Reinhard Kahle
Peter Schroeder-Heister
Robert Stärk (Eds.)

LNCS 2183

Proof Theory in Computer Science

International Seminar, PTCS 2001
Dagstuhl Castle, Germany, October 2001
Proceedings

Springer

Lecture Notes in Computer Science 2183

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

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

Reinhard Kahle Peter Schroeder-Heister
Robert Stärk (Eds.)

Proof Theory in Computer Science

International Seminar, PTCS 2001
Dagstuhl Castle, Germany, October 7-12, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Reinhard Kahle
Peter Schroeder-Heister
Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Sand 13, 72076 Tübingen, Germany
E-mail: {kahle/psh}@informatik.uni-tuebingen.de

Robert Stärk
ETH Zürich, Theoretische Informatik
ETH Zentrum, 8092 Zürich, Switzerland
E-mail: staerk@inf.ethz.ch

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Proof theory in computer science : international seminar ; proceedings /
PTCS 2001, Dagstuhl Castle, Germany, October 7 - 12, 2001. Reinhard Kahle ...
ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ;
Milan ; Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2183)
ISBN 3-540-42752-X

CR Subject Classification (1998): F.4.1, I.2.3, F.4, F.3, I.2, F.2

ISSN 0302-9743
ISBN 3-540-42752-X Springer-Verlag Berlin Heidelberg New York

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

Printed in Germany

Typesetting: Camera-ready by author, date conversion by PTP Berlin, Stefan Sossna
Printed on acid-free paper SPIN 10840525 06/3142 5 4 3 2 1 0

Preface

Proof theory has long been established as a basic discipline of mathematical logic. It has recently become increasingly relevant to computer science. The deductive apparatus provided by proof theory has proved useful for metatheoretical purposes as well as for practical applications. Thus it seemed to us most natural to bring researchers together to assess both the role proof theory already plays in computer science and the role it might play in the future.

The form of a Dagstuhl seminar is most suitable for purposes like this, as Schloß Dagstuhl provides a very convenient and stimulating environment to discuss new ideas and developments. To accompany the conference with a proceedings volume appeared to us equally appropriate. Such a volume not only fixes basic results of the subject and makes them available to a broader audience, but also signals to the scientific community that *Proof Theory in Computer Science* (PTCS) is a major research branch within the wider field of logic in computer science.

Therefore everybody invited to the Dagstuhl seminar was also invited to submit a paper. However, preparation and acceptance of a paper for the volume was not a precondition of participating at the conference, since the idea of a Dagstuhl seminar as a forum for spontaneous and open discussions should be kept. Our idea was that the papers in this volume should be suitable as starting points for such discussions by presenting fundamental results which merit their publication in the Springer LNCS series. The quality and variety of the papers received and accepted rendered this plan fully justified. They are a state-of-the-art sample of proof-theoretic methods and techniques applied within computer science.

In our opinion PTCS focuses on the impact proof theory has or should have on computer science, in particular with respect to programming. Major divisions of PTCS, as represented in this volume, are the following:

1. The *proofs as programs* paradigm in general
2. Typed and untyped systems related to functional programming
3. Proof-theoretic approaches to logic programming
4. Proof-theoretic ways of dealing with computational complexity
5. Proof-theoretic semantics of languages for specification and programming
6. Foundational issues

This list is not intended to be exclusive. For example, there is undoubtedly some overlap between *Automated Deduction* and PTCS. However, since *Automated Deduction* is already a well-established subdiscipline of logic in computer science with its own research programs, many of which are not related to proof theory, we did not include it as a core subject of PTCS.

In the following, we briefly address the topics of PTCS mentioned and indicate how they are exemplified in the contributions to this volume.

1. The most intrinsic relationship between proof theory and computer science, if proof theory is understood as the theory of formal proofs and computer science as the theory of computing, is provided by the fact that in certain formalisms proofs can be evaluated (reduced) to normal forms. This means that proofs can be viewed as representing a (not necessarily deterministic) program for their own evaluation. In particular contexts they allow one to extract valuable information, which may be given, e.g., in the form of particular terms. The idea of considering *proofs as programs*, which in the context of the typed λ-calculus is known as the Curry-Howard-correspondence, is a research program touched upon by most contributions to this volume. The papers by *Baaz & Leitsch* and by *Berger* are directly devoted to it. *Baaz & Leitsch* study the relative complexity of two cut elimination methods and show that they are intrinsically different. *Berger* investigates a proof of transfinite induction given by Gentzen in order to extract algorithms for function hierarchies from it.

2. *Functional programming* has always been at the center of interest of proof theory, as it is based on the λ-calculus. Extensions of the typed λ-calculus, in particular type theories, lead to powerful frameworks suitable for the formalization of large parts of mathematics. The paper by *Alt & Artemov* develops a reflective extension of the typed λ-calculus which internalizes its own derivations as terms. *Dybjer & Setzer* show how indexed forms of inductive-recursive definitions, which would enable a certain kind of generic programming, can be added to Martin-Löf type theory. The main proof-theoretic paradigm competing with type theory is based on type-free applicative theories and extensions thereof within Feferman's general program of explicit mathematics. In his contribution, *Studer* uses this framework in an analysis of a fragment of Java. In particular, he manages to proceed without impredicative assumptions, thus supporting a general conjecture by Feferman.

3. *Logic programming*, which uses the Horn clause fragment of first-order logic as a programming language, is a natural topic of PTCS. Originally it was not developed within a proof-theoretic framework, and its theoretical background is often described in model-theoretic terms. However, it has turned out that a proof-theoretic treatment of logic programming is both nearer to the programmer's way of thinking and conceptually and technically very natural. It also leads to strong extensions, including typed ones which combine features of functional and logic programming. In the present volume, *Elbl* uses proof-theoretic techniques to give "metalogical" operators in logic programming an appropriate rendering.

4. The machine-independent characterization of classes of *computational complexity* not involving explicit bounds has recently gained much attention in proof theory. One such approach, relying on higher-type functionals, is used in *Aehlig et al.*'s paper to characterize the parallel complexity class NC. Another proof-theoretic method based on term rewriting is applied by *Oitavem* in her characterization of PSPACE and is compared and contrasted with other implicit characterizations of this class. *Gordeew* asks the fundamental question of whether functional analysis may serve as an alternative framework in certain subjects of PTCS. He suggests that non-discrete methods may provide powerful tools

for dealing with certain computational problems, in particular those concerning polynomial-time computability.

5. Besides the systematic topics mentioned, the study of *specific languages* is an important aspect of PTCS. In his contribution, *Schmitt* develops and studies a language of iterate logic as the logical basis of certain specification and modeling languages. *Studer* gives a denotational semantics of a fragment of Java. By interpreting Featherweight Java in a proof-theoretically specified language he shows that there is a direct proof-theoretic sense of denotational semantics which differs both from model-theoretic and from domain-theoretic approaches. This shows that the idea of *proof-theoretic semantics* discussed in certain areas of philosophical and mathematical logic, is becoming fruitful for PTCS as well.

6. Finally, two papers concern *foundational aspects* of languages. *Baaz & Fermüller* show that in formalizing identity, it makes a significant difference for uniform provability, whether identity is formulated by means of axioms or by means of a schema. The paper by *Došen & Petrić* presents a coherence result for categories which they call "sesquicartesian", contributing new insights into the equality of arrows (and therefore into the equality of proofs and computations) and its decidability.

We thank the authors and reviewers for their contributions and efforts. We are grateful to the Schloß Dagstuhl conference and research center for acting as our host, and to Springer-Verlag for publishing these proceedings in their LNCS series.

The second and the third editor would like to add that it was Reinhard Kahle's idea to organize a Dagstuhl seminar on PTCS, and that he had the major share in preparing the conference and editing this volume. He would have been the first editor even if his name had not been the first alphabetically.

October 2001

Reinhard Kahle
Peter Schroeder-Heister
Robert Stärk

Program Committee

Arnold Beckmann (Münster)
Roy Dyckhoff (St. Andrews)
Rajeev Goré (Canberra)
Gerhard Jäger (Bern)
Reinhard Kahle (Tübingen)
Dale Miller (Penn State)
Tobias Nipkow (München)
Frank Pfenning (Pittsburgh)
Peter Schroeder-Heister (Tübingen)
Robert Stärk (Zürich)

Additional Reviewers

Steve Bellantoni (Toronto)
Norman Danner (Los Angeles)
Lew Gordeew (Tübingen)
Peter Hancock (Edinburgh)
Jörg Hudelmaier (Tübingen)
Jim Lambek (Montreal)
Daniel Leivant (Bloomington)
Jean-Yves Marion (Nancy)
Ralph Matthes (München)
Hans Jürgen Ohlbach (München)
Christine Paulin-Mohring (Paris)
Thomas Strahm (Bern)

Table of Contents

Linear Ramified Higher Type Recursion and Parallel Complexity

Klaus Aehlig[1,⋆], Jan Johannsen[2,⋆⋆], Helmut Schwichtenberg[1,⋆⋆⋆], and Sebastiaan A. Terwijn[3,†]

[1] Mathematisches Institut, Ludwig-Maximilians-Universität München, Theresienstraße 39, 80333 München, Germany
{aehlig,schwicht}@rz.mathematik.uni-muenchen.de
[2] Institut für Informatik, Ludwig-Maximilians-Universität München Oettingenstraße 67, 80538 München, Germany
jjohanns@informatik.uni-muenchen.de
[3] Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
terwijn@cs.vu.nl

Abstract. A typed lambda calculus with recursion in all finite types is defined such that the first order terms exactly characterize the parallel complexity class NC. This is achieved by use of the appropriate forms of recursion (concatenation recursion and logarithmic recursion), a ramified type structure and imposing of a linearity constraint.

Keywords: higher types, recursion, parallel computation, NC, lambda calculus, linear logic, implicit computational complexity

1 Introduction

One of the most prominent complexity classes, other than polynomial time, is the class NC of functions computable in parallel polylogarithmic time with a polynomial amount of hardware. This class has several natural characterizations in terms of circuits, alternating Turing machines, or parallel random access machines as used in this work. It can be argued that NC is the class of efficiently parallalizable problems, just as polynomial time is generally considered as the correct formalization of feasible sequential computation.

Machine-independent characterizations of computational complexity classes are not only of theoretical, but recently also of increasing practical interest. Besides indicating the robustness and naturalness of the classes in question, they also provide guidance for the development of programming languages [11].

⋆ Supported by the DFG Graduiertenkolleg "Logik in der Informatik"
⋆⋆ Supported by the DFG Emmy Noether-Programme under grant No. Jo 291/2-1
⋆⋆⋆ The hospitality of the Mittag-Leffler Institute in the spring of 2001 is gratefully acknowledged.
† Supported by a Marie Curie fellowship of the European Union under grant no. ERB-FMBI-CT98-3248

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 1–21, 2001.

The earliest such characterizations, starting with Cobham's function algebra for polynomial time [9], used recursions with explicit bounds on the growth of the defined functions. Function algebra characterizations in this style of parallel complexity classes, among them NC, were given by Clote [8] and Allen [1].

More elegant *implicit* characterizations, i.e., without any explicitly given bounds, but instead using logical concepts like ramification or tiering, have been given for many complexity classes, starting with the work of Bellantoni and Cook [4] and Leivant [14] on polynomial time. In his thesis [2], Bellantoni gives such a characterization of NC using a ramified variant of Clote's recursion schemes. A different implicit characterization of NC, using tree recursion, was given by Leivant [15], and refined by Bellantoni and Oitavem [6]. Other parallel complexity classes, viz. parallel logarithmic and polylogarithmic time, were given implicit characterizations by Bellantoni [3], Bloch [7] and Leivant and Marion [16].

In order to apply the approach within the functional programming paradigm, one has to consider functions of higher type, and thus extend the function algebras by a typed lambda calculus. To really make use of this feature, it is desirable to allow the definition of higher type functions by recursion. Higher type recursion was originally considered by Gödel [10] for the analysis of logical systems. Systems with recursion in all finite types characterizing polynomial time were given by Bellantoni et al. [5] and Hofmann [12], based on the first-order system of Bellantoni and Cook [4].

We define an analogous system that characterizes NC while allowing an appropriate form of recursion, viz. logarithmic recursion as used by Clote [8] and Bellantoni [2], in all finite types. More precisely, our system is a typed lambda calculus which allows two kinds of function types, denoted $\sigma \multimap \tau$ and $\sigma \to \tau$, and two sorts of variables of the ground type ι, the *complete* ones in addition to the usual ones, which are called incomplete for emphasis. A function of type $\sigma \to \tau$ can only be applied to complete terms of type σ, i.e., terms containing only complete free variables.

It features two recursion operators LR and CR, the latter corresponding to Clote's [8] concatenation recursion on notation, which can naturally only be applied to first-order functions. The former is a form of recursion of logarithmic length characteristic of all function algebra representations of NC, and here can be applied to functions of all linear types, i.e., types only built up using ι and $\multimap$. The function being iterated, as well as the numerical argument being recurred on have to be complete, i.e., the type of LR is $\sigma \multimap (\iota \to \sigma \multimap \sigma) \to \iota \to \sigma$ for linear σ.

Our analysis clearly reveals the different roles played by the two forms of recursion in characterizing NC: Logarithmic recursion controls the runtime, in that the degree of the polylogarithm that bounds the runtime depends only on the number of occurrences of LR. On the other hand, concatenation recursion is responsible for parallelism; the degree of the polynomial bounding the amount of hardware used depends only on the number of occurrences of CR (and the number of occurences of the constant #.)

The crucial restriction in our system, justifying the use of linear logic notation, is a linearity constraint on variables of higher types: all higher type variables in a term must occur at most once.

The main new contribution in the analysis of the complexity of the system is a strict separation between the term, i.e., the program, and the numerical context, i.e., its input and data. Whereas the runtime may depend polynomially on the former, it may only depend polylogarithmically on the latter.

To make use of this conceptual separation, the algorithm that unfolds recursions computes, given a term and context, a recursion-free term *plus a new context*. In particular, it does not substitute numerical parameters, as this would immediately lead to linear growth, but only uses them for unfolding; in some cases, including the reduction of CR, it extends the context. This way, the growth of terms in the elimination of recursions is kept under control. In earlier systems that comprised at least polynomial time this strict distinction was not necessary, since the computation time there may depend on the *input* superlinearly. Note that any reasonable form of computation will depend at least linearly on the size of the *program*.

A direct extension to higher types of the first-order system of Bellantoni [2] would have a constant for concatenation recursion of linear type $(\iota \multimap \iota) \multimap \iota \multimap \iota$. This causes problems in our analysis because the amount of hardware required depends exponentially on the number of CR in a term, thus we must not allow duplications of this constant during the unfolding of LR. The only way to avoid this is by giving CR the more restrictive typ $(\iota \to \iota) \multimap \iota \to \iota$. This weaker form of concatenation recursion nevertheless suffices to include all of NC, when the set of base functions is slightly extended.

Finally, in order to be able to handle numerals in parallel logarithmic time, we use a tree data structure to store numerals during the computation. Whereas trees are used as the principal data structure in other characterizations of parallel complexity classes [15,16], our system works with usual binary numerals, and trees are only used in the implementation.

2 Clote's Function Algebra for NC

Clote [8] gives a function algebra characterization of NC using two recursion schemes. The class **A** is defined as the least class of functions that contain the constant 0, projections $\pi^n_j(x_1, \ldots, x_n) = x_j$, the binary successors s_0, s_1, bit test *bit*, binary length $|x| := \lceil \log_2(x+1) \rceil$, and $\#$ where $x\#y = 2^{|x|\cdot|y|}$, and is closed under composition and the following two forms of recursion:

A function f is defined by *concatenation recursion on notation* (CRN) from functions g, h_0, h_1 if

$$\begin{aligned} f(0, \overrightarrow{x}) &= g(\overrightarrow{x}) \\ f(s_i(y), \overrightarrow{x}) &= s_{h_i(y,\overrightarrow{x})}(f(y, \overrightarrow{x})) \ , \end{aligned}$$

and f is defined from g, h_0, h_1 and r by weak bounded recursion on notation (WBRN) if there is F such that

$$\begin{aligned} F(0, \overrightarrow{x}) &= g(\overrightarrow{x}) \\ F(s_i(y), \overrightarrow{x}) &= h_i(y, \overrightarrow{x}, F(y, \overrightarrow{x})) \\ F(y, \overrightarrow{x}) &\leq r(y, \overrightarrow{x}) \\ f(y, \overrightarrow{x}) &= F(|y|, \overrightarrow{x}) \ . \end{aligned}$$

Theorem 1 (Clote [8]). *A number-theoretic function f is in $\boldsymbol{A}$ if and only if f is in NC.*

It is easy to see that the recursion scheme WBRN can be replaced by the following scheme: f is defined from g, h and r by *bounded logarithmic recursion* if

$$\begin{aligned} &f(0, \overrightarrow{x}) = g(\overrightarrow{x}) \\ &f(y, \overrightarrow{x}) = h(y, \overrightarrow{x}, f(H(y), \overrightarrow{x})) \qquad \text{for } y > 0 \\ &f(y, \overrightarrow{x}) \leq r(y, \overrightarrow{x}) \ , \end{aligned}$$

where $H(n) := \left\lfloor \frac{n}{2^{\lceil |n|/2 \rceil}} \right\rfloor$ has length about half the length of n. Both forms of recursion produce $\log |y|$ iterations of the step function. We shall denote the function algebra with bounded logarithmic recursion by $\mathbf{A}$ as well.

3 Formal Definition of the System

We use simple types with two forms of abstraction over a single base type ι, i.e., our types are given by the grammar

$$\sigma, \tau ::= \iota \mid \sigma \multimap \tau \mid \sigma \to \tau \ ,$$

and we call the types that are built up from ι and $\multimap$ only the *linear* types.

As the intended semantics for our base type are the binary numerals we have the constants 0 of type ι and s_0 and s_1 of type $\iota \multimap \iota$. Moreover we add constants len of type $\iota \multimap \iota$ and bit of type $\iota \multimap \iota \multimap \iota$ for the corresponding base functions of $\mathbf{A}$.

The functionality of the base function # is split between two constants, a unary # of type $\iota \to \iota$ to produce growth, and sm of type $\iota \multimap \iota \multimap \iota \multimap \iota$ that performs the multiplication of lengths without producing growth. The intended semantics, reflected in the conversion rules below, is $\#n = 2^{|n|^2}$ and $sm(w, a, b) = 2^{|a| \cdot |b|} \bmod 2^{|w|}$.

In order to embed $\mathbf{A}$ into the system, we need two more constants drop of type $\iota \multimap \iota \multimap \iota$ and half of type $\iota \multimap \iota$, intended to denote the functions $drop(n, m) = \left\lfloor \frac{n}{2^{|m|}} \right\rfloor$ and H, respectively.

We allow case-distinction for arbitrary types, so we have a constant d_σ of type $\iota \multimap \sigma \multimap \sigma \multimap \sigma$ for *every* type σ. Recursion is added to the system via the constant LR and parallelism via the constant CR. Their types are

$$\begin{array}{lcll} \mathsf{CR} & : & (\iota \to \iota) \multimap \iota \to \iota & \\ \mathsf{LR}_\sigma & : & \sigma \multimap (\iota \to \sigma \multimap \sigma) \to \iota \to \sigma & \text{for } \sigma \text{ linear} \end{array}$$

Terms are built from variables and constants via abstraction and typed application. We have incomplete variables of every type, denoted by x, y, ... and complete variables of ground type, denoted by $\mathbf{x}$, $\mathbf{y}$, ... All our variables and terms have a fixed type and we add type superscripts to emphasize the type: x^σ, $\mathbf{x}^\iota$, t^σ.

