}$ true and $p\ t \sim^{\beta}$ false but then true $\sim p\ t \sim p\ t' \sim$ false and hence $\sim$ is not consistent. □

In the following we use $\sim$ for any parametric equality.

5 Main results

Given $t : \sigma \to \tau$ we define the graph of t: $t^{\#} : \sigma \iff \tau$ by $u\ t^{\#}\ u' :\iff u' \sim tu$. We show at once the graph theorem, naturality of certain functions and the functor law, because all these results depend upon each other — the situation is very different from System F [AP93].

Proposition 4 (Map properties). *Assume*

$$\boldsymbol{X}\mathbf{OCC}^{+}\rho(\boldsymbol{X};\boldsymbol{Y},\boldsymbol{Z}), \boldsymbol{Y}\mathbf{OCC}^{-}\rho(\boldsymbol{X},\boldsymbol{Y},\boldsymbol{Z}).$$

1. $\rho^{\#}(f^{\#}; {g^{\#}}^{-1}; \sim) \iff \rho(f; g; \sim)$
2. *For $t(X) : \rho(X) \to \rho'(X), f : \sigma \to \tau$ it holds:*

$$t(\tau) \circ \rho(f) \sim \rho'(f) \circ t(\sigma)$$

3. *The η-rules are valid.*
4. *the μ-rules are valid.*
5. $\rho(1_{\sigma}) \sim 1_{\rho(\sigma)}$
6. $\rho(f \circ g; h \circ k) \sim \rho(f; k) \circ \rho(g; h)$

Proof. We show 1-6 by parallel induction over the size of ρ, ρ'. We only consider the case of single argument covariant functors here.

1. The cases for the basic type constructors $\to, \times, +$ are straightforward. We discuss the case for μ in some detail: Let $\rho(X) = \mu Y.\theta(Y, X)$. We have to show:
   (a) $x\rho^{\#}(f^{\#})y \to x\rho(f)^{\#}y$
   We show that

$$x\theta^{\#}(\rho(f)^{\#}, f^{\#})y \to (\mathrm{C}\ x)\rho(f)^{\#}(\mathrm{C}\ y) \tag{1}$$

From this the proposition above follows from the fact that $\rho^{\#}(f^{\#})$ is the least relation with this property. To show 1 assume $x\theta^{\#}(\rho(f)^{\#}, f^{\#})y$ which by ind.hyp. implies $y \sim \theta(\rho(f), f)\ x$. We want to show

$$(\mathrm{C}\ x)\rho(f)^{\#}(\mathrm{C}\ y)$$

which is equivalent to

$$\begin{aligned}\mathrm{C}\ y &\sim \rho(f)(\mathrm{C}\ x)\\ &\sim \mathrm{C} \circ \theta(1, f) \circ \theta(\rho(f), 1)\\ &\sim \mathrm{C} \circ \theta(\rho(f), f)\end{aligned}$$

which follows from the assumption.

(b) $x\rho(f)^{\#}y \to x\rho^{\#}(f^{\#})y$
We show that $P(x) = x\rho^{\#}(f^{\#})\rho(f)\ x$ is closed under the condition for (unary) logical predicates, that is $\rho^{\#}(P)(cx) \to P\ x$ using the closure condition for $\rho^{\#}(f^{\#})$.

2. Parametricity implies

$$x\rho^{\#}(f^{\#})y \to \alpha(\sigma)\ x\rho'^{\#}(f^{\#})\alpha(\sigma)\ x$$

Using 1. we obtain:

$$\rho'(f) \circ \alpha(\sigma) \sim \alpha(\tau) \circ \rho(f)$$

3. We just show η_{μ}: We show that $R = (\mathrm{It}^{X.\rho}\ \mathrm{C})^{\#}$ is closed under $\mu^{\#}$. Hence assume $x\rho^{\#}(R)y$. We have to show $\mathrm{C}\ x\ R\ \mathrm{C}\ y$ that is $(\mathrm{It}\ \mathrm{C})(\mathrm{C}x) \sim (\mathrm{C}y)$. We note that $(\mathrm{It}\ \mathrm{C})(\mathrm{C}x) \sim \mathrm{C}(\rho(\mathrm{It}\ \mathrm{C})x)$ and hence we can reduce the problem to the hypothesis. Using the minimality of $\mu^{\#}$ we conclude $x \sim y \to \mathrm{It}\ \mathrm{C}\ x \sim y$ and hence $\mathrm{It}\ \mathrm{C}\ \sim 1$.
4. Similar to the previous case.
5. Follows directly from the η-rules.
6. For $\to$,+ and $\times$ the composition laws follow directly from the ξ-rules. However for μ and ν the situation is more complicated. Let's consider $\rho(X) = \mu Y.\theta(X, Y), g : \sigma_1 \to \sigma_2, f : \sigma_2 \to \sigma_3$,:

$$\begin{aligned}\rho(f \circ g) &\sim \mathrm{It}(\mathrm{C} \circ \theta(\rho(\sigma_3), f \circ g)\\ \rho(f) \circ \rho(g) &\sim (\mathrm{It}(\mathrm{C} \circ \theta(\rho(\sigma_3), f))) \circ (\mathrm{It}(\mathrm{C} \circ \theta(\rho(\sigma_2), g)))\end{aligned}$$

Let $h = \mathrm{It}(\mathrm{C} \circ \theta(\rho(\sigma_3), f))$ To show that both sides are equal we apply ξ_{μ} (ind.hyp. 4.) which reduces the problem to showing the equivalence of

$$\mathrm{C} \circ \theta(\rho(\sigma_3), f \circ g) \circ \theta(h, \sigma_1)$$

and

$$h \circ \mathrm{C} \circ \theta(\rho(\sigma_2), g).$$

Using $\sim_{\beta}$ on the second term we obtain

$$\mathrm{C} \circ \theta(\rho(\sigma_3), f) \circ \theta(h, \sigma_3) \circ \theta(\rho(\sigma_2), g)$$

Now using naturality (ind.hyp. 2.) we can show that the term is equivalent to

$$\mathrm{C} \circ \theta(\rho(\sigma_3), f) \circ \theta(\rho(\sigma_2), g) \circ \theta(h, \sigma_1)$$

and by applying the ind.hyp. 6. we show the equivalence to the first term. □

Note that [Geu92] does not prove the functor laws for nested μ-types, which is the difficult case. The use of logical relations and naturality may be avoided by an exhaustive case analysis[3] but the present proof seems much more elegant.

Corollary 1. *Parametric theories are strong.*

6 Monotone inductive types

We now turn our attention to general properties of terms having the type of map-terms. It turns out that those are almost canonical.

Proposition 5. *Given* $X\mathbf{OCC}^+\rho(X)$ *and*

$$m(X, Y) : (X \to Y) \to \rho(X) \to \rho(Y)$$

we set

$$\alpha(X) = m(X, X)1_X : \rho(X) \to \rho(X)$$

We can show that

$$\begin{aligned} m(\sigma, \tau)f &= \rho(f) \circ \alpha(\sigma) \\ &= \alpha(\tau) \circ \rho(f) \end{aligned}$$

Proof. The fundamental theorem entails that for any $S : \sigma \iff \sigma'$ and $R : \tau \iff \tau'$ and for $h : \sigma \to \tau, h' : \sigma' \to \tau'$ such that tSt' implies $(f\ t)\ R\ (f'\ t')$ and $a\rho^{\#}(S)a'$ it is the case that

$$(m(\sigma, \tau)\ f\ a)\rho^{\#}(R)\ (m(\sigma', \tau')\ f'\ a')$$

We now set $f' = 1, S = \sim, R = f^{\#}$ and by using the map theorem we obtain the first line. The second line follows by a symmetric argument. □

We can immediately conclude the following corollary which was conjectured by Ralph Matthes in the context of monotone inductive definitions:

Corollary 2. *Given a map term for* $X\mathbf{OCC}^+\rho(X)$

$$m(X, Y) : (X \to Y) \to \rho(X) \to \rho(Y)$$

[3] Suggested by Ralph Matthes.

which satisfies the first functor law

$$m(X,X)1_X = 1_{\rho(X)}$$

Then m is already equal to the proper map:

$$m(X,Y) \sim \lambda f.\rho(f)$$

We introduce the concept of monotone inductive definitions as a generalisation of positive inductive definitions, i.e. we drop the positivity requirement but parametrise the μ-types with a *monotonicity* witness:

Abbreviate the type of monotonicity witnesses for $\rho(X)$ as $\phi_\rho(X,Y) = (X \to Y) \to (\rho(X) \to \rho(Y))$

$$\frac{m(X,Y) : \phi_\rho(X,Y)}{\mu_m X.\rho(X) \in \mathrm{Ty}}$$

$$\begin{aligned} \mathrm{C}_{m,X.\rho} &: \rho(\mu_m X.\rho) \to \mu_m X.\rho \\ \mathrm{It}^\sigma_{m,X.\rho} &: (\rho(\sigma) \to \sigma) \to (\mu_m X.\rho) \to \sigma \\ (\mathrm{It}^\sigma_{m,X.\rho} t) \circ \mathrm{C}_{m,X.\rho} &\sim t \circ (m(\mu_m X.\rho(X), \sigma(\mathrm{It}^{X.\rho,\sigma} t)) \end{aligned}$$

This presentation is appealing because we do not have to define the functors anymore. Moreover we can show:

Corollary 3. *Given $X\mathbf{OCC}^+\rho(X)$ and any $m(X,Y) : \phi_\rho(X,Y)$, $f : \rho(\sigma) \to \sigma$ then we have:*

$$\mathrm{It}^{X.\rho,\sigma}_m \sim \mathrm{It}^{X.\rho,\sigma}((m(\sigma,\sigma)1_\sigma) \circ f)$$

This implies that for positive ρ everything we can define for monotone types is already definable in the standard theory. The analogous facts hold about monotone ν-types.

Note that we do not say anything about the case when X does not appear positively in $\rho(X)$. Indeed, there are interesting examples of monotone but not positive type abstractions like the following monotone type (discovered by Ulrich Berger):

$$\mu_m X.(((X \to \sigma) \to X) \to X) \to X$$

However, it seems that no such m satisfies the first functor law, and we conjecture that they do not increase the computational strength of the system.

Acknowledgements

I would like to thank my colleague Ralph Matthes for interesting discussion on the topic of this paper and the anonymous referees for helpful comments.

References

AP93. Martin Abadi and Gordon Plotkin. A logic for parametric polymorphism. In *Typed Lambda Calculi and Applications – TLCA '93*, pages 361–375, 1993. 343, 350

Buc81. Wilfried Buchholz. The $\omega_{\mu+1}$-rule. In *Iterated Inductive Definitions and Subsystems of Analysis: Recent Proof-Theoretical Studies*, volume 897 of *Lecture Notes in Mathematics*, pages 188–233. 1981. 343, 347

Geu92. Herman Geuvers. Inductive and coinductive types with iteration and recursion. In *Workshop on Types for Proofs and Programs, Båstad*, pages 193–217, 1992. 343, 352

Has90. Ryu Hasegawa. Categorical datatypes in parametric polymorphism. In *Fourth Asian Logic Conference*, 1990. 343

Loa97. Ralph Loader. Equational theories for inductive types. *Annals of Pure and Applied Logic*, 84:175–217, 1997. 343

Mat98. Ralph Matthes. *Extensions of System F by Iteration And Primitive Recursion on Monotone Inductive Types.* PhD thesis, University of Munich, 1998. To appear. 344, 346, 347

Mog. Eugenio Moggi. Email to the Type mailing list. 350

MR91. QingMing Ma and John C. Reynolds. Types, abstraction and parametric polymorphis, part 2. In *Proceedings of the 1991 Mathematics of Programming Semantics Conference*, 1991. 343

Rey83. John C. Reynolds. Types, abstraction and parametric polymorphism. In R.E.A. Mason, editor, *Information Processing 83*, 1983. 343

RR94. E.P. Robinson and G. Rosolini. Reflexive graphs and parametric polymorphism. In S. Abramsky, editor, *Proc. 9th Symposium in Logic in Computer Science*, pages 364–371, Paris, 1994. I.E.E.E. Computer Society. 343

Wad89. Philip Wadler. Theorems for free ! Revised version of a paper appearing in: *4 th International Symposium on Functional Programming Languages and Computer Architecture*, June 1989. 343

On the Complexity of H-Subsumption

Reinhard Pichler

Technische Universität Wien
reini@logic.at

Abstract. The importance of subsumption as a redundancy elimination method in automated theorem proving is generally acknowledged. For a given Herbrand universe H, it can be further strengthened to the so-called H-subsumption, i.e.: A clause D is H-subsumed by a clause set $\mathcal{C}$, iff for every H-ground instance $D\theta$ of D there is a clause $C \in \mathcal{C}$, s.t. C subsumes $D\theta$. In recent time, H-subsumption has gained increasing importance especially in the field of automated model building (cf. e.g. [5], [4], [6]). Furthermore, it can be easily shown that H-subsumption may be incorporated as a redundancy deletion rule into many familiar (resolution- and paramodulation-based) inference systems without destroying the refutational completeness.
However, no satisfactory algorithm for checking H-subsumption has been presented so far. We therefore have to investigate the inherent complexity of H-subsumption in order to explain this lack of efficient algorithms: The main result of this work is a Π_2^p-completeness proof for H-subsumption even if it is subjected to some strong restrictions. Hence, unless the polynomial hierarchy collapses to the first level, H-subsumption is non-polynomially more complex than ordinary subsumption.
Finally we present a new algorithm for H-subsumption whose complexity is compared with previously known algorithms (i.e.: from [5] and [4] on the one hand and from [6] on the other hand). The main advantage of our approach is that the total size of an H-subsumption problem (i.e.: in particular, the term depth of the expressions involved) only has polynomial influence on the overall (time and space) complexity. This is in great contrast to the other two approaches.

1 Introduction

Subsumption is a well established method of redundancy elimination despite its high worst case complexity (i.e.: it is NP-complete, cf. [7], problem LO18). In fact, an efficient implementation of subsumption is an integral part of virtually every practically successful automated theorem prover. For a given Herbrand universe H, (ordinary) subsumption can be extended to an even stronger redundancy criterion, namely to the so-called *H-subsumption* (a term introduced in [6]): A clause D is H-subsumed by a clause set $\mathcal{C}$, iff for every H-ground instance $D\theta$ of D there is a clause $C \in \mathcal{C}$, s.t. C subsumes $D\theta$. In particular in automated model building, H-subsumption plays a crucial role, as recent publications in this field demonstrate (cf., for instance, the c-dissubsumption rule of Caferra et

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 355–371, 1999.

al. in [5] or [4] and the automated model construction algorithm given in [6]). Moreover, it can be easily shown that H-subsumption can be incorporated as a redundancy deletion rule into many familiar (resolution- and paramodulation-based) inference systems without destroying the refutational completeness.

However, no satisfactory algorithm for checking H-subsumption has been presented so far. It is, therefore, the main objective of this work to provide an explanation as to why it seems to be significantly harder to check H-subsumption than ordinary subsumption. The answer given to this question is a Π_2^p-completeness proof for H-subsumption even if it is subjected to some strong restrictions (namely: the clause set $\mathcal{C}$ is a singleton and H consists of 2 elements only). Hence, unless the polynomial hierarchy collapses to the first level, H-subsumption is non-polynomially more complex than ordinary subsumption.

Finally we provide a new algorithm for H-subsumption and compare it with previously known algorithms (i.e.: from [5] and [4] on the one hand and from [6] on the other hand): The main advantage of our approach is that the total size of an H-subsumption problem (i.e.: in particular, the term depth of the expressions involved) only has polynomial influence on the overall (time and space) complexity. This is in great contrast to the other two approaches.

The paper is structured as follows: In Section 2, H-subsumption is contrasted with ordinary subsumption. The compatibility with many common inference systems is shown in 3. In Section 4, the Π_2^p-completeness of H-subsumption is proven even in the case where some strong restrictions are imposed. Finally, in Section 5, we present a new algorithm for H-subsumption and compare its complexity with previously known algorithms. In Section 6, we summarize the main results of this paper and identify some directions for future work. An example provided in the appendix is desigend to illustrate the main ideas of our algorithm.

2 Preliminaries

Unless explicitly stated otherwise, we consider clauses as sets of literals throughout this paper. The following definitions juxtapose ordinary subsumption with H-subsumption:

Definition 2.1. (subsumption) *Let C and D be clauses and let $\mathcal{C}$ be a clause set. Then subsumption of clauses and clause sets, respectively, is defined as follows:*

- *"C subsumes D" (i.e.: $C \leq_{ss} D$), iff there is a substitution θ, s.t. $C\theta \subseteq D$.*
- *"$\mathcal{C}$ subsumes D" (i.e.: $\mathcal{C} \leq_{ss} D$), iff there is a clause $E \in \mathcal{C}$, s.t. $E \leq_{ss} D$.*

Definition 2.2. (H-subsumption) *Let $\mathcal{C}$ be a set of clauses over some Herbrand universe H and let C and D be clauses over H. Then H-subsumption is defined as follows:*

- *"C H-subsumes D" (i.e.: $C \leq_{ss}^{H} D$), iff every H-ground instance $D\theta$ of D is subsumed by C.*

– *"$\mathcal{C}$ H-subsumes D" (i.e.: $\mathcal{C} \leq_{ss}^{H} D$), iff every H-ground instance $D\theta$ of D is subsumed by some clause $E \in \mathcal{C}$.*

By the definition of subsumption, $C \leq_{ss} D$ implies $C \leq_{ss} D\theta$ for any substitution θ over any Herbrand universe H. Hence, (ordinary) subsumption implies H-subsumption for any H. But the converse is, in general, not true as the following example illustrates:

Example 2.1. (ordinary subsumption vs. H-subsumption) Let $C = P(x, y) \vee Q(x) \vee Q(y)$ and $D = P(x, y) \vee Q(a) \vee Q(b)$. Then the following relations hold:

– $C \not\leq_{ss} D$, since there is no substitution θ with domain $\{x, y\}$, s.t. on the one hand $P(x, y)\theta \in D$ and, on the other hand, $Q(x)\theta \in D$ and $Q(y)\theta \in D$.
– For $H = \{a, b\}$, $C \leq_{ss}^{H} D$: Let $D\theta$ be an H-ground instance of D, i.e.: $x\theta \in \{a, b\}$ and $y\theta \in \{a, b\}$. Then $C\theta \leq_{ss} D\theta$.
– For $H = \{a, b, c\}$, $C \not\leq_{ss}^{H} D$: Consider the H-ground substitution $\theta = \{x \leftarrow c, y \leftarrow c\}$. Then the H-ground instance $D\theta$ of D is not subsumed by C.

3 Complete Inference Systems Using H-Subsumption

As was illustrated in the previous section, H-subsumption is a strictly stronger redundancy criterion than ordinary subsumption. In fact, in [13], H-subsumption is compared with other redundancy criteria in the literature and it is shown that H-subsumption is incomparable with the redundancy criteria from [1] and [11], even though the latter criteria (which are based on clause implication) are undecidable on the non-ground level. Hence it is by no means trivial that the refutational completeness of an inference system is preserved if ordinary subsumption is replaced by H-subsumption. In particular, the usual proof found in the literature for the completeness of A-ordering resolution and semantic clash resolution together with ordinary subsumption cannot be simply extended to H-subsumption (e.g. see [10], theorem 4.2.1). This is due to the fact that the lifting property of these resolution refinements together with H-subsumption does not hold, i.e.: Let R_x denote either A-ordering resolution or semantic clash resolution and suppose that $\mathcal{C}' \leq_{ss}^{H} \mathcal{C}$ holds. Then there may exist an R_x-resolvent D from $\mathcal{C}$, s.t. D is not H-subsumed by $\mathcal{C}' \cup \{D' \mid D' \text{ is an } R_x\text{-resolvent from } \mathcal{C}'\}$ (for details, refer to [13]).

Rather than proving the refutational completeness of a concrete inference system when H-subsumption is added, we show the compatibility of H-subsumption with a common completeness proof method, namely *semantic trees* and their extension *transfinite semantic trees*. It is then an easy consequence that H-subsumption is compatible with inference rules like semantic clash resolution, (positive) ordered resolution/paramodulation, blocked ordered resolution/paramodulation and other inference rules, whose completeness is proven via semantic trees in [9] or via transfinite semantic trees in [8], respectively.

Remember from [9] that, in a semantic tree, the nodes are labelled with ground literals (or, more generally, with sets of ground literals). A clause C is

said to *fail* at a node N, if there is a ground instance $C\sigma$ of C, s.t. the nodes along the path from the root to the node N contain the complementary literals from $C\sigma$ as labels. A node N in a semantic tree is called a *failure point* for a clause set $\mathcal{C}$, iff some clause $C \in \mathcal{C}$ fails at N and no clause of $\mathcal{C}$ fails at any node above N. A node N is called an *inference node* for a clause set $\mathcal{C}$, iff N is not a failure point but all nodes immediately below N are failure points. Finally, a semantic tree T is called *closed* for a clause set $\mathcal{C}$, iff every branch of T contains a failure point for $\mathcal{C}$. In order to establish the completeness of a set $\mathcal{R}$ of inference rules via semantic trees, the following properties of $\mathcal{R}$ have to be proven (cf. [9], theorem 2): There is a particular way for associating a finite (closed) semantic tree T with every unsatisfiable clause set $\mathcal{C}$, s.t. there exists an inference node N in T and for the set of clauses $\mathcal{D}$ failing immediately below N, there exists a clause D which fails at N and which is derivable by $\mathcal{R}$ from $\mathcal{D}$.

Suppose that H-subsumption is part of an inference system. Then a clause $D \in \mathcal{C}$ may be deleted from $\mathcal{C}$, if $\mathcal{C} - \{D\} \leq^H_{ss} D$. The following proposition shows that H-subsumption can be easily integrated into inference systems whose completeness is proven via semantic trees:

Proposition 3.1. (H-subsumption and semantic trees) *Let T be a closed semantic tree for some clause set $\mathcal{C}$. Furthermore, let D be a clause in $\mathcal{C}$ which may be deleted by H-subsumption. Then T is also a closed semantic tree for $\mathcal{C} - \{D\}$.*

Proof: Suppose that N is a failure point for $\mathcal{C}$ s.t. D fails at N. Furthermore suppose that D is H-subsumed by some clauses $C_1, \ldots, C_k \in \mathcal{C}$. We have to show that then there exists still a clause $C \in \mathcal{C} - \{D\}$ which fails at N:
By assumption, there is a ground instance D' of D, s.t. the complementary literals of D' occur as labels of the nodes along the path from the root to N. Furthermore, there exists a clause $C \in \mathcal{C}$ s.t. C subsumes D', i.e.: there exists a ground instance C' of C, s.t. all literals of C' are contained in the literals of the ground clause D'. But then C also fails at N. ◇

From proposition 3.1 it follows by an easy induction argument that reduction by H-subsumption does not destroy the refutational completeness of an inference system whose completeness can be proven via semantic trees. In [8], the semantic tree method is extended to transfinite semantic trees so as to cover clauses with equality. Furthermore, some new definitions and a new proof strategy (namely, transfinite induction) are introduced to prove the completeness of several calculi using paramodulation and some refinements thereof via transfinite semantic trees. For details, [8] has to be referred to. However, analogously to proposition 3.1, it can be shown that the "maximum consistent tree" does not grow when a clause set is reduced by H-subsumption. Again it follows easily that calculi whose completeness is proven via transfinite semantic trees, remain complete when reduction by H-subsumption is allowed. Hence, H-subsumption actually does preserve the completeness of the most common (resolution- and paramodulation-based) inference systems.

4 The Complexity of H-Subsumption

In order to investigate the complexity of H-subsumption, we define the following decision problem:

Definition 4.1. (The H-SUB problem) *Let H be some Herbrand universe. Then the H-SUB problem (=* H-subsumption problem*) over H is defined as follows:*

- *instance: $(\mathcal{C} = \{C_1, \ldots, C_n\}, D)$, s.t. $C_1, \ldots, C_n, D$ are clauses over H.*
- *question: Does $\mathcal{C} \leq_{ss}^{H} D$ hold (i.e.: are all H-ground instances $D\theta$ of D subsumed by some clause $C_i \in \mathcal{C}$)?*

The size of a problem instance of H-SUB is given by the total number of symbols. To this end, we assume a simple representation of the clauses as strings of symbols (rather than some more sophisticated representation as directed acyclic graph). As far as the Herbrand universe is concerned, we assume that H is given by its signature. Note however that we do not consider H as part of a problem instance. Instead, the H-SUB problem is considered to be parameterized by the underlying Herbrand universe H. The purpose of this section is to show that the H-SUB problem over the two-elementary Herbrand universe $H = \{0, 1\}$ is Π_2^p-complete. Furthermore, we shall briefly discuss, how this result can be generalized to an arbitrary Herbrand universe.

As usual, the hardness proof is carried out by reducing an already known Π_2^p-hard problem to the new problem H-SUB. To this end, we shall make use of the known Σ_2^p-hard problem QSAT_2 (= *quantified satisfiability with 2 alternations of quantifiers*) or rather its restricted form $3\mathrm{QSAT}_2$. The following definition and theorem are recalled from [14]:

Definition 4.2. ($\mathbf{3QSAT}_i$ problem) *Let $i \geq 1$. The $3QSAT_i$ problem is defined as follows:*

- *instance: $(X_1, \ldots, X_i, E)$, s.t. $X_1, \ldots, X_i$ are pairwise disjoint sets of propositional variables and E is a Boolean formula with propositional variables in $X_1 \cup \ldots \cup X_i$, where E is of the following form:*
  - *(for even i): $E = (l_{11} \wedge l_{12} \wedge l_{13}) \vee \ldots \vee (l_{n1} \wedge l_{n2} \wedge l_{n3})$*
  - *(for odd i): $E = (l_{11} \vee l_{12} \vee l_{13}) \wedge \ldots \wedge (l_{n1} \vee l_{n2} \vee l_{n3})$*
- *question:*
  - *(for even i): Is the quantified Boolean sentence $(\exists X_1)(\forall X_2) \ldots (\forall X_i)E$ satisfiable?*
  - *(for odd i): Is the quantified Boolean sentence $(\exists X_1)(\forall X_2) \ldots (\exists X_i)E$ satisfiable?*

Theorem 4.1. (Σ_i^p-Completeness of $\mathbf{3QSAT}_i$) *The $3QSAT_i$ problem is Σ_i^p-complete.*

Proof: cf. [14], theorem 4.1. ◇

We shall now prove the Π_2^p-completeness of the H-SUB problem over the two-elementary Herbrand universe $H = \{0, 1\}$.

Theorem 4.2. **(Π_2^p-completeness of H-SUB over $H = \{0,1\}$)** *The H-subsumption problem over the Herbrand universe $H = \{0,1\}$ is Π_2^p-complete.*

Proof:

Π_2^p-membership: Let $\mathcal{P} = (\mathcal{C} = \{C_1, \ldots, C_n\}, D)$ be an instance of the H-SUB problem over $H = \{0,1\}$. In order to prove that the complementary problem NOT-H-SUB is in Σ_2^p, we consider the following non-deterministic procedure with oracle:

1. Guess an H-ground instance $D\theta$ of D.
2. For all $i \in \{1, \ldots, n\}$, check by means of a first-order subsumption oracle that $C_i \not\leq_{ss} D\theta$.

This algorithm clearly works in non-deterministically polynomial time. Furthermore, first-order subsumption is a well known NP-complete problem (cf. [7], problem LO18). Hence, the oracle used in the above algorithm is in NP and, therefore, the overall algorithm is in Σ_2^p.

Π_2^p-hardness: By theorem 4.1, we know that the 3QSAT$_2$ problem is Σ_2^p-hard. Therefore, the complementary problem NOT-3QSAT$_2$ is Π_2^p-hard. In order to prove the Π_2^p-hardness of the H-SUB problem, we reduce the NOT-3QSAT$_2$ problem to it:

Let $\mathcal{P} = (X_1, X_2, E)$ be an arbitrary instance of the NOT-3QSAT$_2$ problem, i.e.: $X_1 = \{x_1, \ldots, x_k\}$ and $X_2 = \{y_1, \ldots, y_l\}$ are sets of propositional variables and $E = (l_{11} \wedge l_{12} \wedge l_{13}) \vee \ldots \vee (l_{n1} \wedge l_{n2} \wedge l_{n3})$ is a Boolean formula with propositional variables in $X_1 \cup X_2$. Then we reduce $\mathcal{P}$ to the following instance $\mathcal{H} = (\mathcal{C} = \{C\}, D)$ of the H-SUB problem over $H = \{0,1\}$:

$$C = P(x_1, \ldots, x_k) \vee Q(x_1, \bar{x}_1) \vee \ldots \vee Q(x_k, \bar{x}_k) \vee Q(y_1, \bar{y}_1) \vee \ldots \vee Q(y_l, \bar{y}_l) \vee R(l_{11}, l_{12}, l_{13}) \vee \ldots \vee R(l_{n1}, l_{n2}, l_{n3})$$

$$D = P(x_1, \ldots, x_k) \vee Q(0,1) \vee Q(1,0) \vee R(0,0,0) \vee R(0,0,1) \vee R(0,1,0) \vee R(0,1,1) \vee R(1,0,0) \vee R(1,0,1) \vee R(1,1,0)$$

Note that in $\mathcal{P}$, we use x_i, y_j $\bar{x}_i$ and $\bar{y}_j$ to denote propositional literals, i.e.: x_i, y_j are propositional variables and $\bar{x}_i$, $\bar{y}_j$ are their respective negation. The same notation is used in $\mathcal{H}$ to denote individual variables within first-order formulae. However, no confusion should arise from this, since the meaning is always clear from the context. In fact, this notation was chosen on purpose in order to emphasize the main idea of the equivalence proof given below for the 2 problem instances $\mathcal{P}$ and $\mathcal{H}$: Starting from an interpretation J for the propositional variables x_i, y_j, we shall define a substitution λ on the first-order variables x_i, y_j, $\bar{x}_i$ and $\bar{y}_j$ s.t. a propositional literal (i.e.: x_i, y_j, $\bar{x}_i$ or $\bar{y}_j$, respectively) evaluates to **T** in J, iff λ substitutes the constant 1 for the corresponding first-order variable. Likewise, given a substitution λ on the first-order variables x_i, y_j, $\bar{x}_i$ and $\bar{y}_j$, we shall define an interpretation J for the propositional variables x_i, y_j

s.t. λ substitutes the constant 1 for a first-order variable (i.e.: x_i, y_j, $\bar{x}_i$ or $\bar{y}_j$, respectively), iff the corresponding propositional literal evaluates to $\mathbf{T}$ in J.