Corresponding to the two kinds of function types, there are two forms of abstraction

$$(\lambda x^\sigma.t^\tau)^{\sigma \multimap \tau} \quad \text{and} \quad (\lambda \mathbf{x}^\iota.t^\tau)^{\iota \to \tau}$$

and two forms of application

$$\left(t^{\sigma \multimap \tau}\, s^\sigma\right)^\tau \quad \text{and} \quad (t^{\sigma \to \tau}\, s^\sigma)^\tau \ ,$$

where in the last case we require s to be complete, and a term is called *complete* if all its free variables are. It should be noted that, although we cannot form terms of type $\sigma \to \tau$ with $\sigma \neq \iota$ directly via abstraction, it is still important to have that type in order to express, for example, that the first argument of LR must not contain free incomplete variables.

In the following we omit the type subscripts at the constants d_σ and LR_σ if the type is obvious or irrelevant. Moreover we identify α-equal terms. As usual application associates to the left. A *binary numeral* is either 0, or of the form $\mathsf{s}_{i_1}(\ldots(\mathsf{s}_{i_k}(\mathsf{s}_1 0)))$. We abbreviate the binary numeral $(\mathsf{s}_1 0)$ by 1.

The semantics of ι as binary numerals (rather than binary words) is given by the conversion rule $\mathsf{s}_0\ 0 \mapsto 0$. In the following definitions we identify binary numerals with the natural number they represent. The base functions get their usual semantics, i.e., we add conversion rules $\mathsf{len}\, n \mapsto |n|$, $\mathsf{drop}\, n\, m \mapsto \mathit{drop}(n, m)$, $\mathsf{half}\, n \mapsto H(n)$, $\mathsf{bit}\, n\, i \mapsto \left\lfloor \frac{n}{2^i} \right\rfloor \bmod 2$, $\mathsf{sm}\, w\, m\, n \mapsto \mathit{sm}(w, m, n)$. Moreover, we add the conversion rules

$$\begin{array}{lcl} \mathsf{d}_\sigma\, 0 & \mapsto & \lambda x^\sigma\, y^\sigma\, .\, x \\ \mathsf{d}_\sigma\, (\mathsf{s}_i n) & \mapsto & \lambda x_0^\sigma\, x_1^\sigma\, .\, x_i \\ \# n & \mapsto & \mathsf{s}_0{}^{|n|^2} 1 \\ \mathsf{CR}\, h\, 0 & \mapsto & 0 \\ \mathsf{CR}\, h\, (\mathsf{s}_i\, n) & \mapsto & \mathsf{d}_{(\iota \multimap \iota)}\ (h\, (\mathsf{s}_i n))\, \mathsf{s}_0\, \mathsf{s}_1\, (\mathsf{CR}\, h\, n) \\ \mathsf{LR}\, g\, h\, 0 & \mapsto & g \\ \mathsf{LR}\, g\, h\, n & \mapsto & h\, n\, (\mathsf{LR}\, g\, h\, (\mathsf{half}\, n)) \end{array}$$

Here we always assumed that n, m and $\mathsf{s}_i\, n$ are binary numerals, and in particular that the latter does not reduce to 0. In the last rule, n has to be a binary numeral different from 0.

As usual the reduction relation is the closure of $\mapsto$ under all term forming operations and equivalence is the symmetric, reflexive, transitive closure of the reduction relation. As all reduction rules are correct with respect to the intended semantics and obviously all closed normal terms of type ι are numerals, closed terms t of type ι have a unique normal form that we denote by t^{nf}.

As usual, lists of notations for terms/numbers/ ... that only differ in successive indices are denoted by leaving out the indices and putting an arrow over the notation. It is usually obvious where to add the missing indices. If not we add a dot wherever an index is left out. Lists are inserted into formulae "in the natural way", e.g., $\overrightarrow{hm.} = hm_1, \ldots, hm_k$ and $x\,\overrightarrow{t} = ((x\,t_1)\ldots t_k)$ and $|g| + \overrightarrow{|s|} = |g| + |s_1| + \ldots + |s_k|$. Moreover, by abuse of notation, we denote lists consisting of maybe both, complete and incomplete variables also by $\overrightarrow{x}$.

As already mentioned, we are not interested in all terms of the system, but only in those fulfilling a certain linearity condition.

Definition 1. *A term t is called* linear*, if every variable of higher type in t occurs at most once.*

Since we allow that the variable x does not occur in $\lambda x.t$, our linear terms should correctly be called *affine*, but we keep the more familiar term *linear*.

4 Completeness

Definition 2. *A term $t : \overrightarrow{\iota} \to \iota$ denotes the function $f(\overrightarrow{x})$ if for every $\overrightarrow{n}$, $t\overrightarrow{n}$ reduces to the numeral $f(\overrightarrow{n})$.*

We will sometimes identify a term with the function it denotes.

In order to prove that our term system can denote all functions in NC, we first have to define some auxiliary terms. We define $\mathsf{ones} := \mathsf{CR}(\lambda\mathbf{x}.1)$, then we have that $\mathsf{ones}\, n = 2^{|n|} - 1$, i.e., a numeral of the same length as n consisting of ones only. We use this to define

$$\leq_\ell := \lambda\mathbf{y}\, b\,.\, \mathsf{bit}\,(\mathsf{ones}\,\mathbf{y})\,(\mathsf{len}\, b)\ ,$$

so that $\leq_\ell\ m\, n$ is the characteristic function of $|m| \leq |n|$. We will write $\leq_\ell$ infix in the following. It is used to define

$$\mathsf{max}_\ell := \lambda\mathbf{a}\, b\,.\,\mathsf{d}\,(\mathbf{a} \leq_\ell b)\, b\, \mathbf{a}$$

computing the longer of two binary numerals.

Next we define $\mathsf{rev} := \lambda x.\mathsf{CR}(\lambda\mathbf{i}.\mathsf{bit}\, x\, \mathbf{i})$, so that $\mathsf{rev}\, m\, n$ returns the $|n|$ least significant bits of m reversed. Finally we define the binary predecessor as $\mathsf{p} := \lambda x.\mathsf{drop}\, x\, 1$.

Theorem 2. *For every function $f(\overrightarrow{x})$ in $\boldsymbol{A}$, there is a closed linear term t_f of type $\overrightarrow{\iota} \to \iota$ that denotes f.*

Proof. The proof follows the lines of Bellantoni's [2] completeness proof for his two-sorted function algebra for NC.

We will use the following fact: for every function $f \in \mathbf{A}$ there is a polynomial q_f such that for all $\overrightarrow{n}$, $|f(\overrightarrow{n})| \leq q_f(\overrightarrow{|n|})$. To prove the theorem, we will prove the following stronger claim:

> For every $f(\overrightarrow{x}) \in \mathbf{A}$, there is a closed linear term t_f of type $\iota \to \overrightarrow{\iota} \multimap \iota$ and a polynomial p_f such that for every $\overrightarrow{n}$, $t_f\, w\, \overrightarrow{n}$ reduces to $f(\overrightarrow{n})$ for all w with $|w| \geq p_f(\overrightarrow{|n|})$.

The claim implies the theorem, since by use of the constant $\#$ and the term max_ℓ, we can define terms $w_f : \overrightarrow{\iota} \to \iota$ such that for all $\overrightarrow{n}$, $|w_f\, \overrightarrow{n}| \geq p_f(\overrightarrow{|n|})$.

We prove the claim by induction on the definition of f in the function algebra $\mathbf{A}$ with bounded logarithmic recursion.

If f is any of the base functions $0, s_i, |.|, bit$, then we let $t_f := \lambda \mathbf{w}.c$ where c is the corresponding constant of our system, and for $f = \pi^n_j$ we let $t_f := \lambda \mathbf{w}\, \overrightarrow{x}.x_j$. In these cases we can set $p_f = 0$, and the claim obviously holds.

If f is $\#$, then we set $t_f := \lambda \mathbf{w}\,.\,\mathsf{sm}\,\mathbf{w}$. It holds that $t_f\, w\, a\, b = a \# b$ as long as $|a| \cdot |b| < |w|$, so we set $p_f(x, y) = x \cdot y + 1$.

If f is defined by composition, $f(\overrightarrow{x}) = h(\overrightarrow{g(\overrightarrow{x})})$, then by induction we have terms $t_h, \overrightarrow{t_g}$ and polynomials $p_h, \overrightarrow{p_g}$. We define $t_f := \lambda \mathbf{w}\, \overrightarrow{x}.t_h \mathbf{w} \overrightarrow{(t_g \mathbf{w}\, \overrightarrow{x})}$ and $p_f(\overrightarrow{x}) := p_h(\overrightarrow{q_g(\overrightarrow{x})}) + \overrightarrow{p_g(\overrightarrow{x})}$. The claim follows easily from the induction hypothesis.

Now let f be defined by CRN from g, h_0, h_1, and let t_g, t_{h_i} be given by induction. First we define a function h that combines the two step functions into one, by

$$h := \lambda \mathbf{w}\, y\,.\,\mathsf{d}\, y\, (t_{h_0}\, \mathbf{w}\, (\mathsf{p}\, y))\, (t_{h_1}\, \mathbf{w}\, (\mathsf{p}\, y))$$

then we use this to define a function f' that computes an end-segment of $f(y, \overrightarrow{x})$ reversed, using CR, by

$$\begin{aligned} \mathsf{aux} := \lambda \mathbf{w}\, y\, \overrightarrow{x}\, \mathbf{z}\,.\, d\, (\mathbf{z} \leq_\ell y)\, \left(h\, \mathbf{w}\, (\mathsf{drop}\, y\, (\mathsf{p}\, \mathbf{z}))\, \overrightarrow{x}\right) \\ \left(\mathsf{bit}\, (t_g\, \mathbf{w}\, \overrightarrow{x})\, (|z| - |y| - 1)\right) \\ f' := \lambda \mathbf{w}\, y\, \overrightarrow{x}\,.\, CR\, (\mathsf{aux}\, \mathbf{w}\, y\, \overrightarrow{x})\,, \end{aligned}$$

where $|z| - |y| - 1$ is computed as $\mathsf{len}\, (\mathsf{drop}\, \mathbf{z}\, (\mathsf{s}_1\, y))$. Finally, the computed value is reversed, and t_f is defined by

$$t_f := \lambda \mathbf{w}\, y\, \overrightarrow{x}\,.\,\mathsf{rev}\, (f'\, \mathbf{w}\, y\, \overrightarrow{x}\, \mathbf{w})\, \mathbf{w}\,.$$

In order for this to work, w has to be large enough for g and the h_i to be computed correctly by the inductive hypothesis, thus p_f needs to maximize p_g and the p_{h_i}. Also, w has to be long enough for the concatenation recursion in the definition of f' to actually compute all bits of $f(y, \overrightarrow{x})$, so $|w|$ has to be larger than $|y| + |g(\overrightarrow{x})|$. All this is guaranteed if we set

$$p_f(y, \overrightarrow{x}) := p_g(\overrightarrow{x}) + \sum_{i=1,2} p_{h_i}(y, \overrightarrow{x}) + y + q_g(\overrightarrow{x}) + 1 \; .$$

Finally, let f be defined by bounded logarithmic recursion from g, h and r,

$$\begin{aligned} f(0, \overrightarrow{x}) &= g(\overrightarrow{x}) \\ f(y, \overrightarrow{x}) &= h(y, \overrightarrow{x}, f(H(y), \overrightarrow{x})) && \text{for } y > 0 \\ f(y, \overrightarrow{x}) &\leq r(y, \overrightarrow{x}) \; , \end{aligned}$$

and let t_g and t_h be given by induction. In order to define t_f, we cannot use logarithmic recursion on y since y is incomplete. Instead we simulate the recursion on y by a recursion on a complete argument.

We first define a function Y that yields the values $H^{(k)}(y)$ that are needed in the recursion as

$$\begin{aligned} S &:= \lambda \mathbf{u}\, v^{\iota \multimap \iota}\, y \,.\, \big(\mathsf{d}\,(\mathbf{u} \leq_\ell \mathbf{z})\, y\, (\mathsf{half}\,(v\, y))\big) \\ Y &:= \lambda \mathbf{z}\, \mathbf{w} \,.\, \mathsf{LR}_{\iota \multimap \iota}\, (\lambda y.y)\, S\, \mathbf{w} \; . \end{aligned}$$

We now use this function to define a term f' computing f by recursion on a complete argument $\mathbf{z}$ by

$$\begin{aligned} T := \lambda \mathbf{u}\, v^{\iota \multimap \overrightarrow{\iota} \multimap \iota}\, y\, \overrightarrow{x} \,.\, \Big(&\mathsf{d}\,((Y\, \mathbf{u}\, \mathbf{w}\, y) = 0)\, (t_g\, \mathbf{w}\, y\, \overrightarrow{x}) \\ &\big(t_h\, \mathbf{w}\, (Y\, \mathbf{u}\, \mathbf{w}\, y)\, \overrightarrow{x}\, (v\, y\, \overrightarrow{x})\big)\Big) \\ f' := \lambda \mathbf{w}\, \mathbf{z} \,.\, \mathsf{LR}_{\iota \multimap \overrightarrow{\iota} \multimap \iota}\, (\lambda\, y\, \overrightarrow{x} \,.\, 0)\, T\, \mathbf{z} \end{aligned}$$

where the test $x = 0$ is implemented as $\mathsf{d}\,(\mathsf{bit}\,(\mathsf{s}_0\, x)\,(\mathsf{len}\, x))\, 1\, 0$. Finally, t_f is defined by identifying the complete arguments in f':

$$t_f := \lambda \mathbf{w} \,.\, f'\, \mathbf{w}\, \mathbf{w}$$

To show the correctness of this definition, define

$$p_f(y, \overrightarrow{x}) := 2y + p_h(y, \overrightarrow{x}, q_r(y, \overrightarrow{x})) + p_g(\overrightarrow{x})$$

and fix $y, \overrightarrow{x}$ and w with $|w| \geq p_f(|y|\,, \overrightarrow{|x|})$.

Note that the only values of z for which the function Y is ever invoked during the computation are $H^{(k)}(w)$ for $0 \leq k \leq ||y||$, and that for these values of z,

$Y(z, w, y)$ varies over the values $H^{(k)}(y)$. By a downward induction on k we show that for these values of z,

$$(f'(w, z, y, \overrightarrow{x}) = f(Y(z, w, y), \overrightarrow{x}) \ .$$

This implies the claim for t_f, since $Y(w, w, y) = y$.

The induction basis occurs for $k = ||y||$, where $V(z, w, y) = 0$. Since $|w| \geq 2\,|y|$, we have $z > 0$, thus the recursive step in the definition of f' is used, and the first branch of the case distinction is chosen. Therefore the equality follows from the fact that w is large enough for t_g to compute g correctly.

In the inductive step, we use the fact that $Y(H(z), w, y) = H(Y(z, w, y))$, and that w is large enough for t_h to compute h correctly. Since for $z = H^{(k-1)}(w)$ we have $Y(z, w, y) > 0$, we get

$$\begin{aligned} f'(w, z, y, \overrightarrow{x}) &= t_h(w, Y(z, w, y), \overrightarrow{x}, f'(w, H(z), y, \overrightarrow{x}) \\ &= t_h(w, Y(z, w, y), \overrightarrow{x}, f(Y(H(z), w, y), \overrightarrow{x}) \\ &= t_h(w, Y(z, w, y), \overrightarrow{x}, f(H(Y(z, w, y)), \overrightarrow{x}) \\ &= h(Y(z, w, y), \overrightarrow{x}, f(H(Y(z, w, y)), \overrightarrow{x})) \\ &= f(Y(z, w, y), \overrightarrow{x}) \end{aligned}$$

where the second equality holds by the induction hypothesis. This completes the proof of the claim and the theorem. □

5 Soundness

Definition 3. *The length $|t|$ of a term t is inductively defined as follows: For a variable x, $|x| = 1$, and for any constant c other than* d, *$|c| = 1$, whereas $|\mathsf{d}| = 3$. For complex terms we have the usual clauses $|r\,s| = |r| + |s|$ and $|\lambda x.r| = |r| + 1$.*

The length of the constant d is motivated by the desire to decrease the length of a term in the reduction of a d-redex.

Note that due to our identification of natural numbers with binary numerals, the notation $|n|$ is ambiguous now. Nevertheless, in the following we will only use $|n|$ as the term length defined above which for numerals n differs from the binary length only by one.

Definition 4. *For a list $\overrightarrow{n}$ of numerals, define $|\overrightarrow{n}| := \max(\overrightarrow{|n|})$.*

Definition 5. *A context is a list of pairs (x, n) of variables (complete or incomplete) of type ι and numerals, where all the variables are distinct. If $\overrightarrow{x}$ is a list of distinct variables of type ι and $\overrightarrow{n}$ a list of numerals of the same length, then we denote by $\overrightarrow{x}; \overrightarrow{n}$ the context $\overrightarrow{(x, n)}$.*

Definition 6. *For every symbol c of our language and term t, $\sharp_c(t)$ denotes the number of occurrences of c in t. For obvious aesthetic reasons we abbreviate $\sharp_{\#}(t)$ by $\#(t)$.*

Definition 7. *A term t is called* simple *if t contains none of the constants $\#$,* CR *or* LR.

Bounding the Size of Numerals

Lemma 1. *Let t be a simple, linear term of type ι and $\overrightarrow{x}; \overrightarrow{n}$ a context, such that all free variables in t are among $\overrightarrow{x}$. Then for $t^* := t[\overrightarrow{x} := \overrightarrow{n}]^{\mathrm{nf}}$ we have $|t^*| \leq |t| + |\overrightarrow{n}|$.*

Proof. By induction on $|t|$. We distinguish cases according to the form of t.

Case 1: t is $x\,\overrightarrow{r}$ for a variable x. Since x must be of type ι, $\overrightarrow{r}$ must be empty, and t^* is just one of the numerals in $\overrightarrow{n}$.

Case 2: t is $c\,\overrightarrow{r}$ for a constant c. Here we have four subcases, depending on the constant c.

Case 2a: c is 0, so $\overrightarrow{r}$ is empty and t is already normal.

Case 2b: c is s_i, so t is $c\,r$ for a term r of type ι. Let $r^* := r[\overrightarrow{x} := \overrightarrow{n}]^{\mathrm{nf}}$, by the induction hypothesis we have $|r^*| \leq |r| + |\overrightarrow{n}|$, and therefore we get $|t^*| \leq |r^*| + 1 \leq |t| + |\overrightarrow{n}|$.

Case 2c: c is one of the constants len, half, drop, bit or sm, so t is $c\,r\,\overrightarrow{s}$ for terms $r, \overrightarrow{s}$ of type ι. Let $r^* := r[\overrightarrow{x} := \overrightarrow{n}]^{\mathrm{nf}}$, by the induction hypothesis we have $|r^*| \leq |r| + |\overrightarrow{n}|$, and therefore we get $|t^*| \leq |r^*| \leq |t| + |\overrightarrow{n}|$.

Case 2d: c is d_σ, so t is $\mathsf{d}_\sigma\, s\, u_0\, u_1\, \overrightarrow{v}$, where s is of type ι and u_i are of type σ. Depending on the last bit i of the value of $s[\overrightarrow{x} := \overrightarrow{n}]$, t reduces to the shorter term $t' = u_i\, \overrightarrow{v}$, to which we can apply the induction hypothesis obtaining the normal form t^* with $|t^*| \leq |t'| + |\overrightarrow{n}| < |t| + |\overrightarrow{n}|$.

Case 3: t is $(\lambda x.r)\, s\, \overrightarrow{s}$. Here we have two subcases, depending on the number of occurrences of x in r.

Case 3a: x occurs at most once, then the term $t' := r[x := s]\, \overrightarrow{s}$ is smaller than t, and we can apply the induction hypothesis to t'.

Case 3b: x occurs more than once, and thus is of type ι. Then s is of type ι, so we first apply the induction hypothesis to s, obtaining $s^* := s[\overrightarrow{x} := \overrightarrow{n}]^{\mathrm{nf}}$ with $|s^*| \leq |s| + |\overrightarrow{n}|$. Now we let $t' := r\, \overrightarrow{s}$, and we apply the induction hypothesis to t' and the context $\overrightarrow{x}, y; \overrightarrow{n}, s^*$, so we get

$$|t^*| \leq |t'| + |\overrightarrow{n}, s^*| \leq |t'| + |s| + |\overrightarrow{n}| \ .$$

The last case, where t is $\lambda x.r$, cannot occur because of the type of t. □

Data Structure

We represent terms as parse trees, fulfilling the obvious typing constraints. The number of edges leaving a particular node is called the out-degree of this node. There is a distinguished node with in-degree 0, called the root. Each node is stored in a record consisting of an entry `cont` indicating its kind, plus some pointers to its children. We allow the following kinds of nodes with the given restrictions:

- Variable nodes representing a variable x. Variable nodes have out-degree 0. Every variable has a unique name and an associated register $\mathtt{R}[x]$.

- Abstraction nodes λx representing the binding of the variable x. Abstraction nodes have out-degree one, and we denote the pointer to its child by `succ`.
- For each constant c, there are nodes representing the constant c. These nodes have out-degree 0.
- Application nodes @ representing the application of two terms. The obvious typing constraints have to be fulfilled. We denote the pointers to the two children of an application node by `left` and `right`.
- Auxiliary nodes κ_i representing the composition of type one. These nodes are labeled with a natural number i, and each of those nodes has out-degree either 2 or 3. They will be used to form 2/3-trees (as e.g. described by Knuth [13]) representing numerals during the computation. We require that any node reachable from a $\kappa_\cdot$-node is either a $\kappa_\cdot$ node as well or one of the constants $\mathtt{s}_0$ or $\mathtt{s}_1$.
- Auxiliary nodes κ' representing the identification of type-one-terms with numerals (via "applying" them to 0). The out-degree of such a node, which is also called a "numeral node", either is zero, in which case the node represents the term 0, or the out-degree is one and the edge starting from this node either points to one of the constants $\mathtt{s}_0$ or $\mathtt{s}_1$ or to a $\kappa_\cdot$ node.
- Finally, there are so-called dummy nodes $\diamond$ of out-degree 1. The pointer to the child of a dummy node is again denoted by `succ`. Dummy nodes serve to pass on pointers: a node that becomes superfluous during reduction is made into a dummy node, and any pointer to it will be regarded as if it pointed to its child.

A tree is called a *numeral* if the root is a numeral node, all leaves have the same distance to the root and the label i of every κ_i node is the number of leaves reachable from that node. By standard operations on 2/3-trees it is possible in sequential logarithmic time to

- split a numeral at a given position i.
- find out the i'th bit of the numeral.
- concatenate two numerals.

So using κ' and $\kappa_\cdot$ nodes is just a way of implementing "nodes" labeled with a numeral allowing all the standard operations on numerals in logarithmic time. Note that the length of the label i (coded in binary) of a κ_i node is bounded by the logarithm of the number of nodes.

Normalization Algorithms and Their Complexity

Lemma 2. *Let t be a simple, linear term of type ι and $\overrightarrow{x}; \overrightarrow{n}$ a context such that all free variables in t are among the $\overrightarrow{x}$. Then the normal form of $t[\overrightarrow{x} := \overrightarrow{n}]$ can be computed in time $O(|t| \cdot \log |\overrightarrow{n}|)$ by $O(|t| \cdot |\overrightarrow{n}|)$ processors.*

Proof. We start one processor for each of the nodes of the parse-tree of t, with a pointer to this node in its local register. The registers associated to the variables $\overrightarrow{x}$ in the context contain pointers to the respective numerals $\overrightarrow{n}$, and the registers associated to all other variables are initialized with a `NULL` pointer.

The program operates in rounds, where the next round starts once all active processors have completed the current round. The only processors that will ever do something are those at the application or variable nodes. Thus all processors where $\texttt{cont} \notin \{@, x, \mathsf{d}\}$ can halt immediately. Processors at d nodes do not halt because they will be converted to variable nodes in the course of the reduction.

The action of a processor at an application node in one round depends on the type of its sons. If the right son is a dummy node, i.e., $\texttt{right.cont} = \diamond$, then this dummy is eliminated by setting $\texttt{right} := \texttt{right.succ}$. Otherwise, the action depends on the type of the left son.

- If $\texttt{left.cont} = \diamond$, then eliminate this dummy by setting $\texttt{left} := \texttt{left.succ}$.
- If $\texttt{left.cont} = \lambda x$, then this β-redex is partially reduced by copying the argument $\texttt{right}$ into the register $\texttt{R}[x]$ associated to the variable x. The substitution part of the β-reduction is then performed by the processors at variable nodes. Afterwards, replace the @ and λx nodes by dummies by setting $\texttt{cont} := \diamond$, $\texttt{left.cont} := \diamond$ and $\texttt{succ} := \texttt{left}$.
- If $\texttt{left.cont} \in \{\mathsf{s}_i, \mathsf{len}, \mathsf{half}\}$ and the right son is a numeral, $\texttt{right.cont} = \kappa'$, then replace the current node by a dummy, and let $\texttt{succ}$ point to a numeral representing the result. In the case of s_i and half, this can be implemented by 2/3-tree operations using sequential time $O(\log|\overrightarrow{n}|)$.
  In the case of len, the result is equal to the number i of leaves of the numeral argument. This value is read off the topmost κ_i node, and a numeral of that value is produced. Since i is a number of length $O(\log|\overrightarrow{n}|)$, this can also be done in sequential time $O(\log|\overrightarrow{n}|)$.
- If $\texttt{left.cont} = @$, $\texttt{left.left.cont} \in \{\mathsf{drop}, \mathsf{bit}\}$ and $\texttt{right}$ and $\texttt{left.right}$ both point to numerals, then again replace the current node by a dummy, and let $\texttt{succ}$ point to a numeral representing the result, which again can be computed by 2/3-tree operations in time $O(\log|\overrightarrow{n}|)$.
- If $\texttt{left.cont} = \texttt{left.left.cont} = @$, $\texttt{left.left.left.cont} = \mathsf{sm}$ and all of $\texttt{right}$, $\texttt{left.right}$ and $\texttt{left.left.right}$ point to numerals, then again the current node is replaced by a dummy with $\texttt{succ}$ pointing to the result.
  To compute the result, the lengths i and j are read off the second and third argument, and multiplied. As i and j are $O(\log|\overrightarrow{n}|)$ bit numbers, this can be done in parallel time $O(\log\log|\overrightarrow{n}|)$ by $O(\log^3|\overrightarrow{n}|)$ many processors.
  The product $i \cdot j$ is compared to the length of the first argument; let the maximum of both be k. Now the result is a numeral consisting of a one followed by k zeroes, which can be produced in parallel time $\log^2 k$ by $O(k)$ many processors using the square-and-multiply method, which suffices since $k \leq O(\log|\overrightarrow{n}|)$.
- Finally, if $\texttt{left.cont} = \mathsf{d}$ and $\texttt{right.cont} = \kappa'$, then extract the last bit b of the numeral at $\texttt{right}$, and create two new variables x_0 and x_1. Then reduce the d-redex by replacing the current node and the right son by abstraction nodes, and the left son by a variable node, i.e., setting $\texttt{cont} := \lambda x_0$, $\texttt{right.cont} := \lambda x_1$, $\texttt{succ.right}$, $\texttt{succ.succ} := \texttt{left}$ and $\texttt{left.cont} := x_b$.

A processor with $\texttt{cont} = x$ only becomes active when $\texttt{R}[x] \neq \texttt{NULL}$, and what it does then depends on the type of x.

If x is not of ground type, then the variable x occurs only in this place, so the substitution can be safely performed by setting $\mathtt{cont} := \diamond$ and $\mathtt{succ} := \mathtt{R}[x]$.

If x is of type ι, the processor waits until the content of register $\mathtt{R}[x]$ has been normalized, i.e., it acts only if $\mathtt{R}[x].\mathtt{cont} = \kappa'$. In this case, it replaces the variable node by a dummy, and lets succ point to a newly formed copy of the numeral in $\mathtt{R}[x]$. This copy can be produced in parallel time $O(\log|\overrightarrow{n}|)$ by $|\overrightarrow{n}|$ processors, since the depth of any numeral is bounded by $\log|\overrightarrow{n}|$.

Concerning correctness, note that the tree structure is preserved, since numerals being substituted for type ι variables are explicitly copied, and variables of higher type occur at most once. Obviously, no redex is left when the program halts.

For the time bound, observe that every processor performs at most one proper reduction plus possibly some dummy reductions. Every dummy reduction makes one dummy node unreachable, so the number of dummy reductions is bounded by the number of dummy nodes generated. Every dummy used to be a proper node, and the number of nodes is at most $2\,|t|$, so this number is bounded by $2\,|t|$. Thus at most $4\,|t|$ reductions are performed, and the program ends after at most that many rounds. As argued above, every round takes at most $O(\log|\overrightarrow{n}|)$ operations with $O(|\overrightarrow{n}|)$ many additional processors. □

The next lemma is the key to show that all terms can be normalized in NC: it shows how to eliminate the constants $\#$, LR and CR. As mentioned in the introduction, we have to distinguish between the program, i.e., the term we wish to normalize, and its input, given as a context. The runtime and length of the output term may depend polynomially on the former, but only polylogarithmically on the latter.

Since an ordinary $O(\cdot)$-analysis is too coarse for the inductive argument, we need a more refined asymptotic analysis. Therefore we introduce the following notation:

$$f(n) \lesssim g(n) \;:\Longleftrightarrow\; f(n) \le (1+o(1))g(n)\,,$$

or equivalently $\limsup_{n\to\infty} \frac{f(n)}{g(n)} \le 1$.

Lemma 3. *Let t be a linear term of linear type and $\overrightarrow{x};\overrightarrow{n}$ a context with all free variables of $t[\overrightarrow{x} := \overrightarrow{n}]$ incomplete. Then there are a term $\mathrm{simp}(t,\overrightarrow{x};\overrightarrow{n})$ and a context $\overrightarrow{y};\overrightarrow{m}$ such that $\mathrm{simp}(t,\overrightarrow{x};\overrightarrow{n})[\overrightarrow{y} := \overrightarrow{m}]$ is simple and equivalent to $t[\overrightarrow{x} := \overrightarrow{n}]$, and which can be computed in time*

$$T(|\overrightarrow{n}|) \;\lesssim\; 2^{\sharp_{\mathsf{LR}}(t)} \cdot |t| \cdot (2^{\#(t)} \cdot \log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(t)+2}$$

by

$$P(|\overrightarrow{n}|) \;\lesssim\; |t| \cdot |\overrightarrow{n}|^{2^{\#(t)}(\sharp_{\mathsf{CR}}(t)+2)} \cdot (\log|\overrightarrow{n}|)^{2^{\#(t)}(\sharp_{\mathsf{LR}}(t)+1)}$$

processors, such that

$$|\mathrm{simp}(t,\overrightarrow{x};\overrightarrow{n})| \;\lesssim\; |t| \cdot \left(2^{\#(t)} \cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(t)} \quad \textit{and} \quad |\overrightarrow{m}| \;\lesssim\; |\overrightarrow{n}|^{2^{\#(t)}}\,.$$

The proof is somewhat lengthy, so we sketch it first:

We start by describing the algorithm. It searches for the head-redex and reduces it in the obvious way (and then continues in the same way until the term is normal): in the case of a ground-type β-redex enlarge the context, in the case of a higher type β-redex reduce it in the term; in the case of LR the step term has to be unfolded only logarithmically many times, so we can just form a new term, whereas in the case of CR we have to use parallelism. However, in this case the result of every processor is just a single bit, so the results can be collected efficiently and returned to the *context* (whereas in the case of LR the result is a term of higher type). Note the crucial interplay between the length of the term, the size of the context, the running time and the number of processors needed; therefore we have to provide all four bounds simultaneously.

After the description of the algorithm, a long and tedious (but elementary) calculation follows showing that all bounds indeed hold in every case. The structure of the proof is always the same: in the interesting cases a numerical argument has to be evaluated in order to be able to reduce the redex (i.e., the numeral we recurse on, or the numeral to be put in the context in the case of a ground type β-redex). Then the induction hypothesis yields the size of this numeral and also the amount of time and processors needed. Then calculate the length of the unfolded term. The induction hypothesis for this term yields the amount of time and processors needed for the final computation, and also the bounds for the final output. Summing up all the times (calculation of the numeral, unfolding, final computation) one verifies that the time bound holds as well.

Proof (of lemma 3). By induction on $\sharp_{\mathsf{LR}}(t)$, with a side-induction on $|t|$ show that the following algorithm does it:

By pattern matching, determine in time $O(|t|)$ the form of t, and branch according to the form.

- If t is a variable or one of the constants 0 or d, then return t and leave $\overrightarrow{x}; \overrightarrow{n}$ unchanged.
- If t is $c\,\overrightarrow{s}$, where c is one of the constants s_i, drop, bit, len or sm then recursively simplify $\overrightarrow{s}$, giving $\overrightarrow{s^*}$ and contexts $\overrightarrow{y_j}; \overrightarrow{m_j}$, and return $c\,\overrightarrow{s^*}$ and $\overrightarrow{\overrightarrow{y}}; \overrightarrow{\overrightarrow{m}}$.
- If t is $\mathsf{d}\,r\,\overrightarrow{s}$, then simplify r giving r' and $\overrightarrow{y}; \overrightarrow{m}$. Compute the numeral $r^* := r'[\overrightarrow{y} := \overrightarrow{m}]^{\mathrm{nf}}$, and reduce the redex $\mathsf{d}\,r^*$, giving t', and recursively simplify $t'\,\overrightarrow{s}$ with context $\overrightarrow{x}; \overrightarrow{n}$.
- If t is $\#\,r$ then simplify r giving r' and $\overrightarrow{y}; \overrightarrow{m}$. Compute the numeral $r^* := r'[\overrightarrow{y} := \overrightarrow{m}]^{\mathrm{nf}}$, and return a new variable y' and the context $y'; 2^{|r^*|^2}$.
- If t is $\mathsf{CR}\,h\,r$, then simplify r giving r' and $\overrightarrow{y}; \overrightarrow{m}$, and compute the numeral $r^* := r'[\overrightarrow{y} := \overrightarrow{m}]^{\mathrm{nf}}$.
  Spawn $|r^*|$ many processors, one for each leaf of r^*, by moving along the tree structure of r^*. The processor at bit i of r^* simplifies $h\,\mathbf{z}$ in the context $\overrightarrow{x}, \mathbf{z}; \overrightarrow{n}, \lfloor r^*/2^i \rfloor$ (with $\mathbf{z}$ a new complete variable), giving a term h_i and context $\overrightarrow{y_i}; \overrightarrow{m_i}$, then he computes $h_i^* := h_i[y_i := m_i]^{\mathrm{nf}}$, retaining only the lowest order bit b_i.

The bits $\overrightarrow{b}$ are collected into a 2/3-tree representation of a numeral m, which is output in the form of a new variable z and the context $z; m$.

– t is $\mathsf{LR}\, g\, h\, m\, \overrightarrow{s}$ then simplify m, giving m' and $\overrightarrow{x_m}; \overrightarrow{n_m}$. Normalize m' in the context $\overrightarrow{x}, \overrightarrow{x_m}; \overrightarrow{n}, \overrightarrow{n_m}$, giving m^*. Form k numerals $m_i = \mathsf{half}^i(m^*)$ and sequentially simplify $\overrightarrow{h\, m.}$, giving $\overrightarrow{h'}$. (Of course, more precisely simplify $h\, x$ for a new variable x in the context extended by $x; m_i$.) Then form the term

$$t' := h'_0(h'_1 \ldots (h'_k\, g))\, \overrightarrow{s}$$

and simplify it.

– If t is of the form $\lambda x.r$ then recursively simplify r.

– If t is of the form $(\lambda x.r)\, s\, \overrightarrow{s}$ and x occurs at most once in r then recursively simplify $r[x := s]\, \overrightarrow{s}$.

– If t is of the form $(\lambda x.r)\, s\, \overrightarrow{s}$ and x occurs several times in r, then simplify s giving s' and a context $\overrightarrow{y}; \overrightarrow{m}$. Normalize s' in this context giving the numeral s^*. Then simplify $r\, \overrightarrow{s}$ in the context $\overrightarrow{x}, x; \overrightarrow{n}, s^*$.

For correctness, note that **in the case** $\mathsf{d}\, r\, \overrightarrow{s}$ simplifying r takes time

$$\lesssim\ 2^{\sharp_{\mathsf{LR}}(r)} \cdot |r| \cdot \left(2^{\#(r)} \cdot \log |\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r)+2}$$

and uses

$$\lesssim\ |r| \cdot |\overrightarrow{n}|^{2^{\#(r)}(\sharp_{\mathsf{CR}}(r)+2)}\, (\log |\overrightarrow{n}|)^{2^{\#(r)}(\sharp_{\mathsf{LR}}(r)+1)}$$

many processors. For the output we have $|r'| \lesssim |r| \cdot \left(2^{\#(r)} \cdot \log |\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r)}$ and $|\overrightarrow{m}| \lesssim |\overrightarrow{n}|^{2^{\#(r)}}$. Hence the time used to normalize r' (using the algorithm of lemma 2) is $O(|r'| \cdot \log |\overrightarrow{m}|)$, which is (order of)

$$|r| \cdot \left(2^{\sharp_{\mathsf{LR}}(r)} \cdot \log |\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r)+1}$$

and the number of processors needed is $O(|r'| \cdot |\overrightarrow{m}|) \leq |r| \cdot |\overrightarrow{n}|^{2^{\#(r)}+1}$. Finally, to simplify $t'\, \overrightarrow{s}$ we need time

$$\lesssim\ 2^{\sharp_{\mathsf{LR}}(\overrightarrow{s})} \cdot (|\overrightarrow{s}| + 3) \cdot (2^{\#(\overrightarrow{s})} \cdot \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(\overrightarrow{s})+2}$$

and the number of processors is

$$\lesssim\ (|\overrightarrow{s}| + 3)\, |\overrightarrow{n}|^{2^{\#(\overrightarrow{s})}(\sharp_{\mathsf{CR}}(\overrightarrow{s})+2)}\, (\log \overrightarrow{n})^{2^{\#(\overrightarrow{s})}(\sharp_{\mathsf{LR}}(\overrightarrow{s})+1)}$$

Summing up gives an overall time that is

$$\lesssim\ 2^{\sharp_{\mathsf{LR}}(t)} \cdot (|r| + |\overrightarrow{s}| + 3) \cdot \left(2^{\#(t)} \cdot \log |\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(t)+2}$$

which is a correct bound since $|\mathsf{d}| = 3$. Maximizing gives that the overall number of processors is

$$\lesssim\ |t| \cdot |\overrightarrow{n}|^{2^{\#(t)}(\sharp_{\mathsf{CR}}(t)+2)}\, (\log |\overrightarrow{n}|)^{2^{\#(t)}(\sharp_{\mathsf{LR}}(t)+1)}$$

The length of the output term is

$$\lesssim\ (|\overrightarrow{s}|+3)\cdot\left(2^{\#(\overrightarrow{s})}\cdot\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(\overrightarrow{s})}$$

and the size of the output context is $\lesssim|\overrightarrow{n}|^{2^{\#(\overrightarrow{s})}}$, which suffices.