The above problem transformation can be clearly done in polynomial time (in fact, even linear time suffices). It remains to prove that the problem instances $\mathcal{P}$ and $\mathcal{H}$ are equivalent, i.e.:

$$(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E \text{ is not satisfiable} \Leftrightarrow \{C\} \leq_{ss}^{H} D$$

Satisfiablility of the quantified formula means, that it evaluates to $\mathbf{T}$ in some interpretation. Likewise, unsatisfiability means, that it evaluates to $\mathbf{F}$ in every interpretation. However, the evaluation of this formula is independent of any interpretations, since it contains no free variables. We can therefore simply write
$(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E \equiv \mathbf{T}$ for satisfiability and
$(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E \equiv \mathbf{F}$ for unsatisfiability.
Moreover, the following equivalences hold:

$(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E$ is not satisfiable $\Leftrightarrow$

$(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E \equiv \mathbf{F} \Leftrightarrow$

$\neg(\exists x_1)\ldots(\exists x_k)(\forall y_1)\ldots(\forall y_l)\,E \equiv \mathbf{T} \Leftrightarrow (\forall x_1)\ldots(\forall x_k)(\exists y_1)\ldots(\exists y_l)\,(\neg E) \equiv \mathbf{T}$

We therefore have to prove the following equivalence:

$$(\forall x_1)\ldots(\forall x_k)(\exists y_1)\ldots(\exists y_l)\,(\neg E) \equiv \mathbf{T} \Leftrightarrow \{C\} \leq_{ss}^{H} D$$

– "⇒": Suppose that $(\forall x_1)\ldots(\forall x_k)(\exists y_1)\ldots(\exists y_l)\,(\neg E) \equiv \mathbf{T}$. Furthermore, let $D\theta$ be an arbitrary H-ground instance of D, i.e.: θ is an H-ground substitution with domain $\{x_1,\ldots,x_k\}$ and range $\{0,1\}$. We have to show that there is an H-ground substitution λ s.t. the relation $C\lambda \subseteq D\theta$ holds:
We define the following interpretation I on $\{x_1,\ldots,x_k\}$:

$$I(x_i) := \begin{cases} \mathbf{T} \text{ if } x_i\theta = 1 \\ \mathbf{F} \text{ otherwise} \end{cases}$$

By assumption, $(\forall x_1)\ldots(\forall x_k)(\exists y_1)\ldots(\exists y_l)\,(\neg E) \equiv \mathbf{T}$. Hence, there is an extension J of I to $\{y_1,\ldots,y_l\}$, s.t. $J(\neg E) = \mathbf{T}$. From this we can define the following H-ground substitution λ with domain $\{x_1,\ldots,x_k,\, y_1,\ldots,y_l\} \cup \{\bar{x}_1,\ldots,\bar{x}_k,\, \bar{y}_1,\ldots,\bar{y}_l\}$ and range $\{0,1\}$:

$$x_i\lambda := \begin{cases} 1 \text{ if } J(x_i) = \mathbf{T} \\ 0 \text{ otherwise} \end{cases}$$

$$\bar{x}_i\lambda := \begin{cases} 1 \text{ if } J(\bar{x}_i) = \mathbf{T} \\ 0 \text{ otherwise} \end{cases}$$

$$y_j\lambda := \begin{cases} 1 \text{ if } J(y_j) = \mathbf{T} \\ 0 \text{ otherwise} \end{cases}$$

$$\bar{y}_j\lambda := \begin{cases} 1 \text{ if } J(\bar{y}_j) = \mathbf{T} \\ 0 \text{ otherwise} \end{cases}$$

We claim that for this substitution λ, the relation $C\lambda \subseteq D\theta$ holds:

1. $P(x_1, \ldots, x_k)\lambda = P(x_1, \ldots, x_k)\theta$, since the equivalences $x_i\lambda = 1 \Leftrightarrow J(x_i) = \mathbf{T} \Leftrightarrow I(x_i) = \mathbf{T} \Leftrightarrow x_i\theta = 1$ hold for all $i \in \{1, \ldots, n\}$.
2. $Q(x_i, \bar{x}_i)\lambda \in \{Q(0,1), Q(1,0)\}$ and $Q(y_j, \bar{y}_j)\lambda \in \{Q(0,1), Q(1,0)\}$ holds for all i and all j, since the interpretation J assigns complementary truth values to the propositional variables x_i, y_j on the one hand and the negated literals $\bar{x}_i, \bar{y}_j$ on the other hand. Hence (by the definition of λ) λ also substitutes complementary constants for the first-order variables x_i, y_j on the one hand and $\bar{x}_i, \bar{y}_j$ on the other hand.
3. By the definition of λ, the equivalence

$$R(l_{i1}, l_{i2}, l_{i3})\lambda = R(1,1,1) \Leftrightarrow J(l_{i1} \wedge l_{i2} \wedge l_{i3}) = \mathbf{T}$$

holds for all $i \in \{1, \ldots, n\}$. But, by assumption, $J(\neg E) = \mathbf{T}$, i.e.: no disjunct of E evaluates to $\mathbf{T}$. Hence, for all i: $R(l_{i1}, l_{i2}, l_{i3})\lambda \in \{R(0,0,0), R(0,0,1), R(0,1,0), R(0,1,1), R(1,0,0), R(1,0,1), R(1,1,0)\}$

– "$\Leftarrow$": Suppose that $\{C\} \leq^H_{ss} D$. Furthermore, let I be an arbitrary interpretation on $\{x_1, \ldots, x_k\}$. We have to prove that there is an extension J of I to $\{y_1, \ldots, y_l\}$, s.t. $J(\neg E) = \mathbf{T}$:
We define the following ground H-substitution θ with domain $\{x_1, \ldots, x_k\}$ and range $\{0, 1\}$:

$$x_i\theta := \begin{cases} 1 \text{ if } I(x_i) = \mathbf{T} \\ 0 \text{ otherwise} \end{cases}$$

By assumption, $\{C\} \leq^H_{ss} D$ and, therefore, $C \leq_{ss} D\theta$. Hence, there exists a ground H-substitution λ with domain $\{x_1, \ldots, x_k, \bar{x}_1, \ldots, \bar{x}_k, y_1, \ldots, y_l, \bar{y}_1, \ldots, \bar{y}_l\}$ and range $\{0, 1\}$, s.t. $C\lambda \subseteq D\theta$.
Via λ, we define the following interpretation J on the propositional variables $\{x_1, \ldots, x_k\} \cup \{y_1, \ldots, y_l\}$:

$$J(x_i) := \begin{cases} \mathbf{T} \text{ if } x_i\lambda = 1 \\ \mathbf{F} \text{ otherwise} \end{cases}$$

$$J(y_j) := \begin{cases} \mathbf{T} \text{ if } y_j\lambda = 1 \\ \mathbf{F} \text{ otherwise} \end{cases}$$

The relation $C\lambda \subseteq D\theta$ can now be used to prove that J is of the desired form:

1. $P(x_1, \ldots, x_k)\lambda = P(x_1, \ldots, x_k)\theta$. Hence, $x_i\lambda = x_i\theta$ holds for all i. But then, J is an extension of I to $\{y_1, \ldots, y_l\}$ (i.e.: I and J coincide on $\{x_1, \ldots, x_k\}$) due to the following equivalences: $I(x_i) = \mathbf{T} \Leftrightarrow x_i\theta = 1 \Leftrightarrow x_i\lambda = 1 \Leftrightarrow J(x_i) = \mathbf{T}$ for all $i \in \{1, \ldots, n\}$.
2. $Q(x_i, \bar{x}_i)\lambda \in \{Q(0,1), Q(1,0)\}$ and $Q(y_j, \bar{y}_j)\lambda \in \{Q(0,1), Q(1,0)\}$ holds for all i and all j. Hence, for all $x_i, \bar{x}_i, y_j, \bar{y}_j$, the substitution λ substitutes 1 for the first-order variable in C, iff the corresponding propositional literal evaluates to $\mathbf{T}$ in J. Therefore, the equivalence

$$R(l_{i1}, l_{i2}, l_{i3})\lambda = R(1,1,1) \Leftrightarrow J(l_{i1} \wedge l_{i2} \wedge l_{i3}) = \mathbf{T}$$

again holds for all $i \in \{1, \ldots, n\}$.

3. for all i: $R(l_{i1}, l_{i2}, l_{i3})\lambda \in \{R(0,0,0),\ R(0,0,1),\ R(0,1,0),\ R(0,1,1),\ R(1,0,0), R(1,0,1), R(1,1,0)\}$ and, therefore, no disjunct of E evaluates to $\mathbf{T}$ in J, i.e.: $J(E) = \mathbf{F}$ and, hence, $J(\neg E) = \mathbf{T}$. ◇

Note that the problem transformation in the Π_2^p-hardness proof above resulted in an H-SUB problem where $\mathcal{C} = \{C\}$ is a singleton. Hence, the H-SUB problem is Π_2^p-complete even in the restricted case of a singleton $\mathcal{C}$ and a two-elementary Herbrand universe H. The idea of "encoding" the truth values $\mathbf{T}$ and $\mathbf{F}$ of propositional variables by certain elements of the Herbrand universe in the Π_2^p-hardness proof can be easily extended to an arbitrary Herbrand universe H with 2 or more elements. Hence, the Π_2^p-hardness of H-SUB holds for any non-trivial Herbrand universe. Likewise, the extension of the Π_2^p-membership proof above to the H-SUB problem over an arbitrary but *finite* H is obvious. Unfortunately, the non-deterministic procedure with NP-oracle in the proof above cannot be easily extended to the case of an *infinite Herbrand universe*. In particular, we cannot provide a polynomial bound on the length of the ground instance that is guessed non-deterministically at the beginning of the procedure. In fact, it is not even trivial to prove that H-subsumption is decidable. In [6], the decidability of H-subsumption is proven by providing a limit d on the term depth of (not necessarily ground) instances of D that have to be inspected. However, even though this limit d is polynomial in the size of the input problem, it is not sufficient to ensure that the length of the resulting instance $D\theta$ of D is also polynomially bounded, if H contains at least one function symbol of arity ≥ 2.

5 An H-Subsumption Algorithm

In [12], the H-subsumption of atoms is investigated, i.e.: the case where all clauses in $\mathcal{C}$ as well as the clause D are atoms. The algorithm provided in [12] for testing H-subsumption of atoms works in 2 steps: first the H-subsumption problem is reduced to another kind of problem which is called the *term tuple cover* problem, i.e.: Given a set $M = \{(t_{11}, \ldots, t_{1k}), \ldots, (t_{n1}, \ldots, t_{nk})\}$ of k-tuples of terms over some Herbrand universe H. Is every ground term tuple $(s_1, \ldots, s_k) \in H^k$ an instance of some tuple $(t_{i1}, \ldots, t_{ik}) \in M$? Then the resulting term tuple cover problem is solved by a comparatively efficient algorithm. Unlike previously known algorithms for testing atomic H-subsumption, the algorithm in [12] depends non-polynomially only on the number of atoms (rather than on the total size) of the input atomic H-subsumption problem. The aim of this section is to extend this idea to the H-subsumption problem of clauses: In theorem 5.1 below, we provide a transformation of the clausal H-subsumption problem into the term tuple cover problem. By making use of the algorithm for the term tuple cover problem from [12], we again end up with an algorithm whose overall complexity depends non-polynomially on the number of atoms (rather than on the total size) of the input clausal H-subsumption problem (cf. theorem 5.5).

In analogy to [6], we shall use the notation $G_H(t_1, \ldots, t_k)$ to denote the set of all H-ground instances of the term tuple $(t_1, \ldots, t_k)$. Furthermore, it is

convenient for our purposes to consider clauses as tuples of literals. Hence, in particular, multiple literals and the order of the literals are significant. Moreover, an n-literal subclause D' of a clause $D = M_1 \vee \ldots \vee M_m$ is a clause built from an arbitrary n-tuple of literals from D, i.e.: a clause of the form $D' = M_{k_1} \vee \ldots \vee M_{k_n}$ with $(k_1, \ldots, k_n) \in \{1, \ldots, m\}^n$. We then get the simple subsumption criterion that $C \leq_{ss} D$ holds, iff there exists a subclause D' of D s.t. D' is an instance of C.

The following theorem shows how an H-subsumption problem can be transformed into an equivalent term tuple cover problem via clause unification:

Theorem 5.1. (transformation of the H-subsumption problem) *Let D, $C_1, \ldots, C_n$ be clauses over some Herbrand universe H, where $D = M_1 \vee \ldots \vee M_m$ and $C_i = L_{i1} \vee \ldots \vee L_{in_i}$. Furthermore, let $V(D) = \{x_1, \ldots, x_k\}$ denote the variables occurring in D and suppose that $V(D) \cap V(C_i) = \emptyset$ for all $i \in \{1, \ldots, n\}$ (i.e.: D and the C_i's have no variables in common). Finally, let Θ denote the set of unifiers of a clause C_i with an appropriate subclause D' of D, i.e.:*

$$\Theta = \{\vartheta \mid \exists i \in \{1, \ldots, n\} \text{ and } \exists (k_1, \ldots, k_{n_i}) \in \{1, \ldots, m\}^{n_i} \text{ s.t. } C_i = L_{i1} \vee \ldots \vee L_{in_i} \text{ and } D_{(k_1,\ldots,k_{n_i})} = M_{k_1} \vee \ldots \vee M_{k_{n_i}} \text{ are unifiable with } \vartheta' = mgu(C_i, D_{(k_1,\ldots,k_{n_i})}) \text{ and } \vartheta = \vartheta'|_{V(D)}\}.$$

Then the following equivalence holds:

$$\{C_1, \ldots, C_n\} \leq_{ss}^{H} D \Leftrightarrow \bigcup_{\vartheta \in \Theta} G_H(x_1\vartheta, \ldots, x_k\vartheta) = H^k.$$

Proof:

- "$\Rightarrow$": Suppose that $\{C_1, \ldots, C_n\} \leq_{ss}^{H} D$ holds. Furthermore let $(t_1, \ldots, t_k)$ be an arbitrary ground term tuple in H^k. We have to show that then $(t_1, \ldots, t_k) \in G_H(x_1\vartheta, \ldots, x_k\vartheta)$ for some $\vartheta \in \Theta$:
  Let σ denote the ground substitution $\sigma = \{x_1 \leftarrow t_1, \ldots, x_k \leftarrow t_k\}$. Then $D\sigma$ is a ground instance of D and, therefore, there exists a clause C_i, s.t. $D\sigma = M_1\sigma \vee \ldots \vee M_m\sigma$ is subsumed by C_i. By the subsumption criterion mentioned above, there exists an n_i-tuple of indices $(k_1, \ldots, k_{n_i}) \in \{1, \ldots, m\}^{n_i}$ s.t. $M_{k_1}\sigma \vee \ldots \vee M_{k_{n_i}}\sigma$ is an instance of $C_i = L_{i1} \vee \ldots \vee L_{in_i}$. But then σ is a ground instance of the mgu of these two clauses, i.e.: $\sigma = \vartheta \circ \eta$ for some substitution η, where $\vartheta' = mgu(C_i, D_{(k_1,\ldots,k_{n_i})})$ and $\vartheta = \vartheta'|_{V(D)}$. Hence, in particular, $(t_1, \ldots, t_k) = (x_1\sigma, \ldots, x_k\sigma) = (x_1\vartheta\eta, \ldots, x_k\vartheta\eta) = (x_1\vartheta, \ldots, x_k\vartheta)\eta \in G_H(x_1\vartheta, \ldots, x_k\vartheta)$.
- "$\Leftarrow$": Now suppose that $\bigcup_{\vartheta \in \Theta} G_H(x_1\vartheta, \ldots, x_k\vartheta) = H^k$. Furthermore let $D\sigma$ be an arbitrary H-ground instance of D. We have to show that then $D\sigma$ is subsumed by some clause C_i:
  Let $\boldsymbol{t} = (t_1, \ldots, t_k) \in H^k$ denote the tuple $\boldsymbol{t} = (x_1\sigma, \ldots, x_k\sigma)$. Then $\boldsymbol{t}$ must be contained in $G_H(x_1\vartheta, \ldots, x_k\vartheta)$ for some $\vartheta \in \Theta$. Hence, $\boldsymbol{t} = (x_1\vartheta, \ldots, x_k\vartheta)\eta$ for some substitution η. By the definition of Θ, there exists an index $i \in \{1, \ldots, n\}$ and an n_i-tuple of indices $(k_1, \ldots, k_{n_i}) \in \{1, \ldots, m\}^{n_i}$ s.t. $C_i = L_{i1} \vee \ldots \vee L_{in_i}$ and $D_{(k_1,\ldots,k_{n_i})} = M_{k_1} \vee \ldots \vee M_{k_{n_i}}$ are unifiable with

$\vartheta' = mgu(C_i, D_{(k_1,\ldots,k_{n_i})})$ and $\vartheta = \vartheta'|_{V(D)}$. Hence, in particular, $C_i\vartheta' = D_{(k_1,\ldots,k_{n_i})}\vartheta' = D_{(k_1,\ldots,k_{n_i})}\vartheta$. Thus $D_{(k_1,\ldots,k_{n_i})}\sigma = D_{(k_1,\ldots,k_{n_i})}\vartheta \circ \eta$ is an instance of $C_i\vartheta'$ and also of C_i. But then $D\sigma$ is subsumed by C_i. ◇

The main idea of the term tuple cover algorithm in [12] is to divide the original problem with n term tuples into subproblems s.t. the number of term tuples in each subproblem is strictly smaller than n and the number of subproblems is bounded by n rather than by the total input length. The division into subproblems is based on a well-known partition of the Herbrand universe H. The number and the size of the resulting subproblems are controlled by certain redundancy criteria introduced in [12]. Example 5.1 illustrates the basic idea of partitioning H and of splitting the original problem into subproblems:

Example 5.1. (partitioning H and division into subproblems) Let H be the Herbrand universe with signature $FS(H) = \{f, g, a\}$. Furthermore, let an instance of the term tuple cover problem be given through the set $M = \{(f(x), y), (g(x), y), (x, x), (x, f(y)), (x, g(y))\}$.

Then M covers H^2, iff
$M_a = \{(a, a), (a, f(y)), (a, g(y))\}$ covers $G_H(a) \times H$,
$M_f = \{(f(x), y), (f(x), f(x)), (f(x), f(y)), (f(x), g(y))\}$ covers $G_H(f(x)) \times H$ and $M_g = \{(g(x), y), (g(x), g(x)), (g(x), f(y)), (g(x), g(y))\}$ covers $G_H(g(x)) \times H$.

This, in turn, is equivalent to the following 3 term tuple cover problems:
$\bar{M}_a = \{(a), (f(y)), (g(y))\}$,
$\bar{M}_f = \{(x, y), (x, f(x)), (x, f(y)), (x, g(y))\}$ and
$\bar{M}_g = \{(x, y), (x, g(x)), (x, f(y)), (x, g(y))\}$.

In the above example, the division into subproblems was not problematical at all: Note that all of the 3 subproblems $\bar{M}_f$, $\bar{M}_g$ and $\bar{M}_a$ have strictly less term tuples than the original problem M. This division into subproblems is based on the following partition of H: Let $FS(H)$ denote the set of function symbols of H (constants are considered as function symbols of arity 0). Then every ground term of H has exactly one $f \in FS(H)$ as leading symbol. Hence, H can be partitioned as follows:

$$H = \bigcup_{f \in FS(H)} G_H\Big(f(x_1, \ldots, x_{\alpha(f)})\Big)$$

The reason why things ran so smoothly in example 5.1 is that two different function symbols (namely f and g) occurred as leading symbols of the terms in the first component. Hence, the terms with leading symbol f and the terms with leading symbol g cannot be contained in the same subproblems. But then, we can be sure that the resulting subproblems are strictly smaller than the original problem.

However, there is no guarantee, that two different function symbols actually do occur in the first component (or in some other component, which we can swap with the first one). In order to cope with these situations, several redundancy criteria are proven in [12], which allow the deletion of "problematical" term

tuples. In [12], the following two cases are treated separately, namely Herbrand universes with two or more function symbols of non-zero arity and Herbrand universes with only one such function symbol. It turns out that the latter case is much more difficult to handle than the former one. Moreover, the complexity upper bound obtained in the former case is by far better than in the latter case (i.e.: c^n vs. $n!$, where n denotes the number of tuples).

Due to space limitations we can only sketch the basic ideas of the algorithm for solving the term tuple cover problem over a Herbrand universe with two or more function symbols of non-zero arity. For the other case as well as for any details, the original paper [12] has to be referred to.

It was already illustrated in example 5.1, that the occurrence of two different function symbols as leading symbols in the first component makes the division into subproblems easy. For the cases where only one function symbol or no function symbol at all occurs as leading symbol in the first component, the following redundancy criteria are proven in [12], theorems 4.4 and 4.5, respectively:

Theorem 5.2. (redundancy criterion based on variable positions) *Let H be an arbitrary, infinite Herbrand universe. Furthermore, let $M = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n\}$ be a set of term k-tuples w.r.t. H. Suppose that all terms occurring in the first component of the tuples from M are variables. Then every term tuple $\boldsymbol{t}_i \in M$ whose variable from the first component occurs somewhere else in $\boldsymbol{t}_i$ is* redundant *and may, therefore, be deleted, i.e.: Let $M' := \{\boldsymbol{t}_i = (x, t_{i2}, \ldots, t_{ik}) \in M \mid s.t.\ x$ is a variable that occurs somewhere else in $\boldsymbol{t}_i\}$. Then M covers all of H^k, iff $M - M'$ does.*

Theorem 5.3. (redundancy criterion based on non-variable positions) *Let H be an arbitrary, infinite Herbrand universe with signature $FS(H)$ and let $f \in FS(H)$ be a function symbol of non-zero arity. Furthermore, let $M = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n\}$ be a set of term k-tuples w.r.t. H. Suppose that there exist term tuples in M with a non-variable term in the first component but the function symbol f does not occur as leading symbol of any of these terms. Then every term tuple $\boldsymbol{t} \in M$ with a non-variable term in the first component is* redundant *and may, therefore, be deleted, i.e.: Let $M' := \{\boldsymbol{t}_i = (t_{i1}, \ldots, t_{ik}) \in M \mid t_{i1}$ is a non-variable term$\}$. Then M covers all of H^k, iff $M - M'$ does.*

Furthermore it is shown that the whole first component may be deleted, if the first component of all tuples contains only variables and if these variables occur nowhere else in their tuple. This leads to the construction of a recursive algorithm for the term tuple cover problem, which works according to the following idea:

1. If the first component contains only variables, then we have to distinguish two cases: If there is a tuple, whose variable in the first component occurs somewhere else in the same tuple, then the redundancy criterion from theorem 5.2 is applicable, i.e.: this tuple may be deleted. Otherwise (i.e.: all variables in the first component occur only once in their tuple) the first component may be deleted from all tuples.

2. If some non-variable occurs in the first component of some tuple, then we can again distinguish two cases: If all function symbols (i.e.: at least two, by assumption) with non-zero arity occur as leading symbol of some first component, then the ordinary splitting from example 5.1 is possible. Otherwise (i.e.: at least one function symbol with non-zero arity is missing as leading symbol of the first component) the redundancy criterion from theorem 5.3 is applicable. Hence, all tuples with a non-variable term in the first component may be deleted.

The following theorem on the complexity of the term tuple cover problem follows easily:

Theorem 5.4. (complexity estimation of the term tuple cover problem) *Let H be a Herbrand universe with at least two function symbols of non-zero arity. Then the term tuple cover problem can be decided in time $O(c^n * p(P))$, where $c = |FS(H)|$, n denotes the number of term tuples and $p(P)$ is some polynomial function in the total length P of an input problem instance.*

Proof: cf. [12], theorem 4.8

Together with the transformation of the H-subsumption problem into the term tuple cover problem (from theorem 5.1), we get the following upper bound on the time complexity of H-subsumption:

Theorem 5.5. (complexity estimation of H-subsumption) *Let H be a Herbrand universe with at least two function symbols of non-zero arity. Furthermore, let an instance of the H-subsumption problem be given through the clause set $\mathcal{C} = \{C_1, \ldots, C_n\}$ and the clause D. Then the H-subsumption problem $\mathcal{C} \leq_{ss}^{H} D$ can be decided in time $O(d^{m^N} * q(P))$ for some constant d, where m denotes the number of literals in D, N denotes the total number of literals in the clauses $C_1, \ldots, C_n$ and $q(P)$ is some polynomial function in the total length P of an input problem instance.*

Proof: We assume that an efficient unification algorithm is used where no exponential blow up can occur (cf. [2], chapter 3.3. "Efficient algorithms for $\emptyset$-unification"). Then the problem transformation from theorem 5.1 has to consider $\sum_{i=1}^{n} m^{n_i} \leq m^N$ unification problems, where n_i denotes the number of literals in the clause C_i. Hence, the number of term tuples in the resulting term tuple cover problem is restricted by m^N and their total size is restricted by $m^N * p'(P)$, where $p'(P)$ denotes some polynomial function in the total length P of an input problem instance. By theorem 5.4 above, this term tuple cover problem can be solved in time $O(c^{m^N} * p(m^N * p'(P))$. By appropriately choosing some constant $d > c$ and a polynomial q, this bound can be transformed into $O(d^{m^N} * q(P))$. ◇

Note that our H-subsumption algorithm via the term tuple cover problem has two sources of non-polynomial complexity, namely the number of combinations of literals from D and the number of term tuples in the resulting term tuple

cover problem. By the Π_2^p-hardness of H-subsumption this was somehow to be expected. In fact, the previously known algorithms, which are briefly sketched below, also have two sources of non-polynomial complexity.

In [4], H-subsumption is reduced to an equational problem whose satisfiability is then tested by using the algorithm from [3]. Hence, similar to our algorithm, there are two sources of non-polynomial time complexity, namely: the number of combinations of literals and the exponential complexity of the equational problem solving. However, as has already been discussed in [12], the complexity of the algorithm from [3] is exponential in the whole length of an equational problem. Furthermore, the size of the resulting equational problem in the H-subsumption algorithm of Caferra et al. is determined by the number of combinations of literals from D and also by the number of symbols in the input clauses. Therefore, in contrast to our algorithm, the total size (and, hence, in particular the term depth of the expressions involved) of the original H-subsumption problem has non-polynomial influence on the overall complexity. As far as space complexity is concerned, both our algorithm and the one of Caferra et al. behave in a similar way: The intermediate problem (i.e.: the term tuple cover problem or the equational problem, respectively) has non-polynomial worst case size due to the number of combinations of literals from D.

In [6], partial saturation is applied to D in order to produce an equivalent set $\mathcal{D}$ of clauses where the minimum depth of variable occurrences is greater than the term depth of the clauses in $\mathcal{C} = \{C_1, \ldots, C_n\}$. The resulting H-subsumption problem is then equivalent to ordinary subsumption, i.e.: $\mathcal{C} \leq_{ss}^{H} D \Leftrightarrow \mathcal{C} \leq_{ss}^{H} \mathcal{D} \Leftrightarrow \mathcal{C} \leq_{ss} \mathcal{D}$. Again this algorithm has two sources of non-polynomial time complexity, namely the saturation and the ordinary subsumption. However, the order of these complexity sources is reversed w.r.t. to our algorithms, since the saturation step in some sense corresponds to the term tuple cover problem and the complexity due to the possible combinations of literals from D is hidden in the ordinary subsumption. Hence, the non-polynomial space complexity is determined by the saturation rather than by the possible combinations of literals of D. Our algorithm is rather difficult to compare with the algorithm from [6] since, in general, it is not clear which source of non-polynomial complexity is the most important one. Nevertheless, it should be noted that the existence of a single clause C_i with a particularly big term depth has an enormous influence on the size of the resulting ordinary subsumption problem, i.e.: In the worst case, the clause set $\mathcal{D}$ consists of exponentially many clauses of exponential size. Hence, in contrast to our approach, the size of the expressions involved has non-polynomial influence on the worst case (time and space) complexity when the method from [6] is used.

6 Conclusions and Future Work

It has been shown that H-subsumption is compatible with the most common (resolution- and paramodulation-based) inference systems, although it is a strictly stronger redundancy criterion than ordinary subsumption. The lack of ef-

ficient algorithms for H-subsumption has been explained by proving its Π_2^p-completeness even in a very restricted case. Finally a new H-subsumption algorithm has been presented, whose principal source of complexity – in contrast to previously known algorithms – is the number of atoms rather than the size of the involved expressions.

From the theoretical point of view, the Π_2^p-membership of H-subsumption in case of an infinite Herbrand universe has remained an open question. More practically, a fully satisfactory H-subsumption algorithm has not been found yet. In particular, as was mentioned in Section 5, the algorithm from [6] is preferable to ours in certain situations. A combination of these two algorithms might therefore serve as a good starting point for future efforts to construct a more efficient algorithm.

References

1. L.Bachmair, H.Ganzinger: Rewrite-based Equational Theorem Proving with Selection and Simplification, Journal of Logic and Computation, Vol 4 No 3, pp. 217-247 (1994). 357
2. F.Baader, J.H.Siekmann: Unification Theory, in Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press (1994). 367
3. H. Comon, P. Lescanne: Equational Problems and Disunification, Journal of Symbolic Computation, Vol 7, pp. 371-425 (1989). 368, 368
4. R.Caferra, N.Peltier: Decision Procedures using Model Building Techniques, Proceedings of CSL'95, LNCS 1092, pp.130-144, Springer (1996). 355, 355, 356, 356, 368
5. R.Caferra, N.Zabel: Extending Resolution for Model Construction, Proceedings of JELIA '90, LNAI 478, pp. 153-169, Springer (1991). 355, 355, 356, 356
6. C.Fermüller, A.Leitsch: Hyperresolution and Automated Model Building, Journal of Logic and Computation, Vol 6 No 2, pp.173-230 (1996). 355, 355, 355, 356, 356, 363, 363, 368, 368, 368, 369
7. M.R.Garey, D.S.Johnson: Computers and Intractability, A guide to the Theory of NP-Completeness, W.H. Freeman and Company (1979). 355, 360
8. J.Hsiang, M. Rusinowitch: Proving Refutational Completeness of Theorem-Proving Strategies: The Transfinite Semantic Tree Method, Journal of the ACM, Vol 38 No 3, pp. 559 - 587 (1991). 357, 358, 358
9. R.Kowalski, P.J.Hayes: Semantic Trees in Automated Theorem Proving, in Machine Intelligence 4, pp. 87-101 (1969). 357, 357, 358
10. A.Leitsch: The Resolution Calculus, Springer (Texts in Theoretical Computer Science), Berlin Heidelberg New York (1997). 357
11. R.Nieuwenhuis, A.Rubio: Theorem Proving with ordering and equality constrained clauses, Journal of Symbolic Computation, Vol 11, pp. 1-32 (1995). 357
12. R.Pichler: Algorithms on Atomic Representations of Herbrand Models, in Proceedings of JELIA '98, LNAI 1489, pp. 199-215, Springer (1998). 363, 363, 363, 363, 365, 365, 365, 366, 366, 366, 367, 368, 370, 370, 371

13. R.Pichler: Completeness and Redundancy in Constrained Clause Logic, in Proceedings of FTP'98 (Int. Workshop on First-Order Theorem Proving), Technical Report E1852-GS-981 of Technische Universität Wien, pp. 193–203, available from `http://www.logic.at/ftp98`, Vienna (1998). 357, 357
14. L.J. Stockmeyer: The Polynomial Time Hierarchy, in Journal of Theoretical Computer Science, Vol 3, pp.1-12 (1976). 359, 359

Appendix

A The H-subsumption Algorithm at Work

In Section 5, a new algorithm for testing H-subsumption was provided on the basis of the term tuple cover algorithm from [12]. The following example will help to illustrate this algorithm.