In the case $\#\,r$ we obtain the same bounds for simplification and normalization of r as in the previous case. For r^* we get

$$|r^*|=O(|r'|+|\overrightarrow{m}|)\ \lesssim\ |\overrightarrow{n}|^{2^{\#(r)}}$$

Computing the output now takes time

$$\log|r^*|^2=2^{\#(r)+1}\cdot\log|\overrightarrow{n}|$$

and

$$|r^*|^2\ \lesssim\ |\overrightarrow{n}|^{2^{\#(r)+1}}$$

many processors. Thus the overall time is

$$\lesssim\ 2^{\sharp_{\mathsf{LR}}(r)}\cdot|r|\cdot\left(2^{\#(r)+1}\cdot\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r)+2}$$

and the number of processors is

$$\lesssim\ |r|\cdot|\overrightarrow{n}|^{2^{\#(r)+1}(\sharp_{\mathsf{CR}}(r)+2)}\cdot(\log|\overrightarrow{n}|)^{2^{\#(r)}(\sharp_{\mathsf{LR}}(r)+1)}\,.$$

The length of the output term is 1, and the size of the output context is bounded by $|r^*|^2+1\lesssim|\overrightarrow{n}|^{2^{\#(r)+1}}$, which implies the claim.

In the case $\mathsf{CR}\,h\,r$ note that the arguments $h\colon\iota\to\iota$ and $r\colon\iota$ both have to be present, since t has to be of linear type (and $\mathsf{CR}\colon(\iota\to\iota)\multimap\iota\to\iota$). We obtain the same bounds for simplification and normalization of r and the length of the numeral r^* as in the previous case. Spawning the parallel processors and collecting the result in the end each needs time $\log|r^*|=2^{\#(r)}\cdot|\overrightarrow{n}|$. The main work is done by the $|\overrightarrow{n}|^{2^{\#(r)}}$ many processors that do the simplification and normalization of the step terms. Each of them takes time

$$\begin{aligned}&\lesssim 2^{\sharp_{\mathsf{LR}}(h)}\cdot(|h|+1)\cdot\left(2^{\#(h)}\cdot\log|\overrightarrow{n},r^*|\right)^{\sharp_{\mathsf{LR}}(h)+2}\\&\lesssim 2^{\sharp_{\mathsf{LR}}(h)}\cdot(|h|+1)\cdot\left(2^{\#(h)+\#(r)}\cdot\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+2}\end{aligned}$$

and a number of sub-processors satisfying

$$\begin{aligned}&\lesssim(|h|+1)\cdot|\overrightarrow{n},r^*|^{2^{\#(h)}(\sharp_{\mathsf{CR}}(h)+2)}\cdot(\log|\overrightarrow{n},r^*|)^{2^{\#(h)}(\sharp_{\mathsf{LR}}(h)+1)}\\&\lesssim(|h|+1)\cdot|\overrightarrow{n}|^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{CR}}(h)+2)}\cdot(2^{\#(r)}\log|\overrightarrow{n}|)^{2^{\#(h)}(\sharp_{\mathsf{LR}}(h)+1)}\end{aligned}$$

$$\lesssim (|h|+1)\cdot |\overrightarrow{n}|^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{CR}}(h)+2)}\cdot (\log|\overrightarrow{n}|)^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{LR}}(h)+1)}$$

to compute h_i and $\overrightarrow{y_i}; \overrightarrow{m_i}$ with

$$\begin{aligned}|h_i| &\lesssim (|h|+1)\cdot \left(2^{\#(h)}\cdot \log|\overrightarrow{n}, r^*|\right)^{\sharp_{\mathsf{LR}}(h)}\\ &\lesssim (|h|+1)\cdot \left(2^{\#(h)+\#(r)}\cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)}\end{aligned}$$

and

$$|\overrightarrow{m_i}| \lesssim |\overrightarrow{n}, r^*|^{2^{\#(h)}} \lesssim |\overrightarrow{n}|^{2^{\#(h)+\#(r)}}$$

Now the normal form h_i^* is computed in time

$$O(|h_i|\cdot \log|\overrightarrow{m_i}|) \lesssim (|h|+1)\cdot \left(2^{\#(h)+\#(r)}\cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+1}$$

by

$$O(|h_i|\cdot |\overrightarrow{m_i}|) \lesssim (|h|+1)\,|\overrightarrow{n}|^{2^{\#(h)+\#(r)}+1}\cdot (\log|\overrightarrow{n}|)^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{LR}}(h)+1)}$$

many sub-processors. Summing up the times yields that the overall time is

$$\begin{aligned}&\lesssim 2^{\sharp_{\mathsf{LR}}(r)}\cdot |r|\cdot \left(2^{\#(r)}\cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r)+2}\\ &\quad + 2^{\sharp_{\mathsf{LR}}(h)}\cdot (|h|+1)\cdot \left(2^{\#(h)+\#(r)}\cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+2}\\ &\lesssim 2^{\sharp_{\mathsf{LR}}(t)}\cdot (|r|+|h|+1)\cdot \left(2^{\#(t)}\cdot \log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(t)+2}\end{aligned}$$

The number of sub-processors used by each of the $|r^*|$ processes is

$$\lesssim (|h|+1)\cdot |\overrightarrow{n}|^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{CR}}(h)+2)}\cdot (\log|\overrightarrow{n}|)^{2^{\#(h)+\#(r)}(\sharp_{\mathsf{LR}}(h)+1)}$$

and multiplying this by the upper bound $|\overrightarrow{n}|^{2^{\#(r)}}$ on the number of processes yields that the bound for the total number of processor holds. The output term is of length 1, and the length of the output context is bounded by $|r^*| \lesssim |\overrightarrow{n}|^{2^{\#(r)}}$.

In the case $\mathsf{LR}\, g\, h\, m\, \overrightarrow{s}$ note that, as t has linear type, all the arguments up to and including the m have to be present. Moreover, h is in a complete position, so it cannot contain incomplete free variables, therefore neither can do any of the $\overrightarrow{h'}$; so t' really is linear. Due to the typing restrictions of the LR the step functions $\overrightarrow{hm.}$ have linear type. So in all cases we're entitled to recursively call the algorithm and to apply the induction hypothesis. For calculating m^* we have the same bounds as in the previous cases. We have $k \sim \log|m^*| \lesssim 2^{\#(m)}\log|\overrightarrow{n}|$. The time needed for calculating the $\overrightarrow{h'}$ is

$$\leq k 2^{\sharp_{\mathsf{LR}}(h)}(|h|+1)(2^{\#(h)}\log|\overrightarrow{n}, m^*|)^{\sharp_{\mathsf{LR}}(h)+2}$$

$$\lesssim 2^{\sharp_{\mathsf{LR}}(h)}(|h|+1)(2^{\#(h)+\#(m)}\log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(h)+3}$$

For the length $\left|\overrightarrow{h'}\right|$ we have

$$\begin{aligned}\left|\overrightarrow{h'}\right| &\lesssim (|h|+1)\left(2^{\#(h)}\cdot\log|\overrightarrow{n},m^*|\right)^{\sharp_{\mathsf{LR}}(h)}\\ &\lesssim (|h|+1)\left(2^{\#(h)+\#(m)}\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)}\end{aligned}$$

and the length of the numerals $\overrightarrow{\overrightarrow{n}}$ in the contexts output by the computation of the $\overrightarrow{h'}$ is bounded by $|\overrightarrow{n},m^*|^{2^{\#(h)}} \lesssim (|\overrightarrow{n}|^{2^{\#(m)}})^{2^{\#(h)}} = |\overrightarrow{n}|^{2^{\#(m)+\#(h)}}$. For the length of t' we have

$$\begin{aligned}|t'| &\le k\left|\overrightarrow{h'}\right| + |g| + \overrightarrow{|s|}\\ &\lesssim (|h|+|g|+\overrightarrow{|s|}+1)\left(2^{\#(h)+\#(m)}\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+1}\end{aligned}$$

So the final computation takes time

$$\begin{aligned}&\lesssim 2^{\sharp_{\mathsf{LR}}(t')}\,|t'|\,(2^{\#(t')}\log\left|\overrightarrow{n},\overrightarrow{\overrightarrow{n}}\right|)^{\sharp_{\mathsf{LR}}(t')+2}\\ &\lesssim 2^{\sharp_{\mathsf{LR}}(g)+\overrightarrow{\sharp_{\mathsf{LR}}(s)}}(|h|+|g|+\overrightarrow{|s|}+1)(2^{\#(h)+\#(m)}\log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(h)+1}\\ &\qquad\cdot(2^{\#(g)+\overrightarrow{\#(s)}}\cdot 2^{\#(m)+\#(h)}\log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(g)+\overrightarrow{\sharp_{\mathsf{LR}}(s)}+2}\\ &\lesssim 2^{\sharp_{\mathsf{LR}}(g)+\overrightarrow{\sharp_{\mathsf{LR}}(s)}}(|h|+|g|+\overrightarrow{|s|}+1)(2^{\#(t)}\log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(h)+1+\sharp_{\mathsf{LR}}(g)+\overrightarrow{\sharp_{\mathsf{LR}}(s)}+2}.\end{aligned}$$

So summing up all the times one verifies that the time bound holds. The number of processors needed in the final computation is

$$\begin{aligned}&\lesssim |t'|\cdot\left|\overrightarrow{n},\overrightarrow{\overrightarrow{n}}\right|^{2^{\#(t')}(\sharp_{\mathsf{CR}}(t')+2)}(\log\left|\overrightarrow{n},\overrightarrow{\overrightarrow{n}}\right|)^{2^{\#(t')}(\sharp_{\mathsf{LR}}(t')+1)}\\ &\lesssim (|h|+|g|+\overrightarrow{|s|}+1)\left(2^{\#(h)+\#(m)}\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+1}\\ &\quad\cdot\left(|\overrightarrow{n}|^{2^{\#(m)+\#(h)}}\right)^{2^{\#(t')}(\sharp_{\mathsf{CR}}(t')+2)}\\ &\quad\cdot\left(2^{\#(m)+\#(h)}\log|\overrightarrow{n}|\right)^{2^{\#(t')}(\sharp_{\mathsf{LR}}(t')+1)}\\ &\lesssim (|h|+|g|+\overrightarrow{|s|}+1)\cdot|\overrightarrow{n}|^{2^{\#(m)+\#(h)+\#(t')}(\sharp_{\mathsf{CR}}(t')+2)}\\ &\quad\cdot\left(2^{\#(m)+\#(h)}\log|\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(h)+1+2^{\#(t')}(\sharp_{\mathsf{LR}}(t')+1)}\\ &\lesssim |t|\cdot|\overrightarrow{n}|^{2^{\#(t)}(\sharp_{\mathsf{CR}}(t)+2)}\cdot\left(2^{\#(m)+\#(h)}\log|\overrightarrow{n}|\right)^{2^{\#(t')}(\sharp_{\mathsf{LR}}(h)+\sharp_{\mathsf{LR}}(t')+2)}\\ &\lesssim |t|\cdot|\overrightarrow{n}|^{2^{\#(t)}(\sharp_{\mathsf{CR}}(t)+2)}\cdot(\log|\overrightarrow{n}|)^{2^{\#(m)+\#(h)+\#(t')}(\sharp_{\mathsf{LR}}(h)+\sharp_{\mathsf{LR}}(t')+2)}\end{aligned}$$

The context finally output is bounded by $\lesssim \left|\overrightarrow{n},\overrightarrow{\overrightarrow{n}}\right|^{2^{\#(t')}} \lesssim |\overrightarrow{n}|^{2^{\#(m)+\#(h)+\#(t')}}$. The length of the final output is bounded by

$$\lesssim |t'|\cdot(2^{\#(t')}\log\left|\overrightarrow{n},\overrightarrow{\overrightarrow{n}}\right|)^{\sharp_{\mathsf{LR}}(t')}$$

$$\begin{aligned}
&\lesssim (|h| + |g| + \overrightarrow{|s|} + 1)(2^{\#(h)+\#(m)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(h)+1} \\
&\qquad\qquad \cdot (2^{\#(t')+\#(m)+\#(h)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(t')} \\
&\lesssim (|h| + |g| + \overrightarrow{|s|} + 1)(2^{\#(t')+\#(m)+\#(h)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(t')+\sharp_{\mathsf{LR}}(h)+1}
\end{aligned}$$

So all bounds hold in this case.

In the case $\lambda x.r$ note that due to the fact that t has linear type x has to be incomplete, so we're entitled to use the induction hypothesis.

In the case $(\lambda x.r)\, s\, \overrightarrow{s}$ **with *several* occurrences of** x in r note that due to the fact that t is linear, x has to be of ground type (since higher type variables are only allowed to occur once). The time needed to calculate s' is bounded by

$$2^{\sharp_{\mathsf{LR}}(s)}\, |s|\, (2^{\#(s)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(s)+2}$$

and the number of processors is not too high. For the length of s' we have

$$|s'| \;\lesssim\; |s| \cdot (2^{\#(s)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(s)}$$

and $|\overrightarrow{m}| \lesssim |\overrightarrow{n}|^{2^{\#(s)}}$. So the time for calculating s^* is bounded by

$$\lesssim\; |s'| \log |\overrightarrow{n}, \overrightarrow{m}| \;\lesssim\; |s| \cdot (2^{\#(s)} \log |\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(s)+1}$$

For the length of the numeral s^* we have

$$|s^*| \leq |s'| + |\overrightarrow{n}, \overrightarrow{m}| \;\lesssim\; |\overrightarrow{n}|^{2^{\#(s)}}$$

So the last computation takes time

$$2^{\sharp_{\mathsf{LR}}(r\,\overrightarrow{s})} \cdot |r\,\overrightarrow{s}| \left(2^{\#(r\,\overrightarrow{s})+\#(s)} \log |\overrightarrow{n}|\right)^{\sharp_{\mathsf{LR}}(r\,\overrightarrow{s})+2}.$$

Summing up, the time bound holds. The number of processors needed for the last computation is bounded by

$$\begin{aligned}
&\lesssim |r\,\overrightarrow{s}| \cdot \left(|\overrightarrow{n}|^{2^{\#(s)}}\right)^{2^{\#(r\,\overrightarrow{s})}(\sharp_{\mathsf{CR}}(r\,\overrightarrow{s})+2)} \left(2^{\#(s)} \log |\overrightarrow{n}|\right)^{2^{\#(r\,\overrightarrow{s})}(\sharp_{\mathsf{LR}}(r\,\overrightarrow{s})+1)} \\
&\lesssim |t| \cdot |\overrightarrow{n}|^{2^{\#(s)+\#(r\,\overrightarrow{s})}(\sharp_{\mathsf{CR}}(r\,\overrightarrow{s})+2)} (\log |\overrightarrow{n}|)^{2^{\#(s)+\#(r\,\overrightarrow{s})}(\sharp_{\mathsf{LR}}(r\,\overrightarrow{s})+1)}
\end{aligned}$$

The context finally output is bounded by $|\overrightarrow{n}, s^*|^{2^{\#(r\,\overrightarrow{s})}} \lesssim |\overrightarrow{n}|^{2^{\#(s)+\#(r\,\overrightarrow{s})}}$.

In all other cases the bounds trivially hold. □

We conclude that a term of linear type can be simplified by an NC algorithm, where the degree of the runtime bound only depends on the number of occurrences of LR, and the degree of the hardware bound only depends on the number of occurrences of # and CR. More precisely, we have the following corollary.

Corollary 1. *The term* $\mathrm{simp}(t, \overrightarrow{x}; \overrightarrow{n})$ *and the new context* $\overrightarrow{y}; \overrightarrow{m}$ *in the above lemma can be computed in time*

$$T(|\overrightarrow{n}|) \;\leq\; O((\log|\overrightarrow{n}|)^{\sharp_{\mathsf{LR}}(t)+2})$$

by a number of processors satisfying

$$P(|\overrightarrow{n}|) \;\leq\; O(|\overrightarrow{n}|^{2^{\#(t)}(\sharp_{\mathsf{CR}}(t)+3)}) \;.$$

Theorem 3. *Let* t *be a linear term of type* $\overrightarrow{\iota} \to \iota$*. Then the function denoted by* t *is in NC.*

Proof. Let $\overrightarrow{n}$ be an input, given as 2/3-tree representations of numerals, and $\overrightarrow{\mathbf{x}}$ complete variables of type ι. Using Lemma 3, we compute $t' := \mathrm{simp}(t\,\overrightarrow{\mathbf{x}}, \overrightarrow{\mathbf{x}}; \overrightarrow{n})$ and a new context $\overrightarrow{y}; \overrightarrow{m}$ with $|t'| \leq (\log|\overrightarrow{n}|)^{O(1)}$ and $|\overrightarrow{m}| \leq |\overrightarrow{n}|^{O(1)}$ in time $(\log|\overrightarrow{n}|)^{O(1)}$ by $|\overrightarrow{n}|^{O(1)}$ many processors.

Then using Lemma 2 we compute the normal form $t'[\overrightarrow{y} := \overrightarrow{m}]^{\mathrm{nf}}$ in time $O(|t'| \cdot \log|\overrightarrow{m}|) = (\log|\overrightarrow{n}|)^{O(1)}$ by $O(|t'|\,|\overrightarrow{m}|) = |\overrightarrow{n}|^{O(1)}$ many processors.

Hence the function denoted by t is computable in polylogarithmic time by polynomially many processors, and thus is in NC. □

From Theorems 2 and 3 we immediately get our main result:

Corollary 2. *A number-theoretic function* f *is in NC if and only if it is denoted by a linear term of our system.*

References

1. B. Allen. Arithmetizing uniform NC. *Annals of Pure and Applied Logic*, 53(1):1–50, 1991.
2. S. Bellantoni. *Predicative Recursion and Computational Complexity.* PhD thesis, University of Toronto, 1992.
3. S. Bellantoni. Characterizing parallel time by type 2 recursions with polynomial output length. In D. Leivant, editor, *Logic and Computational Complexity*, pages 253–268. Springer LNCS 960, 1995.
4. S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.
5. S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104:17–30, 2000.
6. S. Bellantoni and I. Oitavem. Separating NC along the δ axis. Submitted, 2001.
7. S. Bloch. Function-algebraic characterizations of log and polylog parallel time. *Computational Complexity*, 4:175–205, 1994.
8. P. Clote. Sequential, machine independent characterizations of the parallel complexity classes $ALogTIME$, AC^k, NC^k and NC. In S. Buss and P. Scott, editors, *Feasible Mathematics*, pages 49–69. Birkhäuser, 1990.
9. A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of the second International Congress on Logic, Methodology and Philosophy of Science*, pages 24–30, 1965.