Example A.1. (Testing H-subsumption via a transformation into term tuples) Let H be the Herbrand universe with signature $\Sigma = \{a, f, g\}$ and let $(\mathcal{C} = \{C_1, C_2, C_3, C_4\}, D)$ denote an instance of the H-subsumption problem over H, where the clauses C_i and D are defined as follows:

$$\begin{array}{lll} D &= P(a, x_1) \vee P(f(x_2), x_3) \vee P(x_4, a) & \\ C_1 &= P(a, f^3(y_1)) \vee P(f(y_2), a) & C_3 = P(f^2(y_1), f(y_2)) \\ C_2 &= P(y_1, y_2) \vee P(f(y_2), g(y_3)) & C_4 = P(f(a), y_1) \end{array}$$

Let $V(D) = \{x_1, x_2, x_3, x_4\}$ denote the set of variables in D. Then the problem transformation from theorem 5.1 yields the set $\Theta = \{\vartheta'_1|_{V(D)}, \ldots, \vartheta'_7|_{V(D)}\}$ of mgu's, where the substitutions ϑ'_1 through ϑ'_7 are defined as follows:

$\vartheta'_1 = mgu(C_1, D_{(1,2)}) = mgu(P(a, f^3(y_1)) \vee P(f(y_2), a), P(a, x_1) \vee$
$P(f(x_2), x_3)) = \{x_1 \leftarrow f^3(y_1), x_3 \leftarrow a, y_2 \leftarrow x_2\}$
$\vartheta'_2 = mgu(C_1, D_{(1,3)}) = mgu(P(a, f^3(y_1)) \vee P(f(y_2), a), P(a, x_1) \vee P(x_4, a)) =$
$= \{x_1 \leftarrow f^3(y_1), x_4 \leftarrow f(y_2)\}$
$\vartheta'_3 = mgu(C_2, D_{(1,2)}) = mgu(P(y_1, y_2) \vee P(f(y_2), g(y_3)), P(a, x_1) \vee$
$P(f(x_2), x_3)) = \{x_1 \leftarrow x_2, x_3 \leftarrow g(y_3), y_1 \leftarrow a, y_2 \leftarrow x_2)\}$
$\vartheta'_4 = mgu(C_2, D_{(2,2)}) = mgu(P(y_1, y_2) \vee P(f(y_2), g(y_3)), P(f(x_2), x_3) \vee$
$P(f(x_2), x_3)) = \{x_2 \leftarrow g(y_3), x_3 \leftarrow g(y_3), y_1 \leftarrow f(g(y_3)), y_2 \leftarrow g(y_3)\}$
$\vartheta'_5 = mgu(C_3, D_{(2)}) = mgu(P(f^2(y_1), f(y_2)), P(f(x_2), x_3)) =$
$= \{x_2 \leftarrow f(y_1), x_3 \leftarrow f(y_2)\}$
$\vartheta'_6 = mgu(C_4, D_{(2)}) = mgu(P(f(a), y_1), P(f(x_2), x_3)) = \{x_2 \leftarrow a, y_1 \leftarrow x_3\}$
$\vartheta'_7 = mgu(C_4, D_{(3)}) = mgu(P(f(a), y_1), P(x_4, a)) = \{x_4 \leftarrow f(a), y_1 \leftarrow a\}$

The term tuple cover procedure from [12] is then called with the following set of term tuples:

$$M = \{\big(f^3(y_1), x_2, a, x_4\big),\ \big(f^3(y_1), x_2, x_3, f(y_2)\big),\ \big(x_2, x_2, g(y_3), x_4\big),\ \big(x_1, g(y_3), g(y_3), x_4\big),\ \big(x_1, f(y_1), f(y_2), x_4\big),\ \big(x_1, a, x_3, x_4\big),\ \big(x_1, x_2, x_3, f(a)\big)\}$$

The term tuple sets and the actions carried out by the recursive calls of the term tuple cover procedure from [12] are given below:

original problem:
$M = \{(f^3(y_1), x_2, a, x_4), (f^3(y_1), x_2, x_3, f(y_2)), (x_2, x_2, g(y_3), x_4),$
$(x_1, g(y_3), g(y_3), x_4), (x_1, f(y_1), f(y_2), x_4), (x_1, a, x_3, x_4), (x_1, x_2, x_3, f(a))\}$

The first 2 tuples may be deleted from M by the redundancy criterion from theorem 5.3: $M' = \{(x_2, x_2, g(y_3), x_4), (x_1, g(y_3), g(y_3), x_4),$
$(x_1, f(y_1), f(y_2), x_4), (x_1, a, x_3, x_4), (x_1, x_2, x_3, f(a))\}$.

The first tuple may be deleted from M by the redundancy criterion from theorem 5.2: $M'' = \{(x_1, g(y_3), g(y_3), x_4), (x_1, f(y_1), f(y_2), x_4),$
$(x_1, a, x_3, x_4), (x_1, x_2, x_3, f(a))\}$.

Now the first column (consisting only of variables with a single occurrence) may be ignored: $M''' = \{(g(y_3), g(y_3), x_4), (f(y_1), f(y_2), x_4),$
$(a, x_3, x_4), (x_2, x_3, f(a))\}$.

splitting into subproblems: The term tuple cover problem
$M_a = \{(x_3, x_4), (x_3, f(a))\}$ corresponds to the condition that
$\bar{M}_a = \{(a, x_3, x_4), (a, x_3, f(a))\}$ covers $G_H(a) \times H^2$.

Analogously, the following term tuple cover problems correspond to the requirement of covering $G_H(f(x)) \times H^2$ and $G_H(g(x)) \times H^2$, resp.:
$M_f = \{(y_1, f(y_2), x_4), (y_1, x_3, f(a))\}$.
$M_g = \{(y_3, g(y_3), x_4), (y_3, x_3, f(a))\}$.
(Note that the last tuple from M''' is contained in every subproblem.)

subproblem M_a: $(x_3, x_4) \in M_a$ already covers all of H^2.
Hence, in particular, M_a covers H^2.

subproblem M_f: The first column constains only variables with a single occurrence and may, therefore, be ignored:
$M'_f = \{(f(y_2), x_4), (x_3, f(a))\}$.
Now the first tuple may be deleted by theorem 5.3: $M''_f = \{(x_3, f(a))\}$.
The first column may be ignored and theorem 5.3 may be applied to the remaining component: $M'''_f = \emptyset$.
By theorem 5.3, the term tuple sets M_f and M'''_f are equivalent.
Hence, the subproblem M_f is a negative instance of the term tuple cover problem. But then the original set of term tuples M does not cover H^4 either.

By the transformation in theorem 5.1, the original H-subsumption problem and the term tuple cover problem given through M are equivalent. Hence, $\mathcal{C}$ does not H-subsume D.

Complexity Classes and Rewrite Systems with Polynomial Interpretation

G. Bonfante, A. Cichon, J.Y Marion and H. Touzet

Loria, Calligramme project,
B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France
{bonfante,cichon,marionjy,touzet}@loria.fr

Abstract. We are concerned with functions over words which are computable by means of a rewrite system admitting polynomial interpretation termination proofs. We classify them according to the interpretations of successor symbols. This leads to the definition of three classes, which turn out to be exactly the poly-time, linear exponential-time and doubly linear exponential time computable functions. As a consequence, we also characterize the linear space computable functions.

1 Introduction

We are interested in studying the relationship between termination orderings for rewrite systems and feasible computation. One might suspect that polynomial interpretations, introduced in [9], would be a good candidate for the investigation of small complexity classes of functions. Various implementations have been carried out [2,6]. However, it has been shown in [8] that a rewrite system with a polynomial interpretation termination proof can admit doubly-exponential derivation lengths.

The work of [3] initiated an alternative analysis of the effects of termination proofs based on the use of polynomial interpretations. It was shown that a particularly important aspect was the interpretations of the constructors which were restricted to the set $\mathbb{N} = \{0, s\}$, 0 a constant and s a unary successor symbol.

The present paper reconsiders the situation. Firstly, we study functions over strings, that is, rewrite systems based on several successors. Indeed words are the canonical domains for computations. Secondly, we provide a stratification of the systems which is related to the kind of polynomials interpreting the successors.

We show that PTIME functions are characterized by rewrite systems admitting polynomial interpretation termination proofs where the successors are interpreted by *linear* polynomials with leading coefficient 1. As a consequence, functions over $\mathbb{N}$ turn out to be precisely LINSPACE functions.

Within the same framework, linear exponential time functions, i.e. functions in $\text{ETIME} = \text{DTIME}(2^{0(n)})$, are characterized by rewrite systems admitting polynomial interpretation termination proofs where the successors are interpreted by *linear* polynomials.

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 372–384, 1999.

The linear doubly-exponential time functions, i.e. in $\mathrm{E}_2\textsc{time} = \textsc{Dtime}(2^{2^{0(n)}})$, are characterized by rewrite systems admitting polynomial interpretation termination proofs where the successors are interpreted by *non-linear* polynomials.

Machine independant characterizations of complexity classes were originated by Cobham [4]. His approach is by mean of 'bounded recursion on notation' in which rates of growth of functions are limited by functions already defined in the class. In contrast, in our work, polynomial interpretations impose an external condition on rewrite rules. Therefore, the different kinds of polynomial interpretations of successors are somewhat akin to the notion of data tiering recently introduced in [1,10]. It is also worth mentioning the characterization of Ptime functions over finite models in [7], because both consider basically the same system: the Herbrand-Gödel equations.

This paper is organized as follows. Section 2 defines functions computed by rewrite systems with polynomial interpretation termination proofs. Then, the main results with their consequences are presented in Theorem 1. In Section 3, we establish the characterization of Ptime. In Section 4, we examine Linspace. The last two sections are devoted to exponential classes and the proofs use results of the previous sections.

2 Computability & polynomial interpretation

The term rewriting notations used throughout are based on [5]. In particular, let $\mathcal{F}$ be a finite set of symbols of fixed arity and $\mathcal{V}$ a denumerable set of variables. $\mathcal{T}(\mathcal{F},\mathcal{V})$ is the term algebra built up from $\mathcal{F}$ and $\mathcal{V}$ and $\mathcal{T}(\mathcal{F})$ is the set of ground terms. The relation $\xrightarrow{+}$ ($\xrightarrow{*}$) denotes the transitive (reflexive-transitive) closure of $\rightarrow$. If u and v are two terms of $\mathcal{T}(\mathcal{F},\mathcal{V})$, we write $u \xrightarrow{!} v$ to mean that $u \xrightarrow{*} v$ and v is in normal form.

2.1 Functions definable by rewrite systems

We shall concentrate on functions over words which are computed by rewrite systems. For this, we specify a set of constructors $\mathbb{W}$ and a disjoint set of defined symbols $\mathbb{F}$. The set of constructors $\mathbb{W}$ contains at least one constant symbol. All other constructors are unary and are called *successors*.

The set of rewrite rules $\mathcal{R}$ over the term algebra $\mathcal{T}(\mathbb{F}\cup\mathbb{W},\mathcal{V})$ defines each function symbol of $\mathbb{F}$. We always assume that $\mathcal{R}$ is terminating (strongly normalizable) and also that $\mathcal{R}$ is confluent and determines for each term a unique normal form in the constructor set $\mathcal{T}(\mathbb{W})$. Indeed we shall in fact be dealing mostly with orthogonal systems which are therefore confluent and which provide the standard presentation of equational definitions of functions. Nevertheless, orthogonality is not a crucial property whereas confluence is.

It is convenient to present a rewrite system as a tuple $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f\rangle$, where

- $\mathbb{W}$ is the set of constructors;
- $\mathbb{F}$ is the set of defined function symbols;

- $\mathcal{R}$ is the set of rewrite rules;
- f is the main function symbol of $\mathbb{F}$;

Given an alphabet Σ and a constructor set $\mathbb{W}$, define an *encoding function* α as an injective morphism of $\Sigma \to \mathbb{W}$ which is extended canonically to Σ^* in the usual manner :

$$\begin{array}{rl} \alpha(\varepsilon) = \epsilon & (\varepsilon \text{ is the empty word of } \Sigma^*) \\ \alpha(iu) = \alpha(i)(\alpha(u)) & (i \in \Sigma). \end{array}$$

A function $\phi : (\Sigma^*)^n \mapsto \Sigma^*$ is said to be *computed* by the rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f \rangle$ if there is a pair (α, β) of encoding functions such that, for all $w_1, \ldots, w_n, v \in \Sigma^*$

$$f(\alpha(w_1), \ldots, \alpha(w_n)) \xrightarrow{!} \beta(v) \Leftrightarrow \phi(w_1, \ldots, w_n) = v$$

The pair (α, β) allows us to define semantics for a given system. α will serve to define the meaning of an input and β the meaning of an output. This means that we shall be able to use different sets of constructors for inputs and outputs.

Example 1. The rewrite system $\langle \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{Sq}\}, \mathbb{N}, \mathtt{Sq} \rangle$ where $\mathbb{N}$={0,s} defined the tally square function.

$$\mathcal{A} \begin{cases} \mathtt{Add}(0, y) \to y \\ \mathtt{Add}(sx, y) \to s(\mathtt{Add}(x, y)) \\ \mathtt{Mul}(0, y) \to 0 \\ \mathtt{Mul}(sx, y) \to \mathtt{Add}(y, \mathtt{Mul}(x, y)) \\ \mathtt{Sq}(x) \to \mathtt{Mul}(x, x) \end{cases}$$

Each number n is represented in unary, say over $\{|\}^*$, by $\underline{n} = |^n$. Then, $\mathtt{Sq}$ is interpreted by one encoding function α : $\alpha(\epsilon) = 0$ and $\alpha(\underline{n+1}) = s(\alpha(\underline{n}))$. Hence, $\mathtt{Sq}(\alpha(\underline{n})) \xrightarrow{!} \alpha(\underline{n^2})$

2.2 Polynomial interpretation

A *polynomial interpretation termination proof* for a rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f \rangle$ consists in the assignment to each function symbol g of $\mathbb{F} \cup \mathbb{W}$, a polynomial $[g]$ with non-negative integer coefficients which satisfies the following conditions:

- If the arity of g is k then $[g]$ is a polynomial with k variables.
- $[g]$ is monotonic, that is,

$$n < m \Rightarrow [g](\cdots, n, \cdots) < [g](\cdots, m, \cdots)$$

- If the arity of g is 0 then $[g] > 0$, otherwise $[g](n_1, \cdots, n_k) > n_i$, for $i = 1, n$ and for all $n_j > 0$, $j = 1, k$. This last condition is necessary for Lemma 2.

$[\,]$ is extended canonically to terms as follows

$$[g(t_1, \ldots, t_n)] = [g]([t_1], \ldots, [t_n]).$$

Lastly, $[\,]$ must ensure that for all rules $l \to r$ of $\mathcal{R}$

$$[l] > [r]$$

for all values of variables greater than or equal to the minimum of the interpretations of the constants.

Example 2. The rewrite system of example 1 admits the following interpretation

$$\begin{aligned} [0] &= 2 \\ [s](x) &= x + 1 \\ [\mathtt{Add}](x, y) &= 2x + y, \\ [\mathtt{Mul}](x, y) &= 3xy \\ [\mathtt{Sq}](x) &= 3x^2 + 1 \end{aligned}$$

From now on, a rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f \rangle$ with a polynomial interpretation $[\,]$ is denoted by $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] \rangle$.

We end this section by giving some general properties of such systems which will be used later. Firstly, whenever $u \xrightarrow{+} v$, $[v] < [u]$. This observation implies that the length of any derivation starting from a term t is bounded by $[t]$.

Lemma 1. *A rewrite system with a polynomial interpretation is terminating.*

Define the *size* $|t|$ of a term t as follows:

$$|t| = \begin{cases} 1 & \text{if } t \text{ is a constant or a variable} \\ \sum_{i=1}^{n} |t_i| + 1 & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Lemma 2. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] \rangle$ be a rewrite system. For all terms t,*

$$|t| \leq [t].$$

Proof. The proof goes by induction on $|t|$. The result is obvious if t is a constant because $|t| = 1$ and also $[t] > 0$ by assumption on $[\,]$. Otherwise, t is of the form $f(t_1, \ldots, t_n)$ and so

$$\begin{aligned} |f(t_1, \ldots, t_n)| &= 1 + \textstyle\sum_{i=1,n} |t_i| \text{ (by size def.)} \\ &\leq 1 + \textstyle\sum_{i=1,n} [t_i] \text{ (by ind. assumption)} \end{aligned}$$

Since $[f]$ is monotonic, for all $n > 0$, $[f](\cdots, n+1, \cdots) \geq [f](\cdots, n, \cdots) + 1$. Also, we have $[f](1, \cdots, 1) > 1$ because $[f(x_1, \cdots, x_n)] > x_i$. Therefore,

$$\textstyle\sum_{i=1,n} [t_i] < [f(t_1, \cdots, t_n)]$$

And so

$$\begin{aligned} |f(t_1, \ldots, t_n)| &\leq 1 + \textstyle\sum_{i=1,n} [t_i] \\ &\leq [f(t_1, \cdots, t_n)] \end{aligned}$$

2.3 Classes of functions and results

We shall now examine the intent of the successor interpretations in computations, as initiated by [8,3]. These interpretations will play a central role throughout as the following example shows.

Example 3. On the constructor set $\mathbb{N}_1 = \{0, s, q\}$, we define the factorial function `Fact` by

$$\begin{cases} \texttt{Fact}(0) \to s(0) \\ \texttt{Tr}(q(x)) \to s(\texttt{Tr}(x)) \\ \texttt{Tr}(0) \to 0 \\ \texttt{Fact}(q(x)) \to \texttt{Mul}(s(\texttt{Tr}(x)), \texttt{Fact}(x)) \end{cases}$$

The rules for `Mul` are provided in example 1. Clearly, $\texttt{Fact}(q^n(0)) \xrightarrow{+} s^{n!}(0)$, but we have two representations for natural numbers. Inputs are encoded with q whereas outputs with s. To explictly say what the function computed by `fact` is, we provide an encoding pair (α, β). As before, over $\{|\}^*$, we set $\alpha(|) = q$ and $\beta(|) = s$. Hence, `Fact` represents the factorial function since $\texttt{Fact}(\alpha(\underline{n})) \xrightarrow{!} \beta(\underline{n!})$.

Now, to give a polynomial interpretation of `Fact`, we must interpret both successors s and q in a different way, say by $[s](x) = x+1$ and $[q](x) = 3(x+2)^2$. It is then routine to check that the following interpretation works for `Tr` and `Fact`:

$$[\texttt{Tr}](x) = x + 1$$
$$[\texttt{Fact}](x) = x + 2$$

`Tr` is a coercion function that translates data of some kind to data of a lower kind.

In fact, successor interpretations fall into three categories

kind 0: polynomials of the form $P(X) = X + c$ $(c > 0)$,
kind 1: polynomials of the form $P(X) = aX + b$ $(a > 1, b \geq 0)$
kind 2: polynomials of the form $P(X) = aX^d + R(X)$ $(a > 0,\ d > 1$ and R is a polynomial of degree strictly less than $d)$

This classification allows us to distinguish three classes of rewrite systems.

Definition 1. *Let $i \in \{0, 1, 2\}$. A rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] \rangle$ is $\Pi(i)$ if, for each successor s of $\mathbb{W}$, the interpretation $[s]$ is a polynomial of kind less than or equal to i. A function ϕ is* $\Pi(i)$-computable *if there is a $\Pi(i)$ rewrite system which computes ϕ.*

For instance, example 2 shows that addition and multiplication are $\Pi(0)$-computable functions and example 3 that the factorial function is $\Pi(2)$-computable. We now state our main result.

Theorem 1.

1. *The $\Pi(0)$-computable functions are exactly the* Ptime *functions.*

2. *The $\Pi(1)$-computable functions are exactly the* ETIME *functions, that is, the functions computable in time $2^{O(n)}$.*
3. *The $\Pi(2)$-computable functions are exactly the* E_2TIME *functions, that is, the functions computable in time $2^{2^{O(n)}}$.*

Proof. The proof goes as follows. Lemma 6 shows how to simulate $\Pi(0)$-computable functions in PTIME. The converse is established in Lemma 9.

The characterization of ETIME is a consequence of Lemma 13 and Lemma 11. Lastly, E_2TIME is obtained by applying Lemmas 17 and 15

A consequence of the above result on PTIME is a characterization of LINSPACE in Section 4.

Now say that a language L is recognized by a rewrite system if the characteristic function of L is computable by this rewrite system. It is obvious that languages recognized in time $2^{O(n)}$ are $\Pi(1)$-computable. Actually, every language recognized in time $2^{O(n^k)}$ is $\Pi(1)$-computable, *with respect to a polynomial time reduction.* For this, we use a standard padding argument. Indeed, let L be a DTIME(2^{n^k}) language. Suppose that L is *accepted* by the TM M. Then, construct $L' = \{x@0^{|x|^k} \text{ with } x \in L\}$, that is, each word of L' is a word of L padded by extra 0's. Thus, L' is recognized in time $2^{O(n)}$ using M. Then, the characteristic function of L' is $\Pi(1)$-computable.

2.4 General properties

For use as Lemmas, we now establish some properties of $\Pi(i)$-systems.

Firstly, we show a property of weak closure under composition of $\Pi(i)$-computable functions, enabling us to combine systems to define functions in a modular way.

Lemma 3. *Let ϕ_i be a $\Pi(i)$-computable function and ϕ be a $\Pi(0)$-computable function. The function ψ defined by $\psi(\boldsymbol{x}, \boldsymbol{y}) = \phi(\phi_i(\boldsymbol{x}), \boldsymbol{y})$ is $\Pi(i)$-computable.*

Proof. Assume that ϕ is computed by the $\Pi(0)$-rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] \rangle$ with the pair (α, β) of encoding functions. Assume that ϕ_i is computed by the $\Pi(i)$-rewrite system $\langle \mathcal{R}_i, \mathbb{F}_i, \mathbb{W}_i, f_i, [\,]_i \rangle$ with the encoding (α_i, β_i). Without loss of generality, we also suppose that both $\mathbb{F}$ and $\mathbb{F}_i$ are disjoint and also that $\mathbb{W}$ and $\mathbb{W}_i$ are two disjoint copies of the same set of constructors. This means that there is a one-one mapping τ_{α_i} from $\mathbb{W}_i$ to $\mathbb{W}$ verifying $\alpha = \tau_{\alpha_i} \circ \alpha_i$. The key point of the construction is the introduction of a coercion function $\mathtt{Tr}$ that translates terms of $\mathbb{W}_i$ into terms of $\mathbb{W}$. $\mathtt{Tr}_{\alpha_i}$ is defined thus :

$$\begin{array}{rl} \mathtt{Tr}_{\alpha_i}(\epsilon) \to \tau(\epsilon) & \epsilon \in \mathbb{W}_i \text{ is 0-ary} \\ \mathtt{Tr}_{\alpha_i}(s(x)) \to \tau(s)\mathtt{Tr}_{\alpha_i}(x) & s \in \mathbb{W}_i \text{ is a successor} \end{array}$$

There is also a function τ_{β_i} from $\mathbb{W}_i$ to $\mathbb{W}$ verifying $\alpha = \tau_{\beta_i} \circ \beta_i$, and so we define $\mathtt{Tr}_{\beta_i}$ similarly.

The function ψ is computed by $\langle \mathcal{R}_*, \mathbb{F} \cup \mathbb{F}_i, \mathbb{W} \cup \mathbb{W}_i, f_*, [\,]_* \rangle$ where the set of rules $\mathcal{R}_*$ just contains the set $\mathcal{R} \cup \mathcal{R}_i$, the above rules for $\mathtt{Tr}$ and the following rule for f:

$$f_*(\boldsymbol{x}, y_1, \ldots, y_n) \to f(\mathtt{Tr}_{\beta_i}(f_i(\boldsymbol{x})), \mathtt{Tr}_{\alpha_i}(y_1), \ldots, \mathtt{Tr}_{\alpha_i}(y_n))$$

$\mathcal{R}_*$ is a confluent rewrite system, since the sets of defined symbols $\mathbb{F}$ and $\mathbb{F}_i$ are disjoint and f_* is a new symbol. The encoding pair interpreting f_* is (α_i, β).

The polynomial interpretation $[\,]_*$ is an extension of $[\,]$ and $[\,]_i$.

$$\begin{aligned}
[X]_* &= [X], \text{ if } X \in \mathbb{F} \cup \mathbb{W} \\
[X]_* &= [X]_i, \text{ if } X \in \mathbb{F}_i \cup \mathbb{W}_i \\
[f_*]_*(\boldsymbol{x}, y_1, \ldots, y_n) &= [f]_*([\mathtt{Tr}]_*[f_i]_*(\boldsymbol{x}), [\mathtt{Tr}]_*(y_1), \cdots, [\mathtt{Tr}]_*(y_n)) + 1 \\
[\mathtt{Tr}_{\alpha_i}]_*(x) &= ax \\
[\mathtt{Tr}_{\beta_i}]_*(x) &= ax
\end{aligned}$$

where $a = \max\{c;\ [\epsilon] = c \text{ or } [s](x) = x + c \text{ where } \epsilon, s \in \mathbb{W}\} + 1$.

The next Lemma is convenient for determining upper bounds on the runtime of $\Pi(i)$-computable functions. The proposition asserts that both (1) the length of the derivation and (2) the size of any terms involved during the reduction process are polynomially bounded in the interpretation of the inputs.

Definition 2. *A class $\mathcal{C}$ of unary increasing functions over natural numbers* accommodates polynomials *iff for all $\phi \in \mathcal{C}$, for all polynomials P with natural number coefficients, there is a function $\phi' \in \mathcal{C}$ such that $P(\phi(x)) \leq \phi'(x)$, for all $x > 0$.*

In the sequel, we shall deal with three main classes of functions: polynomials, exponentials $\{2^{cx}\ ;\ c > 0\}$ and doubly exponentials $\{2^{2^{cx}}\ ;\ c > 0\}$. It is clear that these classes accommodate polynomials.

Lemma 4. *Let $\mathcal{C}$ be a class of functions which accommodates polynomials. Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,] \rangle$ be a rewrite system. Assume that there is $\phi \in \mathcal{C}$ such that, for all terms t in $\mathcal{T}(\mathbb{W})$, $[t] \leq \phi(|t|)$. Then there is $\phi' \in \mathcal{C}$, such that for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$ and $f \in \mathbb{F}$, the following holds:*

(i) $[f(t_1, \cdots, t_n)] \leq \phi'(\max\{|t_i|\ ;\ 1 \leq i \leq n\})$.
(ii) *The length of any derivation starting from $f(t_1, \cdots, t_n)$ is bounded by* $\phi'(\max\{|t_i|\ ;\ 1 \leq i \leq n\})$.
(iii) *If $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq \phi'(\max\{|t_i|\ ;\ 1 \leq i \leq n\})$.*

Proof. Let $m = \max\{|t_i|, 1 \leq i \leq n\}$. By hypothesis, we have

$$\begin{aligned}
[f(t_1, \ldots, t_n)] &\leq [f](\phi(m), \ldots, \phi(m)) \\
&\leq \phi'(m) \text{ for some } \phi' \text{ in } \mathcal{C}
\end{aligned}$$

So (i) is proved. (ii) is a consequence of (i) since the length of any derivation is bounded by the polynomial interpretation of the term reduced. Finally, Lemma 2 implies $|v| \leq [v]$. And since $[v] \leq [f(t_1, \cdots, t_n)]$, we complete (iii) by applying (i) again.

3 PTIME is $\Pi(0)$

3.1 $\Pi(0)$-computable functions are PTIME

Lemma 5. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\ ] \rangle$ be a $\Pi(0)$ rewrite system. Then there is a constant c such that for all $t \in \mathcal{T}(\mathbb{W})$, we have $[t] \leq c.|t|$.*

Proof. By construction of $\Pi(0)$, for all constructors in $\mathbb{W}$, there exists $c > 0$ such that $[\epsilon] \leq c$ and $[s](x) \leq x + c$. So, for all $t \in \mathcal{T}(\mathbb{W})$, we have, by composition of interpretations, $[t] \leq c.|t|$.

Corollary 1. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\ ] \rangle$ be a $\Pi(0)$ rewrite system. Then there is a polynomial P_f such that for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,*

1. *the length of any derivation starting from $f(t_1, \ldots, t_n)$ is bounded by $P_f(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$,*
2. *if $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq P_f(\max\{|t_i| \ ; \ 1 \leq i \leq n\})$.*

Proof. (1) is a consequence of Lemma 4-(ii) and Lemma 5. (2) is a consequence of Lemma 4-(iii) and Lemma 5.

Lemma 6. *If ϕ is $\Pi(0)$-computable, then ϕ is in* PTIME.

Proof. Let Σ be the alphabet for ϕ. Suppose that ϕ is computed by the system $\langle \mathcal{R}, \mathbb{W}, \mathbb{F}, f, [\ ] \rangle$ in $\Pi(0)$ and let $w_1, \ldots, w_n$ be words in Σ^*. Since all derivations starting from $f(\alpha(w_1), \ldots, \alpha(w_n))$ leads to the same normal form, we take any strategy to compute the normal form of $f(\alpha(w_1), \ldots, \alpha(w_n))$. The key point is that at any step the size of terms, which we produce by replacing a redex, is always bounded by $O(\max(|w_i|)^p)$ (see (2) of Corollary 1). So, in time bounded by $O(\max(|w_i|)^p)$, we select a redex and replace it by the right hand side of the corresponding rule of $\mathcal{R}$. The reduction lasts for at most $O(\max(|w_i|)^q)$ (see (1) of Corollary 1). It follows that the runtime of the computation of $\phi(w_1, \ldots, w_n)$ is bounded by $O(\max(|w_i|)^{p+q})$.

3.2 PTIME functions are $\Pi(0)$-computable

In this section we show that PTIME functions are $\Pi(0)$-computable by simulating a Turing Machine by a rewrite system with polynomial interpretations. The outline of the proof is as follows. Firstly, we construct a rewrite system in $\Pi(0)$ that simulates T steps of the computation of a Turing machine. This will be done in Lemma 7. This part is independant from PTIME: it does not take advantage of the polynomial bound on the computation runtime. Then we show in Lemma 8 that polynomials can be computed by $\Pi(0)$ rewrite systems. Finally, we conclude by composing both results.

Encoding a time bounded Turing machine It is convenient to consider multi-stack Turing machines, abbreviated STM. (Although two-stack Turing Machines are sufficiently expressive to delineate PTIME, we shall need more than two-stacks in Section 4 to deal with unary computation.) Formally, a k-stack TM, $\mathcal{M}$, is defined by a tuple $\mathcal{M} =< \Sigma, \epsilon, Q, q_0, Q_f, \delta >$ where Σ is the alphabet and ϵ is the bottom stack symbol; Q is the set of states with $q_0 \in Q$ as initial state and $Q_f \subseteq Q$ is the set of final states; and finally the transition function is $\delta : Q \times (\Sigma \cup \{\epsilon\})^k \to Q \times (\Sigma^*)^k$.

The meaning of $\delta(q, a_1, \cdots, a_k) = (q', u_1, \cdots, u_k)$ is the following. The STM is in state q and the letter on the top of the ith stack is a_i. Then, the STM replaces each a_i by the word u_i and switches to the state q'.

A configuration of the machine is a tuple $< q, w_1, \cdots, w_k >$ where $q \in Q$ is the current state, and w_i is the content of the ith stack. Let $\Rightarrow$ be the relation which provides the next configuration of δ. We define for $\mathcal{M}$ a function $F[\mathcal{M}] : (\Sigma^*)^k \mapsto \Sigma^*$ by $F[\mathcal{M}](w_1, \cdots, w_k) = r_k$ iff r_k is the content of the kth stack at the end of the computation which starts with $(q_0, w_1, \cdots, w_k)$.

Lemma 7. *Let $\mathcal{M} =< \Sigma, \epsilon, Q, q_0, Q_f, \delta >$ be a STM. Then there is a $\Pi(0)$-computable function ϕ_M such that, for each input $(w_1, \cdots, w_k)$, if $\mathcal{M}$ halts in less than t steps then $\phi_M(t, w_1, \cdots, w_k) = F[\mathcal{M}](w_1, \cdots, w_k)$.*

Proof. We construct a rewrite system $\mathcal{R}_\delta$ which computes ϕ_M as follows:

- Constructors are $\mathbb{W} = \{s_i \mid i \in \Sigma\} \cup \{\epsilon\}$,
- the defined symbols are $\{\mathtt{q}\ ;\ q \in Q\}$, where, for all $q \in Q$, $\mathtt{q}$ is a function symbol of arity $k+1$. The first parameter corresponds to the remaining computation runtime and the k other parameters to stacks,
- the encoding pair is (α, α) with $\alpha(i) = s_i$ for all $i \in \Sigma$.

Let s be s_0. The rewrite rules are the following :

- If $\delta(q, a_1, \cdots, a_k) = (q', u_1, \cdots, u_k)$ then

$$\mathtt{q}(st, \alpha(a_1)x_1, \cdots, \alpha(a_k)x_k) \to \mathtt{q}'(t, \alpha(u_1)x_1, \cdots, \alpha(u_1)x_k)$$

- Otherwise, $q_f \in Q_f$ and $\mathtt{q}_f(st, x_1, \cdots, x_k) \to x_k$.

It is straightforward to verify that

$$< q, w_1, \cdots, w_k > \Rightarrow < q', w'_1, \cdots, w'_k >$$

if and only if

$$\mathtt{q}(st, \alpha(w_1), \cdots, \alpha(w_k)) \to \mathtt{q}'(t, \alpha(w'_1), \cdots, \alpha(w'_k))$$

and that

$$q \text{ is a final state iff } \mathtt{q}(t, \alpha(w_1), \cdots, \alpha(w_k)) \to \alpha(w_k)$$

Therefore, if $\mathcal{M}$ halts in less than t steps on inputs $w_1, \ldots, w_k$ then the result of the computation is provided by the normal form of $\mathtt{q}_0(s^t(0), \alpha(w_1), \cdots, \alpha(w_k))$, which is exactly $F[\mathcal{M}](w_1, \cdots, w_k)$. So ϕ_M is represented by $\mathtt{q}_0$.

Lastly, we interpret each function symbol by

$$\begin{aligned} [\epsilon] &= 2 \\ [s_i](x) &= x + 1 && \forall i \in \Sigma \\ [\mathtt{q}](t, x_1, \cdots, x_k) &= k.c.t + x_1 + \cdots + x_k && \forall q \in Q \end{aligned}$$

where the constant c is strictly greater than the interpretation of any word that is involved in the definition of δ. More precisely, associate to the ith transition rule $\delta(q, a_1, \cdots, a_k) = (q', u_1, \cdots, u_k)$ the constant $c_i = \max\{[a_j], [u_j]\}$, and set $c = \max\{c_i\} + 1$. We conclude that the system is $\Pi(0)$.

Simulating Poly-time

Lemma 8. *Each polynomial is $\Pi(0)$-computable.*

Proof. Each polynomial is defined by composition of addition and multiplication. Example 1 shows that addition and multiplication are $\Pi(0)$-computable functions. Lemma 3 leads then to the conclusion.

Lemma 9. *If ϕ is in* PTIME*, then ϕ is $\Pi(0)$-computable.*

Proof. Let ϕ be a function, computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by a polynomial P. The function ϕ is obtained by composing ϕ_M, as defined in Lemma 7, and P. Since ϕ_M and P are both $\Pi(0)$-computable, ϕ is also $\Pi(0)$-computable by lemma 3.

4 A characterization of LINSPACE

A function is in LINSPACE if it is computed by a multi-stack Turing machine over an alphabet with at least two letters running in linear space. Actually, there is an alternative characterization of LINSPACE due to [7]. This result will allow us to give a new characterization of LINSPACE with functions over $\{0, s\}$.

Theorem 2. *A function ϕ is computable over a multi-stack Turing machine over a unary alphabet in polynomial-time iff ϕ is in* LINSPACE*.*

Proof. The proof follows essentially [10]. Let ϕ be an m-ary function. Assume that ϕ is computed by a k-stack Turing machine M which works on the unary alphabet $\{|\}$ and whose runtime is bounded by $P(w_1, \cdots, w_m)$ for some polynomial P and for all inputs w_1,...,w_m. From M, we construct a $(k+1)$-stack Turing machine N over the binary alphabet $\{0, 1\}$. The stack i, $i = 1, k$, of N contains the binary representation of the number of |'s in the ith stack of M. Now, observe that M's operations just consist in adding or removing some fixed amount of |'s. So when, M adds c |'s to some stack, for example, N will also add c to the same stack in base two. But, this is easily performed by N with the spare stack. Thus, the size of each stack of N is bounded by $O(\log(\max_{i=1,m}(w_i)))$.

Conversely, assume that ϕ is computed by a k-stack Turing machine M over, say, $\{a,b\}^*$. Define $\underline{u}$ to be the dyadic representation of the word $u \in \{a,b\}^*$ (that is, $\underline{\epsilon} = 0$, $\underline{au} = 2.\underline{u} + 1$ and $\underline{bu} = 2.\underline{u} + 2$). We build a $(k+2)$-stack Turing machine N over the unary alphabet $\{|\}$ as follows. If there is u in the stack i of M then there are $\underline{u}$ |'s in the stack i of N. Say that M pushes a onto a stack, N doubles the number of |'s in this stack and then adds |. To multiply by two, N uses the two extra stacks to duplicate the stack. N proceeds similarly to push or pop a word given by the transition function of M. The runtime of M is bounded by $2^{c.n}$ where n is the maximum size of the inputs. So, the runtime of N is linear in $2^{c.n}$, that is, polynomial in the greatest input value.

Theorem 3. *A function over the domain $\mathbb{N} = \{s, 0\}$ is $\Pi(0)$-computable iff it is* Linspace.

Proof. Let ϕ be a function computed by the $\Pi(0)$-rewrite system $\langle \mathcal{R}, \mathbb{F}, \mathbb{N}, f, [\,]\rangle$. By Lemma 6, ϕ is computable in polynomial time over a unary alphabet. So, by Theorem 2, ϕ is in Linspace. Conversely, if ϕ is in Linspace, then Theorem 2 yields that ϕ is computable in polynomial time on some stack TM which works on a unary alphabet. Therefore, by Lemma 9, ϕ is $\Pi(0)$-computable.

5 Etime is $\Pi(1)$

Define Etime as the class of functions which are computable in time bounded by $2^{O(n)}$ on Turing machines, where n is the maximum size of the inputs.

5.1 $\Pi(1)$-computable functions are Etime

Lemma 10. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,]\rangle$ be a $\Pi(1)$ rewrite system. Then there is a constant c such that, for all $t \in \mathcal{T}(\mathbb{W})$, $[t] \leq 2^{c|t|}$.*

Proof. By definition of $\Pi(1)$, for all constructors in $\mathbb{W}$, there is a constant c such that $[\epsilon] \leq c$ and and $[s](x) \leq c.x$, when $x > 0$. It is clear that

$$[s(t)] \leq c.[t] \ \leq c.2^{c.|t|} \ \leq 2^{c(|t|+1)} = 2^{c|st|}$$

Corollary 2. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\ ]\rangle$ be a $\Pi(1)$ rewrite system. Then there is a constant c such that, for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,*

1. *the length of any derivation starting from $f(t_1, \ldots, t_n)$ is bounded by $2^{c \max\{|t_i| \,;\, 1 \leq i \leq n\}}$,*
2. *if $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq 2^{c \max\{|t_i| \,;\, 1 \leq i \leq n\}}$.*

Proof. Consequence of Lemma 4 and Lemma 10.

Lemma 11. *If ϕ is $\Pi(1)$-computable, then ϕ is in* Etime.

Proof. See the proof of Lemma 6.

5.2 ETIME functions are $\Pi(1)$-computable

Lemma 12. *Let c be a constant. The function $\lambda n.\ 2^{c.n}$ is $\Pi(1)$-computable.*

Proof. Let $\langle \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{E}, \{0, s, q\}, \mathtt{E}\rangle$ be the rewrite system defined as in example 1 with the following rules :

$$\begin{cases} \mathtt{E}(0) \to s0 \\ \mathtt{E}(qx) \to \mathtt{Add}(\mathtt{E}(x), \ldots \mathtt{Add}(\mathtt{E}(x), 0)) \quad (2^c \text{ occurrences of } \mathtt{Add}) \end{cases}$$

$\mathtt{E}$ represents $\lambda n.\ 2^{cn}$ through the encoding pair (α, β) where $\alpha(|) = q$ and $\beta(|) = s$. The rewrite system is $\Pi(1)$:

$$[q](x) = 2^{c+1}(x+3)$$
$$[\mathtt{E}](x) = x + 3$$

Lemma 13. *If ϕ is in* ETIME*, then ϕ is $\Pi(1)$-computable.*

Proof. Let ϕ be a function, computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by an exponential 2^{cn} for some constant c. The function ϕ can be obtained by composing ϕ_M as defined in Lemma 7 and $\lambda n\ 2^{cn}$. Since ϕ_M is $\Pi(0)$ and $\lambda n\ 2^{cn}$ is $\Pi(1)$, ϕ is $\Pi(1)$-computable by Lemma 3.

6 $\Pi(2)$ is E_2TIME

6.1 $\Pi(2)$-computable functions are E_2TIME

Lemma 14. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\,]\rangle$ be a $\Pi(2)$ rewrite system. Then there is a constant $c > 0$ such that, for all $t \in \mathcal{T}(\mathbb{W})$, $[t] \leq 2^{2^{c|t|}}$.*

Proof. By definition of $\Pi(2)$, there is a constant a such that $[\epsilon] \leq a$ and $[s](x) \leq ax^a$, for $x > 0$ and for all constructors. Define $c = 2a$. It is easy to verify that

$$[s(t)] \leq a.[t]^a \leq a.(2^{2^{c.|t|}})^a \leq (2^{2^{c.|t|}})^{2a} \leq 2^{2^{c.|t|}.2a} \leq 2^{2^{c.(|t|+1)}}$$

Corollary 3. *Let $\langle \mathcal{R}, \mathbb{F}, \mathbb{W}, f, [\ ]\rangle$ be a $\Pi(2)$ rewrite system. Then there is a constant c such that, for all terms $t_1, \ldots, t_n \in \mathcal{T}(\mathbb{W})$,*

1. *the length of any derivation starting from $f(t_1, \ldots, t_n)$ is bounded by $2^{2^{c \max\{|t_i|\ ;\ 1 \leq i \leq n\}}}$.*
2. *if $f(t_1, \ldots, t_n) \xrightarrow{+}_{\mathcal{R}} v$, then $|v| \leq 2^{2^{c.\ \max\{|t_i|\ ;\ 1 \leq i \leq n\}}}$.*

Proof. Consequences of Lemmas 4 and 14.

Lemma 15. *If ϕ is $\Pi(2)$-computable, then ϕ is in* E_2TIME.

6.2 E_2TIME functions are $\Pi(2)$-computable

Lemma 16. *Let c be a constant. The function $\lambda n\ 2^{2^{cn}}$ is $\Pi(2)$-computable.*

Proof. As in example 1, we define $\langle \mathcal{A}, \{\mathtt{Add}, \mathtt{Mul}, \mathtt{D}\}, \{0, s, q\}, \mathtt{D}\rangle$ where $\mathtt{D}$ is defined thus,

$$\begin{cases} \mathtt{D}(0) \to ss0 \\ \mathtt{D}(qx) \to \mathtt{Mul}(\mathtt{D}(x), \ldots \mathtt{Mul}(\mathtt{D}(x), s0)) \quad (2^c \text{ occurrences of } \mathtt{Mul}) \end{cases}$$

$\mathtt{D}$ represents $\lambda n\ 2^{2^{cn}}$ through the encoding pair (α, β) where $\alpha(1) = q$ and $\beta(1) = s$. The rewrite system is $\Pi(2)$:

$$[q](x) = (3^{2^c}2)(x+3)^{2^c}$$
$$[\mathtt{D}](x) = x + 3$$

Lemma 17. *If ϕ is in E_2TIME, then ϕ is $\Pi(2)$-computable.*

Proof. Let ϕ be a function computed by a Turing machine $\mathcal{M}$, such that the time of computation is bounded by a doubly exponential function, D. The function ϕ is obtained by composing ϕ_M as defined in Lemma 7 and D. Since ϕ_M is $\Pi(0)$-computable and D is $\Pi(2)$-computable, Lemma 3 implies that ϕ is $\Pi(2)$-computable.

References

1. S. Bellantoni and S. Cook, *A new recursion-theoretic characterization of the polytime functions*, Computational Complexity,2, 1992, p. 97–110.
2. A. Ben Cherifa and P. Lescanne, *Termination of rewriting systems by polynomial interpretations and its implementation.* Science of computer Programming 9 (1987), p. 131-159.
3. E.A Cichon and P. Lescanne, *Polynomial interpretations and the complexity of algorithms.* CADE'11 (1992), p 139-147.
4. A. Cobham, *The intrinsic computational difficulty of functions*, ed. Y. Bar-Hillel, Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science, North-Holland, Amsterdam, 1962, p. 24-30.
5. N. Dershowitz and J.P. Jouannaud, *Rewrite systems.* Handbook of Theoretical Computer Science vol.B, North-Holland.
6. J. Giesl, *Generating polynomial orderings for termination proofs.* RTA-95, Lecture Notes in Computer Science 914, p. 427-431.
7. Y. Gurevich, *Algebras of feasible functions.*, Twenty Fourth Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1983, p. 210–214.
8. D. Hofbauer and C. Lautemann, *Termination proofs and the length of derivations.* RTA-88, Lecture Notes in Computer Science 355.
9. D.S. Lankford, *On proving term rewriting systems are Noetherien.* Technical Report Memo MTP-3, louisiana Technical University, Ruston, LA (1979).
10. D. Leivant, *Predicative recurrence and computational complexity I: Word recurrence and poly-time*, Feasible Mathematrics II, ed. Peter Clote and Jeffery Remmel, Birkhauser-Boston, 1994.

RPO Constraint Solving Is in NP

Paliath Narendran[1,*], Michael Rusinowitch[2], and Rakesh Verma[3,**]

[1] Institute of Programming and Logics, Department of Computer Science, State University of New York at Albany, Albany, NY 12222, USA
dran@cs.albany.edu
[2] LORIA, 615, rue du jardin botanique, BP 101, 54602 Villers les Nancy cedex, France
rusi@loria.fr
[3] Dept. of Comp. Science, University of Houston, Houston, TX 77204, USA
rmverma@cs.uh.edu

Abstract. A new decision procedure for the existential fragment of ordering constraints expressed using the recursive path ordering is presented. This procedure is nondeterministic and checks whether a set of constraints is solvable over the *given* signature, i.e., the signature over which the terms in the constraints are defined. It is shown that this nondeterministic procedure runs in polynomial time, thus establishing the membership of this problem in the complexity class NP for the first time.

1 Introduction

Reduction orderings have been developed for testing the termination of term rewrite systems. Among these orderings the Recursive Path Ordering (RPO) [3] is one of the most used since it gives simple syntactic conditions for orienting equations as rules. An optimal way to apply RPO is to orient an equation $u = v$ as $u \to v$ if for all ground substitutions θ, $u\theta \succ_{rpo} v\theta$. When the precedence is total this amounts to checking the unsatisfiability of the *ordering constraint* $v \succ_{rpo} u \vee u \sim_{rpo} v$. For instance, $h(f(x,x), f(y,y), k(z)) = h(f(x,y), f(x,y), z)$ cannot be oriented with the standard version of RPO. However the *constrained* version allows for an immediate left-to-right orientation.

Many extensions of RPO have been designed in order to orient equations of the kind above [16,9,13,17]. Since these extensions (in the multiset case) coincide with RPO on ground terms for total precedence, *constrained* RPO can be considered as subsuming them: if two terms are comparable with any of these extensions they are indeed comparable with *constrained* RPO.

Symbolic constraints are also widely used in automated deduction for narrowing the search space ([6,11,15]). They are commonly formulas to be interpreted in a Herbrand universe (a term algebra). Deduced formulas inherit constraints

* Research supported in part by the NSF grants CCR-9712396 and INT-9401087
** Research supported in part by NSF grant CCR-9732186

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 385–398, 1999.

from their antecedents. These constraints reduce the number of potential instantiations of the formulas they label. For instance, unification or equational constraints prevent redundant deductions by forbidding equality replacements in subformulas that have been frozen as constraints. Ordering constraints based on RPO and its variants allow us to express theorem-proving strategies such as ordered resolution, ordered paramodulation [11,15] and unfailing completion [7]. Note that the completeness of the above strategies only assumes that the ordering can be extended to a simplification ordering that is total on ground terms. Although RPO is not total on ground terms it can be extended easily in such a way.

An important breakthrough in this area was the proof by H. Comon that the *Lexicographic Path Ordering* (LPO) constraint solving problem is decidable [1]. Subsequently R. Nieuwenhuis [14] showed that the problem is NP-complete. In the case of RPO, the decidability of constraint solving was proved by J.P. Jouannaud and M. Okada [8]. In fact, they gave an algorithm for a general ordering combining both RPO and LPO. The RPO case is also considered by [10] where a polynomial bound on the depths of the terms in a solution is derived. This is the only upper bound that has been given for RPO constraint solving so far, as far as we know; however, it does not help us show that the solvability problem is in NP. In this paper we show it to be in NP for the first time, and therefore NP-complete since it is NP-hard by [14]. Unlike [8] our algorithm does not use any predecessor function. Such a function creates big terms in the solving process and it is unlikely that the algorithm in [8] could be modified into a NP one.

Note also that the proof that LPO constraint solving is NP cannot be adapted for RPO since it relies heavily on the fact that a non-natural term is a subterm of its successor, which is not the case for RPO.

Given a total precedence on function symbols, our non-deterministic algorithm for solving a constraint C is very simple: The first step is standard ([1]) and amounts to guessing a linear ordering L on the set of subterms of C. In the second step we introduce a quadratic number of new variables in the chain L to get L'. Finally we check the so-called *feasibility of gaps* where a gap in a chain is a subsystem of the chain with non-variable terms only occurring at its ends. A gap is feasible if it is entailed by former (earlier) gaps (in the chain). If all gaps of L' are feasible and L' implies C then the algorithm returns "satisfiable".

2 Basic Definitions

We assume that F is a finite set of function symbols given with their arity. $T(F, V)$ is the set of finite terms built on F and an alphabet V of (first-order) *variable symbols*. $\equiv$ denotes syntactic equality of terms. $T(F)$ is the set of terms which do not contain any variables. A multiset over a set X is a function M from X to the natural numbers. Any ordering $>$ on X can be extended to an ordering $\gg$ on finite multisets over X as follows: $M \gg N$ if a) $M \neq N$ and b) whenever $N(x) > M(x)$ then $M(y) > N(y)$ for some $y > x$. Note that if $>$ is well-founded so is $\gg$.

Recursive Path Ordering (*rpo*) as originally defined by Dershowitz [3] (sometimes referred to as *multiset path ordering* or *mpo* to distinguish it from *lpo*) is defined as follows:

Given a finite total precedence $\succ_F$ on functions,

$$s = f(s_1, \ldots, s_m) \succ^F_{rpo} g(t_1, \ldots, t_n) = t$$

if and only if one of the following holds:

1. $f \succ_F g$ and $s \succ^F_{rpo} t_i$ for all $1 \leq i \leq n$
2. $f = g$, $m = n$ and
$$\{s_1, \ldots, s_m\} \succ\!\!\succ^F_{rpo} \{t_1, \ldots, t_m\}$$
3. There exists a j, $1 \leq j \leq m$, such that $s_j \succ^F_{rpo} t$ or $s_j \sim_{rpo} t$.

where $\sim_{rpo}$ is defined as "permutatively equal," or, in other words, the terms are treated as *unordered* trees. The multiset extension of $\succ_{rpo}$ used above, namely $\succ\!\!\succ_{rpo}$, is defined in terms of this equivalence.

An *RPO constraint solving problem* is a quantifier-free first-order formula constructed from terms and the binary predicate symbols "$\succ$" and "$=$". The formula is interpreted in $T(F)$, with $\succ$ as $\succ_{rpo}$ and $=$ as $\sim_{rpo}$.

Note that the smallest ground term is also the smallest constant a. Let f be the lowest nonconstant function symbol. Let $\{a_k \succ_F a_{k-1} \succ_F \ldots \succ_F a_1\}$ be the possibly empty set of constants smaller than f and let $Const = \{a_k \succ_F a_{k-1} \succ_F \ldots \succ_F a_1 \succ_F a\}$. For readability we present in the body of the paper the case where f is of arity 2 and in the Appendix we give the modifications needed for handling the general case.

Let g be the smallest symbol greater than f; then the first non-successor term w.r.t. $\succ_{rpo}$ is $g(a, \ldots, a_{arity(g)})$. We assume in the following that g has arity 2. The proof only needs trivial change in all other cases (including arity 0 or 1).

The set of ground terms up to the permutative equivalence, $T(F)/\sim_{rpo}$, is well-ordered by $\succ_{rpo}$. It has been proved (see e.g. [10]) that $\circ$, defined below, is a successor function on this well-ordered set:

$$\circ(s) = \begin{cases} a_{i+1} & \text{if } s = a_i \text{ and } i < k, \\ f(a, a) & \text{if } s = a_k, \\ f(a, s) & \text{if } s = h(s_1, \ldots, s_n) \text{ and } h \succ_F f, \\ f(a, \circ(s_2)) & \text{if } s = f(s_1, s_2) \text{ and } s_1 \sim_{rpo} s_2, \\ f(\circ(s_1), s_2) & \text{if } s = f(s_1, s_2) \text{ and } s_1 \prec_{rpo} s_2. \\ f(s_1, \circ(s_2)) & \text{if } s = f(s_1, s_2) \text{ and } s_2 \prec_{rpo} s_1. \end{cases}$$

Note that $\circ$ increases the depth of a term by at most 1. For instance one has $\circ(f(f(a, a), f(a, a))) = f(a, f(a, f(a, a)))$.

3 Example

We consider a precedence $a < f < g$ with f and g as above and the problem:

$$s \equiv f(g(g(u,u),g(v,v)),u) \prec t \equiv f(g(g(u,v),g(u,v)),f(a,a))$$

Guessing a linear ordering between subterms allows us to get for instance:

$$a \prec u = v \prec f(a,a) \prec g(a,a) \prec g(u,u) = g(u,v) = g(v,v) \prec g(g(u,v),g(u,v)) = g(g(u,u),g(v,v)) \prec s \prec t$$

In the following we shall call such a system a *simple* system: the equality relations are compatible; also any inequality that can be decomposed according to the RPO definition is entailed from inequalities to its left. However this simple system is not satisfiable since no term can be inserted between a and $f(a,a)$: we say in this case that the gap $a \prec u \prec f(a,a)$ is not feasible. A satisfiable simple system could have been obtained if we had instead guessed $u = a$.

4 Linear and simple systems

The first step of our procedure follows the same strategy as [1]. Given an *RPO-constraint solving problem* we transform it, in non-deterministic polynomial time, into a *linear system,* defined as follows, by first guessing the relations between all subterms occurring in C:

Definition 1. A *linear system* L is a linear order of the form

$$a \ \# \ \ldots \ \# \ g(a,a) \ \# \ \ldots$$

which is closed under subterm and where each symbol $\#$ is either $\prec$ or $=$. In other words, if t is in the system then every subterm s of t has to occur to the left of it. For ease of exposition, we assume that the terms a and $g(a,a)$ are in L.

Note that any solution of C satisfies some linear system extracted from C. Therefore satisfiability of C is reduced to checking that some linear system extracted from C is satisfiable and entails C. Entailment can be verified easily by simple evaluation of C using direct information from the linear system. We show now how to check the satisfiability of a linear system.

At the expense of polynomial-time work we can assume that

- Terms in a linear system will be always represented in such a way that the arguments to each function are given in ascending order in terms of $\prec$.

We further assume that L is *rpo-closed,* which is defined as follows:

A relation $\succ$ on $T(F,V)$ is *rpo-closed* if (i) $a \succ s$ is not in the relation for any term s and (ii) for every $f(s_1,\ldots,s_m) \succ^* g(t_1,\ldots,t_n)$ one of the following holds:

1. $f \succ_F g$ and $s \succ^* t_i$ for all $1 \leq i \leq m$
2. $f = g$, $m = n$ and
$$\{s_1, \ldots, s_m\} \succcurlyeq^* \{t_1, \ldots, t_m\}$$
3. There exists a j, $1 \leq j \leq m$, such that $s_j \succ^* t$ or $s_j \equiv t$.

where $\succcurlyeq^*$ is the multiset extension of $\succ^*$, the transitive closure of $\succ$.

It can be checked in polynomial time that a linear system is rpo-closed.

We first concentrate on the equational part $eq(L)$ of a linear system L. We do not want to solve it explicitly since replacing every variable by its solution may generate exponential size terms. We rather adopt a technique similar to the *DAG solved form* approach by considering the equivalence relation generated by all equalities in L, and denoted by $=_L$.

Definition 2. *A linear system L is* eq-feasible *if:*

1. *two terms with different root function symbols are not equivalent modulo $=_L$.*
2. *a term and a proper subterm of it are not equivalent modulo $=_L$.*
3. *$f(s_1, \ldots, s_n) =_L f(t_1, \ldots, t_n)$ implies $s_i =_L t_i$ for $i = 1, \ldots, n$*

It is clear that a satisfiable L is eq-feasible. The last requirement is satisfied because the arguments are ordered w.r.t. $\prec$. We also have:

Proposition 1 *It can be checked in polynomial time that a linear system is eq-feasible. If L is eq-feasible then every ground substitution whose domain is the set of variables in L is a solution of $eq(L)$.*

This proposition shows that once the system is eq-feasible we only need to show existence of solutions for the inequalities.

Definition 3. *A* simple system *is an eq-feasible linear system.*

Given a simple system S we shall "identify" a term with its equivalence class modulo $=_S$. For instance a variable is a class that only contains variables. The root symbol of a non-variable term is the root symbol of any non-variable term of its class. The arguments of a non-variable term are the classes of the arguments of any non-variable term of its class (recall that arguments are ordered w.r.t. $\prec$).

Definition 4. *A* gap *in a system is a subchain with non-variable terms at both ends and only variables in the middle.*

5 Membership in NP

Let S be an rpo-closed simple system

$$a \prec \ldots \prec g(a, a) \prec \ldots$$

The function π denotes the position of a term on S from the left. Thus $\pi(a) = 0$.

Let i be an integer. An i-expansion of S between positions j and k is an introduction of new variables $z_1, \ldots, z_i$ between j and k.

By a *natural number term* we mean a term t such that $t \prec g(a, a)$. We use N to denote the set of all natural number terms. The numerical value ν of a natural number term is defined as follows: $\nu(a) = 0$ and, for $s \neq a$, $\nu(s) = k$ if s is the k^{th} successor of a.

Note that computing the set of $\nu(b)$ for all ground terms b occurring in a simple system S is polynomial in $|S|$.

The following easy result will be very useful in the proof. It can be checked by induction on $f(t_1, t_2)$ w.r.t. $\prec_{rpo}$.

Fact: Given natural number terms $f(s_1, s_2) \prec_{rpo} f(t_1, t_2)$, we have $\nu(f(t_1, t_2)) - \nu(f(s_1, s_2)) = \nu(s_2) - \nu(s_1) + \nu(t_1) + 1/2[(\nu(t_2) + \nu(s_2) + 2)(\nu(t_2) - \nu(s_2) - 1)] + 1$.

Lemma 1. *Let t_1 and t_2 be two terms in a simple system $S =$*

$$a \prec \ldots \prec t_1 \prec \ldots \prec t_2 \prec \ldots$$

and let θ be a solution of S. Let $\delta = \pi(t_2) - \pi(t_1)$. Then $\circ^{\delta}(\theta(t_1)) \preceq \theta(t_2)$.

For the sake of exposition we often use the phrase "k terms are possible between s and t" to mean $\circ^k(s) \prec t$.

Lemma 2. *Let t_1 and t_2 be two terms in a simple system $S =$*

$$a \prec \ldots t_1 \ldots t_2 \ldots \prec g(a, a)$$

and let θ be a solution of S. Let $\Delta = \nu(\theta(t_2)) - \nu(\theta(t_1)) - 1$ and $\delta = \pi(t_2) - \pi(t_1) - 1$. Then, for all $k < (\Delta - \delta)$ there is a k-expansion S' of S between $\pi(t_1)$ and $\pi(t_2)$ such that θ can be extended to a solution of S'.

A more general statement is the following:

Lemma 3. *Let t_1 and t_2 be two terms in a simple system $S =$*

$$a \prec \ldots \prec t_1 \prec \ldots \prec t_2 \prec \ldots$$

and let θ be a solution of S. Let $\delta = \pi(t_2) - \pi(t_1) - 1$. Then, for all $m > \delta$ the following holds: if $\circ^m(\theta(t_1)) \prec \theta(t_2)$, then there is an $(m - \delta)$-expansion S' of S between $\pi(t_1)$ and $\pi(t_2)$ such that θ can be extended to a solution of S'.

Definition 5. *A gap* $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ *is* rigid *if and only if none of the* x_i*'s occur in* t*.* G *is* elastic *if and only if it is not rigid, i.e., if* x_i *occurs in* t *for some* $1 \leq i \leq n$*.*

Definition 6. *A rigid gap* $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ *in a simple system* S *is* feasible *if and only if one of the following holds:*

(a) s *and* t *are ground natural-number terms,* s *is a constant, and* $\nu(t) - \nu(s) - 1 \geq n$*.*

(b) $s = f(s_1, s_2)$*,* $t = f(t_1, t_2)$*,* $t \prec g(a, a)$ *and*

$$(\pi(s_2) - \pi(s_1)) + \pi(t_1) + \frac{(\pi(t_2) + \pi(s_2) + 2)(\pi(t_2) - \pi(s_2) - 1)}{2} \geq n$$

(c) *The root symbol of* t *is some* $h \succ f$

(d) $g(a, a) \prec s$*,* $s = f(s_1, s_2)$*,* $t = f(t_1, t_2)$*, and either*

(i) $s_2 = t_2$ *and* $\pi(t_1) - \pi(s_1) \geq n + 1$*, or*

(ii) $\pi(t_2) - \pi(s_2) > 1$*, or*

(iii) $\pi(t_2) - \pi(s_2) = 1$ *and* $(\pi(s_2) - \pi(s_1)) + \pi(t_1) \geq n + 1$

(e) $g(a, a) \preceq s$*,* $s = h(s_1, s_2)$*,* $t = f(t_1, t_2)$*,* $h \succ f$*, and* $\pi(t_1) \geq n$*. (*h *could be the same as* g*.)*

Note that cases (a) and (b) deal with the case when the gap needs to be filled with natural number terms.