10. K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.
11. M. Hofmann. Programming languages capturing complexity classes. *ACM SIGACT News*, 31(2), 2000. Logic Column 9.
12. M. Hofmann. Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 104:113–166, 2000.
13. D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1998.
14. D. Leivant. Stratified functional programs and computational complexity. In *Proc. of the 20th Symposium on Principles of Programming Languages*, pages 325–333, 1993.
15. D. Leivant. A characterization of NC by tree recurrence. In *Proc. 39th Symposium on Foundations of Computer Science*, pages 716–724, 1998.
16. D. Leivant and J.-Y. Marion. A characterization of alternating log time by ramified recurrence. *Theoretical Computer Science*, 236:193–208, 2000.

Reflective λ-Calculus

Jesse Alt and Sergei Artemov*

Cornell University,
Ithaca, NY 14853, U.S.A.
{jma35, artemov}@cs.cornell.edu

Abstract. We introduce a general purpose typed λ-calculus $\boldsymbol{\lambda}^{\infty}$ which contains intuitionistic logic, is capable of internalizing its own derivations as λ-terms and yet enjoys strong normalization with respect to a natural reduction system. In particular, $\boldsymbol{\lambda}^{\infty}$ subsumes the typed λ-calculus. The Curry-Howard isomorphism converting intuitionistic proofs into λ-terms is a simple instance of the internalization property of $\boldsymbol{\lambda}^{\infty}$. The standard semantics of $\boldsymbol{\lambda}^{\infty}$ is given by a proof system with proof checking capacities. The system $\boldsymbol{\lambda}^{\infty}$ is a theoretical prototype of reflective extensions of a broad class of type-based systems in programming languages, provers, AI and knowledge representation, etc.

1 Introduction

According to the Curry-Howard isomorphism, the calculus of intuitionistic propositions (types) and the calculus of typed λ-terms (proof terms) constitute a pair of isomorphic though distinct structures. Combining those logical and computational universes has been considered a major direction in theoretical logic and applications (propositions-as-types, proofs-as-programs paradigms, etc., cf. [5,7,11,12]). Modern computational systems are often capable of performing logical derivations, formalizing their own derivations and computations, proof and type checking, normalizing λ-terms, etc. A basic theoretical prototype of such a system would be a long anticipated joint calculus of propositions (types) and typed λ-terms (proofs, programs). There are several natural requirements for such a calculus raising from the intended reading of the type assertion $t:F$ as a proposition *t is a proof of F*.

1. A type assertion $t:F$ should be treated as a legitimate proposition (hence a type). The intended informal semantics of $t:F$ could be *t is a proof of F* or *t has type F*. In particular, $t:F$ could participate freely in constructing new types (propositions). Such a capacity would produce types containing λ-terms (proofs, programs) inside. For example, the following types should be allowed $t:F \to s:G$, $t:F \wedge s:G$, $s:(t:F)$, etc. In programmistic terms this means a possibility of encoding computations inside types. A system should contain

* The research described in this paper was supported in part by ARO under the MURI program "Integrated Approach to Intelligent Systems", grant DAAH04-96-1-0341, by DARPA under program LPE, project 34145.

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 22-37, 2001.

both the **intuitionistic logic** (as a calculus of types) and the **simply typed λ-calculus**.

2. Such a system should contain the **reflection principle** $t : F \to F$ representing a fundamental property of proofs *if t is a proof of F then F holds*. An alternative type-style reading of this principle says *if t of type F then F is inhabited*.

3. A system should contain the explicit **proof checking** (the **type checking**) **operation** "!" and the principle $t{:}F \to !t{:}(t{:}F)$. The informal reading of "!" is that given a term t the term $!t$ describes a computation verifying $t{:}F$ (i.e. that t is a proof of F, or that t has type F). The proof checking and the reflection principles enable us to naturally connect relevant types up and down from a given one. Such a possibility has been partly realized by means of modal logic in modal principles $\Box F \to F$ and $\Box F \to \Box\Box F$ respectively, though this presentation lacks the explicit character of typed terms (cf. [3,6]).

4. A system should accommodate the Curry-Howard isomorphism that maps natural derivations in intuitionistic logic to well defined typed λ-terms. A fundamental closure requirement suggests generalizing the Curry-Howard isomorphism to the **Internalization Property** of the system: *if $A_1, A_2, \ldots, A_n \vdash B$ then for fresh variables $x_1, x_2, \ldots, x_n$ and some term $t(x_1, x_2, \ldots, x_n)$*

$$x_1{:}A_1, x_2{:}A_2, \ldots, x_n{:}A_n \vdash t(x_1, x_2, \ldots, x_n){:}B.$$

5. Desirable properties of terms in such system include strong normalizability with respect to β-reduction, projection reductions, etc., as well as other natural properties of typed λ-terms.

The main goal of this paper is to find such a system. The reflective λ-calculus $\boldsymbol{\lambda^{\infty}}$ introduced below is the minimal system that meets these specifications.

The Logic of Proofs **LP** ([1,2,3]) where proofs are represented by advanced combinatory terms (called *proof polynomials*), satisfy properties 1–4 above. The property 5 is missing in **LP** since the combinatory format does not admit reductions and thus does not leave a room for nontrivial normalizations.

Reflexive λ-terms correspond to proof polynomials over intuitionistic logic in the basis $\{\to, \wedge\}$. Those polynomials are built from constants and variables by two basic operations: "·" (application), "!" (proof checker). The logical identities for those proof polynomials are described by the corresponding fragment of the Logic of Proofs **LP**. In **LP** the usual set of type formation rules is extended by a new one: given a proof polynomial p and formula F build a new type (propositional formula) $p{:}F$. Valid identities in this language are given by propositional formulas in the extended language generated by the calculus having axioms

A0. Axiom schemes and rules of intuitionistic logic
A1. $t{:}F \to F$ *"verification"*
A2. $t{:}(F \to G) \to (s{:}F \to (t \cdot s){:}G)$ *"application"*
A3. $t{:}F \to !t{:}(t{:}F)$ *"proof checker"*

Rule *axiom necessitation*:
given a constant c and an axiom F from A0–A3 infer $c{:}F$.

The standard semantics of proof polynomials is operations on proofs in a proof system containing intuitionistic logic. The operation "application" has the usual meaning as applying a proof t of $F \to G$ to a proof s of F to get a proof $t \cdot s$ of G. The "proof checking" takes a proof t of F and returns a proof $!t$ of the sentence t *is a proof of* F. Logic of Proofs in the combinatory format finds applications in those areas where modal logics, epistemic logics, logics of knowledge work.

The λ-version of the whole class of proof polynomials is not normalizing. Indeed, the important feature of proof polynomials is their polymorphism. In particular, a variable can be assigned a finite number of arbitrary types. If a variable x has two types A and $A \to A$ then the λ-term $(\lambda x.xx) \cdot (\lambda x.xx)$ does not have a normal form. In order to find a natural normalizing subsystem we limit our considerations to λ-terms for single-conclusion (functional) proof systems, i.e. systems where each proof proves only one theorem. Such terms will have unique types. Proof polynomials in the combinatory format for functional proof systems has been studied in [8,9].

Requirements 1–4 above in the λ-term format inevitably lead to the system $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ below. The key idea of having nested copies of basic operations on λ-terms in $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ can be illustrated by the following examples. By the Internalization Property, a propositional derivation $A \to B, A \vdash B$ yields a λ-term derivation[1]

$$x_1{:}(A \to B), y_1{:}A \vdash (x_1 \circ y_1){:}B,$$

which yields the existence of a term $t(x_2, y_2)$ such that

$$x_2{:}x_1{:}(A \to B), y_2{:}y_1{:}A \vdash t(x_2, y_2){:}(x_1 \circ y_1){:}B.$$

(here and below ":" is meant to be right-associative). Naturally, we opt to accept such t as a basic operation and call it $\circ^2$. The defining identity for $\circ^2$ is thus

$$x_2{:}x_1{:}(A \to B), y_2{:}y_1{:}A \vdash (x_2 \circ^2 y_2){:}(x_1 \circ y_1){:}B,$$

or, in the alternative notation t^F for type assigmnents

$$x_2^{x_1^{A \to B}}, y_2^{y_1^A} \vdash (x_2 \circ^2 y_2)^{(x_1 \circ y_1)^B}.$$

Likewise, the Internalization Property yields the existence of the operation $\circ^3$ such that

$$x_3{:}x_2{:}x_1{:}(A \to B), y_3{:}y_2{:}y_1{:}A \vdash (x_3 \circ^3 y_3){:}(x_2 \circ^2 y_2){:}(x_1 \circ y_1){:}B$$

Similar nested counterparts appear for λ-abstraction, pairing, and projections.

[1] Here "$\circ$" denotes application on λ-terms.

New series of operations are generated by the reflection and proof checking principles. The reflection has form $t : A \vdash A$. The internalized version of this derivation gives a unary operation $\Downarrow$ such that

$$x_1 : t : A \vdash \Downarrow x_1 : A.$$

Likewise, for some unary operation $\Downarrow^{\mathbf{2}}$

$$x_2 : x_1 : t : A \vdash \Downarrow^{\mathbf{2}} x_2 : \Downarrow x_1 : A,$$

etc.

The proof checking derivation $t : A \vdash !t : t : A$ gives rise to a unary operation $\Uparrow$ such that

$$x_1 : t : A \vdash \Uparrow x_1 : !t : t : A.$$

Further application of Internalization produces $\Uparrow^{\mathbf{2}}$, $\Uparrow^{\mathbf{3}}$, etc. such that

$$x_2 : x_1 : t : A \vdash \Uparrow^{\mathbf{2}} x_2 : \Uparrow x_1 : !t : t : A,$$

$$x_3 : x_2 : x_1 : t : A \vdash \Uparrow^{\mathbf{3}} x_3 : \Uparrow^{\mathbf{2}} x_2 : \Uparrow x_1 : !t : t : A,$$

etc.

Theorem 1 below demonstrates that such a set of nested operations on λ-terms is in fact necessary and sufficient to guarantee the Internalization Property for the whole of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$. Therefore, there is nothing arbitrary in this set, and nothing is missing there either.

Normalization of terms depends upon a choice of reductions, which may vary from one application to another. In this paper we consider a system of reductions motivated by our provability reading of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$. We consider a normalization process as a kind of search for a direct proof of a given fixed formula (type). In particular, we try to avoid changing a formula (type) during normalization. This puts some restriction on our system of reductions. As a result, the reflective λ-terms under the chosen reductions are strongly normalizable, but not confluent. A given term may have different normal forms. An additional motivation for considering those reductions is that they extend the usual set of λ-calculus reductions and subsume strong normalization for such components of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ as the intuitionistic calculus and the simply typed λ-calculus.

2 Reflective λ-Calculus

The reflective λ-calculus $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ below is a joint calculus of propositions (types) and proofs (λ-terms) with rigid typing. Every term and all subterms of a term carry a fixed type. In other words, in $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ we assume a Church style rigid typing rather than a Curry style type assignment system.

2.1 Types and Typed Terms

The language of reflective λ-calculus includes

propositional letters $p_1, p_2, p_3, \ldots$

type constructors (connectives) $\rightarrow, \wedge$

term constructors (functional symbols): unary ! , $\Uparrow^n$, $\Downarrow^n$, $\boldsymbol{\pi}_0^n$, $\boldsymbol{\pi}_1^n$; binary $\circ^n$, $\mathbf{p}^n$, for $n = 1, 2, 3 \ldots$

operator symbols ':', λ^1, λ^2, ..., λ^n, ...;

a countably infinite supply of *variables* $x_1, x_2, x_3, \ldots$ of each type F (definition below), each variable x is a term of its unique pre-assigned type.

Types and *(well-typed, well-defined, well-formed) terms* are defined by a simultaneous induction according to the calculus $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ below.

1. Propositional letters are *(atomic) types*
2. *Types* (formulas) F are built according to the grammar

$$F = p \mid F \rightarrow F \mid F \wedge F \mid t{:}F,$$

where p is an atomic type, t a well-formed term having type F. Types of format $t:F$ where t is a term and F a type are called *type assertions* or *quasi-atomic types*. Note that only correct type assertions $t:F$ are syntactically allowed inside types. The informal semantics for $t:F$ is *t is a proof of F*; so a formula

$$t_n{:}t_{n-1}{:}\ldots{:}t_1{:}A$$

can be read as "t_n is a proof that t_{n-1} is a proof that ... is a proof that t_1 proves A". For the sake of brevity we will refer to types as terms of depth 0.

3. *Inhabited types* and well-formed terms (or terms for short) are constructed according to the calculus $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ below.

A derivation in $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ is a rooted tree with nodes labelled by types, in particular, type assertions. Leaves of a derivation are labelled by axioms of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ which are arbitrary types or type assertions $x:F$ where F is a type and x a variable of type F. Note that the set of axioms is thus also defined inductively according to $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$: as soon as we are able to establish that F is a type (in particular, for a quasi-atomic type $s{:}G$ this requires establishing by means of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ that s indeed is a term of type G), we are entitled to use variables of type F as new axioms.

A *context* is a collection of quasi-atomic types $x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$ where x_i, x_j are distinct variables for $i \neq j$. A derivation tree is *in a context Γ* if all leaves of the derivation are labelled by some quasi-atomic types from Γ.

A step down from leaves to the root is performed by one of the inference rules of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$. Each rule comes in levels $n = 0, 1, 2, 3, \ldots$. A rule has one or two premises which are types (in particular, type assertions), and a conclusion. The intended reading of such a rule is that if premises are inhabited types, then the conclusion is also inhabited. If the level of a rule is greater than 0, then the premise(s) and the conclusion are all type assertions. Such a rule is regarded also as a term formation rule with the intended reading: *the conclusion $t:F$ is a correct type assertion provided the premise(s) are correct.*

If $t:F$ appears as a label in (the root of) a derivation tree, we say that t is a *term of type* F. We also refer to terms as well-defined, well-typed, well-formed terms.

In $\boldsymbol{\lambda^\infty}$ we use the natural deduction format, where derivations are represented by proof trees with assumptions, both open (charged) and closed (discharged). We will also use the sequent style notation for derivations in $\boldsymbol{\lambda^\infty}$ by reading $\Gamma \vdash F$ as an $\boldsymbol{\lambda^\infty}$-derivation of F in Γ. Within the current definition below we assume that $n = 0, 1, 2, \ldots$ and $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$. In particular, if $n = 0$ then $\boldsymbol{v}$ is empty. We also agree on the following vector-style notations:

$\boldsymbol{t}:A$ denotes $t_n:t_{n-1}:\ldots:t_1:A$ (e.g. $\boldsymbol{t}:A$ is A, when $n=0$),

$\boldsymbol{t}:\{A_1, A_2, \ldots, A_n\}$ denotes $\{t_1:A_1, t_2:A_2, \ldots, t_n:A_n\}$,

$\lambda^n\boldsymbol{x}.\boldsymbol{t}:B$ denotes $\lambda^n x_n.t_n:\lambda^{n-1}x_{n-1}.t_{n-1}:\ldots:\lambda x_1.t_1:B$,

$(\boldsymbol{t} \circ^n \boldsymbol{s}):B$ denotes $(t_n \circ^n s_n):(t_{n-1} \circ^{n-1} s_{n-1}):\ldots:(t_1 \circ s_1):B$,

$\Uparrow^n\boldsymbol{t}:B$ denotes $\Uparrow^n t_n:\Uparrow^{n-1}t_{n-1}:\ldots:\Uparrow t_1:B$,

likewise for all other functional symbols of $\boldsymbol{\lambda^\infty}$.

Derivations are generated by the following clauses. Here A, B, C are formulas, Γ a finite set of types, $\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u}$ are n-vectors of pseudo-terms, $\boldsymbol{x}$ are n-vectors of variables, $n = 0, 1, 2, \ldots$.

	Natural deduction rule	*Its sequent form*
(Ax)	$\boldsymbol{x}:A$	$\Gamma \vdash \boldsymbol{x}:A$, if $\boldsymbol{x}:A$ is in Γ

(λ) $$\frac{\boldsymbol{t}:B}{\lambda^n\boldsymbol{x}.\boldsymbol{t}:(A\to B)}$$

provided $x_n:x_{n-1}:\ldots:x_1:A$, x_i occurs free neither in t_j for $i \neq j$ nor in $A \to B$. Premises corresponding to $x_n:x_{n-1}:\ldots:x_1:A$ (if any) are discharged. In the full sequent form this rule is

$$\frac{\Gamma, x_n:x_{n-1}:\ldots:x_1:A \vdash t_n:t_{n-1}:\ldots:t_1:B}{\Gamma \vdash \lambda^n x_n.t_n:\lambda^{n-1}x_{n-1}.t_{n-1}:\ldots:\lambda x_1.t_1:(A\to B)}$$

where none of $\boldsymbol{x}$ occurs free in the conclusion sequent.

All the rules below do not bind/unbind variables.

(App) $$\frac{\boldsymbol{t}:(A\to B) \quad \boldsymbol{s}:A}{(\boldsymbol{t} \circ^n \boldsymbol{s}):B} \qquad \frac{\Gamma \vdash \boldsymbol{t}:(A\to B) \quad \Gamma \vdash \boldsymbol{s}:A}{\Gamma \vdash (\boldsymbol{t} \circ^n \boldsymbol{s}):B}$$

(p) $$\frac{\boldsymbol{t}:A \quad \boldsymbol{s}:B}{\mathbf{p}^n(\boldsymbol{t},\boldsymbol{s}):(A\wedge B)} \qquad \frac{\Gamma \vdash \boldsymbol{t}:A \quad \Gamma \vdash \boldsymbol{s}:B}{\Gamma \vdash \mathbf{p}^n(\boldsymbol{t},\boldsymbol{s}):(A \wedge B)}$$

$$(\boldsymbol{\pi})\quad \frac{\boldsymbol{t}:(A_0 \wedge A_1)}{\boldsymbol{\pi}_i^n \boldsymbol{t}:A_i}\ (i=0,1) \qquad \frac{\Gamma \vdash \boldsymbol{t}:(A_0 \wedge A_1)}{\Gamma \vdash \boldsymbol{\pi}_i^n \boldsymbol{t}:A_i}\ (i=0,1)$$

$$(\Uparrow)\quad \frac{\boldsymbol{t}:u:A}{\Uparrow^n \boldsymbol{t}:!u:u:A} \qquad \frac{\Gamma \vdash \boldsymbol{t}:u:A}{\Gamma \vdash \Uparrow^n \boldsymbol{t}:!u:u:A}$$

$$(\Downarrow)\quad \frac{\boldsymbol{t}:u:A}{\Downarrow^n \boldsymbol{t}:A} \qquad \frac{\Gamma \vdash \boldsymbol{t}:u:A}{\Gamma \vdash \Downarrow^n \boldsymbol{t}:A}$$

Remark 1. The intuitionistic logic for implication/conjunction and λ-calculus are the special cases for rules with $n = 0$ and $n = 1$ only, respectively, if we furthermore restrict all of the displayed formulas to types which do not contain quasi-atoms.