Lemma 4. *Let* $S = a \prec \ldots \prec g(a, a)$ *be a simple system and* θ *be a solution of* S*. Let* $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ *be a rigid gap in* S*. Then there is a* k*-expansion* ($k \leq n$) S' *of* S *between 0 and* $\pi(s)$ *such that*

(a) G *is feasible in* S'*, and*

(b) θ *can be extended to a solution of* S'*.*

Proof: If s and t are ground natural-number terms then any solution θ of S satisfies $\nu(\theta(t)) - \nu(\theta(s)) - 1 = \nu(t) - \nu(s) - 1 \geq n$. Hence G is feasible in S and we can take S for S'. Otherwise, $s = f(s_1, s_2), t = f(t_1, t_2)$ and $\nu(\theta(t)) - \nu(\theta(s)) - 1$ is equal to

$$(\nu(\theta(s_2)) - \nu(\theta(s_1))) + \nu(\theta(t_1)) + \frac{(\nu(\theta(t_2)) + \nu(\theta(s_2)) + 2)(\nu(\theta(t_2)) - \nu(\theta(s_2)) - 1)}{2}$$

Since θ is a solution we have $\nu(\theta(t)) - \nu(\theta(s)) - 1 \geq n$. Assume that the following expression E_π

$$(\pi(s_2) - \pi(s_1)) + \pi(t_1) + \frac{(\pi(t_2) + \pi(s_2) + 2)(\pi(t_2) - \pi(s_2) - 1)}{2}$$

is less than n. We can show that by adding new variables to S we can raise the values of π so that E_π gets bigger than n. Assume for instance that $\pi(t_2) - \pi(s_2) > 1$. If $\pi(s_2) - \pi(s_1)$ (resp. $\pi(t_1)$, $\pi(t_2)$, $\pi(s_2)$, $\pi(t_2) - \pi(s_2)$) is less than $\nu(\theta(s_2)) - \nu(\theta(s_1))$ (resp. $\nu(\theta(t_1))$, $\nu(\theta(t_2))$, $\nu(\theta(s_2))$, $\nu(\theta(t_2)) - \nu(\theta(s_2))$) then by adding a new variable between s_2 and s_1 (resp. before t_1, before t_2, before s_2, between t_2 and s_2) we get a new system S' with function π' such that $E_{\pi'} > E_\pi$. More precisely, the extra variable should always be added between two non-variable terms s', t' delimiting a subinterval of m variables ($m \geq 0$) with $\nu(\theta(s')) - \nu(\theta(t')) - 1 > m$. Note that the only situation where we cannot extend the system by a new variable is when $\pi(s_2) - \pi(s_1) = \nu(\theta(s_2)) - \nu(\theta(s_1))$, $\pi(t_1) = \nu(\theta(t_1))$, $\pi(t_2) = \nu(\theta(t_2))$, $\pi(s_2) = \nu(\theta(s_2))$ and $\pi(t_2) - \pi(s_2) = \nu(\theta(t_2)) - \nu(\theta(s_2))$. However in this case $E_\pi \geq n$ and no extension is needed. This shows that by adding at most n variables to S we get a new system S' such that $E_{\pi'} \geq n$ and therefore the gap G is feasible in S'. When $\pi(t_2) - \pi(s_2) \leq 1$ the reasoning is similar. □

Lemma 5. *Let S be a simple system and and θ be a solution of S. Let $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ be a rigid gap in S where $g(a, a) \preceq s$. Then there is a k-expansion ($k \leq n$) S' of S between 0 and $\pi(s)$ such that*

(a) G is feasible in S', and

(b) θ can be extended to a solution of S'.

Proof: If the root symbol of t is some $h \succ f$, then G is feasible in S and we may take S for S'. Otherwise let $t = f(t_1, t_2)$.

Let us assume that $s = f(s_1, s_2)$. If $\theta(s_2) = \theta(t_2)$ then ($s_2 = t_2$, and) $\circ^n(\theta(s_1)) \prec \theta(t_1)$. Hence we can insert at most n new variables between t_1 and s_1 such that G becomes feasible in the extended system S' and θ can be extended to a solution of S'. If $\circ(\theta(s_2)) = \theta(t_2)$ then for some ordinals k, k', $\circ^k(\theta(s_1)) = \theta(s_2)$ and $\circ^{k'}(a) = \theta(t_1)$. If one of the ordinals, say k, is not finite then we can add n new variables between s_1 and s_2 to get a new system S' such that $\pi(s_1) - \pi(s_2) \geq n$. If both k, k' are finite then we must have $k + k' \geq n$ for θ to be a solution. By adding variables between s_1 and s_2 and before t_1 we get again a suitable system S'. If $\circ(\theta(s_2)) \prec_{rpo} \theta(t_2)$ then it is always possible to introduce a new variable such that $\pi(t_2) - \pi(s_2) > 1$.

Suppose the root symbol of s is some $h \succ f$. The case $s \prec t_2$ is impossible: if t_2 is a non-variable term $s \prec \ldots \prec t$ is not a gap; if t_2 is a variable $s \prec \ldots \prec t$ is not rigid. So the only possible case is when $s = t_2$. Since θ is a solution, if $n > 0$ then $\circ^n(\theta(s)) = f(\circ^{n-1}(a), s) \prec_{rpo} \theta(f(t_1, s))$. Hence $\circ^{n-1}(a) \prec_{rpo} \theta(t_1)$ and there is a way to insert n new variables before t_1 such that G gets feasible in the extended system S' and θ can be extended to a solution of S'. □

Lemma 6. *Let S be a simple system and $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ be a feasible rigid gap in S. Let S' be such that $S = S' \cup G \cup S''$, where every term in S' is below s and every term in S'' is above t in the ordering. Then S' is solvable if and only if $S' \cup G$ is solvable.*

Proof: Let θ be a solution of S'. Note that $\theta(s)$ and $\theta(t)$ are both ground terms, since G is a rigid gap. Suppose $S' \cup G$ is not solvable. Then it must be that there

is a $k \leq n$ such that $\circ^k(\theta(s)) = \theta(t)$. Hence the root symbol of t must be f; let $t = f(t_1, t_2)$ for some $t_1 \preceq t_2$. We now consider different cases depending on the root symbol of s and also where the gap is situated—in the natural number part or to the right of $g(a, a)$.

$\mathbf{root(s) = h \neq f}$: Thus $\circ^k(\theta(s)) = f(\bar{q}, \theta(s))$ where $\nu(\bar{q}) = k - 1$. In other words, $\theta(t_1) = \bar{q}$ and hence $\nu(\theta(t_1)) = k - 1$. But since the gap is feasible, by case (e) in the definition of feasibility it must be that $\pi(t_1) \geq n$. Since $t_1 \prec f(t_1, t_2)$, t_1 belongs to S'. Hence, due to Lemma 1, it is impossible that θ is a solution to S', since $k \leq n$.

$\mathbf{s = f(s_1, s_2) \prec t \prec g(a, a)}$: Since $\theta(s)$ and $\theta(t)$ are natural numbers, $\nu(\theta(t)) - \nu(\theta(s)) - 1$ is equal to:

$$(\nu(\theta(s_2)) - \nu(\theta(s_1))) + \nu(\theta(t_1)) + \frac{(\nu(\theta(t_2)) + \nu(\theta(s_2)) + 2)(\nu(\theta(t_2)) - \nu(\theta(s_2)) - 1)}{2}$$

Since gap G is feasible, by Lemma 1, it is not hard to show that this value must be at least n.

$\mathbf{g(a, a) \prec s = f(s_1, s_2) \prec t}$: Clearly $g(a, a) \preceq s_2 \preceq t_2$. If $s_1 \equiv s_2$ then $\circ(\theta(s)) = f(a, \circ(\theta(s_2)))$. Otherwise, i.e., if $s_1 \prec s_2$, then $\circ(\theta(s)) = f(\circ(\theta(s_1)), \theta(s_2))$. We now split into cases based on the relationship between s_2 and t_2.

$s_2 \equiv t_2$: Then it must be that $\circ^k(\theta(s_1)) = \theta(t_1)$. Hence the feasibility condition (d)(i), which applies to this gap, could not have been fulfilled.

$\theta(t_2) = \circ(\theta(s_2))$: Here condition (d)(iii) is what ensures feasibility. By Lemma 1, the number of terms possible between $\theta(s_2)$ and $\theta(s_1)$ must be at least $\pi(s_2) - \pi(s_1)$. Similarly, the number of terms possible between $\theta(t_1)$ and a must be at least $\pi(t_1)$. Hence it is not hard to see that $\theta(t)$ cannot be equal to $\circ^k(\theta(s))$ for any $k \leq n$.

$\circ(\theta(s_2)) \prec_{rpo} \theta(t_2)$: Note that $\theta(s) \prec_{rpo} f(a, \circ(\theta(s_2)))$. Since $g(a, a) \prec_{rpo} \theta(s_2)$, it is impossible that $\circ^m(f(a, \circ(\theta(s_2)))) \equiv \theta(t)$ for any m. □

Lemma 7. *Let S be a simple system and $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ be an elastic gap in S. Let S' be such that $S = S' \cup G \cup S''$, where every term in S' is below s and every term in S'' is above t in the ordering. Then S' is solvable if and only if $S' \cup G$ is solvable.*

Proof: Let θ be a solution of S' and let i be the largest index such that x_i occurs in t. We shall extend θ to a solution of $S' \cup G$. Therefore we define: $\theta(x_j) = \circ^j(\theta(s))$ for $1 \leq j \leq i - 1$. In the following we show that a ground term v can be selected such that:

$$\theta(s) \prec_{rpo} v \preceq_{rpo} \circ^{n-i}(v) \prec_{rpo} \theta_v(t)$$

where $\theta_v = \theta \cup \{x_i/v\}$. This will be sufficient for proving the lemma since the variables between x_i and t do not occur in t. We shall take $v = f(a, \circ^{n-i}(\theta(x)))$ where $x = x_{i-1}$ if $i - 1 > 0$ and s otherwise.

case $t \prec g(a,a)$: In this case we have $t \equiv f(t_1, t_2)$ with $t_1 \preceq t_2$ for some terms t_1, t_2 in S. Observe that $x_i \equiv t_1$ or $x_i \equiv t_2$ otherwise x_i would belong to a gap in S'.

subcase $x_i \equiv t_1 \equiv t_2$: Note that for any $u \prec_{rpo} g(a,a)$

$$\nu(f(u,u)) - \nu(u) = \nu(u) + 1$$

Now since $\nu(v) > n - i$, we have $\nu(f(v,v)) - \nu(v) > n - i$. Hence $\circ^{n-i}(v) \prec_{rpo} \theta_v(t)$.

subcase $x_i \equiv t_2 \not\equiv t_1$: If t_1 is not a variable then $t_1 \preceq s$ in S'. If t_1 is a variable then this variable occurs before x_i in S', by the choice of x_i. Hence, $\theta(t_1)$ is a ground term. Since

$$\circ^{n-i}(v) = f(\circ^{n-i}(a), \circ^{n-i}(\theta(x))) \prec_{rpo} f(\theta(t_1), f(a, \circ^{n-i}(\theta(x))))$$

we conclude again $\circ^{n-i}(v) \prec_{rpo} \theta_v(t)$.

subcase $x_i \equiv t_1 \not\equiv t_2$: Then $x \prec t_2$ in S'. Since $t_2 \prec t$, t_2 has to be a variable occurring between x_i and t. But this is impossible since x_i is the maximal variable to occur in t.

case $g(a,a) \preceq s \prec \ldots \prec t$: Let $t \equiv h(t_1, t_2)$. As above we have that $x_i \equiv t_2$. Note also that $\theta_v(t)$ is ground by the choice of x_i. We discuss according to the root symbol h of t:

subcase $h \neq f$: Since $g(a,a) \preceq_{rpo} \theta(x)$,

$$\circ^{n-i}(v) = f(\circ^{n-i}(a), \circ^{n-i}(\theta(x))) \prec_{rpo} \theta(h(t_1, x_i))$$

subcase $h = f$:

$$\circ^{n-i}(v) = f(\circ^{n-i}(a), \circ^{n-i}(\theta(x))) \prec_{rpo} \theta(f(t_1, x_i))$$

since $\circ^{n-i}(a) \prec_{rpo} \circ^{n-i}(\theta(x)) \prec_{rpo} \theta(x_i)$. □

A simple system S is *gap-consistent* if and only if every rigid gap in it is feasible.

Lemma 8. *Let S be a simple system and $G = (s \prec x_1 \prec \ldots \prec x_n \prec t)$ be the rightmost gap in S. Let S' be such that $S = S' \cup G \cup S''$, where every term in S' is below s and every term in S'' is above t in the ordering. If G is elastic, then S is solvable if and only if S' is solvable.*

Lemma 9. *Every gap-consistent simple system is solvable.*

From the proofs of Lemmas 6 and 7, it is clear that we can derive a polynomial depth-bound for a solution of a gap-consistent system. Together with the main theorem below this gives us another derivation of the result of [10].

Theorem 2. *A simple system S is solvable iff it admits a gap-consistent extension S' of size $t * (v + 1)$ where v is the number of variables in S and t is the number of non-variable terms in S.*

Proof: We prove the "only if" part by induction on t. The base case is when $t = 2$. Then the simple system S is of the form

$$a \prec x_1 \prec \ldots \prec x_k \prec g(a, a) \prec x_{k+1} \prec \ldots \prec x_v$$

which is clearly gap-consistent by condition (c) in the definition. Thus we can take $S' = S$. Otherwise $S = S_i \cup G \cup S_u$ where $G = (s \prec \ldots \prec t)$ is the rightmost gap of S. Let θ be a solution of S, k ($k \leq v$) be the number of variables in G, and t_i (resp. t_u) be the number of non-variable terms in S_i (resp. S_u). If G is elastic, then by Lemma 8 S is solvable iff S_i is solvable, and we apply the induction hypothesis to S_i. Now assume that G is rigid. By Lemmas 4 and 5 there is a k-expansion S_i' of S_i such that G is feasible in $S_i' \cup G \cup S_u$ and this system has a solution extending θ. Let us apply the induction hypothesis to S_i': There is an expansion S_i'' of S_i' of size bounded by $t_i * (v + 1)$ which is gap-consistent and admits a solution extending θ. Now $S_i'' \cup G \cup S_u$ is also gap-consistent with a solution extending θ and has size bounded by: $t_i * (v + 1) + k + t_u$ which is less than $t * (v + 1)$. □

Example: Consider the simple system

$$a \prec f(a, a) \prec g(a, a) \prec u \prec v \prec f(u, u) \prec x_1 \prec x_2 \prec f(f(a, a), v)$$

The rightmost gap $(f(u, u) \prec x_1 \prec x_2 \prec f(f(a, a), v))$ is not feasible unless $\pi(v) - \pi(u) > 1$, since (d)(ii) is the only condition that is useful here. (Condition (d)(iii) will not work since $\pi(f(a, a)) = 1$.) If we introduce a new variable w between u and v, the system becomes gap-consistent. Note that the gap $(g(a, a) \prec u \prec w \prec v \prec f(u, u))$ is elastic.

6 Conclusion

In this paper, we have given a novel, non-deterministic decision procedure for ordering constraints expressed using the recursive path ordering. This procedure requires only polynomial-time for verification after the initial guessing phase. Hence, we established the membership of this problem in the complexity class NP. Since the problem is NP-hard, our result implies that it is NP-complete. We are also able to derive the depth-bound of [10] through a significantly different

approach. Note that in the case of extended signatures the result can be obtained much more easily as in [14] since every rpo-closed eq-feasible linear system is satisfiable.

Extension to RPO with status is not very difficult because the successor function only depends on the status of f (see [8]). Hence the method outlined by Nieuwenhuis [14] can be adapted here. Details will be given in the full paper.

References

1. H. Comon. Solving inequations in term algebras. In *Proc. 5th IEEE Symp. Logic in Computer Science (LICS),* Philadelphia, June 1990. 386, 386, 388
2. H. Comon and R. Treinen. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science* 176, Apr. 1997.
3. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science* 17(3): 279–301. 1982. 385, 387
4. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation* 3(1):69–115, Feb. 1987.
5. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–309. North-Holland, 1990.
6. H. Ganzinger, R. Nieuwenhuis, and P. Nivela. The Saturate System, 1995. Software and documentation available from: `www.mpi-sb.mpg.de`. 385
7. J. Hsiang and M. Rusinowitch. On word problems in equational theories. In T. Ottmann, editor, *14th International Colloquium on Automata, Languages and Programming (ICALP),* volume 267 of *Lecture Notes in Computer Science*, pages 54–71, Karlsruhe, Germany, July 1987. Springer-Verlag. 386
8. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm ordering is decidable. In *18th International Colloquium on Automata, Languages and Programming (ICALP),* volume 510 of *Lecture Notes in Computer Science*, pages 455-468, Madrid, Spain, July 1991. Springer-Verlag. 386, 386, 386, 396
9. D. Kapur, P. Narendran and G. Sivakumar. A path ordering for proving termination of term rewriting systems. In H. Ehrig (ed.), *10th CAAP,* volume 185 of *Lecture Notes in Computer Science*, pages 173-187, Berlin, March 1985. 385
10. D. Kapur and G. Sivakumar. Maximal Extensions of Simplification Orderings. presented at the 15$^{\text{th}}$ conference on the *Foundations of Software Technology and Theoretical Computer Science (FSTTCS-15),* Bangalore, India, December 1995. 386, 387, 395, 395
11. C. Kirchner, H. Kirchner, and M. Rusinowitch. Deduction with symbolic constraints. *Revue Française d'Intelligence Artificielle*, 4(3):9–52, 1990. Special issue on automatic deduction. 385, 386
12. M.S. Krishnamoorthy and P. Narendran. Note on recursive path ordering. *TCS* 40, pages 323-328. 1985.
13. P. Lescanne. On the recursive decomposition ordering with lexicographical status and other related orderings. *J. of Automated Reasoning* 6 (1) 39-49 (1990). 385
14. R. Nieuwenhuis. Simple LPO constraint solving methods. *Inf. Process. Lett.* 47(2):65–69, Aug. 1993. 386, 386, 396, 396

15. R. Nieuwenhuis and A. Rubio. Theorem proving with ordering constrained clauses. In D. Kapur, editor, Proceedings of 11th Conf. on Automated Deduction, Saratoga Springs, NY, 1992, volume 607 of *Lecture Notes in Artificial Intelligence,* pages 477–491, June 1992, Springer-Verlag. 385, 386
16. D. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Technical Report R-78-943, U. of Illinois, Dept of Computer Science, 1978. 385
17. J. Steinbach. Extensions and comparison of simplification orderings. Proceedings of 3rd RTA, Chapel Hill, NC, volume 355 of *Lecture Notes in Computer Science,* pages 434-448, 1989, Springer-Verlag. 385

Appendix

We now consider the case when the arity of f, the lowest function symbol, is greater than 2. For ease of presentation we define gaps between ordered tuples of terms. An *ordered tuple* is a tuple $(s_1, \ldots, s_n)$ where $s_1 \preceq \ldots \preceq s_n$. Given natural number terms (or ordered tuples of natural number terms) s and t, let $[s, t]$ denote the gap between s and t including both. For instance, $[a, f(a, a, a)] = 2$ if we assume f is ternary and a is the only constant below f in the ordering. The following lemma can be shown:

Lemma A.1 *Let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be ground natural number terms, where $p_i \preceq p_{i+1}$ and $q_i \preceq q_{i+1}$ for $1 \leq i < n$. Then*

$$[(p_1, \ldots, p_n), (q_1, \ldots, q_n)] = 1 + \sum_{i=1}^{n} \binom{\nu(q_i) + i - 1}{i} - \binom{\nu(p_i) + i - 1}{i}$$

Proof: By induction using the measure (length of the tuple, $\nu(q_n) - \nu(p_n)$). □

Corollary A.11 *Let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be ground natural number terms, where $n > 1$ and $p_i \preceq p_{i+1}$ and $q_i \preceq q_{i+1}$ for $1 \leq i < n$. If $\nu(q_n) - \nu(p_n) \geq 2$, then $[(p_1, \ldots, p_n), (q_1, \ldots, q_n)] \geq \nu(q_n) + 2$.*

Proof: The worst-case scenario is where $p_1 = \ldots = p_n$, $q_1 = \ldots = q_{n-1} = a$ and $\nu(q_n) = \nu(p_n)+2$. Then $[(p_n, \ldots, p_n), (a, \ldots, a, q_n)] = 2 + \binom{\nu(p_n)+1+n}{n} - \binom{\nu(p_n)+n}{n}$ $= 2 + \binom{\nu(p_n)+n}{n-1} \geq 2 + \nu(p_n) + 2$. □

Corollary A.12 *Let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be ground natural number terms, where $n > 1$ and $p_i \preceq p_{i+1}$ and $q_i \preceq q_{i+1}$ for $1 \leq i < n$. If $\nu(q_n) - \nu(p_n) = 1$, then $[(p_1, \ldots, p_n), (q_1, \ldots, q_n)] \geq \nu(p_n) - \nu(p_1) + \nu(q_{n-1}) + 2$.*

Condition (b) can now be changed to

(b′) $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$, $t \prec g(a, a)$ and

$$\sum_{i=1}^{m} \binom{\pi(t_i) + i - 1}{i} - \binom{\pi(s_i) + i - 1}{i} > n$$

Note that this can be checked in polynomial time since m, $\pi(s_i)$, etc. are effectively given in unary.

Let us now consider condition (d). Let $p = f(p_1, \ldots, p_m)$, $q = f(q_1, \ldots, q_m)$ be ground terms and $g(a, a) \prec p$. Let $k \in \{1, \ldots, m\}$ be the largest number such that $p_k \prec q_k$. The gap between p and q is infinite if and only if $q_k \succ g(a, a)$ and one of the following holds:

(i) $k = 1$ and the gap between p_1 and q_1 is infinite
(ii) $k > 1$ and $q_k \succ \circ(p_k)$
(iii) $k > 1$ and $g(a, a) \preceq q_{k-1}$
(iv) $p_2 \prec p_k$
(v) $k > 1$ and the gap between p_1 and p_2 is infinite

It is easy to see that when none of the above 5 conditions hold, either $k = 1$ and the gap between p_1 and q_1 is finite, or it has to be that $q_k = \circ(p_k)$, $p_2 = \ldots = p_k$, the gap between p_1 and p_2 is finite and $q_1, \ldots, q_{k-1}$ are natural number terms. In the latter case the gap between p and q is

$$(\text{gap between } p_1 \text{ and } p_2) + [(a, a, \ldots, a), (q_1, \ldots, q_{k-1})]$$

Thus the new condition in place of (d) is

(d′) $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$, $t \prec g(a, a)$. Let $k \in \{1, \ldots, m\}$ be the largest number such that $s_k \prec t_k$. Now either
(i) $t_k \prec g(a, a)$ and $\sum_{i=1}^{k} \binom{\pi(t_i)+i-1}{i} - \binom{\pi(s_i)+i-1}{i} > n$, or

(ii) $k = 1$ and $\pi(t_k) - \pi(s_k) > n$, or

(iii) $k > 1$, $g(a, a) \preceq t_k$ and $\pi(t_k) - \pi(s_k) > 1$, or

(iv) $k > 1$ and $g(a, a) \preceq t_{k-1}$, or

(v) $g(a, a) \preceq t_k$ and $s_2 \prec s_k$, or

(vi) $k > 1$ and $\pi(s_2) - \pi(s_1) + \sum_{i=1}^{k-1} \binom{\pi(t_i)+i-1}{i} > n$

In condition (e), the root symbol of $s = h(s_1, \ldots, s_{arity(h)})$ is greater than f in the ordering. The only case where the gap between s and t is finite is when $t = f(t_1, \ldots, t_m)$, $t_m = s$ and $t_1, \ldots, t_{m-1}$ are natural number terms.

Hence condition (e) can be reformulated as

(e′) $g(a, a) \preceq s, s = h(s_1, \ldots, s_{arity(h)}), t = f(t_1, \ldots, t_m), h \succ f$ and either
(i) $\sum_{i=1}^{m-1} \binom{\pi(t_i)+i-1}{i} \geq n$, or

(ii) $g(a, a) \preceq t_i$ for some $1 \leq i \leq m - 1$

Quantifier Elimination in Fuzzy Logic

Matthias Baaz[1] and Helmut Veith[2]

[1] Institut fär Algebra und Diskrete Mathematik
Technische Universität Wien, A-1040 Austria
baaz@logic.at
[2] Institut fär Informationssysteme
Technische Universität Wien, A-1040 Austria
veith@dbai.tuwien.ac.at

Abstract. We investigate quantifier elimination of first order logic over fuzzy algebras. Fuzzy algebras are defined from continuous t-norms over the unit interval, and subsume Łukasiewicz [28,29], Gödel [16,12] and Product [19] Logic as most prominent examples.
We show that a fuzzy algebra has quantifier elimination iff it is one of the abovementioned logics. Moreover, we show quantifier elimination for various extensions of these logics, and observe other model-theoretic properties of fuzzy algebras.
Further considerations are devoted to approximation of fuzzy logics by finite-valued logics.

1 Introduction

In the study of fuzzy and many-valued logics, logicians have mainly stressed *algebraic structures* such as MV algebras [6], product algebras [19], etc. These algebras are conceived as variants of Boolean algebras; they facilitate algebraic characterizations and eventually lead to Hilbert-style axiomatizations of the fuzzy logics under consideration, cf. [18].

Most computer science applications [40] of fuzzy logic however follow another, more *arithmetically* oriented tradition; fuzzy logics are considered to be and used as mathematical tools for specifying real number constraints on the unit interval. Previous theoretical research in this field includes McNaughton's characterization of Łukasiewicz logic [31] as well as extensive work on triangular norms [22].

This paper is intended to initiate the systematic study of fuzzy logic from the viewpoint of quantifier elimination; thus it contributes to arithmetic fuzzy logic in a genuinely logical manner: We consider first order logic where the first order variables range over the unit interval $[0, 1]$ and the function symbols are interpreted by the arithmetic truth functions of a fuzzy logic. In this framework, propositional fuzzy formulas are the terms of the first order language, and can be considered as *polynomials* in this *fuzzy algebra*.

The motivation for this work is twofold: On the one hand, a first oder framework is the natural way for specifying fuzzy conditions and constraints in many

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 399–414, 1999.

applications. On the other hand, a better understanding of fuzzy semantics is of independent interest for the foundations of fuzzy logics and AI, and may shed light on long time open questions such as the existence of analytic calculi for Łukasiewicz logic.

Let us review the *key concepts* and the *main contributions* of this paper:

Triangular Logics. We focus on fuzzy logics whose truth functions are defined by continuous triangular norms (t-norms) [36,37]. These logics (which we shall call *triangular logics* for short) have attracted a lot of research in recent years, since on the one hand they retain an appealing theory akin to the theory of Boolean algebras in classical logic, while on the other hand they subsume major fuzzy formalisms such as Łukasiewicz logic Ł, Product logic Π, and Gödel logic G. In turn, every continuous t-norm is the ordinal sum of the Łukasiewicz, Product and Gödel t-norms [27]. The most thorough and recent treatment of triangular logics can be found in [18].

First Order Quantifier Elimination. Quantifier elimination is one of the most prominent and important features of logical theories; as van den Dries puts it [10], "experience has shown that in the absence of quantifier elimination the content of completeness and decidability is disappointing." The theory of real closed fields (or Tarski algebra) is the most prominent and well-studied example of a first order theory with quantifier elimination, see [10]. Tarski algebra is also the natural mathematical theory to consider when talking about arithmetic in the unit interval; in fact, interpreting fuzzy logic in Tarski algebra is the usual way to show decidability of many fuzzy logics. Starting with the seminal paper [9], sophisticated *computational methods* have been developed for fast quantifier elimination in real closed fields. However, the worst case complexity of the best algorithms for deciding Tarski algebra is in EXPSPACE (see, e.g. [4,17]), and even deciding existential sentences is PSPACE-complete [5].

Main Contributions. The main contributions of the present paper can be summarized as follows:

- In section 3, we show that a fuzzy algebra has quantifier elimination iff it belongs to one of Ł, Π, or G. The proof for Ł is a geometric argument exploiting McNaughton's characterization of the Ł truth functions.
- This result is followed by mostly model theoretic observations and corollaries aimed at clarifying the algebraic status of fuzzy algebras; we consider categoricity, and in particular show that Ł, Π, and G are $\mathcal{O}$-minimal. Therefore, geometric methods like cell decomposition can be applied [9,34,23]. This gives hope to construct algorithms for quantifier elimination which run fast in practice.
- While traditionally, fuzzy logic has focussed on the real interval - which is natural when dealing with continuous norms – we shall see in Sections 4 and 5, that in fact, rational fuzzy algebras also appear very natural as *Fraïssé approximations* of finite valued logics, and as *prime models* of the theory of

important fuzzy algebras. Moreover, we characterize quantifier elimination in the finitely generated subalgebras; these finitely generated subalgebras correspond to the finite valued Łukasiewicz and Gödel logics.

Related Work. Some results about Łukasiewicz logic – in particular, model completeness as well as the existence of a prime model for Łukasiewicz logic – have already been proven in [25,26] by exploiting the close relationship between Łukasiewicz algebras and Abelian lattice-ordered groups [14,15,8]. We note that our proof methods are different from the mentioned ones, in particular we give a direct proof of quantifier elimination from which most of the other results follow.

Organization of the Paper. In Section 2, we survey triangular logics; most of the presentation is based on [18]. Section 3 contains the proof of quantifier elimination, and Section 4 is devoted to the other model-theoretic and complexity observations. Section 5 deals with approximation by finite-valued logics.

2 Triangular Logics and Extensions

Fuzzy Algebra. The truth functions of the fuzzy logics (*the matrix of the logic*) are defined from a *continuous triangular norm* t, i.e., a two-place function defined on $[0,1]$ which is continuous, and satisfies the following axioms [24]:

T1 $t(0,x) = 0$ and $t(1,x) = x$
T2 $x \leq u \ \wedge \ y \leq z \Longrightarrow t(x,y) \leq t(u,z)$
T3 $t(x,y) = t(y,x)$
T4 $t(t(x,y),z) = t(x,t(y,z))$

Note that continuity is not expressed by the axioms.

In triangular logics, a t-norm t defines the truth function of conjunction ($\&$); implication ($\Rightarrow$) then is defined by the right adjunct of t, i.e., by the unique truth function i such that

I1 $z \leq i(x,y)$ iff $t(x,z) \leq y$

Since t is continuous, we obtain that $i(x,y) = \max\{z | t(x,z) \leq y\}$. Negation ($\neg$) is defined by $i(x,0)$.

Definition Let t be a continuous t-norm. The *fuzzy algebra* generated by t is the algebra

$$\mathbb{F}_t = ([0,1], t, i, 0, 1, \leq),$$

where $[0,1]$ is the real unit interval, and $\leq$ is the ordinary linear order. If the truth functions can be restricted to the rational unit interval, the *rational fuzzy algebra* $\mathbb{Q}\,\mathbb{F}_t$ is defined likewise for the rational unit interval. □

Remarks:

1. We do not distinguish between the truth functions and the syntactical connectives of triangular logics, i.e., we treat fuzzy formulas as **polynomials in the fuzzy algebra**, and use t and & as prefix and infix notation of the same operation.
2. It is easy to see that $x \leq y$ is definable by $i(x, y) = 1$; thus, $\leq$ is just included in the language for convenience. In a minimalistic mood, we could as well remove 1 from the signature.
3. We shall write $x \equiv y$ as a shorthand for $(x \Rightarrow y)\&(y \Rightarrow x)$.

Triangular logics comprise Łukasiewicz, Gödel, and Product Logic as special cases; the truth functions of these logics are summarized in figure 1. The corresponding fuzzy algebras are denoted by (rational) $\mathbb{L}, \mathbb{P}, \mathbb{G}$. (Essentially, $\mathbb{L}$ and Ł are used interchangeably; however we use $\mathbb{L}$, $\mathbb{P}$, and $\mathbb{G}$ when we speak about the fuzzy algebra *per se*, and Ł, Π, G when we speak of the logic in general.)

Ling's Theorem. For a given norm t, let E be the set $\{x \in [0,1] : t(x,x) = x\}$ of idempotencies of t. The complement $[0,1] - E$ is is the union of countably many non-overlapping open intervals, whose closure $cl([0,1] - E)$ corresponds to a class $\{I_i : i \in \mathcal{I}\}$ of closed intervals.

Ling's theorem [27] says that the triangular logics are characterized by those t-norms where the restriction of the t-norm to any interval I_j is isomorphic to either $\mathbb{L}$ or $\mathbb{P}$, and where the value of the t-norm is equal to the minimum norm (Gödel norm) whenever the arguments belong to different intervals. Consult [18] for a formal statement of the theorem.

Therefore, any investigation of triangular logics will focus on these basic triangular logics first.

		Algebra		
Connective	**Case Distinction**	$\mathbb{G}$	$\mathbb{L}$	$\mathbb{P}$
$x\&y$		$\min(x,y)$	$\max(0, x+y-1)$	$x \times y$
$x \Rightarrow y$	$x \leq y$	1	1	1
$x \Rightarrow y$	$x > y$	y	$1 - x + y$	$\frac{y}{x}$
$\neg 0$		1	1	1
$\neg x$	$x > 0$	0	$1 - x$	0

Fig. 1. The truth functions of the basic triangular logics.

As in previous work [2], we consider extensions of triangular logics by $0-1$ projections [1] and root operators [21].

Projections. The projection operators are defined by $\triangle x = \lfloor x \rfloor$ and $\triangledown x = \lceil x \rceil$. Since $\triangledown$ and $\triangle$ are mutually definable in Ł (by $\triangle x = \neg \triangledown \neg x$) and $\triangledown$ is definable already in G and Π, it suffices to consider extensions of those logics by $\triangle$.

The $\triangle$ operator is very powerful because it allows to formulate quantifier free statements about the fuzzy algebra within the fuzzy logic.

$$\begin{aligned} x \leq y &= \triangle(x \Rightarrow y) \\ x = y &= \triangle(x \Rightarrow y) \& \triangle(y \Rightarrow x) \\ x > y &= (x \geq y) \& \neg(x = y) \end{aligned}$$

Thus, $\triangle$ is a kind of *introspection* operator: We say that a formula is *crisp* if every atomic subformula is in the scope of a $\triangle$ operator. The following statement was shown in [2]:

Fact 1 *The crisp formulas of a logic with $\triangle$ and t-norm t can be identified with the quantifier free fragment of first order logic over the fuzzy algebra $\mathbb{F}_t$. Thus, if a fuzzy algebra has quantifier elimination, then all first order properties are expressible in the corresponding triangular logic with $\triangle$.*

Root Operators. The root operators $\Box_i(x) = y$ are defined as the maximal solution of the equation

$$\underbrace{y \& \ldots \& y}_{i \text{ times}} = x.$$

In fact, the solution is unique except for the case of $x = 0$ in Łukasiewicz logic. The root operators for Ł are defined by $\Box_k(x) = \frac{y}{k} + \frac{k-1}{k}$. The root operators for Π are defined by $\Box_k(x) = x^{\frac{1}{k}}$. For G, the roots are just the identity function. It has been shown in [2] that for Ł we do not have to deal with $\Box_k$, but can use the much simpler function $\Box_k^* = \frac{x}{k}$, since $\Box_k$ and $\Box_k^*$ are mutually expressible in Ł.

For a fuzzy algebra $\mathbb{F}$, let $\mathbb{F}^{\Box}$ denote its expansion by all root operators, and $\mathbb{F}^{\triangle}$ its expansion by the projection operator $\triangle$.

Disjunction and McNaughton's Theorem. In the framework of t-norms, disjunction can be defined by the co-norm $1-t(1-x, 1-y)$. However, following previous work [18], we do *not* include disjunction as a basic operation. The following result shows that Gödel conjunction and disjunction (i.e., arithmetic minimum and maximum) are definable in all triangular logics, and thus, disjunction is definable in Gödel logic:

Fact 2 ([18]) *There exists two formulas* $\min(x, y)$ *and* $\max(x, y)$ *which are identic to Gödel conjunction and Gödel disjunction in all triangular logics.*

Therefore, we can use $\min(x, y)$ and $\max(x, y)$ as abbreviations in propositional formulas. As for disjunction in Łukasiewicz logic, we use a very deep and useful algebraic characterization due to McNaughton [31], and its recent generalization [2]:

Theorem 1 *(i)* [31] *A function* $f : [0,1]^n \to [0,1]$ *is representable by an* $\mathbb{L}$*-polynomial if and only if* f *is continuous, and there is a finite number of polynomials*

$$g_i(x_1, \ldots, x_n) = b_i + \mathfrak{a}_i \mathbf{x} = b_i + \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, N$$

with integer coefficients b_i, a_{ij} *such that for every* $n-$*tuple* $(t_1, \ldots, t_n) \in [0,1]^n$ *there exists an* $k, 1 \leq k \leq N$ *with*

$$f(t_1, \ldots, t_n) = g_k(t_1, \ldots, t_n).$$

(ii) [2] A function $f : [0,1]^n \to [0,1]$ *is representable by an* $\mathbb{L}^{\Box}$*-polynomial if and only if* f *is continuous, and there is a finite number of polynomials like in (i), whose coefficients are rational numbers.*

Thus, it follows that Łukasiewicz disjunction is definable in Łukasiewicz logic. As another consequence, it is easy to see that cut-off addition and subtraction can be defined in Ł:

$$x \dot{-} y \;=\; \max(0, x - y) \quad \text{and} \quad x \dot{+} y \;=\; \min(1, x + y).$$

3 Elimination of Quantifiers

3.1 Łukasiewicz Algebra

Theorem 2 Th($\mathbb{L}$) *admits quantifier elimination.*

The following proof uses some terminology and arguments from the theory of polyhedra and linear programming; cf. [39] for background material. In particular recall that a polytope can be described as the intersection of finitely many closed halfspaces; these halfspaces are called the *supporting halfspaces* of the polytope. For brevity, we use the following terminology: By a *popen* polytope we mean the intersection of finitely many **p**ossibly open halfspaces.

Proof: Every equality $f = g$ is equivalent to $f \dot{-} g = 0 \;\wedge\; g \dot{-} f = 0$, and thus we may w.l.o.g. suppose that all equalities have the form $h = 0$. Similarly, every inequality $f \neq g$ can be rewritten as $f \dot{-} g \neq 0 \vee g \dot{-} f \neq 0$. We say that a formula is *reduced* if it is of the form $f = 0$ or $f \neq 0$.

Obviously, it is sufficient to treat the case of $(\exists x)\Phi(x, \mathbf{x})$ where Φ is a conjunction of reduced formulas with free variables among $\mathbf{x} = x_1, \ldots, x_n$.

Consider a conjunction $\bigwedge_{i=1}^{n} f_i(x, \mathbf{x}) = 0$. Clearly, there exists an x satisfying all conditions if there exists one for the atomic formula $\max(f_1, \ldots, f_i) = 0$. Similarly, $\bigwedge_{i=1}^{m} g_i(x, \mathbf{x}) \neq 0$ can be rewritten as $\min(g_1, \ldots, g_m) \neq 0$. Since both min and max are $\mathbb{L}$-polynomials, we have reduced the quantifier elimination to the case of

$$(\exists x) f(x, \mathbf{x}) = 0 \;\wedge\; g(x, \mathbf{x}) \neq 0.$$

Thus, we consider the $k+1$-dimensional set of satisfying assignments $S = \{(x, \mathbf{x}) \mid f(x, \mathbf{x}) = 0 \ \wedge \ g(x, \mathbf{x}) \neq 0\}$, i.e., the formula

$$(\exists x)(x, \mathbf{x}) \in S.$$

From Theorem 1 (i) it follows that in the natural topology on $[0,1]^{k+1}$ the boundaries of S are piecewise linear in the variables $x, \mathbf{x}$. Moreover, S can be written as the finite union of convex popen polytopes $S_1, \ldots, S_s$. Let us therefore without loss of generality suppose that S is a convex popen polytope.

Every convex popen polytope S can be described as the intersection of a finite number of possibly open supporting half-spaces, i.e. as a conjunction $\bigwedge_{i \in I} \pi_i$ of formulas

$$\pi_i := \mathfrak{a}_\mathfrak{i}\mathbf{x} + b_i \prec^i 0, \quad 1 \leq i \leq h$$

where $\prec^i$ is either $<$ or $\leq$. (We need not consider the case of $=$, since it is definable by $\leq$.) Without loss of generality we suppose that among the π_i all formulas of the form $v \geq 0$ and $v \leq 1$ for $v \in \mathbf{x}$ are contained.

For each $i, 1 \leq i \leq h$, π_i is equivalent to one of the following inequations, where f_i is a rational linear polynomial in $\mathbf{x}$:

1. $x \leq f_i(\mathbf{x})$
2. $x < f_i(\mathbf{x})$
3. $x \geq f_i(\mathbf{x})$
4. $x > f_i(\mathbf{x})$

Let the sets $I_\leq$, $I_<$, $I_\geq$, $I_>$ be the subsets of I containing the indices for the different cases.

Then $\bigwedge_{i \in I} \pi_i$ is equivalent to

$$\bigwedge_{i \in I_\leq, j \in I_\geq} f_j \leq f_i \quad \wedge \quad \bigwedge_{i \in I_<, j \in I_\geq} f_j < f_i \quad \wedge$$

$$\wedge \quad \bigwedge_{i \in I_\leq, j \in I_>} f_j < f_i \quad \wedge \quad \bigwedge_{i \in I_<, j \in I_>} f_j < f_i$$

Thus, it remains to show that rational inequations of the form $f_j < f_i$ and $f_j \leq f_i$ can be expressed in $FO(\mathbb{L})$. Let us consider the case of $f_j \leq f_i$. First note that it can be rewritten to the form $h \geq 0$, such that h is a polynomial in $\mathbf{x}$ with *integer* coefficients only. (This can be achieved by multiplication with the least common denominator.) Theorem 1 (i) says that the cut-off version $\max(1, \min(0, h))$ is expressible by an $\mathbb{L}$-polynomial h', and thus, $h' \geq 0$ is the formula in question. The case of $f_j < f_i$ is completely analogous. This completes the proof. □

Daniele Mundici has stated [33] that an alternative proof for quantifier elimination can be obtained using the theory of lattice-ordered groups [14,15] and the Γ operator [8].

3.2 Product Algebra

To simplify the treatment of product logic, t is reasonable to obtain an alternative view of $\mathbb{P}$ than that we had before. Let R^+ denote the set of non-negative real numbers, and R^∞ its extension by an infinite element ∞. Instead of $[0,1]$, we consider truth values from R^∞, such that 0 denotes truth, and ∞ denotes falsity. Using an isomorphism

$$i : x \mapsto \begin{cases} \log x^{-1} & \text{if } x > 0 \\ \infty & \text{if } x = 0 \end{cases}$$

it is an easy exercise for the reader to show that the truth functions are translated into well-known linear functions:

$$x \wedge y = x + y \qquad x \longrightarrow y = y \dot{-} x \qquad \neg x = \infty \dot{-} x = \begin{cases} 0 & \text{if } x = \infty \\ \infty & \text{otherwise} \end{cases}$$

Thus, the well-known operations $+$ and $\dot{-}$ on the real line are extended to ∞ in the straightforward way: $x + \infty = \infty$, $x \dot{-} \infty = 0$, and $\infty \dot{-} x = \infty$ for $x \neq \infty$.

Remark: It is easy to see that this new view of $\mathbb{P}$ also translates to rational fuzzy algebras: although the image $i([0,1]_\mathbb{Q})$ of the rational interval under the mapping i is not a subset of the rationals, $i([0,1]_\mathbb{Q})$ is isomorphic to $\mathbb{Q}\mathbb{P}$.

Note that polynomials without ∞ (i.e., without falsity) are continuous. For the case of ∞, we need the notion of a pre-assignment which was used before in [18] (but not given any name there.) A pre-assignment is an assignment which fixes which variables obtain truth value ∞, and which have (yet unknown) real number truth values.

Lemma 1 *[18] Let $f(\mathbf{x})$ be a Π polynomial, and p be a pre-assignment. Then, under the pre-assignment p, f is equivalent either to ∞ or to a Π polynomial f^p which does not contain ∞.*

An important property of Π polynomials is that after eliminating ∞, they are continuous and homogeneous, i.e., they evaluate to 0 if all their variables are assigned 0.

On the other hand, homogeneous linear polynomials can be expressed by Π polynomials:

Lemma 2 *Let $f \leq g$ be a linear homogeneous inequation with rational coefficients. Then $f \leq g$ can be expressed in* $\mathrm{Th}(\Pi)$.

Proof: First, by moving monoms to the left or right, both sides of the inequation contain additions only. Then, by multiplication with the least common denominator, the inequation has the wished form. □

Corollary 1 *Lemma 2 holds analogously for $f = g$ and $f < g$.*

Theorem 3 Th($\mathbb{P}$) *admits quantifier elimination.*

Proof: Recall that by the above remarks, the rational product algebra $\mathbb{Q}\,\mathbb{P}$ can be identified with a structure over the non-negative rational numbers with additive operations. In fact, we will show in Theorem 8 that $\mathbb{Q}\,\mathbb{P}$ is the unique prime model of the product fuzzy algebra. Therefore, it suffices to consider quantifier elimination over the algebra $\mathbb{Q}\,\mathbb{P}$.

From Lemma 1 it follows that we can for all variables make a case distinction by pre-assignments and therefore without loss of generality consider a formula

$$(\exists x_0) \bigwedge_{0 \leq i \leq n} x_i \neq \infty \ \wedge \ \phi(x_0, x_1, \ldots, x_n)$$

where ϕ does not contain ∞. To avoid notational inconveniences, we shall in the following consider formulas *where all variables are required to range over* Q^+, *i.e., the non-negative rational numbers.*

Again, by the same argument as in the proof of Theorem 2 we can without loss of generality assume the following form:

$$(\exists x_0) f(x_0, \mathbf{x}) = 0 \ \wedge \ g(x_0, \mathbf{x}) \neq 0$$

Since Π-polynomials without ∞ are continuous, we are dealing with homogeneous continuous and piecewise linear polynomials now, and therefore, by similar arguments as for Łukasiewicz logic we can decompose the set S of satisfying assignments into popen convex polytopes, whose boundaries are described by linear polynomials with rational coefficients. The crucial difference here is that *all polynomials in* $+, \dot{-}, 0$ *are homogeneous.* (In other words, all faces of all polytopes contain the origin.) By Lemma 2, such equations are expressible in Th(Π). □

3.3 Gödel Algebra and General t-norms

Theorem 4 Th($\mathbb{G}$) *admits elimination of quantifiers.*

Proof: This is a trivial consequence of quantifier elimination for linear orders with constants for the endpoints. □

Thus, we can give a characterization of fuzzy algebras with quantifier elimination:

Theorem 5 *If a fuzzy algebra has quantifier elimination, then it is isomorphic to one of* $\mathbb{L}$, $\mathbb{P}$ *or* $\mathbb{G}$.

Proof: It remains to show that all other fuzzy algebras do not have quantifier elimination. By Ling's Theorem [27], such logics have at least one idempotence different from $0, 1$. Thus, consider the formula stating "there is no idempotence between x and y", i.e.,

$$(\forall z) x < z < y \longrightarrow t(z, z) \neq z.$$

Obviously, this property is not expressible by a quantifier-free formula. □

3.4 Extensions of Fuzzy Algebras

In a similar way, we obtain quantifier elimination for extended fuzzy algebras:

Corollary 2 *All quantifier elimination results for a fuzzy algebra* $\mathbb{F}$ *remain valid for the extensions* $\mathbb{F}^{\triangle}, \mathbb{F}^{\Box}, \mathbb{F}^{\triangle\Box}$ *by projection and divisibility operators. Moreover, every formula is in fact equivalent to a quantifier-free formula over* $\mathbb{F}$*, i.e., without projection and divisibility operators.*

Proof: Suppose without loss of generality that the formula in question does not contain nested function symbols, and that all atomic formulas are of the form $x = f(\mathbf{x})$. (This can be achieved easily by introducing new *quantified* variables.) Then we can replace every occurrence of $x = \Box_k(y)$ and $x = \triangle y$ by its first order definition. The result then follows from Theorems 2, 3, and 4. □

4 Consequences and Model-Theoretic Observations

Quantifier elimination has many model-theoretic consequences which give us a good idea of fuzzy algebras. We state some of the results without defining the model-theoretic terminology; the reader is referred to [20] for background.

As mentioned in the introduction, some of the following consequences were shown for $\mathbb{L}$ previously in [25,26]:

Corollary 3 *Let* $\mathbb{F}$ *be any of* $\mathbb{L}$, $\mathbb{P}$, *or* $\mathbb{G}$. *Then*

1. $\mathbb{F}$ *is model-complete, i.e., all models of* $\mathrm{Th}(\mathbb{F})$ *are existentially closed models of* $\mathrm{Th}(\mathbb{F})$*, and all embeddings among models of* $\mathrm{Th}(\mathbb{F})$ *are elementary.*
2. $\mathrm{Th}(\mathbb{F})$ *is equivalent to an* $\forall\exists$ *theory.*
3. $\mathrm{Th}(\mathbb{F})$ *has the strong amalgamation property.*
4. $\mathrm{Th}_{\forall}(\mathbb{F})$*, i.e., the substructures of* $\mathrm{Th}(\mathbb{F})$*, have the amalgamation property.*
5. $\mathrm{Th}(\mathbb{F})$ *is not categorical in uncountable powers.*

Proof: Model-completeness is an immediate consequence of quantifier elimination [20, Theorem 8.3.1], and the Chang-Łós-Suszko Theorem [20, 6.5.9] implies item 2. The amalgamation properties follow from [20, Theorem 8.4.1, Corollary 8.6.3]. The result about categoricity follows from $\aleph_0$-instability (due to the presence of an order). □

We shall deal with countable categoricity in Section 5.

In fact, item 2. of Theorem 3 has been made more precise; since [25] is available in Italian only, we repeat the statement:

Theorem 6 ([25]) *Th*$(\mathbb{L})$ *is axiomatized by the following axiom schema:*

1. *The universal closure of the propositional axioms for Łukasiewicz logic, as e.g. in [18].*
2. *Totality of the natural order, i.e.,* $\forall xy\ i(x,y) = 1 \ \vee \ i(y,x) = 1$

3. *Two divisibility axiom schemata where p ranges over all prime numbers, and* px *denotes* $\underbrace{x\dot{+}\ldots\dot{+}x}_{k\text{times}}$*:* $\forall x\exists y\ py = x$ *and* $\exists x(p-1)x = \neg x$

We say that a linearly ordered structure $\mathbb{F}$ is $\mathcal{O}$-minimal, if every first order definable set (with parameters from $\mathbb{F}$) is a *finite union of points and intervals.* $\mathcal{O}$-minimality is an extremely important property which was introduced by van den Dries, and investigated in depth in [34,23]. $\mathcal{O}$-minimality leads to a deep structure theory for definable sets in $n-$dimensional space. It was shown in [23] that if a structure $\mathbb{F}$ is $\mathcal{O}$-minimal, then all models of Th($\mathbb{F}$) are $\mathcal{O}$-minimal. (In other words, the theory of an $\mathcal{O}$-minimal structure is *strongly* $\mathcal{O}$-minimal.) Thus, $\mathcal{O}-$minimality is a very strong property in absence of stability. In fact, an $\mathcal{O}$-minimal theory is categorical with respect to the class of uncountable saturated orders, cf. [23].

Theorem 7 $\mathbb{L}$*,* $\mathbb{P}$*, and* $\mathbb{G}$ *are* $\mathcal{O}$*-minimal.*

Proof: This is an immediate consequence of quantifier elimination. □

$\mathcal{O}$-minimality can also be shown for other fuzzy algebras whose norms are definable in the theory of real closed fields. $\mathcal{O}$-minimality implies the existence of prime models. It is not hard to show that the prime model of Th($\mathbb{F}$) is the rational fuzzy algebra $\mathbb{Q}\mathbb{F}$ for $\mathbb{F} = \mathbb{L}, \mathbb{P}, \mathbb{G}$:

Theorem 8 *Let* $\mathbb{F}$ *be one of* $\mathbb{L}, \mathbb{P}, \mathbb{G}$*. Then* $\mathbb{Q}\mathbb{F}$ *is the unique prime model of* Th($\mathbb{F}$)*.*

Proof: By [7, Theorem 2.3.4], it suffices to show that $\mathbb{Q}\mathbb{F}$ is countable atomic. The latter means that every tuple $a_1, \ldots, a_n$ of domain elements from $\mathbb{Q}\mathbb{F}$ satisfies a complete formula in Th($\mathbb{F}$). In $\mathbb{Q}\mathbb{L}$, every rational number r is first order definable by a formula $\phi_r(x)$ without parameters; suitable Ł-polynomials can be obtained using Theorem 1. Thus, $\bigwedge_{1\leq i\leq n} \phi_{a_i}(x_i)$ is the complete formula in question.

For $\mathbb{Q}\mathbb{G}$, a complete formula just has to state the order type of the variables. Thus, the result is proven for $\mathbb{Q}\mathbb{G}$ as well. (Note that since there are only finitely many order types, it follows that $\mathbb{Q}\mathbb{G}$ is ω-categoric by the Ryll-Nardzewski-Engeler Theorem.)

Let us finally turn to $\mathbb{Q}\mathbb{P}$; as argued above, the domain of $\mathbb{Q}\mathbb{P}$ can be identified with the non-negative rationals. Since ∞ as a constant is trivially first order definable, we can w.l.o.g. consider domain elements $a_1, \ldots, a_n$ different from ∞. Then each of the a_i can be described by a rational equation in the other domain elements. By Corollary 1, this is expressible in $\mathbb{Q}\mathbb{P}$ by a formula $\phi(x_1, \ldots, x_n)$. Since there are no rational constants except 0 in the language, ϕ is complete. □

Of course, quantifier elimination also holds for $\mathbb{Q}\mathbb{L}$, $\mathbb{Q}\mathbb{P}$, and $\mathbb{Q}\mathbb{G}$. For $\mathbb{Q}\mathbb{L}$, we can even say more:

Corollary 4 *If a set is definable in* $\mathbb{Q}\mathbb{L}$ *with parameters, then it is also definable without parameters.*

Proof: Since all rationals in the unit interval are first order definable (cf. the proof of Theorem 8), the result follows by Theorem 2. □

5 Approximation by Finite-Valued Logics

Finite-Valued Triangular Logics. For G and Ł, there exist finite-valued logics which can be constructed as finitely generated subalgebras of the rational fuzzy algebras $\mathbb{Q}\mathbb{G}$ and $\mathbb{Q}\mathbb{L}$. Indeed, for each natural number $i \geq 2$, there exist unique finitely generated subalgebras of size i. The corresponding $i-$valued logics are denoted $\mathbb{G}_i$ and $\mathbb{L}_i$ respectively. It is clear that product logic does not have nontrivial finite subalgebras. Therefore, our treatment of finite-valued logics is necessarily confined to Łukasiewicz and Gödel logic. However, a quite surprising characterization of the finitely generated subalgebras of $\mathbb{Q}\mathbb{P}$ is given below.

General considerations about approximations by finite-valued logics can be found in [3].

Fraïssé Approximations. Let us first review some important facts from model theory which allow a very smooth approximation of rational fuzzy algebras by their finitely generated subalgebras. The exposition heavily relies on [20].

Let $\mathbb{F}$ be a structure. For a set $D \subseteq \mathbb{F}$, let $[D]_{\mathbb{F}}$ (or just $[D]$) denote the substructure of $\mathbb{F}$ generated by D. $\mathbb{F}$ is locally finite, if for finite D, $[D]$ is also finite. $\mathbb{F}$ is uniformly locally finite, if $|[D]|$ is finite, and bound by some function $f(|D|)$. $\mathbb{F}$ is ultrahomogeneous, if every isomorphism between finitely generated substructures of $\mathbb{F}$ extends to an automorphism of $\mathbb{F}$. The class of all finitely generated substructures of $\mathbb{F}$ is called the age age($\mathbb{F}$) of $\mathbb{F}$.

Let now $\mathbb{K}$ be a class of structures, for example the age of some given structure. Then the following properties allow to construct a limit of the structures in $\mathbb{K}$.

HP Hereditary Property.
If $A \in \mathbb{K}$ and B is a finitely generated substructure of A then B is isomorphic to some structure in $\mathbb{K}$.

JEP Joint Embedding Property.
IF $A, B \in \mathbb{K}$ then there is $C \in \mathbb{K}$ such that both A and B are embeddable in C.

AP Amalgamation Property.
If $A, B, C \in \mathbb{K}$, and $e : A \to B, f : A \to C$ are embeddings, then there are $D \in \mathbb{K}$, and embeddings $g : B \to D, h : C \to D$ such that $ge = hf$.

The following theorems about approximation are essentially due to Fraïssé [13]:

Theorem 9 *1. Let τ be a signature, and $\mathbb{K}$ be a non-empty finite or countable set of finitely generated $\tau-$structures which has the HP and the JEP. Then $\mathbb{K}$ is the age of some finite or countable structure.*

2. *If moreover,* τ *is countable and has has the AP, then there is a* $\tau-$*structure* limit($\mathbb{K}$)*, unique up to isomorphism, such that*
   (a) limit($\mathbb{K}$) *has cardinality* $\leq \omega$*,*
   (b) $\mathbb{K} =$ age(limit($\mathbb{K}$))*, and*
   (c) limit($\mathbb{K}$) *is ultrahomogeneous.*
3. *Let* τ *be countable, and* $\mathbb{F}$ *a finite or countable structure which is ultrahomogeneous. Then* age($\mathbb{F}$) *is non-empty, has at most countably many isomorphism types, and satisfies HP, JEP, and AP.*

5.1 Gödel Logic

A first trivial example of this effect is Gödel logic. Since the matrix of Gödel logic is definitionally equivalent with the linear order over $[0, 1]$, it is easy to see that the finite-valued Gödel logics approximate infinite-valued Gödel logic. The following results are very easy:

Theorem 10 $\mathbb{Q}\mathbb{G}$ *is the Fraïssé limit of* age($\mathbb{G}$)*. Therefore, the finite-valued Gödel logics approximate rational Gödel logic, i.e., the* $\mathbb{G}_i$ *satisfy HP, JEP and AP.*

Since ultrahomogeneous finite structures have quantifier elimination [20, Corrolary 8.4.2], we obtain the following result:

Theorem 11 $\mathbb{G}_i$ *has quantifier elimination iff* $i \leq 3$*.*

Proof: Consider the formula saying "there is no intermediate element between x and y", i.e., $\neg(\exists z)x < z < y$. Then this property is not expressible without quantifiers if $i > 3$. □

Corollary 5 $\mathbb{G}_i$ *is ultrahomogeneous iff* $i \leq 3$*.*

Moreover, [20, Corollary 7.4.2] together with Ling's Theorem yield the following:

Corollary 6 $\mathbb{G}$ *is the unique* ω*-categoric fuzzy algebra, i.e. the unique fuzzy algebra* $\mathbb{F}$*, s.t.* Th($\mathbb{F}$) *is* ω*-categoric.*

5.2 Łukasiewicz Logic

Lemma 3 age($\mathbb{Q}\mathbb{L}$) *contains exactly the finite Łukasiewicz algebras.*

The lemma follows from the following exacter statement:

Lemma 4 *[2] Let* $D = \{\frac{p_1}{q_1}, \frac{p_2}{q_2}, \cdots, \frac{p_n}{q_n}\}$ *where* $gcd(p_i, q_i) = 1, q_i \neq 1$ *for all* $1 \leq i \leq n$*. Then* D *generates the subalgebra* $\mathbf{L}_{l+1}$ *of* $\mathbf{L}$*, where* l *is the least common multiple of the denominators* $\{q_1, \ldots, q_n\}$*.*

Corollary 7 $\mathbb{QL}$ *is locally finite, but not uniformly locally finite.*

Proof: Local finiteness is an immediate consequence of Lemma 4. To see that $\mathbb{QL}$ is not uniformly locally finite, just note that $\frac{1}{n}$ generates $\mathbb{L}_{n+1}$, and thus, every $\mathbb{L}_i$ can be generated even by a single element. □

Corollary 8 *All* $\mathbb{L}_i$ *are ultrahomogeneous, and thus, have quantifier elimination.*

Corollary 9 $\mathbb{QL}$ *is ultrahomogeneous.*

The theorem now follows immediately:

Theorem 12 $\mathbb{QL}$ *is the Fraïssé limit of* age($\mathbb{L}$). *Thus, the finite-valued Łukasiewicz logics approximate rational Łukasiewicz logic, i.e., the* $\mathbb{L}_i$ *satisfy HP, JEP and AP.*

For more consequences of Fraïssé approximability, in particular about existentially closed models, consult [20,26].

5.3 Product Logic

Lemma 5 *Up to isomorphism,* $\mathbb{QP}$ *has exactly one finitely generated substructure, i.e.,* $|\text{age}(\mathbb{QP})| = 1$.

Proof: Consider like above $\mathbb{QP}$ over the domain of the non-negative rationals plus infinity, and let $a_1, \ldots, a_n$ be a sequence of generators. Since there are no rational constants except 0 available, we may w.l.o.g. assume that all the a_i are natural numbers, whose greatest common divisor is 1. Therefore, exactly the natural numbers are generated. □

For obvious reasons, we shall call this natural subalgebra $\mathbb{NP}$. Thus, $\mathbb{NP}$ can be written similarly to the fuzzy algebras:

$$(\{1, 2, 3, \ldots\}, +, \dot{-}, \infty, 0, \geq)$$

For trivial reasons, we have approximation results as for $\mathbb{QL}$ and $\mathbb{QG}$:

Fact 3 $\mathbb{QP}$ *is ultrahomogeneous.*

Theorem 13 $\mathbb{QP}$ *is the Fraïssé limit of* $\mathbb{NP}$.

However, in contrast to the other finitely generated substructures, $\mathbb{NP}$ does not have quantifier elimination:

Theorem 14 $\mathbb{NP}$ *does not have quantifier elimination.*

Proof: The same formula as in the proof of Theorem 11 serves as a counterexample. □

Acknowledgements

We are indebted to Daniele Mundici for his numerous comments and for providing us with references [25,26], and to Peter Klement for discussions about t-norms. Moreover, we thank Katrin Seyr and Cristinel Mateis for help in translating [25,26].