Example 1. Here are some examples of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$-derivations in the sequent format (cf. 3.2). We skip the trivial axiom parts for brevity.

$$1)\quad \frac{y:x:A \vdash \Downarrow y:A}{\vdash \lambda y.\Downarrow y:(x:A \to A)} \qquad 2)\quad \frac{y:x:A \vdash \Uparrow y:!x:x:A}{\vdash \lambda y.\Uparrow y:(x:A \to !x:x:A)}$$

However, one can derive neither $\vdash A \to x:A$, nor $\vdash \lambda x.!x:(A \to x:A)$, since x occurs free there.

$$3)\quad \frac{\dfrac{u:x:A, v:y:B \vdash \mathbf{p}^2(u,v):\mathbf{p}(x,y):(A \wedge B)}{u:x:A \vdash \lambda^2 v.\mathbf{p}^2(u,v):\lambda y.\mathbf{p}(x,y):(B \to (A \wedge B))}}{\vdash \lambda^2 uv.\mathbf{p}^2(u,v):\lambda xy.\mathbf{p}(x,y):(A \to (B \to (A \wedge B)))}$$

$$4)\quad \frac{\dfrac{\dfrac{u:x:A, v:y:B \vdash \mathbf{p}^2(u,v):\mathbf{p}(x,y):(A \wedge B)}{u:x:A, v:y:B \vdash \Uparrow \mathbf{p}^2(u,v):!\mathbf{p}(x,y):\mathbf{p}(x,y):(A \wedge B)}}{\vdash \lambda uv.\Uparrow \mathbf{p}^2(u,v):(x:A \to (y:B \to !\mathbf{p}(x,y):\mathbf{p}(x,y):(A \wedge B)))}}$$

Note that unlike in the previous example we cannot introduce λ^2 in place of λ at the last stage here since the resulting sequent would be

$$\vdash \lambda^2 uv.\Uparrow \mathbf{p}^2(u,v):\lambda xy.!\mathbf{p}(x,y):(A \to (B \to \mathbf{p}(x,y):(A \wedge B)))$$

containing abstraction over variables x, y which are left free in the conclusion, which is illegal.

Here is an informal explanation of why such a derivation should not be permitted. Substituting different terms for x and y in the last sequent produces different types from $A \to (B \to \mathbf{p}(x,y):(A \wedge B))$, whereas neither of the terms $\lambda xy.!\mathbf{p}(x,y)$ and $\lambda^2 uv.\Uparrow \mathbf{p}^2(u,v)$ changes after such substitutions. This is bad syntactically, since the same terms will be assigned different types. Semantically this is bad either, since this would violate the one proof - one theorem convention.

Proposition 1. *(Closure under substitution) If $t(x)$ is a well-defined term, x a variable of type A, s a term of type A free for x in $t(x)$, then $t(s)$ is a well-defined term of the same type as $t(x)$.*

Proposition 2. *(Uniqueness of Types) If both $t : F$ and $t : F'$ are well-typed terms, then $F \equiv F'$.*

Theorem 1. *(Internalization Property for $\boldsymbol{\lambda^\infty}$) Let $\boldsymbol{\lambda^\infty}$ derive*

$$A_1, A_2, \ldots, A_m \vdash B.$$

Then one can build a well-defined term $t(x_1, x_2, \ldots, x_m)$ with fresh variables $\boldsymbol{x}$ such that $\boldsymbol{\lambda^\infty}$ also derives

$$x_1 : A_1, x_2 : A_2, \ldots, x_m : A_m \vdash t(x_1, x_2, \ldots, x_m) : B.$$

Proof. We increment n at every node of the derivation $A_1, A_2, \ldots, A_m \vdash B$. The base case is obvious. We will check the most principal step clause $(\boldsymbol{\lambda})$ leaving the rest as an exercise. Let the last step of a derivation be

$$\frac{\Gamma, y_n : y_{n-1} : \ldots : y_1 : A \vdash t_n : t_{n-1} : \ldots : t_1 : B}{\Gamma \vdash \lambda^n y_n.t_n : \lambda^{n-1} y_{n-1}.t_{n-1} : \ldots : \lambda y_1.t_1 : (A \to B)}\ .$$

By the induction hypothesis, for some term $s(\boldsymbol{x}, x_{m+1})$ of fresh variables $\boldsymbol{x}, x_{m+1}$

$$\boldsymbol{x} : \Gamma, x_{m+1} : y_n : y_{n-1} : \ldots : y_1 : A \vdash s(\boldsymbol{x}, x_{m+1}) : t_n : t_{n-1} : \ldots : t_1 : B.$$

Apply the rule $(\boldsymbol{\lambda})$ for $n+1$ to obtain

$$\boldsymbol{x} : \Gamma \vdash \lambda^{n+1} x_{m+1}.s : \lambda^n y_n.t_n : \lambda^{n-1} y_{n-1}.t_{n-1} : \ldots : \lambda y_1.t_1 : (A \to B),$$

and put $t(x_1, x_2, \ldots, x_m) = \lambda^{n+1} x_{m+1}.s(\boldsymbol{x}, x_{m+1})$.

2.2 Reductions in the $\boldsymbol{\lambda^\infty}$Calculus

Definition 1. *For $n = 1, 2, \ldots$, the redexes for $\boldsymbol{\lambda^\infty}$ and their contracta (written as $t \triangleright t'$, for t a redex, t' a contractum of t) are:*

- $(\lambda^n x_n.t_n) \circ^n s_n : \ldots : (\lambda x_1.t_1) \circ s_1 : F \ \triangleright\ t_n[s_n/x_n] : \ldots : t_1[s_1/x_1] : F$ *(β_n-contraction)*
- $\boldsymbol{\pi}_i^n \mathbf{p}^n(t_0^n, t_1^n) : \boldsymbol{\pi}_i^{n-1} \mathbf{p}^{n-1}(t_0^{n-1}, t_1^{n-1}) : \ldots : \boldsymbol{\pi}_i \mathbf{p}(t_0^1, t_1^1) : F \ \triangleright\ t_i^n : t_i^{n-1} : \ldots : t_i^1 : F,\ i = 0, 1$
- $\Downarrow^n \Uparrow^n t_n : \Downarrow^{n-1} \Uparrow^{n-1} t_{n-1} : \ldots : \Downarrow \Uparrow t_1 : F \triangleright\ t_n : t_{n-1} : \ldots : t_1 : F$

Before defining the reduction relation on reflective λ-terms (Definition 5) we will come with some motivations (Remarks 2 and 3).

Remark 2. The system should allow types of normalized terms to contain terms which are not normal. The object, then, is to normalize a term of any type, regardless of the redundancy of the type, and focus on eliminating redundancy in the term. It has been noted, for instance, that the statement of the normalization property for a term system, contains "redundancy", because it says that something which is not normal, is equivalent to a normal thing.

Remark 3. In the simply-typed and untyped λ-calculi, reduction of terms is defined as contraction of the redexes which occur as subterm occurrences of the term. For $\boldsymbol{\lambda^{\infty}}$, the situation is a little trickier. We want to be sure that, when reducing a term t_n of type $t_{n-1} : \ldots : t_1 : A$, we end up with a well-typed term t'_n having type $t'_{n-1} : \ldots : t'_1 : A$. I.e., we don't want the formula A to change under reduction, while the intermediary terms t_j, may be allowed to change in a predictable way.

An example, illustrating the difficulties arising when we allow any subterm occurrence of a redex to contract, is the $\boldsymbol{\lambda^{\infty}}$-term

$$\dfrac{\dfrac{x_3 : x_2 : x_1 : A \vdash \Uparrow^2 x_3 : \Uparrow x_2 : !x_1 : x_1 : A}{x_3 : x_2 : x_1 : A \vdash \Downarrow^2 \Uparrow^2 x_3 : \Downarrow\Uparrow x_2 : x_1 : A}}{\vdash \lambda x_3.\Downarrow^2\Uparrow^2 x_3 : (x_2 : x_1 : A \to \Downarrow\Uparrow x_2 : x_1 : A)}$$

Let us use a blend of type assertion notations t^F and $t : F$ to show the type of the subterm $\Downarrow^2\Uparrow^2 x_3$ of the resulting term more explicitly.

$$[\lambda x_3.(\Downarrow^2\Uparrow^2 x_3)^{\Downarrow\Uparrow x_2:x_1:A}]^{(x_2:x_1:A \to \Downarrow\Uparrow x_2:x_1:A)}$$

In principle, the subterm $(\Downarrow^2\Uparrow^2 x_3)^{\Downarrow\Uparrow x_2:x_1:A}$ is a redex. However, from the proof theoretical point of view it would be a mistake to allow the whole term to reduce to

$$[\lambda x_3.x_3]^{(x_2:x_1:A \to x_2:x_1:A)}.$$

Indeed, the formula (type) τ

$$x_2 : x_1 : A \to \Downarrow\Uparrow x_2 : x_1 : A$$

is a nontrivial formalized special case of so-called *subject extension*. We do not want to change this formula (type) in a normalization process to

$$x_2 : x_1 : A \to x_2 : x_1 : A$$

which is a trivial proposition. Speaking proof theoretically, while normalizing we are looking for a normal proof of a given proposition (here it is formula τ) rather than changing a proposition itself in a process of finding a "better" proof of it. Abandoning τ in the normalization process would not allow us to find a natural normal inhabitant (normal proof) of τ. The principal difficulty here is that the main term is "lower level" than the subterm occurrence being contracted. Accordingly, we develop a notion of subterm occurrences sitting properly in a term.

Definition 2. *For t a well-defined $\boldsymbol{\lambda^{\infty}}$-term, t is of level n (written, $\ell ev(t) = n$) if the bottom rule in the derivation tree has superscript n.*

Definition 3. *For t_0 a subterm occurrence of a $\boldsymbol{\lambda^{\infty}}$-term t, with $\ell ev(t_0) = n$, we say that t_0 is properly embedded in t ($t_0 \sqsubseteq_p t$) if there are no subterm occurrences in t containing t_0 of the forms: $\lambda^m x.s$, $s \circ^m r$, $\mathbf{p}^m(s, r)$, $\boldsymbol{\pi}_i^m s$, $\Downarrow^m s$, with $m < n$*

Speaking vaguely, we require that redexes be properly embedded in all subterms except, may be, the "proof checkers" $\Uparrow^m s$. We will show below that this requirement suffices for type preservation, and strong normalization of $\boldsymbol{\lambda}^{\infty}$-terms.

We may now define reduction of $\boldsymbol{\lambda}^{\infty}$-terms.

Definition 4. **(a)** *For any well-defined $\boldsymbol{\lambda}^{\infty}$-terms t and t', t one-step reduces to t' ($t \succ_1 t'$) if t contains a properly embedded subterm occurrence s of the form of one of the redexes above, $s \triangleright s'$, and $t' \equiv t[s'/s]$.*

(b) *The reduction relation $\succeq$ is the reflexive and transitive closure of $\succ_1$.*

Equipped with the above-defined notion of reduction, $\boldsymbol{\lambda}^{\infty}$ enjoys (suitably modified versions of) the most important properties of the simply typed lambda calculus. The elementary property of type preservation under reduction holds, in a modified form, namely, for $t : F$ a well-typed term, if $t \succ_1 t'$ then $t' : F'$ is well-typed, for a naturally determined type F'. More precisely,

Proposition 3. *(Type Preservation) Let t_n be a well-typed term of level n, having type $t_{n-1} : \ldots : t_1 : A$. If $t_n \succ_1 t'_n$, then t'_n is a well-typed level n term. Furthermore, t'_n has type $t'_{n-1} : \ldots : t'_1 : A$, with $t_i \succ_1 t'_i$ or $t_i \equiv t'_i$, for $1 \leq i \leq n-1$.*

Proof. $t_n \succ_1 t'_n$, by definition, only if there exists $s_n \sqsubseteq_p t_n, s_n \triangleright s'_n$, and $t'_n \equiv t_n[s'_n/s_n]$. We proceed by induction on $|t_n|$, and on the number of subterm occurrences between t_n and s_n, and construct a derivation in $\boldsymbol{\lambda}^{\infty}$ of the new term t'_n. The construction of this derivation is analogous to the construction of derivations in intuitionistic logic under detour reduction.

We get a form of subject reduction as an immediate consequence:

Corollary 1. *(Subject Reduction) $\Gamma \vdash \boldsymbol{t} : A \Rightarrow \Gamma \vdash \boldsymbol{t'} : A$, for $t_n \succ_1 t'_n$.*

Remark 4. It is easy to see that the derivation of t'_n obtained in the proof above, is uniquely determined by t'_n, and the derivation of t_n. In this way, we associate to each reduction of terms, a corresponding reduction of derivations. Under this operation of simultaneous term and derivation reduction, we see how the internalization property of $\boldsymbol{\lambda}^{\infty}$ is a generalization of the Curry-Howard isomorphism. Namely, the square in Figure 1 commutes, where ι is the map given by the Internalization Property, and $\succ_1$ is the map given by one-step reduction.

Example 2. We may modify the term from the above example, to obtain a term with a properly embedded redex:

$$\dfrac{\dfrac{x_3 : x_2 : x_1 : A \vdash \Uparrow^2 x_3 : \Uparrow x_2 : !x_1 : x_1 : A}{x_3 : x_2 : x_1 : A \vdash \Downarrow^2 \Uparrow^2 x_3 : \Downarrow \Uparrow x_2 : x_1 : A}}{\vdash \lambda^2 x_3 . \Downarrow^2 \Uparrow^2 x_3 : \lambda x_2 . \Downarrow \Uparrow x_2 : (x_1 : A \to x_1 : A)}$$

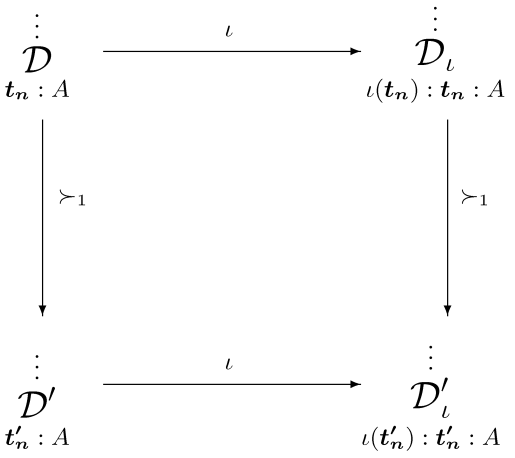


Fig. 1. *Generalization of the Curry-Howard isomorphism*

The (properly embedded) redex $\Downarrow^2\Uparrow^2 x_3$ contracts to x_3, and the term derivation and type are modified accordingly:

$$\frac{x_3 : x_2 : x_1 : A \vdash x_3 : x_2 : x_1 : A}{\vdash \lambda^2 x_3.x_3 : \lambda x_2.x_2 : (x_1 : A \to x_1 : A)}$$

The result really is a normal proof of the statement inside the parentheses.

Proposition 4. *The reduction relation $\succeq$ is not confluent.*

Proof. By a counterexample. Consider the following well-defined terms (missing types can be easily recovered).

$\mathbf{p}^2(x_2, y_2) : \mathbf{p}(x_1, y_1) : (A \wedge B)$
$\lambda x_2.\mathbf{p}^2(x_2, y_2) : (x_1 : A \to \mathbf{p}(x_1, y_1) : (A \wedge B))$
$\lambda x_2.\mathbf{p}^2(x_2, y_2)) \circ z : \mathbf{p}(x_1, y_1) : (A \wedge B)$
$\lambda^2 y_2.[(\lambda x_2.\mathbf{p}^2(x_2, y_2)) \circ z] : \lambda y_1.\mathbf{p}(x_1, y_1) : (B \to (A \wedge B))$

If we call the above term $t_2 : t_1 : (B \to (A \wedge B))$,
then $t_2 \circ^2 \boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2) : t_1 \circ \boldsymbol{\pi}_1\mathbf{p}(u_1, w_1) : (A \wedge B)$ reduces to both:
(1) $t_2[\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)/y_2] : t_1[\boldsymbol{\pi}_1\mathbf{p}(u_1, w_1)/y_1] : (A \wedge B)$ and
(2) $t_2 \circ^2 u_2 : t_1 \circ u_1 : (A \wedge B)$,

which further reduces to
(3) $t_2[u_2/y_2] : t_1[u_1/y_1] : (A \wedge B)$.

And, there's no way to reduce (1) to (3), since $\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)$ isn't properly embedded in $t_2[\boldsymbol{\pi}_1^2\mathbf{p}^2(u_2, w_2)/y_2]$.

3 Strong Normalization of $\boldsymbol{\lambda^{\infty}}$-Terms

Definition 5. **(a)** *A reduction sequence is any sequence of terms* $t_0, \ldots, t_n \ldots$ *such that, for* $0 \leq j < n$, $t_j \succ_1 t_{j+1}$. **(b)** *A term* t *is strongly normalizing* ($t \in \mathbf{SN}$) *if every possible reduction sequence for* t *is finite.* **(c)** *If* t *is strongly normalizing,* $\nu(t) =$ *the least* n *such that every reduction sequence for* t *terminates in fewer than* n *steps.*

We now show that all terms of $\boldsymbol{\lambda^{\infty}}$ are strongly normalizing. The proof follows the method of Tait (cf. [7]). This method defines a class of *reducible* terms by induction on type. We are concerned with types, but will ignore variations in type, *up to a point.*

Definition 6. *More precisely, a term* t *is* n*-cushioned of type* A *if* t *has type* $t_{n-1} : \ldots : t_1 : A$, *where the* t_i *are* $\boldsymbol{\lambda^{\infty}}$*-terms, for* $1 \leq i \leq n-1$. *Note that this is a weaker requirement than* $\ell ev(t) = n$. *Indeed,* $\ell ev(t) = n$ *implies* t *is* k*-cushioned, for* $k \leq n$, *and it could also be that* t *is* m*-cushioned, for some* $m > n$.

For the proof, we'll define sets $\mathtt{RED}^n_T$ of reducible terms for n-cushioned terms of type T, by induction on T. We then prove some properties of these sets of terms, including the property that all terms in the sets are strongly normalizing. Finally, strong normalization of the calculus is established by showing that all $\boldsymbol{\lambda^{\infty}}$-terms are in a set $\mathtt{RED}^n_T$.

Definition 7. *If* T *is a type, we define the depth of* T, $|T|$, *inductively:*

$|p| = 0$, *for* p *an atomic proposition*
$|A \wedge B| = |A \to B| = max(|A|, |B|) + 1$
$|u : A| = |A| + 1$

Definition 8. *The sets* $\mathtt{RED}^n_T$ *of reducible* n*-cushioned terms of type* T *are defined by induction on* $|T|$*:*

1. *For* t *of atomic type* p, $t \in \mathtt{RED}^n_p$ *if* t *is strongly normalizing.*
2. *For* t *of type* $A \to B$, $t \in \mathtt{RED}^n_{A \to B}$ *if, for all* $s \in \mathtt{RED}^n_A$, $t \circ^n s \in \mathtt{RED}^n_B$.
3. *For* t *of type* $A_0 \wedge A_1$, $t \in \mathtt{RED}^n_{A_0 \wedge A_1}$ *if* $\boldsymbol{\pi}^n_i t \in \mathtt{RED}^n_{A_i}$, *for* $i = 0, 1$.
4. *For* t *of type* $u : A$, $t \in \mathtt{RED}^n_{u:A}$ *if* $\Downarrow^n t \in \mathtt{RED}^n_A$ *and* $t \in \mathtt{RED}^{n+1}_A$

Proposition 5. *For* t *a well-typed term, if* t *is strongly normalizing, then* $\Downarrow^n t$ *is too.*

Proof. If t is not of the form $\Uparrow^n t_0$, then every reduction sequence for $\Downarrow^n t$ has less or equal to $\nu(t)$ terms in it. If t is of the form $\Uparrow^n t_0$, then every reduction sequence for $\Downarrow^n t$ has length $\leq \nu(t) + 1$.

Definition 9. *A term* t *is neutral if* t *is not of the form* $\lambda^n x.t_0$, $\mathbf{p}^n(t_0, t_1)$, *or* $\Uparrow^n t_0$.

Proposition 6. *For T any type, all terms $t \in \mathtt{RED}^n_T$ have the following properties:*

(CR 1) *If $t \in \mathtt{RED}^n_T$, then t is strongly normalizing.*
(CR 2) *If $t \in \mathtt{RED}^n_T$ and $t \succeq t'$, then $t' \in \mathtt{RED}^n_{T'}$, for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : T \succeq t' : u'_{n-1} : u'_{n-2} : \ldots : u'_1 : T'$*
(CR 3) *If t is neutral, and $t' \in \mathtt{RED}^n_T$ for all t' such that $t \succ_1 t'$, then $t \in \mathtt{RED}^n_T$*
(CR 4) *If $t \in \mathtt{RED}^n_T$, then $t \in \mathtt{RED}^{n-1}_{u_1:T}$, for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : T$.*

Proof. Proof by induction on $|T|$.

Step 1 T is atomic, i.e. $T \equiv p$.
(CR 1) Trivial.
(CR 2) If t is **SN**, and $t \succeq t'$, then t' is **SN**, so t' is reducible.
(CR 3) Let $M = max(\nu(t') \mid t \succ_1 t')$. There are finitely many t' such that $t \succ_1 t'$, so M is finite. And, $\nu(t) \leq M + 1$.
(CR 4) $t \in \mathtt{RED}^n_p$ yields t is **SN** thus $\Downarrow^{n-1} t$ is **SN**, by above proposition. Therefore, $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_p$ and $t \in \mathtt{RED}^{n-1}_{u_1:p}$
Step 2 $T \equiv (A \to B)$.
(CR 1) Let x be a variable of type A. Then $t \circ^n x \in \mathtt{RED}^n_B$, and by induction, $t \circ^n x$ is **SN**. Since every reduction sequence for t is embedded in a reduction sequence of $t \circ^n x$, t is **SN**. This argument works since $t \succeq t'$ implies that the subterm occurrence being contracted is properly embedded in t, which means this subterm occurrence must have $lev < n + 1$. Under those conditions $t \succeq t'$ implies $t \circ^n x \succeq t' \circ^n x$.
(CR 2) Let $t \succeq t'$ for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : T$, and take $s \in \mathtt{RED}^n_A$ for $s : w_{n-1} : w_{n-2} : \ldots : w_1 : A$. Then $t \circ^n s \in \mathtt{RED}^n_B$, and $t \circ^n s \succeq t' \circ^n s$. By induction, $t' \circ^n s$ is reducible. Since s was arbitrary, t' is reducible.
(CR 3) Let t be neutral, and suppose that t' is reducible, for all t' such that $t \succ_1 t'$. Let $s \in \mathtt{RED}^n_A$. By induction, s is **SN**. We prove, by induction on $\nu(s)$ that if $t \circ^n s \succ_1 (t \circ^n s)'$, $(t \circ^n s)'$ is reducible. There are two possibilities for $(t \circ^n s)'$:
1. $(t \circ^n s)'$ is $t' \circ^n s$, with $t \succ_1 t'$. Then t' is reducible, so $t' \circ^n s$ is.
2. $(t \circ^n s)'$ is $t \circ^n s'$, with $s \succ_1 s'$. Then s' is reducible, so $t \circ^n s'$ is.

By the induction hypothesis, $t \circ^n s$ is reducible, because it's neutral. Thus, t is reducible.
(CR 4) We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A \to B}$. Let $r \in \mathtt{RED}^{n-1}_A$. By induction, r is **SN**; and by **(CR 1)**, t is **SN**. Thus, by the above proposition, $\Downarrow^{n-1} t$ is **SN**, and we can induct on $\nu(r) + \nu(\Downarrow^{n-1} t)$. Base: $\nu(r) + \nu(\Downarrow^{n-1} t) = 0$. Then $\Downarrow^{n-1} t \circ^{n-1} r$ is normal (and it is neutral), so by induction, $\Downarrow^{n-1} t \circ^{n-1} r \in \mathtt{RED}^{n-1}_B$. Induction: in one step, $\Downarrow^{n-1} t \circ^{n-1} r$ reduces to
• $(\Downarrow^{n-1} t)' \circ^{n-1} r$ - reducible by induction
• $\Downarrow^{n-1} t \circ^{n-1} r'$ - reducible by induction.
Hence, by induction hypothesis **(CR 3)**, $\Downarrow^{n-1} t \circ^{n-1} r \in \mathtt{RED}^{n-1}_B$ for all $r \in \mathtt{RED}^{n-1}_A$. Thus, $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A \to B}$, and $t \in \mathtt{RED}^{n-1}_{u_1:(A \to B)}$.
Step 3 $T \equiv (A \wedge B)$.

(CR 1)–**(CR 3)** Similar to the previous step. Note that $t \succeq t'$ yields $\boldsymbol{\pi}_i^n t \succeq \boldsymbol{\pi}_i^n t'$, for $t : u_{n-1} : u_{n-2} : \ldots : u_1 : (A_0 \wedge A_1)$.

(CR 4) By **(CR 1)**, t is **SN**. We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{A_0 \wedge A_1}$, i.e. that $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t) \in \mathtt{RED}^{n-1}_{A_i}$, for $i = 0, 1$. By induction on $\nu(\Downarrow^{n-1} t)$. Base $\nu(\Downarrow^{n-1} t) = 0$. Then $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t)$ is both neutral and normal. By I.H. **(CR 3)**, $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t) \in \mathtt{RED}^{n-1}_{A_i}$. Induction: in one step $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t)$ reduces to $\boldsymbol{\pi}_i^{n-1}(\Downarrow^{n-1} t')$ which is reducible, by I.H..

Step 4 $T \equiv (u : A)$.

(CR 1) $t \in \mathtt{RED}^n_{u:A}$ yields $t \in \mathtt{RED}^{n+1}_A$, thus. by I.H., t is **SN**.

(CR 2) Let $t \succeq t'$. By I.H. **(CR 2)**, $t' \in \mathtt{RED}^{n+1}_{A'}$. By I.H. **(CR 4)**, $t' \in \mathtt{RED}^n_{u':A'}$.

(CR 3) Suppose $t' \in \mathtt{RED}^n_{u':A'}$ for all t' such that $t \succ_1 t'$. Then $t' \in \mathtt{RED}^{n+1}_{A'}$ for all t' such that $t \succ_1 t'$. By I.H. **(CR 3)**, $t \in \mathtt{RED}^{n+1}_A$. By I.H. **(CR 4)**, $t \in \mathtt{RED}^n_{u:A}$.

(CR 4) Let $t \in \mathtt{RED}^n_{u:A}$. By **(CR 1)**, t is **SN**. Thus, by Proposition 5, $\Downarrow^{n-1} t$ is **SN**. We need to show that $\Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_{u:A}$, i.e. that $\Downarrow^{n-1} t \in \mathtt{RED}^n_A$ and $\Downarrow^{n-1} \Downarrow^{n-1} t \in \mathtt{RED}^{n-1}_A$. Both are easily shown by induction on $\nu(\Downarrow^{n-1} t)$ and on $\nu(\Downarrow^{n-1} \Downarrow^{n-1} t)$, respectively, by using **(CR 3)** inductively.

Proposition 7. *1. If, for all $u \in \mathtt{RED}^n_A$, $t[u/x] \in \mathtt{RED}^n_B$, then $\lambda^n x.t \in \mathtt{RED}^n_{A \to B}$.*
2. If t_0, t_1 are reducible, then $\boldsymbol{p}^n(t_0, t_1)$ is reducible.
3. If $t \in \mathtt{RED}^n_{u:A}$, then $\Uparrow^n t \in \mathtt{RED}^n_{!u:u:A}$.
4. If $t \in \mathtt{RED}^n_A$ and $m < n$, then $\Uparrow^m t \in \mathtt{RED}^{n+1}_A$.

Proof. For part (1): Let $u \in \mathtt{RED}^n_A$. We'll show that $(\lambda^n x.t) \circ^n u$ is reducible, by induction on $\nu(u) + \nu(t)$. $(\lambda^n x.t) \circ^n u \succ_1$:

- $t[u/x]$, which is reducible by hypothesis.
- $(\lambda^n x.t') \circ^n u$ with $t \succ_1 t'$. By induction, this is reducible.
- $(\lambda^n x.t) \circ^n u'$ with $u \succ_1 u'$. By induction, this is reducible.

Parts (2) and (3) are established similarly. Part 4 is by induction on $|A|$. The case A is atomic is handled in the obvious way, implications and conjunctions are handled by induction on $\nu(t) + \nu(r)$ and $\nu(t)$ respectively. Finally, the case where A is $u : B$ follows from the induction hypothesis.

Proposition 8. *Let t be a term. Suppose the free variables of t are among $x_1, \ldots, x_k$, of levels $n_1, \ldots, n_k$ and having types $U_1, \ldots, U_k$. Let $u_1, \ldots, u_k$ be terms with $u_1 \in \mathtt{RED}^{n_1}_{U_1}, \ldots, u_k \in \mathtt{RED}^{n_k}_{U_k}$. Then the term $t[u_1/x_1, \ldots, u_k/x_k]$ (written $t[\boldsymbol{u}/\boldsymbol{x}]$), is reducible.*

Proof. **Step 1** t entered by **(Ax)**. Trivial.
Step 2 Let t be $\lambda^n y_n.t_n^0 : \lambda^{n-1} y_{n-1}.t_{n-1}^0 : \ldots : \lambda y_1.t_1^0 : (A \to B)$. By I.H., $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible (i.e., is in some reducibility class) for all $v_n \in \mathtt{RED}^m_T$, where m is the level of y_n, and T is y_n's type (so y_n has type $r_{m-1} : \ldots : r_1 : T$, and $m \geq n$). By **(CR 4)**, and the definition of the classes $\mathtt{RED}^k_{w:F}$, this implies that $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible for all $v_n \in \mathtt{RED}^n_A$. Now, using **(CR 4)** and the definition again, $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}]$ is reducible implies that $t_n^0[v_n/y_n, \boldsymbol{u}/\boldsymbol{x}] \in \mathtt{RED}^n_B$. By Proposition 7, $\lambda^n y_n.t_n^0[\boldsymbol{u}/\boldsymbol{x}] (\equiv t[\boldsymbol{u}/\boldsymbol{x}])$ is reducible.

Step 3 The other six cases are handled similarly, using either the definition of the reducibility classes or Proposition 7.

As a corollary, we have:

Theorem 2. *All well-defined terms of* $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ *are strongly normalizable.*

4 Conclusion

1. Proof polynomials from the Logic of Proofs ([3]) represent the reflective idea in the combinatory terms format, with many postulated constant terms and three basic operations only. Such an approach turned out to be fruitful in proof theory, modal and epistemic logics, logics of knowledge, etc. Proof polynomials are capable of extracting explicit witnesses from any modal derivation and thus may be regarded as a logical basis for the quantitative theory of knowledge. On the other hand, such areas as typed programming languages, theorem provers, verification systems, etc., use the language of λ-terms rather than combinatory terms. Technically speaking, in the λ format there are no postulated constants, but there are many (in our case infinitely many) operations on terms. Those operations create certain redundancies in terms, and the theory of eliminating those redundancies (i.e. the normalization process) plays an important role. Reflective λ-calculus $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ is a universal superstructure over formal languages based on types and λ-terms. It offers a much richer collection of types, in particular the ones capable of encoding computations in types. We wish to think that a wide range of systems based on types and terms could make use of such a capability.

2. The very idea of reflective terms is native to both provability and computability. The extension of the Curry-Howard isomorphism offered by the reflective λ-calculus $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ captures reflexivity in a uniform abstract way. This opens a possibility of finding more mutual connections between proofs and programs. In this paper we have built strongly normalizing reflective λ-terms that correspond to some proper subclass of proof polynomials. It is pivotal to study exact relations between $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$, Intuitionistic Modal Logic and the Intuitionistic Logic of Proofs. Such an investigation could bring important new rules to $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$, and eventually to programming languages.

3. Reflective λ-calculus is capable of internalizing its own reductions as well as its own derivations. If a term $t' : F'$ is obtained from $t : F$ by a reduction, then the corresponding derivation (well-formed term) $s : t : F$ can be transformed to a term $s' : t' : F'$. This takes care of the cases when, for example, an abstraction term is obtained not by the abstraction rule, but by a projection reduction. Such a step can be internalized in $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ as well.

4. A more concise equivalent axiomatization of $\boldsymbol{\lambda}^{\boldsymbol{\infty}}$ could probably be obtained by formalizing Internalization as an atomic rule of inference rather than a derived rule. This route is worth pursuing.

5. There are many natural systems of reductions for reflective λ-terms. For this paper we have picked one with well-embedded redexes on the basis of the

proof theoretical semantics. We conjecture that the system with unlimited reductions enjoys both strong normalization and confluence.

Acknowledgment. Sergei Artemov suggested the system $\boldsymbol{\lambda}^{\infty}$ and wrote sections 1, 2.1, 4. Jesse Alt made a major contribution to finding the proof of the strong normalization theorem $\boldsymbol{\lambda}^{\infty}$ and wrote most of sections 2.2, 3.

Special thanks are due to Robert Constable and Anil Nerode for supporting this research. The authors are also indebted to Volodya Krupski, Roma Kuznets, Alesha Nogin, Lena Nogina, and the anonymous referee for a reading different versions of this paper which led to valuable improvements.

References

1. S. Artemov, "Operational Modal Logic," *Tech. Rep. MSI 95-29*, Cornell University, December 1995
   `http://www.cs.cornell.edu/Info/People/artemov/MSI95-29.ps`
2. S. Artemov, "Uniform provability realization of intuitionistic logic, modality and lambda-terms", Electronic Notes on Theoretical Computer Science, vol. 23, No. 1. 1999 `http://www.elsevier.nl/entcs`
3. S. Artemov, "Explicit provability and constructive semantics", The *Bulletin for Symbolic Logic*, v.7, No. 1, pp. 1-36, 2001
   `http://www.cs.cornell.edu/Info/People/artemov/BSL.ps`.
4. H. Barendregt, "Lambda calculi with types". In *Handbook of Logic in Computer Science, vol.2*, Abramsky, Gabbay, and Maibaum, eds., Oxford University Press, pp. 118-309, 1992
5. R. Constable, "Types in logic, mathematics and programming". In *Handbook in Proof Theory*, S. Buss, ed., Elsevier, pp. 683-786, 1998
6. D. de Jongh and G. Japaridze, "Logic of Provability", in S. Buss, ed., *Handbook of Proof Theory*, Elsevier, pp. 475-546, 1998
7. J.-Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types*, Cambridge University Press, 1989.
8. V.N. Krupski, "Operational Logic of Proofs with Functionality Condition on Proof Predicate", Lecture Notes in Computer Science, v. 1234, *Logical Foundations of Computer Science' 97, Yaroslavl'*, pp. 167-177, 1997
9. V.N. Krupski, "The single-conclusion proof logic and inference rules specification", Annals of Pure and Applied Logic, v.110, No. 1-3, 2001 (to appear).
10. A. Nerode, "Applied Logic". In *The merging of Disciplines: New Directions in Pure, Applied, and Computational Mathematics*, R.E. Ewing, K.I. Gross, C.F. Martin, eds., Springer-Verlag, pp. 127-164, 1986
11. A. Troelstra, "Realizability". In *Handbook in Proof Theory*, S. Buss, ed., Elsevier, pp. 407-474, 1998
12. A.S. Troelstra and H. Schwichtenberg, *Basic Proof Theory*, Cambridge University Press, 1996.

A Note on the Proof-Theoretic Strength of a Single Application of the Schema of Identity

Matthias Baaz[1] and Christian G. Fermüller[2]

[1] Institut für Algebra und Computermathematik E118.2,
Technische Universität Wien, A–1040 Vienna, Austria,
[2] Institut für Computersprachen E185,
Technische Universität Wien, A–1040 Vienna, Austria
{baaz,chrisf}@logic.at

Abstract. Every instance of the schema of identity is derivable from identity axioms. However, from a proof theoretic perspective the schema of identity is quite powerful. Even one instance of it can be used to prove formulas uniformly where no fixed number of instances of identity axioms is sufficient for this purpose.

1 Introduction

In this paper we investigate the impact of replacing instances of the schema of identity by instances of identity axioms in valid Herbrand disjunctions. More precisely, we consider valid sequents of form

$$\Xi, s = t \supset (B(s) \supset B(t)) \vdash \exists \overline{x} A(r^p, \overline{x})$$

where Ξ consists of identity axioms, A is quantifier free, and r^p is some "parameter term", for which the index p can be thought of as its size. Obviously $s = t \supset (B(s) \supset B(t))$ can be derived from identity axioms and thus be replaced (in the sequent above) by appropriate instances of identity axioms. We compare the number of formulas in the corresponding Herbrand disjunctions

$$\Xi', s = t \supset (B(s) \supset B(t)) \vdash A(r^p, \overline{t_1}), \ldots, A(r^p, \overline{t_n})$$

and

$$\Xi', \Pi \vdash A(r^p, \overline{t_1}), \ldots, A(r^p, \overline{t_n})$$

respectively, where Ξ', Π consist of instances of identity axioms. On the one hand we provide examples where $|\Pi|$ (i.e., the number of formulas in Π) cannot be uniformly bounded (i.e., depends on the size of the parameter term r^p). On the other hand we show that if the parameter term is composed of a "fanning out" function symbol then such bounds exist. (We call a unary function symbol f fanning out if $f^m(x) \neq f^n(x)$ is provable whenever $m \neq n$, for all $m, n \in \mathbb{N}$.)

The length of a Herbrand disjunction is uniformly bounded by the length of a proof and the logical complexity of the end sequent. (This follows, e.g.,

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 38–48, 2001.

from Gentzen's Midsequent Theorem; see, e.g., [Takeuti, G., 1980].) Therefore our results can be interpreted as statements on the short provability of formulas involving complex terms using the schema of identity, i.e. its proof theoretic strength in the literal sense. In general, the short (uniform) provability is lost when the schema of identity is replaced by identity axioms.

2 Basic Notions

Syntax

Terms are built up, as usual, from *constant* and *function symbols*. In particular, we will use 0 and c as constant symbols, $\mathbf{s}$ and $\mathbf{f}$ as unary function symbols, and $+$ and $\diamond$ as binary function symbols for which we use infix notation. *Atoms* (atomic formulas) consist of an n-ary *predicate symbol* ($n \geq 1$) applied to n terms (arguments). Throughout the paper we refer to the binary predicate symbol $=$, for which we use infix notation, as "identity". *Formulas* are either atoms or built up from atoms using the usual connectives ($\neg, \wedge, \vee, \supset$) and quantifiers ($\forall$, $\exists$). $\bigwedge_{i=1}^{n} A_i$ abbreviates $A_1 \wedge \ldots \wedge A_n$. $\neg s = t$ is written as $s \neq t$. *Tuples of variables or terms* are indicated by over-lines: e.g., $\exists \overline{x} A(\overline{x})$ abbreviates $\exists x_1 \ldots \exists x_n A(x_1, \ldots, x_n)$ for some $n \geq 1$. By $\|A\|$ we denote the number of atoms in a formula A.