References

1. Baaz, M.: Infinite-Valued Gödel Logics with 0-1-Projections and Relativizations. In *Proceedings of Gödel 96 - Kurt Gödel's Legacy*, LNL 6, Springer Verlag, pp.23-33. 402
2. Baaz, M., Veith, H.: Interpolation in Fuzzy Logic. *Archive for Mathematical Logic*, accepted for publication. 402, 403, 403, 403, 404, 411
3. Baaz, M., Zach, R.: Approximating propositional calculi by finite-valued logics. In *Proc. 24th Int. Symp. on Multiple-Valued Logic*, pp. 257-263. IEEE Press, Los Alamitos, May 25-27, 1994. 410
4. Ben-Or, M., Kozen, D.: The Complexity of Elementary Algebra and Geometry. *Journal of Computer and System Sciences* 32, pp. 251-264, 1986. 400
5. Canny, J.: Some Algebraic and Geometric Computations in PSPACE, Proc. STOC 1988, pp. 460-467. 400
6. Chang, C.C.: Algebraic Analysis of Many-Valued Logics. Trans.Amer.Math.Soc. vol. 88 (1958), pp. 467-490. 399
7. Chang, C.C., Keisler, H.J.: Model Theory. North Holland, 1973. 409
8. Cignoli, R., Mundici, D.: An Invitation to Chang's MV Algebras. In *Advances in algebra and model theory*, R. Goebel, M. Droste eds., pp. 171-197, Gordon and Breach, Publishers, 1997. 401, 405
9. Collins, E.G.: Quantifier Elimination for Real Closed Fields by Cylindrical Decomposition. In *Automata Theory and Formal Languages. 2nd GI Conference*, LNCS 33, pp. 134-183, 1975, Springer Verlag. 400, 400
10. van den Dries, L.: Tarski's Elimination Theory for Real Closed Fields. *The Journal of Symbolic Logic* 53/1, 1988, pp. 7-19. 400, 400
11. van den Dries, L.: Algebraic Theories with Definable Skolem Functions. The Journal of Symbolic Logic, Volume 49, Number 2, 1984, pp. 625-629.
12. Dummett, M.: A Propositional Logic with Denumerable Matrix. *Journal of Symbolic Logic* Vol. 24, 1959, 96-107. 399
13. Fraïssé, R.: Sur l'extension aux relations de quelques propriétés des ordres. *Ann.Sci.École Norm. Sup.*, 71, 1954, pp.363-388. 410
14. Glass, A.M.W., Pierce, K.R.: Existentially complete abelian lattice-ordered groups. *Trans. Amer. Math. Society* 261, 255-270, 1980. 401, 405
15. Glass, A.M.W., Pierce, K.R.: Equations and inequations in lattice-ordered groups. In *Ordered Groups*, J.E. Smith, G.O. Kenny, R.N. Ball, eds., pp. 141-171, Dekker, New York, 1980. 401, 405
16. Gödel, K.: Zum Intuitionistischen Aussagenkalkäl. *Ergebnisse eines mathematischen Kolloquiums* 4, 1933, 34-38. 399
17. Grigoriev, D.Yu.: Complexity of Deciding Tarski Algebra. *J.Symbolic Computation* 5, pp. 65-108, 1988. 400
18. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer 1998. 399, 400, 401, 402, 403, 403, 406, 406, 408

19. Hájek, P., Godo, L., Esteva, F.: A complete many-valued logic with product-conjunction. *Archive for Math. Logic 35/3*, 191-208, 1996. 399, 399
20. Hodges, W.: Model Theory. Cambridge University Press 1993. 408, 408, 408, 408, 410, 411, 411, 412
21. Höhle, U.: Commutative residuated l-monoids. In [22], pp. 53-106. 402
22. Höhle, U., and Klement, P., Eds.: Non-classical logics and their applications to fuzzy subsets (A handbook of the mathematical foundations of the fuzzy set theory), Kluwer, 1995. 399, 414
23. Knight, J., Pillay, A., Steinhorn, C.: Definable Sets in Ordered Structures 2, *Transactions of the American Mathematical Society*, 295/2, pp. 593-605. 400, 409, 409, 409
24. Kreiser, L., Gottwald, S., Stelzner, W., eds.: Nichtklassische Logik. Akademie-Verlag, Berlin 1988. 401
25. Lacava, F., Saeli, D.: Sul model-completamento della teoria delle Ł-catene. Bolletino Unione Matematica Italiana (5) 14-A, 107-110, 1977. 401, 408, 408, 408, 413, 413
26. Lacava, F.: Alcune prorietá delle Ł-algebre e delle Ł-algebre esistenzialmente chiuse. Bolletino Unione Matematica Italiana (5) 16-A, 360-366, 1979. 401, 408, 412, 413, 413
27. Ling, H.C.: Representation of associative functions. *Publ. Math. Debrecen 12*, 182-212, 1965. 400, 402, 407
28. Łukasiewicz, J.: Zagadnienia prawdy (The problems of truth). In *Księga pamiątkowa XI zjazdu lekarzy i przyrodników polskich* 1922, 84-85,87. 399
29. Łukasiewicz, J.: Philosophische Bemerkungen zu mehrwertigen Systemen der Aussagenlogik. *Comptes Rendus de la Societe des Science et de Lettres de Varsovie*, cl.iii 23 (1930), 51-77. 399
30. Marker, D.: Introduction to the Model Theory of Fiels. In *Model Theory of Fields*, LNL 5, chapter I, Springer Verlag.
31. McNaughton, R.: A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic 16*, 1-13, 1951. 399, 403, 404
32. Mundici, D.: Satisfiability in Many-Valued Sentential Logic is NP-complete. *Theoretical Computer Science* 52 (1987), pp.145-153.
33. Mundici, D.: Personal Communication. 405
34. Pillay, A., Steinhorn, C.: Definable Sets in Ordered Structures 1, *Transactions of the American Mathematical Society*, 295/2, pp. 565-592. 400, 409
35. Rourke, C.P., Sanderson, B.J.: Introduction to piecewise-linear topology. Springer 1972.
36. Schweizer, B., Sklar, A.: Statistical Metric Spaces. *Pacific Journal of Mathematics 10*, 313-334, 1960. 400
37. Schweizer, B., Sklar, A.: Associative functions and statistical triangle inequalities. *Publicationes Mathematicae Debrecen 8*, 169-186, 1961. 400
38. Tarski, A.: A Decision Method of Elementary Algebra and Geometry. Berkeley, University of California Press, 1951.
39. Yemelichev, V.A., Kovalev, M.M., Kravtsov, M.K.: Polytopes, Graphs and Optmimisation. Cambridge Univ ersity Press, 1984. 404
40. Zadeh, L.A.: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. Selected Papers by Lofti A. Zadeh ed. by G.J. Klir and B. Yuan. World Scientific Publishing, 1996. 399

Many-Valued First-Order Logics with Probabilistic Semantics

Thomas Lukasiewicz

Institut für Informatik, Universität Gießen,
Arndtstraße 2, D-35392 Gießen, Germany
lukasiewicz@informatik.uni-giessen.de

Abstract. We present n-valued first-order logics with a purely probabilistic semantics. We then introduce a new probabilistic semantics of n-valued first-order logics that lies between the purely probabilistic semantics and the truth-functional semantics of the n-valued Łukasiewicz logics Ł$_n$. Within this semantics, closed formulas of classical first-order logics that are logically equivalent in the classical sense also have the same truth value under all n-valued interpretations. Moreover, this semantics is shown to have interesting computational properties. More precisely, n-valued logical consequence in disjunctive logic programs with n-valued disjunctive facts can be reduced to classical logical consequence in $n-1$ layers of classical disjunctive logic programs. Moreover, we show that n-valued logic programs have a model and a fixpoint semantics that are very similar to those of classical logic programs. Finally, we show that some important deduction problems in n-valued logic programs have the same computational complexity like their classical counterparts.

1 Introduction

One way to give semantics to n-valued first-order logics is to use a truth-functional approach and to define the semantics inductively as it is well-known from classical first-order logics. Another way is to use a probabilistic approach as illustrated by the following example.

Let us assume that we have two experts in a certain field and that these two experts fixed a common first-order language to express their knowledge. Moreover, let us assume that the two experts have a common domain over which their first-order language is interpreted, that they also have a common interpretation of all function symbols, but that each of them has its own interpretation of the predicate symbols (that is, each of the two experts stands for a classical first-order interpretation). Finally, let us assume that it is our job to unify the knowledge of the two experts (that is, to define an interpretation that combines the two classical first-order interpretations). Now, there might be closed formulas on which the two experts agree and those on which they disagree: given a closed formula α, it may be the case that they both think that α is **true**, that they both think that α is **false**, or that one of them thinks that α is **true** and the other one that α is **false**. If we assume that both experts are equally credible, then we could model this situation by assigning α the truth values 1, 0, or $\frac{1}{2}$,

G. Gottlob, E. Grandjean, K. Seyr (Eds.): CSL'98, LNCS 1584, pp. 415–430, 1999.

respectively. That is, the truth value of a closed formula α is equal to the relative number of experts who think that α is **true**.

Note that we can similarly merge two finite sets of closed formulas over a common classical first-order language. More precisely, if α is contained in both sets, then the truth value of α is 1. If α is contained in one of the sets, then the truth value of α is greater than or equal to $\frac{1}{2}$. Finally, if α is not contained in any of the sets, then the truth value of α is greater than or equal to 0. A more general view on merging knowledge-bases is given, for example, in [33].

Note also that another way of combining classical first-order interpretations of a finite number of experts is given in [13]. In detail, each formula α of a modal logic is assigned as a truth value the set of all experts who think that α is **true**. That is, the work in [13] keeps track of each single expert, while our approach abstracts to relative numbers of experts (since we are interested in the totally ordered set of truth values $\{\frac{0}{k}, \frac{1}{k}, \ldots, \frac{k}{k}\}$ with $k \geq 2$).

Analyzing our *2-experts world* more deeply, we realize that the truth value assignment to formulas respects the laws of probability. That is, the truth value of a closed formula is in $[0, 1]$, the truth value of all tautologies is 1, the truth value of the disjunction $\alpha \vee \beta$ of mutually exclusive closed formulas α and β is the sum of the truth values of α and β, and logically equivalent closed formulas have the same truth value. But, instead of allowing all real numbers in $[0, 1]$ as truth values, we just consider the members of $\{0, \frac{1}{2}, 1\}$ as truth values.

We also realize that the 2-experts world cannot be expressed by any truth-functional 3-valued logic, since the truth value of the conjunction $\alpha \wedge \beta$ of two closed formulas α and β is generally not a function of the truth values of α and β! For example, if the two experts disagree on a closed formula α, then the truth value of both α and its negation $\neg\alpha$ is $\frac{1}{2}$. Thus, the conjunction $\alpha \wedge \alpha$ also gets the truth value $\frac{1}{2}$, whereas the contradiction $\alpha \wedge \neg\alpha$ always gets the truth value 0.

Finally, we observe that our 2-experts world can easily be generalized to a *k-experts world* with $k \geq 2$ and the $k + 1$ truth values $\frac{0}{k}, \frac{1}{k}, \ldots, \frac{k}{k}$.

We are arrived at the topic of this paper, which is to explore the area between the purely probabilistic and the truth-functional semantics of n-valued first-order logics. Our aim is to bring together the advantages of two different approaches. While the probabilistic formalism provides a well-defined semantics, truth-functional logics are often more compelling from the computational side (see, for example, [22] and [23] for the subtleties of probabilistic propositional deduction).

In this paper, we now present a new probabilistic semantics of n-valued first-order logics that lies between the purely probabilistic semantics and the truth-functional semantics given by the n-valued Łukasiewicz logics Ł_n. More precisely, the main contributions of this paper can be summarized as follows:

- We present n-valued first-order logics with a purely probabilistic semantics.
- We introduce $\text{Ł}_n^\star$-interpretations, which give n-valued first-order formulas a new probabilistic semantics that lies between the purely probabilistic semantics and the truth-functional semantics of the n-valued Łukasiewicz logics Ł_n.
- The introduced $\text{Ł}_n^\star$-interpretations have the following nice property: closed formulas of classical first-order logics that are logically equivalent in the classical sense also have the same truth value under all $\text{Ł}_n^\star$-interpretations.

- We show that disjunctive logic programming with n-valued disjunctive facts in $\text{Ł}_n^\star$ is reducible to classical disjunctive logic programming in $n-1$ layers.
- We present a model and a fixpoint semantics for n-valued logic programming in $\text{Ł}_n^\star$, which are very similar to those of classical logic programming.
- We show that some important special cases of n-valued logic programming in $\text{Ł}_n^\star$ have the same computational complexity like their classical counterparts.

The rest of this paper is organized as follows. In Section 2, we present n-valued first-order logics with purely probabilistic semantics. In Section 3, we introduce $\text{Ł}_n^\star$-interpretations. Sections 4 and 5 discuss disjunctive logic programs with n-valued disjunctive facts and n-valued logic programs, respectively. In Section 6, we discuss important related work. In Section 7, we summarize the main results and give an outlook on future research.

2 Many-Valued First-Order Logics

Let Φ be a first-order vocabulary that contains a set of function symbols and a set of predicate symbols. Let $\mathcal{X}$ be a set of variables. The first part of the following definitions are the usual ones from classical first-order logics. We briefly introduce them here to fix a clear technical background.

We define *terms* by induction as follows. A term is a variable from $\mathcal{X}$ or an expression of the kind $f(t_1,\ldots,t_k)$, where f is a function symbol of arity $k \geq 0$ from Φ and $t_1,\ldots,t_k$ are terms. *Formulas* are defined by induction as follows. If p is a predicate symbol of arity $k \geq 0$ from Φ and $t_1,\ldots,t_k$ are terms, then $p(t_1,\ldots,t_k)$ is a formula (called *atomic formula*). If F and G are formulas, then $\neg F$, $(F \wedge G)$, $(F \vee G)$, and $(F \leftarrow G)$ are formulas. If F is a formula and x is a variable from $\mathcal{X}$, then $\forall x\, F$ and $\exists x\, F$ are formulas.

An *interpretation* $I = (D, \pi)$ consists of a nonempty set D, called *domain*, and a mapping π that assigns to each function symbol from Φ a function of right arity over D and to each predicate symbol from Φ a predicate of right arity over D. A *variable assignment* σ is a mapping that assigns to each variable from $\mathcal{X}$ an element from D. For a variable x from $\mathcal{X}$ and an element d from D, we write $\sigma[x/d]$ to denote the variable assignment that is identical to σ except that it assigns d to x. The variable assignment σ is by induction extended to terms by $\sigma(f(t_1,\ldots,t_k)) = \pi(f)(\sigma(t_1),\ldots,\sigma(t_k))$ for all terms $f(t_1,\ldots,t_k)$. The *truth* of formulas F in I under σ, denoted $I \models_\sigma F$, is inductively defined as follows:

- $I \models_\sigma p(t_1,\ldots,t_k)$ iff $(\sigma(t_1),\ldots,\sigma(t_k)) \in \pi(p)$.
- $I \models_\sigma \neg F$ iff not $I \models_\sigma F$, and $I \models_\sigma (F \wedge G)$ iff $I \models_\sigma F$ and $I \models_\sigma G$.
- $I \models_\sigma \forall x\, F$ iff $I \models_{\sigma[x/d]} F$ for all $d \in D$.
- The truth of the remaining formulas in I under σ is defined by expressing $\vee$, $\leftarrow$, and $\exists$ in terms of $\neg$, $\wedge$, and $\forall$ as usual.

A formula F *is true* in I, or I is a *model* of F, denoted $I \models F$, iff F is true in I under all variable assignments σ. I is a *model* of a set of formulas $\mathcal{F}$, denoted $I \models \mathcal{F}$, iff I is a model of all formulas in $\mathcal{F}$. $\mathcal{F}$ is *satisfiable* iff a model of $\mathcal{F}$ exists. A formula F is a *logical consequence* of a set of formulas $\mathcal{F}$, denoted $\mathcal{F} \models F$, iff each model of $\mathcal{F}$ is also a model of F.

We adopt the classical conventions to eliminate parentheses. Moreover, universally quantified formulas are defined as usual. Finally, the classical notions of ground terms, ground formulas, and ground instances of formulas are adopted.

2.1 Many-Valued First-Order Logics of Probabilities in $\mathbf{Pr}_n$

Next, we introduce a purely probabilistic approach to n-valued first-order logics. For this purpose, let $TV = \{0, \frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1\}$ with $n \geq 3$ denote the set of *truth values*. Note that the classical truth values **false** and **true** correspond to 0 and 1, respectively. We define the syntax and the probabilistic semantics of n-valued formulas as follows:

An *n-valued formula* is an expression of the kind $tv(F) \geq c$ with a classical formula F and a truth value c from TV (note that $tv(F) \geq c$ will mean that the truth value of F is greater than or equal to c, while $tv(F) \geq c$ together with $tv(\neg F) \geq 1 - c$ will express that the truth value of F is exactly c).

A *Pr_n-interpretation* $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ consists of $n-1$ interpretations $(D, \pi_1), \ldots, (D, \pi_{n-1})$ over the same domain D such that $\pi_1(f) = \cdots = \pi_{n-1}(f)$ for all function symbols f from Φ. The *truth value* $\mathcal{I}_\sigma(F)$ of a formula F in the Pr_n-interpretation $\mathcal{I}$ under a variable assignment σ is defined by:

$$\mathcal{I}_\sigma(F) \;=\; |\{i \in [1\,{:}\,n-1] \mid (D, \pi_i) \models_\sigma F\}| \,/\, (n-1)\,.$$

An n-valued formula $tv(F) \geq c$ *is true* in $\mathcal{I}$ under σ iff $\mathcal{I}_\sigma(F) \geq c$. An n-valued formula N *is true* in $\mathcal{I}$, or $\mathcal{I}$ is a *model* of N, denoted $\mathcal{I} \models N$, iff N is true in $\mathcal{I}$ under all variable assignments σ. $\mathcal{I}$ is a *model* of a set of n-valued formulas $\mathcal{N}$, denoted $\mathcal{I} \models \mathcal{N}$, iff $\mathcal{I}$ is a model of all n-valued formulas in $\mathcal{N}$. $\mathcal{N}$ is *satisfiable* iff a model of $\mathcal{N}$ exists. N is a *logical consequence* of $\mathcal{N}$, denoted $\mathcal{N} \models N$, iff each model of $\mathcal{N}$ is also a model of N.

Finally, note that we canonically define ground n-valued formulas and ground instances of n-valued formulas.

2.2 Many-Valued First-Order Logics in the Łukasiewicz Logics $\mathbf{Ł}_n$

Among all the truth-functional n-valued logics, the Łukasiewicz logic $\mathrm{Ł}_n$ seems to be the one closest in spirit to our n-valued logic of probabilities. We now briefly describe how n-valued formulas are interpreted in $\mathrm{Ł}_n$. For this purpose, we just have to define $\mathrm{Ł}_n$-interpretations and the truth value of n-valued formulas in $\mathrm{Ł}_n$-interpretations under variable assignments. Note that our $\mathrm{Ł}_n$-interpretations are defined in a nonstandard way by adapting Pr_n-interpretations (of course, this approach is technically quite clumsy, but it will be useful to understand the relationship between Pr_n- and $\mathrm{Ł}_n$-interpretations).

An *$Ł_n$-interpretation* $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ consists of $n-1$ interpretations $(D, \pi_1), \ldots, (D, \pi_{n-1})$ over the same domain D such that $\pi_1(f) = \cdots = \pi_{n-1}(f)$ for all function symbols f from Φ. The *truth value* $\mathcal{I}_\sigma(F)$ of a formula F in the $\mathrm{Ł}_n$-interpretation $\mathcal{I}$ under a variable assignment σ is inductively defined by:

- $\mathcal{I}_\sigma(p(t_1, \ldots, t_k)) \;=\; |\{i \in [1\,{:}\,n-1] \mid (D, \pi_i) \models_\sigma p(t_1, \ldots, t_k)\}| \,/\, (n-1)\,.$
- $\mathcal{I}_\sigma(\neg F) \;=\; 1 - \mathcal{I}_\sigma(F)$ and $\mathcal{I}_\sigma(F \wedge G) \;=\; \min(\mathcal{I}_\sigma(F), \mathcal{I}_\sigma(G))$.

- $\mathcal{I}_\sigma(F \vee G) = \max(\mathcal{I}_\sigma(F), \mathcal{I}_\sigma(G))$ and $\mathcal{I}_\sigma(F \leftarrow G) = \min(1, \mathcal{I}_\sigma(F) - \mathcal{I}_\sigma(G) + 1)$.
- $\mathcal{I}_\sigma(\forall x\, F) = \min\{\mathcal{I}_{\sigma[x/d]}(F) \mid d \in D\}$.
- $\mathcal{I}_\sigma(\exists x\, F) = \max\{\mathcal{I}_{\sigma[x/d]}(F) \mid d \in D\}$.

3 Many-Valued First-Order Logics of Probabilities in Ł$^\star_n$

In this section, we define a new probabilistic semantics of n-valued formulas that lies between Pr_n- and Ł$_n$-interpretations. In this new semantics, we keep the nice property of Pr_n-interpretations that closed formulas of the same classical truth value under all classical interpretations are also of the same n-valued truth value under all n-valued interpretations. In particular, contradictory and tautological closed formulas always have the truth values 0 and 1, respectively.

The new semantics is obtained by adapting Pr_n-interpretations to Ł$_n$-interpretations. More precisely, we give up some properties of Ł$_n$-interpretations (to keep the above-mentioned nice properties of Pr_n-interpretations) by being more cautious in using the truth tables of Ł$_n$. That is, not all logical combinations of formulas are truth-functionally interpreted. We just interpret logical combinations of ground atomic formulas by the truth tables of Ł$_n$. To realize this, we must impose a certain restriction on Pr_n-interpretations:

Theorem 1 *Let $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ be a Pr_n-interpretation and let σ be a variable assignment. Let each $d \in D$ have a ground term t_d with $\sigma(t_d) = d$.*

It holds $\mathcal{I}_\sigma(A \wedge B) = \min(\mathcal{I}_\sigma(A), \mathcal{I}_\sigma(B))$ for all ground atomic formulas A and B iff there exists a permutation μ of the elements in $[1\!:\!n-1]$ with $\pi_{\mu(1)}(p) \supseteq \pi_{\mu(2)}(p) \supseteq \cdots \supseteq \pi_{\mu(n-1)}(p)$ for all predicate symbols p from Φ.

Proof. '$\Leftarrow$': Let A and B be ground atomic formulas. If μ is a permutation of the elements in $[1\!:\!n-1]$ with $\pi_{\mu(1)}(p) \supseteq \pi_{\mu(2)}(p) \supseteq \cdots \supseteq \pi_{\mu(n-1)}(p)$ for all predicate symbols p from Φ, then $(D, \pi_{\mu(i)}) \models_\sigma A$ and $(D, \pi_{\mu(i)}) \models_\sigma B$ for exactly all $1 \leq i \leq \mathcal{I}_\sigma(A) \cdot (n-1)$ and $1 \leq i \leq \mathcal{I}_\sigma(B) \cdot (n-1)$, respectively. Hence, we get $(D, \pi_{\mu(i)}) \models_\sigma A \wedge B$ for exactly all $1 \leq i \leq \min(\mathcal{I}_\sigma(A), \mathcal{I}_\sigma(B)) \cdot (n-1)$.

'$\Rightarrow$': For all ground atomic formulas A let $I_A = \{i \in [1\!:\!n-1] \mid (D, \pi_i) \models_\sigma A\}$. For all ground atomic formulas A and B: if $\mathcal{I}_\sigma(A \wedge B) = \min(\mathcal{I}_\sigma(A), \mathcal{I}_\sigma(B))$, then $\mathcal{I}_\sigma(A) \leq \mathcal{I}_\sigma(B)$ iff $I_A \subseteq I_B$. Hence, we can define the desired permutation μ by $\mu(i) = \min((\bigcap\{I_A \mid A \in GAF, \mathcal{I}_\sigma(A) \cdot (n-1) \geq i\}) \setminus \{\mu(1), \ldots, \mu(i-1)\})$ for all $i \in [1\!:\!n-1]$, where GAF denotes the set of all ground atomic formulas. Note that we canonically assume $\bigcap \emptyset = [1\!:\!n-1]$. □

3.1 Ł$^\star_n$-Interpretations

Inspired by Theorem 1, we can now introduce a natural probabilistic semantics of n-valued formulas that lies between the purely probabilistic semantics of Pr_n-interpretations and the truth-functional semantics of Ł$_n$-interpretations. We do this by defining Ł$^\star_n$-interpretations and the truth value of n-valued formulas in Ł$^\star_n$-interpretations under variable assignments.

An *Ł$^\star_n$-interpretation* $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ consists of $n-1$ interpretations $(D, \pi_1), \ldots, (D, \pi_{n-1})$ over the same domain D such that $\pi_1(f) = \cdots = \pi_{n-1}(f)$

for all function symbols f from Φ and

$$\pi_1(p) \supseteq \pi_2(p) \supseteq \cdots \supseteq \pi_{n-1}(p) \text{ for all predicate symbols } p \text{ from } \Phi. \quad (1)$$

The *truth value* $\mathcal{I}_\sigma(F)$ of a formula F in the $\text{Ł}_n^\star$-interpretation $\mathcal{I}$ under a variable assignment σ is defined by:

$$\mathcal{I}_\sigma(F) \;=\; |\{i \in [1\,{:}\,n-1] \,|\, (D, \pi_i) \models_\sigma F\}| \,/\, (n-1)\,.$$

3.2 Properties of $\text{Ł}_n^\star$-Interpretations

In this subsection, we explore some basic properties of $\text{Ł}_n^\star$-interpretations. We start with showing that (1) is propagated through the structure of all formulas that are built without the logical connectives $\neg$ and $\leftarrow$.

We say that the truth value of a formula F is *compressed* in the $\text{Ł}_n^\star$-interpretation $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ under the variable assignment σ iff

$$\mathcal{I}_\sigma(F) \;=\; \max(\{i \in [1\,{:}\,n-1] \,|\, (D, \pi_i) \models_\sigma F\} \cup \{0\}) \,/\, (n-1)\,.$$

Lemma 2 *For all formulas F that are built without the logical connectives $\neg$ and $\leftarrow$, all $\text{Ł}_n^\star$-interpretations $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$, and all variable assignments σ, the truth value of F is compressed in $\mathcal{I}$ under σ.*

Proof. The claim is proved by induction on the structure of all formulas that are built without the logical connectives $\neg$ and $\leftarrow$. Let $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$ be an $\text{Ł}_n^\star$-interpretations and let σ be a variable assignment.

If F is an atomic formula $p(t_1, \ldots, t_k)$, then the truth value of F is compressed in $\mathcal{I}$ under σ by the definition of $\text{Ł}_n^\star$-interpretations. If F is of the form $G \wedge H$ or $G \vee H$, then the truth values of G and H are compressed in $\mathcal{I}$ under σ by the induction hypothesis. Hence, also the truth values of $G \wedge H$ and $G \vee H$ are compressed in $\mathcal{I}$ under σ. Moreover, we get $\mathcal{I}_\sigma(G \wedge H) = \min(\mathcal{I}_\sigma(G), \mathcal{I}_\sigma(H))$ and $\mathcal{I}_\sigma(G \vee H) = \max(\mathcal{I}_\sigma(G), \mathcal{I}_\sigma(H))$. If F is of the form $\forall x\, G$ or $\exists x\, G$, then for all $d \in D$ the truth value of G is compressed in $\mathcal{I}$ under $\sigma[x/d]$ by the induction hypothesis. Hence, also the truth values of $\forall x\, G$ and $\exists x\, G$ are compressed in $\mathcal{I}$ under σ. Furthermore, we get $\mathcal{I}_\sigma(\forall x\, G) = \min\{\mathcal{I}_{\sigma[x/d]}(G) \,|\, d \in D\}$ and $\mathcal{I}_\sigma(\exists x\, G) = \max\{\mathcal{I}_{\sigma[x/d]}(G) \,|\, d \in D\}$. □

Next, we focus on the relationship between $\text{Ł}_n^\star$- and Ł_n-interpretations. We show that $\text{Ł}_n^\star$- and Ł_n-interpretations coincide in the semantics they give to all the formulas that are built without the logical connectives $\neg$ and $\leftarrow$ and in the semantics they give to all logical combinations of these formulas.

Lemma 3 *For all formulas F and G that are built without the logical connectives $\neg$ and $\leftarrow$, all variables x in $\mathcal{X}$, all $\text{Ł}_n^\star$-interpretations $\mathcal{I} = (D, \pi_1, \ldots, \pi_{n-1})$, and all variable assignments σ:*

$$\mathcal{I}_\sigma(\neg F) = 1 - \mathcal{I}_\sigma(F) \quad (2)$$

$$\mathcal{I}_\sigma(F \wedge G) = \min(\mathcal{I}_\sigma(F), \mathcal{I}_\sigma(G)) \quad (3)$$

$$\mathcal{I}_\sigma(F \vee G) = \max(\mathcal{I}_\sigma(F), \mathcal{I}_\sigma(G)) \quad (4)$$

$$\mathcal{I}_\sigma(F \leftarrow G) = \min(1, \mathcal{I}_\sigma(F) - \mathcal{I}_\sigma(G) + 1) \quad (5)$$

$$\mathcal{I}_\sigma(\forall x\, F) = \min\{\mathcal{I}_{\sigma[x/d]}(F) \mid d \in D\} \tag{6}$$
$$\mathcal{I}_\sigma(\exists x\, F) = \max\{\mathcal{I}_{\sigma[x/d]}(F) \mid d \in D\}\,. \tag{7}$$

Proof. The claim (2) is immediate (for all formulas F) by the definition of the truth value of a formula. The claims (3) and (4) hold by the proof of Lemma 2. The claim (5) follows from $\mathcal{I}_\sigma(F \leftarrow G) = \mathcal{I}_\sigma(F \vee \neg G)$ (since the implication connective $\leftarrow$ is semantically expressed in terms of $\neg$ and $\vee$), from $\mathcal{I}_\sigma(F \vee \neg G) = \mathcal{I}_\sigma(F \wedge G) - \mathcal{I}_\sigma(G) + 1$ by the definition of the truth value of a formula, and from (3). Finally, the claims (6) and (7) hold by the proof of Lemma 2. □

Note that Lemma 3 does not hold for all formulas F and G (that is, Ł$_n^\star$- and Ł$_n$-interpretations do not coincide completely in the semantics they give to classical first-order formulas). More precisely, the semantics of Ł$_n^\star$-interpretations cannot be defined inductively by *any* truth table. We give a counter-example to show this. If F and G are atomic formulas with $\mathcal{I}_\sigma(F) = \frac{1}{n-1}$ and $\mathcal{I}_\sigma(G) = \frac{n-2}{n-1}$, then we get $\mathcal{I}_\sigma(F \wedge G) = \min(\mathcal{I}_\sigma(F), \mathcal{I}_\sigma(G)) = \frac{1}{n-1}$. However, also $\mathcal{I}_\sigma(F) = \frac{1}{n-1}$ and thus $\mathcal{I}_\sigma(\neg F) = \frac{n-2}{n-1}$, but it always holds $\mathcal{I}_\sigma(F \wedge \neg F) = 0$.

Especially for handling uncertain beliefs, Ł$_n^\star$-interpretations provide a more intuitive semantics than Ł$_n$-interpretations, as the following example shows.

3.3 Example

Assume an agent has the belief $\frac{1}{2}$ in an event F. Ł$_n^\star$-interpretations then assign the belief 0 to the false event $F \wedge \neg F$ and the belief 1 to the true event $F \vee \neg F$. Ł$_n$-interpretations, however, assign the belief $\frac{1}{2}$ to both $F \wedge \neg F$ and $F \vee \neg F$.

Assume now an agent has the belief $\frac{1}{2}$ in the events F and G. Ł$_n^\star$-interpretations then assign the same belief 1 to the logically equivalent events $F \leftarrow G$ and $F \vee \neg G$. Ł$_n$-interpretations, however, assign the different beliefs 1 and $\frac{1}{2}$ to $F \leftarrow G$ and $F \vee \neg G$, respectively.

4 Disjunctive Logic Programming with Many-Valued Disjunctive Facts in Ł$_n^\star$

In the previous section, we introduced Ł$_n^\star$-interpretations and we showed that their characterizing property (1) is propagated through the structure of all the formulas that are built without $\neg$ and $\leftarrow$. Hence, each n-valued knowledge-base that just contains n-valued formulas of the form $tv(F) \geq 1$ and n-valued formulas that are built without $\neg$ and $\leftarrow$ has a unique splitting into $n-1$ layers of classical knowledge-bases. Moreover, as we show in this section, in disjunctive logic programs with n-valued disjunctive facts, we can reduce logical consequences in Ł$_n^\star$ to classical logical consequences in $n-1$ layers of classical disjunctive logic programs. This result is of particular interest for handling uncertain knowledge in disjunctive deductive databases, since it can directly be used to generalize their extensional part to many-valued disjunctive facts.

4.1 Logical Consequence in Ł$_n^\star$

A *disjunctive fact* is a universally quantified formula $\forall(A_1 \vee \cdots \vee A_k)$ with atomic formulas $A_1, \ldots, A_k$ and $k \geq 1$. A *disjunctive program clause* is a universally

quantified formula $\forall(A_1 \vee \cdots \vee A_k \vee \neg B_1 \vee \cdots \vee \neg B_l)$ with atomic formulas $A_1, \ldots, A_k, B_1, \ldots, B_l$, where $k \geq 1$ and $l \geq 0$. An *n-valued disjunctive fact* is an n-valued formula $tv(F) \geq c$ with a disjunctive fact F. A *disjunctive logic program with n-valued disjunctive facts* is a finite set of n-valued formulas $tv(P) \geq 1$ with disjunctive program clauses P and of n-valued disjunctive facts.

In the sequel, let $\mathcal{P}$ be a disjunctive logic program with n-valued disjunctive facts. Let the Herbrand base $HB_{\mathcal{P}}$ comprise all ground atomic formulas that are constructed with function symbols and predicate symbols from $\mathcal{P}$ (we assume that $\mathcal{P}$ contains at least one 0-ary predicate symbol or at least one predicate symbol and one 0-ary function symbol). We subsequently restrict our considerations to *Herbrand $\text{Ł}_n^\star$-interpretations* $\mathcal{I} = (I_1, \ldots, I_{n-1})$, which consist of $n-1$ classical Herbrand interpretations $I_1, \ldots, I_{n-1}$ with $HB_{\mathcal{P}} \supseteq I_1 \supseteq \cdots \supseteq I_{n-1}$.

We now split $\mathcal{P}$ into $n-1$ layers of disjunctive logic programs. For each truth value $d > 0$ from TV let $\mathcal{P}_d$ contain all disjunctive program clauses P for which there exists a truth value $c \geq d$ from TV with $tv(P) \geq c \in \mathcal{P}$.

The next theorem shows the crucial result that certain logical consequences in $\text{Ł}_n^\star$ in disjunctive logic programs with n-valued disjunctive facts can be reduced to logical consequences in $n-1$ layers of classical disjunctive logic programs.

Theorem 4 *Let $\mathcal{P}$ be a disjunctive logic program with n-valued disjunctive facts. For all n-valued disjunctive facts $tv(F) \geq d$ with $d > 0$:*

$$\mathcal{P} \models tv(F) \geq d \text{ in } \text{Ł}_n^\star \text{ iff } \mathcal{P}_d \models F\,.$$

Proof. '$\Leftarrow$': Let $\mathcal{I} = (I_1, \ldots, I_{n-1})$ be a model of $\mathcal{P}$. Then, we get $I_i \models P$ for all $tv(P) \geq c \in \mathcal{P}$ and all $1 \leq i \leq c \cdot (n-1)$ (here, we use that $\mathcal{P}$ contains only n-valued disjunctive facts and n-valued formulas of the form $tv(P) \geq 1$). Hence, the classical Herbrand interpretation $I_{d\cdot(n-1)}$ is a model of $\mathcal{P}_d$. Since $\mathcal{P}_d$ logically entails F, the interpretation $I_{d\cdot(n-1)}$ is also a model of F. Hence, since $I_1 \supseteq \cdots \supseteq I_{n-1}$, each classical Herbrand interpretation I_i with $1 \leq i \leq d \cdot (n-1)$ is a model of F. Thus, we also get $\mathcal{I} \models tv(F) \geq d$.

'$\Rightarrow$': Let I be a classical Herbrand interpretation that is a model of $\mathcal{P}_d$. We define a model $\mathcal{I} = (I_1, \ldots, I_{n-1})$ of $\mathcal{P}$ as follows. Let $I_i = HB_{\mathcal{P}}$ for all $1 \leq i < d \cdot (n-1)$ (this works, since $\mathcal{P}$ is a disjunctive logic program with n-valued disjunctive facts) and $I_i = I$ for all $d \cdot (n-1) \leq i \leq n-1$ (this would also work for more general n-valued formulas in $\mathcal{P}$). Now, since $\mathcal{P}$ logically entails $tv(F) \geq d$, each classical Herbrand interpretation I_i with $1 \leq i \leq d \cdot (n-1)$ is a model of F. Hence, in particular the interpretation $I_{d\cdot(n-1)}$ is a model of F. That is, the interpretation I is a model of F. □

Finally, Theorem 4 also shows that logical consequences in $\text{Ł}_n^\star$ in disjunctive logic programs with n-valued disjunctive facts are independent of the number n of truth values. More precisely, for all disjunctive logic programs with n-valued disjunctive facts $\mathcal{P}$, n-valued disjunctive facts $tv(F) \geq d$, and $m = kn - k + 1$ with $k \geq 1$, we get $\mathcal{P} \models tv(F) \geq d$ in $\text{Ł}_n^\star$ iff $\mathcal{P} \models tv(F) \geq d$ in $\text{Ł}_m^\star$.

4.2 Example

Let $\mathcal{P}$ be the following disjunctive logic program with 4-valued disjunctive facts (a, b, and c are 0-ary function symbols, p, q, r, and s are 1-ary predicate symbols, t is a 2-ary predicate symbol, and X and Y are variables):

$$\mathcal{P}' = \{t(X,Y) \leftarrow q(X) \wedge r(Y),\ t(X,Y) \leftarrow q(X) \wedge s(Y),\ r(Y) \vee s(Y) \leftarrow p(Y)\}$$
$$\mathcal{P} = \{tv(P) \geq 1 \mid P \in \mathcal{P}'\} \cup \{tv(p(b)) \geq 2/3,\ tv(q(a)) \geq 2/3,\ tv(r(c) \vee s(c)) \geq 1/3\}\,.$$

We get the following three layers of classical disjunctive logic programs:

$$\mathcal{P}_{1/3} = \mathcal{P}' \cup \{p(b),\ q(a),\ r(c) \vee s(c)\}$$
$$\mathcal{P}_{2/3} = \mathcal{P}' \cup \{p(b),\ q(a)\}$$
$$\mathcal{P}_{1} = \mathcal{P}'\,.$$

We easily verify that $\mathcal{P}_{1/3} \models t(a,c)$ and $\mathcal{P}_{2/3} \models t(a,b)$. Hence, by Theorem 4, $\mathcal{P} \models tv(t(a,c)) \geq 1/3$ and $\mathcal{P} \models tv(t(a,b)) \geq 2/3$ in $\text{Ł}_4^\star$ (and also in $\text{Ł}_7^\star$, $\text{Ł}_{10}^\star$, ...). Moreover, we easily verify that neither $\mathcal{P}_{2/3} \models t(a,c)$ nor $\mathcal{P}_1 \models t(a,b)$. Hence, by Theorem 4, neither $\mathcal{P} \models tv(t(a,c)) \geq 2/3$ nor $\mathcal{P} \models tv(t(a,b)) \geq 1$ in $\text{Ł}_4^\star$ (and these logical consequences also do not hold in $\text{Ł}_7^\star$, $\text{Ł}_{10}^\star$, ...).

5 Many-Valued Logic Programming in $\text{Ł}_n^\star$

In this section, we focus on n-valued logic programming in $\text{Ł}_n^\star$. Differently from disjunctive logic programming with n-valued facts in $\text{Ł}_n^\star$, it cannot be reduced to classical logic programming in $n-1$ layers. However, the semantic background of $\text{Ł}_n^\star$-interpretations easily provides a model and a fixpoint semantics for n-valued logic programming in $\text{Ł}_n^\star$, which generalize the model and the fixpoint semantics of classical logic programming (see, for example, [1]).

5.1 Model and Fixpoint Semantics

A *program clause* is a universally quantified formula $\forall(A \vee \neg B_1 \vee \cdots \vee \neg B_l)$ with atomic formulas $A, B_1, \ldots, B_l$ and $l \geq 0$. An *n-valued program clause* is an n-valued formula $tv(P) \geq c$ with a program clause P. We abbreviate n-valued program clauses $tv(\forall(H \vee \neg B_1 \vee \cdots \vee \neg B_k)) \geq c$ by $(H \leftarrow B_1 \wedge \cdots \wedge B_k)[c,1]$. An *$n$-valued logic program* is a finite set of n-valued program clauses.

In the sequel, let $\mathcal{P}$ be an n-valued logic program and let the Herbrand base $HB_\mathcal{P}$ be defined like in Section 4.1. Again, we restrict our considerations to Herbrand $\text{Ł}_n^\star$-interpretations, which are now identified with fuzzy sets [35] that are mappings from $HB_\mathcal{P}$ to TV. More precisely, each Herbrand $\text{Ł}_n^\star$-interpretation $(I_1, I_2, \ldots, I_{n-1})$ is identified with the fuzzy set $\boldsymbol{I}\colon HB_\mathcal{P} \to TV$ that is defined by $\boldsymbol{I}(A) = \max(\{c \in TV \setminus \{0\} \mid A \in I_{c\cdot(n-1)}\} \cup \{0\})$ for all $A \in HB_\mathcal{P}$. We subsequently use bold symbols to denote such fuzzy sets.

The fuzzy sets $\boldsymbol{\emptyset}$ and $\boldsymbol{HB}_\mathcal{P}$ are defined by $\boldsymbol{\emptyset}(A) = 0$ and $\boldsymbol{HB}_\mathcal{P}(A) = 1$ for all $A \in HB_\mathcal{P}$. Finally, we define the intersection, the union, and the subset relation for fuzzy sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ as usual by $\boldsymbol{S}_1 \cap \boldsymbol{S}_2 = \min(\boldsymbol{S}_1, \boldsymbol{S}_2)$, $\boldsymbol{S}_1 \cup \boldsymbol{S}_2 = \max(\boldsymbol{S}_1, \boldsymbol{S}_2)$, and $\boldsymbol{S}_1 \subseteq \boldsymbol{S}_2$ iff $\boldsymbol{S}_1 = \boldsymbol{S}_1 \cap \boldsymbol{S}_2$, respectively.

The semantics of n-valued program clauses under Herbrand $\text{Ł}_n^\star$-interpretations is already defined by the semantics of n-valued formulas under Pr_n-interpretations. However, to get a rough idea how to define an immediate consequence operator, it might be useful to briefly focus on the technical details that additionally follow from the properties of $\text{Ł}_n^\star$-interpretations:

Lemma 5 *For all interpretations $\boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}$, all variable assignments σ, and all n-valued program clauses $(H \leftarrow B_1 \wedge \cdots \wedge B_k)[c,1]$ in $\mathcal{P}$:*

$$tv(H \vee \neg B_1 \vee \cdots \vee \neg B_k) \geq c \text{ is true in } \boldsymbol{I} \text{ under } \sigma \text{ iff}$$
$$\boldsymbol{I}_\sigma(H) \geq c - 1 + \min(\boldsymbol{I}_\sigma(B_1), \ldots, \boldsymbol{I}_\sigma(B_k)) .$$

Proof. By (3) and (5) of Lemma 3, we get directly $\boldsymbol{I}_\sigma(H \vee \neg B_1 \vee \cdots \vee \neg B_k) = \boldsymbol{I}_\sigma(H \leftarrow B_1 \wedge \cdots \wedge B_k) = \min(1, \boldsymbol{I}_\sigma(H) - \min(\boldsymbol{I}_\sigma(B_1), \ldots, \boldsymbol{I}_\sigma(B_k)) + 1)$. Hence, the n-valued formula $tv(H \vee \neg B_1 \vee \cdots \vee \neg B_k) \geq c$ is true in $\boldsymbol{I}$ under σ iff $\min(1, \boldsymbol{I}_\sigma(H) - \min(\boldsymbol{I}_\sigma(B_1), \ldots, \boldsymbol{I}_\sigma(B_k)) + 1) \geq c$. This already shows the claim, since $1 \geq c$ for all truth values $c \in TV$. □

We define the monotonic immediate consequence operator $\boldsymbol{T}_\mathcal{P}$ on the set of all subsets of $\boldsymbol{HB}_\mathcal{P}$ as follows. For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}$ and $H \in HB_\mathcal{P}$:

$$\boldsymbol{T}_\mathcal{P}(\boldsymbol{I})(H) = \max(\{c - 1 + \min(\boldsymbol{I}(B_1), \ldots, \boldsymbol{I}(B_k)) \mid (H \leftarrow B_1 \wedge \cdots \wedge B_k)[c,1]$$
$$\text{is a ground instance of an } n\text{-valued program clause in } \mathcal{P}\} \cup \{0\}) .$$

Note that we canonically define $\min(\boldsymbol{I}(B_1), \ldots, \boldsymbol{I}(B_k)) = 1$ for $k = 0$.

For all $\boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}$, we define $\boldsymbol{T}_\mathcal{P} \uparrow \omega(\boldsymbol{I})$ as the union of all $\boldsymbol{T}_\mathcal{P} \uparrow n(\boldsymbol{I})$ with $n < \omega$, where $\boldsymbol{T}_\mathcal{P} \uparrow 0(\boldsymbol{I}) = \boldsymbol{I}$ and $\boldsymbol{T}_\mathcal{P} \uparrow (n+1)(\boldsymbol{I}) = \boldsymbol{T}_\mathcal{P}(\boldsymbol{T}_\mathcal{P} \uparrow n(\boldsymbol{I}))$ for all $n < \omega$.

Next, we show that the immediate consequence operator $\boldsymbol{T}_\mathcal{P}$ makes sense. That is, that each model of $\mathcal{P}$ corresponds to a pre-fixpoint of $\boldsymbol{T}_\mathcal{P}$:

Lemma 6 *$\boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}$ is a model of $\mathcal{P}$ iff $\boldsymbol{T}_\mathcal{P}(\boldsymbol{I}) \subseteq \boldsymbol{I}$.*

Proof. The claim follows immediately from Lemma 5. □

To arrive at the main theorem, it just remains to show the continuity of $\boldsymbol{T}_\mathcal{P}$:

Lemma 7 *$\boldsymbol{T}_\mathcal{P}$ is continuous.*

Proof sketch. We must show that $\bigcup\{\boldsymbol{T}_\mathcal{P}(\boldsymbol{I}_i) \mid i \geq 0\} = \boldsymbol{T}_\mathcal{P}(\bigcup\{\boldsymbol{I}_i \mid i \geq 0\})$ for all sequences $\boldsymbol{I}_0 \subseteq \boldsymbol{I}_1 \subseteq \cdots \subseteq \boldsymbol{HB}_\mathcal{P}$. This can be done similarly to van Emden's proof of continuity of the operator T_P in his quantitative deduction [34]. □

Finally, the desired model and fixpoint semantics of n-valued logic programs is expressed by the following important theorem:

Theorem 8 $\bigcap\{\boldsymbol{I} \mid \boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}, \boldsymbol{I} \models \mathcal{P}\} = lfp(\boldsymbol{T}_\mathcal{P}) = \boldsymbol{T}_\mathcal{P} \uparrow \omega(\emptyset)$.

Proof. The first equality holds, since the monotonic operator $\boldsymbol{T}_\mathcal{P}$ has a least fixpoint $lfp(\boldsymbol{T}_\mathcal{P})$ that is equal to its least pre-fixpoint. Now, by Lemma 6, this least pre-fixpoint is equal to $\bigcap\{\boldsymbol{I} \mid \boldsymbol{I} \subseteq \boldsymbol{HB}_\mathcal{P}, \boldsymbol{I} \models \mathcal{P}\}$. The second equality follows from the continuity of $\boldsymbol{T}_\mathcal{P}$ by Lemma 7. □

Theorem 8 provides a characterization of logical consequence in $\text{Ł}_n^\star$ as follows. The n-valued logic program $\mathcal{P}$ logically entails in $\text{Ł}_n^\star$ the ground n-valued atomic formula $tv(A) \geq d$ iff $\boldsymbol{T}_\mathcal{P}{\uparrow}\omega(\emptyset)(A) \geq d$.

Hence, since the fixpoint semantics is independent of the number n of truth values, the notion of logical consequence in $\text{Ł}_n^\star$ in n-valued logic programs is also independent of the number n of truth values. In detail, for all n-valued logic programs $\mathcal{P}$, ground n-valued atomic formulas $tv(A) \geq d$, and $m = kn - k + 1$ with $k \geq 1$, we get $\mathcal{P} \models tv(A) \geq d$ in $\text{Ł}_n^\star$ iff $\mathcal{P} \models tv(A) \geq d$ in $\text{Ł}_m^\star$.

5.2 Example

Let $\mathcal{P}$ be the following 4-valued logic program (a, b, c, and d are 0-ary function symbols, e and p are 2-ary predicate symbols, and X, Y, and Z are variables):

$$\begin{aligned}\mathcal{P} = \{&(e(a,b) \leftarrow)[1,1],\ (e(b,c) \leftarrow)[1,1],\ (e(c,d) \leftarrow)[2/3,1],\\ &(p(X,Y) \leftarrow e(X,Y))[1,1],\ (p(X,Z) \leftarrow p(X,Y) \wedge p(Y,Z))[2/3,1]\}\,.\end{aligned}$$

The line of computation of the fixpoint iteration is reported by the following members of $\boldsymbol{T}_\mathcal{P}{\uparrow}1(\emptyset)$, $\boldsymbol{T}_\mathcal{P}{\uparrow}2(\emptyset)$, $\boldsymbol{T}_\mathcal{P}{\uparrow}3(\emptyset)$, and $\boldsymbol{T}_\mathcal{P}{\uparrow}4(\emptyset)$:

$$\begin{aligned}(e(a,b),1),(e(b,c),1),(e(c,d),2/3) &\in \boldsymbol{T}_\mathcal{P}{\uparrow}1(\emptyset)\\ (p(a,b),1),(p(b,c),1),(p(c,d),2/3) &\in \boldsymbol{T}_\mathcal{P}{\uparrow}2(\emptyset)\\ (p(a,c),2/3),(p(b,d),1/3) &\in \boldsymbol{T}_\mathcal{P}{\uparrow}3(\emptyset)\\ (p(a,d),1/3) &\in \boldsymbol{T}_\mathcal{P}{\uparrow}4(\emptyset)\,.\end{aligned}$$

Hence, $\mathcal{P}$ logically entails in $\text{Ł}_4^\star$ (and also in $\text{Ł}_7^\star$, $\text{Ł}_{10}^\star$, ...) the ground 4-valued atomic formula $tv(p(a,d)) \geq 1/3$, since $(p(a,d),1/3) \in \boldsymbol{T}_\mathcal{P}{\uparrow}\omega(\emptyset) = \boldsymbol{T}_\mathcal{P}{\uparrow}4(\emptyset)$.

5.3 Computational Complexity

In this section, we analyze the computational complexity of three special cases of deciding whether a ground n-valued atomic formula $tv(A) \geq c$ is a logical consequence in $\text{Ł}_n^\star$ of an n-valued logic program. They generalize the decision problems that define the computational complexity of propositional logic programming, the data complexity of datalog, and the program complexity of datalog (see, for example, [5] for a survey). Crucially, we now show that the computational complexities of the three deduction problems in $\text{Ł}_n^\star$ are no worse than the computational complexities of their more specialized classical counterparts.

Theorem 9 *Let $\mathcal{P}$ be a ground n-valued logic program, let A be a ground atomic formula, and let c be a truth value from TV. The problem of deciding whether $tv(A) \geq c$ is a logical consequence in $\text{Ł}_n^\star$ of $\mathcal{P}$ is P-complete.*

Proof. The problem is P-hard, since it generalizes the P-complete problem of deciding whether a ground atomic formula is a classical logical consequence of a ground classical logic program.

It is in P, since the number of applications of the operator $\boldsymbol{T}_\mathcal{P}$ to reach $\boldsymbol{T}_\mathcal{P} \uparrow \omega(\emptyset)$ is bounded by $n - 1$ times the number of ground atomic formulas

in $\mathcal{P}$ and since each application of $\boldsymbol{T}_\mathcal{P}$ runs in polynomial time in the size of $\mathcal{P}$. Note that, in fact, the number of applications of $\boldsymbol{T}_\mathcal{P}$ to reach $\boldsymbol{T}_\mathcal{P}{\uparrow}\omega(\emptyset)$ is even bounded by the number of ground atomic formulas in $\mathcal{P}$. To show this, let us simply assume that $\boldsymbol{T}_\mathcal{P}{\uparrow}\omega(\emptyset)$ contains a pair (A, tv) that is not contained in $\boldsymbol{T}_\mathcal{P}{\uparrow}m(\emptyset)$, where m is the number of ground atomic formulas in $\mathcal{P}$. Hence, the derivation tree of (A, tv) must contain at least one path from a leaf to the root (A, tv) with first a node (B, tv_1) and thereafter another node (B, tv_2), where B is a ground atomic formula and $tv_1, tv_2 \in TV$ with $tv_1 < tv_2$. However, the immediate consequence operator $\boldsymbol{T}_\mathcal{P}$ shows that the truth values are decreasing along every path from a leaf to the root. That is, we also get $tv_1 \geq tv_2$. □

Theorem 10 *Let all function symbols in Φ be of arity 0.*

a) Let $\mathcal{P}$ be a fixed n-valued logic program, let $\mathcal{F}$ be a varying set of ground n-valued atomic formulas, let A be a ground atomic formula, and let c be a truth value from TV. The problem of deciding whether $tv(A) \geq c$ is a logical consequence in $\text{Ł}_n^\star$ of $\mathcal{P} \cup \mathcal{F}$ is P-complete.

b) Let $\mathcal{P}$ be a varying n-valued logic program, let $\mathcal{F}$ be a fixed set of ground n-valued atomic formulas, let A be a ground atomic formula, and let c be a truth value from TV. The problem of deciding whether $tv(A) \geq c$ is a logical consequence in $\text{Ł}_n^\star$ of $\mathcal{P} \cup \mathcal{F}$ is DEXPTIME-complete.

Proof. a) The problem is P-hard, since it generalizes the P-complete problem that defines the data complexity of datalog.

It is in P by Theorem 9, since the number of all ground instances of n-valued program clauses in $\mathcal{P} \cup \mathcal{F}$ (which are constructed with all 0-ary function symbols from $\mathcal{P} \cup \mathcal{F}$) is polynomial in the input size of $\mathcal{F}$.

b) The problem is DEXPTIME-hard, since it generalizes the DEXPTIME-complete problem that defines the program complexity of datalog.

It is in DEXPTIME by Theorem 9, since the number of all ground instances of n-valued program clauses in $\mathcal{P} \cup \mathcal{F}$ (which are built with all 0-ary function symbols from $\mathcal{P} \cup \mathcal{F}$) is exponential in a polynom of the input size of $\mathcal{P}$. □

6 Related Work

The literature already contains quite extensive work on probabilistic logics and on truth-functional many-valued logics separately. However, to the best of our knowledge, an integration of both has never been studied so far.

Probabilistic propositional logics and their various dialects are thoroughly studied in the literature (see, for example, the pioneering work in [29] and [10]). Their extensions to probabilistic first-order logics can be classified into first-order logics in which probabilities are defined over a set of possible worlds and those in which probabilities are given over the elements of the domain (see, for example, [2] and [15]). The first ones are suitable for representing degrees of belief, while the latter are appropriate for describing statistical knowledge. The same classification holds for approaches to probabilistic logic programming (see, for example, [27], [26], [28], [24], and [25]).

Many approaches to truth-functional finite-valued logic programming are restricted to three or four truth values (see, for example, [18], [11], [12], [3], [30], and [8]). Among these approaches, the one closest in spirit to n-valued logic programming in $\text{Ł}_n^\star$ is the 3-valued approach in [18]. Note that in contrast to [18], however, n-valued logic programming in $\text{Ł}_n^\star$ is especially much more general and also well-grounded on a probabilistic semantics.

Our n-valued logic programming in $\text{Ł}_n^\star$ is closely related to van Emden's infinite-valued quantitative deduction in [34]. More precisely, both approaches have a common semantic background in probabilistic logic programming as introduced in [24]. That is, interpreted by probability distributions over possible worlds that satisfy an extension of (1), the probabilistic logic programs in [24] coincide in their fixpoint semantics with n-valued logic programs in $\text{Ł}_n^\star$ (if the implication connective is interpreted as material implication) and with van Emden's quantitative logic programs (if the implication connective is interpreted as conditional probability). Note that probability distributions over possible worlds represent experts worlds in which the number of experts is variable and in which the (not necessarily equal) credibilities of the experts sum up to 1.

Crucially, in contrast to van Emden's quantitative deduction, our n-valued logic programming in $\text{Ł}_n^\star$ does not have to struggle with an infinite number of truth values in its proof theory (which will be presented in future work).

Both n-valued logic programming in $\text{Ł}_n^\star$ and also van Emden's quantitative deduction can be regarded as special cases of generalized annotated logic programming (where annotations may contain variables and function symbols: see, for example, [16]). Moreover, n-valued logic programming in $\text{Ł}_n^\star$ is generalized by signed formula logic programming as discussed in [21]. However, we think that it is important in its own right, since generalized annotated logic programming and also signed formula logic programming (with their increased expressiveness) generally do not seem to have the computational properties of classical logic programming (see [19] and [21]). Finally, we also consider n-valued logic programming in $\text{Ł}_n^\star$ as a new important starting point for giving a natural semantics to disjunctions and negations in many-valued logic programming (which is a very interesting topic of future research). In particular, the implied natural semantics of negations will differ from the classical semantics of negations in current truth-functional many-valued logic programming.

Note that annotated logic programs with only constants as annotations (like the closely related signed formula approach in [14]) are not expressive enough to generalize n-valued logic programs in $\text{Ł}_n^\star$, since they cannot represent truth value assignments on the level of n-valued program clauses (they can just represent logical relationships between n-valued facts).

Finally, the way in which truth values are propagated through the implication connective of many-valued program clauses distinguishes our n-valued logic programming in $\text{Ł}_n^\star$ from many other approaches like, for example, the propositional approach in [9] and the work in [17] (we interpret the implication connective as material implication, while the latter approaches define the truth value of the head of a many-valued program clause as conjunction of the truth value of the many-valued program clause itself and the truth value of its body).

7 Summary and Outlook

We presented n-valued first-order logics with a purely probabilistic semantics. We then introduced $\text{Ł}_n^\star$-interpretations as a new probabilistic semantics of n-valued first-order logics that lies between the purely probabilistic semantics and the truth-functional semantics of the n-valued Łukasiewicz logics Ł_n.

We showed that $\text{Ł}_n^\star$-interpretations have very interesting properties from both the semantic and the computational point of view. In particular, closed formulas of classical first-order logics that are logically equivalent in the classical sense also have the same truth value under all $\text{Ł}_n^\star$-interpretations. Moreover, disjunctive logic programming with n-valued disjunctive facts in $\text{Ł}_n^\star$ can be reduced to classical disjunctive logic programming in $n-1$ layers. Moreover, n-valued logic programs in $\text{Ł}_n^\star$ were shown to have a model and a fixpoint semantics that are very similar to those of classical logic programs. Finally, some important deduction problems in n-valued logic programs in $\text{Ł}_n^\star$ were shown to have the same computational complexity like their classical counterparts.

Future research may concern the extension of $\text{Ł}_n^\star$-interpretations to probability distributions over possible worlds (by which we obtain experts worlds in which the number of experts is variable and in which their credibilities is not necessarily equal). Another topic of future research is n-valued logic programming in $\text{Ł}_n^\star$ with negations and disjunctions. Finally, it would be interesting to extend the language of n-valued formulas to also allow reasoning on the level of the full n-valued first-order language (similar to, for example, [15]).

8 Acknowledgements

I am very grateful to the referees for their useful comments. I am also very grateful to Thomas Eiter for having presented this paper at the conference.

References

1. K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 10, pages 493–574. The MIT Press, 1990. 423
2. F. Bacchus, A. Grove, J. Y. Halpern, and D. Koller. From statistical knowledge bases to degrees of beliefs. *Artif. Intell.*, 87:75–143, 1996. 426
3. H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theor. Comput. Sci.*, 68:135–154, 1989. 427
4. R. Carnap. *Logical Foundations of Probability.* University of Chicago Press, Chicago, 1950.
5. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. In *Proc. of the 12th Annual IEEE Conference on Computational Complexity*, pages 82–101, 1997. 425
6. B. de Finetti. *Theory of Probability.* J. Wiley, New York, 1974.
7. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. In *Proc. of the 14th International Conference on Logic Programming*, pages 391–405, 1997.
8. J. P. Delahaye and V. Thibau. Programming in three-valued logic. *Theor. Comput. Sci.*, 78:189–216, 1991. 427
9. G. Escalada-Imaz and F. Manyà. Efficient interpretation of propositional multi-valued logic programs. In *Proc. 5th International Conference on Information Pro-*

cessing and Management of Uncertainty in Knowledge-Based Systems (IPMU '94), volume 945 of *LNCS*, pages 428–239. Springer, 1995. 427
10. R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Inf. Comput.*, 87:78–128, 1990. 426
11. M. Fitting. Partial models and logic programming. *Theor. Comput. Sci.*, 48:229–255, 1986. 427
12. M. Fitting. Bilattices and the semantics of logic programming. *J. Logic Program.*, 11(1–2):91–116, 1991. 427
13. M. Fitting. Many-valued modal logics II. *Fundam. Inf.*, 17:55–73, 1992. 416, 416
14. R. Hähnle. Exploiting data dependencies in many-valued logics. *J. Appl. Non-Class. Log.*, 6(1):49–69, 1996. 427
15. J. Y. Halpern. An analysis of first-order logics of probability. *Artif. Intell.*, 46:311–350, 1990. 426, 428
16. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Logic Program.*, 12(3–4):335–367, 1992. 427
17. L. V. S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proc. of the International Logic Programming Symposium*, pages 254–268, 1994. 427
18. J.-L. Lassez and M. J. Maher. Optimal fixedpoints of logic programs. *Theor. Comput. Sci.*, 39:15–25, 1985. 427, 427, 427
19. S. M. Leach and J. J. Lu. Query processing in annotated logic programming: Theory and implementation. *J. Intell. Inf. Syst.*, 6(1):33–58, 1996. 427
20. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 2nd ed., 1987.
21. J. J. Lu. Logic programming with signs and annotations. *J. Log. Comput.*, 6(6):755–778, 1996. 427, 427
22. T. Lukasiewicz. Magic inference rules for probabilistic deduction under taxonomic knowledge. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence*, pages 354–361. Morgan Kaufmann Publishers, 1998. 416
23. T. Lukasiewicz. Probabilistic deduction with conditional constraints over basic events. In *Principles of Knowledge Representation and Reasoning: Proc. of the 6th International Conference*, pages 380–391. Morgan Kaufmann Publishers, 1998. 416
24. T. Lukasiewicz. Probabilistic logic programming. In *Proc. of the 13th European Conference on Artificial Intelligence*, pages 388–392. J. Wiley & Sons, 1998. 426, 427, 427
25. R. T. Ng. Semantics, consistency, and query processing of empirical deductive databases. *IEEE Trans. Knowl. Data Eng.*, 9(1):32–49, 1997. 426
26. R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Inf. Comput.*, 101:150–201, 1992. 426
27. R. T. Ng and V. S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *J. Autom. Reasoning*, 10(2):191–235, 1993. 426
28. R. T. Ng and V. S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Inf. Comput.*, 110:42–83, 1994. 426
29. N. J. Nilsson. Probabilistic logic. *Artif. Intell.*, 28:71–88, 1986. 426
30. T. C. Przymusinski. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artif. Intell.*, 49:309–343, 1991. 427
31. N. Rescher. *Many-valued Logic*. McGraw-Hill, New York, 1969.
32. J. B. Rosser and A. R. Turquette. *Many-valued Logics*. North-Holland, 1952.
33. V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Trans. Database Syst.*, 19(2):291–331, 1994. 416
34. M. H. van Emden. Quantitative deduction and its fixpoint theory. *J. Logic Program.*, 3(1):37–53, 1986. 424, 427
35. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965. 423

Author Index