By an *expression* we mean either a term or a formula. It is convenient to view an expression as a rooted and ordered tree, where the inner nodes are function symbols (the root possibly is a predicate symbol) and the leaf nodes are constants or variables. A *position* p in an expression A is a sequence of edges in a path leading from the root to a node n_p of A. Two positions are overlapping if one is a proper subsequence of the other. In writing $A(t_1, \ldots, t_n)$ we implicitly refer, for each $i \in \{1, \ldots, n\}$, to some — possibly empty — set π_i of positions in the expression A in which t_i occurs, where no two positions in $\bigcup_{1 \leq i \leq n} \pi_i$ are overlapping. The indicated positions will be made explicit only when necessary. In reference to $A(t_1, \ldots, t_n)$ we denote the result of replacing t_i at the corresponding positions π_i with the term s_i by $A(s_1, \ldots, s_n)$. (Note that not necessarily *all* occurrences of t_i in A are indicated.) $g[s]$ indicates that the term s occurs as a *proper* subterm — at certain positions $p_1, \ldots p_n$, $n \geq 1$ — in g. In reference to $g[s]$ we denote by $g[t]$ the result of replacing s at these positions with the term t.

"$\equiv$" denotes *syntactical identity* between expressions. For any term t and unary function symbol f we define: $f^0(t) \equiv t$ and $f^{n+1}(t) \equiv f^n(f(t))$. Similarly if $\odot$ is a binary function symbol (for which we use infix notation) we define: $[t]^0_{\odot} \equiv t$ and $[t]^{n+1}_{\odot} \equiv (t \odot [t]^n_{\odot})$.

A variable is called *fresh* with respect to some context (e.g., a sequent) if it does not occur elsewhere in that context. By a *fresh copy* F' of a formula F we mean a variable renaming copy of F, such that the variables occurring in F' do not occur in the context.

(Variable) *substitutions* are functions from variables to terms that are homomorphically extended to expressions and sequences of formulas as usual. The

result of applying the substitution σ to an expression E is written as $\sigma(E)$. We assume familiarity with the concept of a *most general (simultaneous) unifier* of a finite set $\{(E_1, F_1), \ldots, (E_n, F_n)\}$ of pairs of expressions (see, e.g., [Leitsch, A., 1997]).

Axiomatizing Identity

The context of our investigation is pure, classical first-order logic. However we are mainly interested in the effects of different treatments of the identity predicate. We will frequently refer to the following list of axioms, called *identity axioms*:

- Reflexivity: $x = x$
- Symmetry: $x = y \supset y = x$
- Transitivity: $(x = y \wedge y = z) \supset x = z$
- Congruence (functions): for every n-ary function symbol f and every $1 \leq i \leq n$: $x_i = y_i \supset f(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots x_n)$
- Congruence (predicates): for every n-ary predicate symbol P and every $1 \leq i \leq n$: $x_i = y_i \supset (P(x_1, \ldots, x_n) \supset P(x_1, \ldots, x_{i-1}, y_i, x_{i+1}, \ldots x_n))$

The Classical Sequent Calculus LK

As underlying deduction system we will use Gentzen's (in every sense of the word) classical **LK**. This, however, is only a matter of taste: our results can be easily translated to other formats of deduction. We will not have to present any formal **LK**-derivations in detail here and thus simply refer to, e.g., [Takeuti, G., 1980] for formal definitions of sequents, rules, derivations etc. Sequents are written as $\Gamma \vdash \Delta$, where Γ and Δ are sequences of formulas. By $|\Gamma|$ we denote the number of occurrences of formulas in Γ.

All *tautological*, i.e., propositionally valid sequents are (cut-free) derivable in a number of steps depending only on the logical complexity of the formulas in the end sequent.

Gentzen's Midsequent Theorem provides a version of Herbrand's Theorem for sequents (see [Takeuti, G., 1980]).

Note that the sequents in an **LK**-derivation can be skolemized without increase of proof length and that Herbrand disjunctions can be embedded in "re-skolemized" derivations whose length does not depend on the term structure (cf. [Baaz, M. and Leitsch, A., 1994], [Baaz, M. and Leitsch, A., 1999].) Consequently the results below can be re-interpreted for quantified formulas.

3 One Application of the Schema of Identity

The schema of identity is given by

$$x = y \supset (A(x) \supset A(y)) \qquad \text{(ID)}$$

where A is a formula and x, y are variables. The schema

$$x = y \supset t(x) = t(y) \tag{ID$^=$}$$

can be seen as a special case of ID (where $A(z) \equiv t(x) = t(z)$) combined with an appropriate instance of the axiom of reflexivity. More generally, we show that a single instance of ID is equivalent to some finite number of instances of ID$^=$ that involve the same pair of terms. (By an instance of a schema we mean a formula of the given form, where the variables have been replaced by terms.)

Proposition 1.
(a) For every conjunction $\bigwedge_{i=1}^{n}(s = t \supset r_i(s) = r_i(t))$ *of instances of* ID$^=$ *there is a tautological sequent*

$$\Gamma, s = t \supset (A(s) \supset A(t)) \vdash \bigwedge_{i=1}^{n}(s = t \supset r_i(s) = r_i(t))$$

where Γ *consists of at most* n *instances of the reflexivity axiom and* $\|A\| \leq 2n$.
(b) For every instance $s = t \supset (A(s) \supset A(t))$ *of* ID *there is a tautological sequent*

$$\Gamma, s = t \supset r_1(s) = r_1(t), \ldots, s = t \supset r_n(s) = r_n(t) \vdash s = t \supset (A(s) \supset A(t))$$

where Γ *consists of instances of congruence axioms for predicate symbols and* $|\Gamma| \leq a\|A\|$, *where* a *is the maximal arity of predicate symbols occurring in* A.

Proof. (a) Let

$$A(t) = \bigwedge_{i=1}^{n}(s = t \supset r_i(s) = r_i(t))$$

and

$$A(s) = \bigwedge_{i=1}^{n}(s = s \supset r_i(s) = r_i(s))$$

Let the instances Γ of the reflexivity axiom be $r_1(s) = r_1(t), \ldots, r_n(s) = r_n(t)$. Then one can easily derive the sequent $\Gamma, s = t \supset (A(s) \supset A(t)) \vdash A(t)$, q.e.d.

(b) Let $r_1(s), \ldots, r_n(s)$ be *all* terms that occur as arguments of predicates (atoms) in $A(s)$. Using appropriate instances of the congruence axioms we obtain a proof of

$$\Gamma \vdash \underbrace{r_1(s) = r_1(t) \supset (\ldots \supset (r_n(s) = r_n(t) \supset (A(s) \supset A(t)) \ldots)}_{R}$$

where $|\Gamma|$ is bounded by the number of argument positions in A, i.e., $|\Gamma| \leq a\|A\|$, where a is the maximal arity of predicates occurring in A. Note that

$$s = t \supset r_1(s) = r_1(t), \ldots, s = t \supset r_n(s) = r_n(t), R \vdash s = t \supset (A(s) \supset A(t))$$

is tautological. Using the cut rule we thus obtain a proof of the sequent, as required. ∎

The following theorem (together with Propositions 2) shows that a single application of the schema of identity can replace an unbounded number of instances of identity axioms. (Remember that, by Proposition 1, the two instances of $\mathrm{ID}^{=}$ correspond to a single instance of ID in presence of some additional instances of congruence axioms.)

Theorem 1. *Let* $A(u,x,y) \equiv ((x+y=y \supset x=0) \wedge 0+0=0) \supset u=0$. *(We refer to* all *positions at which the variables* u,x,y *occur.) For all* $p \in \mathbb{N}$ *there are terms* $r_1^p, r_2^p, t_1^p, t_2^p$ *such that the sequent*

$$0+0=0 \supset r_1^p(0+0)=r_1^p(0), 0+0=0 \supset r_2^p(0+0)=r_2^p(0), T \vdash A([0]_+^p, t_1^p, t_2^p)$$

is tautological, where T *is an instance of the transitivity axiom.*

Proof. Let

$$r_1^p(0+0) \equiv [0]_+^p + ([0]_+^{p-1} + \ldots + ([0]_+^2 + 0)\ldots)$$

where "$(0+0)$" in "$r_1^p(0+0)$" refers to *all* occurrences of subterms of this form. Let

$$r_2^p(0+0) \equiv [0]_+^{p-1} + (\ldots + ([0]_+^2 + \underline{(0+0)})\ldots)$$

where "$(0+0)$" in "$r_2^p(0+0)$" refers only to the rightmost (i.e., the underlined) occurrence of $0+0$. Note that $r_2^p(0+0) \equiv r_1^p(0)$. The needed instance T of the transitivity axiom is

$$(r_1^p(0+0)=r_1^p(0) \wedge r_2^p(0+0)=r_2^p(0)) \supset r_1^p(0+0)=r_2^p(0).$$

Let

$$t_1^p \equiv [0]_+^p$$

and

$$t_2^p \equiv [0]_+^{p-1} + (\ldots + ([0]_+^2 + 0)\ldots).$$

Note that $t_2^p \equiv r_2^p(0)$ and $t_1^p + t_2^p \equiv r_1(0+0)$; t_1^p is identical to the parameter term and the conclusion of the instance of the transitivity axiom is identical to instance of the premise in $x+y=y \supset x=0$. Consequently the sequent is of form

$$N \supset B_1,\ N \supset B_2,\ (B_1 \wedge B_2) \supset B \ \vdash\ ((B \supset C) \wedge N) \supset C$$

and therefore tautological. ∎

Remark 1. A variant of this statement for number theory has been proven in [Yukami, T., 1984]. See also [Baaz, M. and Pudlák, P., 1993].

Remark 2. Note that only few properties of "+" and "0" are used. In particular, one may think of "0" as the neutral element of an arbitrary group (w.r.t. +). From this observation it follows that the implication $x+y=y \supset x=0$ can be replaced by pure identities in principle.

As mentioned above, each instance of the schema of identity is provable from identity axioms. However, the instances of $\mathrm{ID}^{=}$ in the tautological sequent of Theorem 1 cannot be replaced by a number of instances of identity axioms that is *independent* of the "size" p of the parameter term $[0]^p_+$, as the following proposition shows.

Proposition 2. *Let $\Gamma_p \vdash \Delta_p$ be tautological, where Γ_p consists of instances of identity axioms, and Δ_p consists of instances of $A([0]^p_+, x, y)$ (for A as in Theorem 1). $|\Gamma_p| + |\Delta_p|$ cannot be uniformly bounded by a constant.*

Proof. Indirect. Assume that there are constant numbers a and b such that for all p we have $|\Gamma_p| = a$ and $|\Delta_p| = b$. We look at a term minimal generalization of the sequent $\Gamma_p \vdash \Delta_p$. I.e., we replace the formulas in Δ_p by formulas $A(u, x_i, y_i)$, where the x_i, y_i $(1 \leq i \leq b)$ are pairwise distinct variables distinct also from u. Moreover, we replace each formula in Γ_p by a fresh copy of the identity axiom I_j of which it is an instance. The resulting sequent

$$I_1, \ldots, I_a \vdash A(u, x_1, y_1), \ldots, A(u, x_b, y_b)$$

is no longer tautological. However, by simultaneously unifying all those pairs of occurrences of atoms $(s = t, s' = t')$ for which the corresponding atoms in $\Gamma_p \vdash \Delta_p$ are identical, we regain a tautological sequent

$$\sigma(I_1), \ldots \sigma(I_a) \vdash A(\sigma(u), \sigma(x_1), \sigma(y_1)), \ldots, A(\sigma(u), \sigma(x_b), \sigma(y_b))$$

where σ is the most general unifier. This sequent is independent of p, but the term $[0]^p_+$ is an instance of $\sigma(u)$. Therefore $\sigma(u)$ must be of form $0 + (\ldots(0 + z)\ldots)$ where z is a variable. However, this implies that the sequent cannot be tautological. (Standard arithmetic provides a counter example.) ∎

Remark 3. The above statement can be strengthened by allowing also instances of valid universal number theoretic formulas in Γ_p.

4 Monadic Parameter Terms

Looking at the formulation of Theorem 1 one might be seduced to believe that it is essential that the parameter term $[0]^p_+$ is built up using a binary function symbol. However an analogous fact also holds for monadic parameter terms:

Theorem 2. *Let $A(u, x, y) \equiv (x + y = y \supset x = 0) \wedge \mathbf{f}(x) = x) \supset u = 0)$. (We refer to* all *positions at which the variables u, x, y occur.) For all $p \in \mathbb{N}$ there are terms $r^p_1, r^p_2, t^p_1, t^p_2$ such that the sequent*

$$\mathbf{f}(0) = 0 \supset r^p_1(\mathbf{f}(0)) = r^p_1(0), 0 + 0 = 0 \supset r^p_2(0 + 0) = r^p_2(0), T \vdash A(\mathbf{f}^p(0), t^p_1, t^p_2)$$

is tautological, where T is an instance of the transitivity axiom.

Proof. Let

$$r_1^p(z) \equiv \mathbf{f}^{p-1}(z) + (\ldots + (z + 0)\ldots).$$

Therefore

$$r_1^p(\mathbf{f}(0)) \equiv \mathbf{f}^p(0) + (\ldots + (\mathbf{f}(0) + 0)\ldots)$$

and

$$r_1^p(0) \equiv \mathbf{f}^{p-1}(0) + (\ldots + (0 + 0)\ldots)$$

Note that this implies that also the leftmost formula of the sequent is an instance of $\mathrm{ID}^{=}$. Let

$$r_2^p(z) \equiv \mathbf{f}^{p-1}(0) + (\ldots + (\mathbf{f}(0) + z\ldots)).$$

Setting

$$t_1^p \equiv \mathbf{f}^p(0)$$

and

$$t_2^p \equiv \mathbf{f}^{p-1}(0) + (\ldots + (\mathbf{f}(0) + 0)\ldots)$$

it is straightforward to check that the sequent is tautological (analogously to the proof of Theorem 1.) ∎

Remark 4. Note that in the proof above *two* instances of the schema of identity are used. (See also the final remark, at the end of the paper.)

Again, one can show that the two instances of $\mathrm{ID}^{=}$ cannot be replaced by a uniformly bounded number of instances of identity axioms:

Proposition 3. *Let $\Gamma_p \vdash \Delta_p$ be tautological, where Γ_p consists of instances of identity axioms and Δ_p consists of instances of $A(\mathbf{f}^p(0), x, y)$ (for A as in Theorem 2). $|\Gamma_p| + |\Delta_p|$ cannot be uniformly bounded by a constant.*

Proof. Analogous to the proof of Proposition 2.

5 Cases Where the Schema of Identity Is "Weak"

Definition 1. *We call a unary function symbol* $\mathbf{s}$ fanning out *if there is a set of axioms* $\mathsf{NFP}_\mathbf{s}$ *such that for all $m \neq n$ the formula $\mathbf{s}^m(x) \neq \mathbf{s}^n(x)$ is provable from* $\mathsf{NFP}_\mathbf{s}$ *and the identity axioms.*

We show that — in contrast to the results in the last section — "short" proofs using one instance of the schema of identity can be transformed into "short" proofs using only identity axioms if the parameter term is composed of a fanning out function symbol.

Theorem 3. *Let the propositional sequent* (H^{id})

$$\Xi, s = t \supset r_1(s) = r_1(t), \ldots, s = t \supset r_k(s) = r_k(t) \vdash A(\mathbf{s}^p(0), \overline{t_1}), \ldots, A(\mathbf{s}^p(0), \overline{t_n})$$

be provable in **LK**, *where* Ξ *consists of instances of identity axioms, and* k *as well as* $|\Xi|$ *are independent of the size* p *of the parameter term* $\mathbf{s}^p(0)$.

If $\mathbf{s}$ *is fanning out than the instances of* $\mathrm{ID}^{=}$ *are redundant in the following sense: A sequent*

$$\Xi', \Omega \vdash A(\mathbf{s}^p(0), \overline{t'_1}), \ldots, A(\mathbf{s}^p(0), \overline{t'_r}) \qquad (H^{\circ})$$

is provable in **LK**, *where* $r \leq 2n$, Ω *is a (possibly empty) sequence of instances axioms in* $\mathsf{NFP}_{\mathbf{s}}$ *and* Ξ *is a sequence of instances of identity axioms and both* $|\Xi|$ *and* $|\Omega|$ *are independent of the size* p *of the parameter term* $\mathbf{s}^p(0)$.

Proof. **Case 1:** $s \equiv t$. In this case the instances of $\mathrm{ID}^{=}$ in (H^{id}) clearly can be replaced by the instances $r_1(s) = r_1(s), \ldots, r_k(s) = r_k(s)$ of the reflexivity axiom to obtain the required Herbrand disjunction (H°).

Case 2: $s \equiv \mathbf{s}^{\ell_1}(0)$ and $t \equiv \mathbf{s}^{\ell_2}(0)$, $\ell_1 \neq \ell_2$. Since

$$s \neq t \vdash s = t \supset r_i(s) = r_i(t)$$

is a tautology we can transform the derivation of (H^{id}) into one of

$$\Xi, s \neq t \vdash A(\mathbf{s}^p(0), \overline{t_1}), \ldots, A(\mathbf{s}^p(0), \overline{t_n}) \qquad (1)$$

By the form of s and t and the fact that $\mathbf{s}$ is fanning out $s \neq t$ is derivable from $\mathsf{NFP}_{\mathbf{s}}$ and identity axioms. In other words, there exists a tautological sequent

$$\Xi, \Pi \vdash A(\mathbf{s}^p(0), \overline{t_1}), \ldots, A(\mathbf{s}^p(0), \overline{t_n})$$

where Π consists of instances of axioms in $\mathsf{NFP}_{\mathbf{s}}$ and identity axioms. However, in general, the size of s and t will depend on the depth p of the parameter, which implies that $|\Pi|$ is dependent on p as well. Therefore we transform sequent (1) into another tautological sequent

$$\Xi', s' \neq t' \vdash A(\mathbf{s}^p(0), \overline{t'_1}), \ldots, A(\mathbf{s}^p(0), \overline{t'_n})$$

where the sizes of terms s' and t' are independent of p. To this end we replace in (1) all occurrences of $\mathbf{s}^p(0)$ by a fresh variable u, all indicated occurrences of terms in $\overline{t_1}, \ldots, \overline{t_n}$ by pairwise distinct and new variables, each formula in Ξ by a fresh copy of the identity axiom of which it is an instance, as well as $s \neq t$ by $v \neq w$, where v, w are fresh variables as well. Of course, the resulting sequent

$$\Xi^{\circ}, v \neq w \vdash A(u, \overline{x_1}), \ldots, A(u, \overline{x_n}) \qquad (2)$$

is not tautological in general. However, by simultaneously unifying in (2) all those pairs of occurrences of atoms (P°, Q°) for which the corresponding atoms P and Q in (1) are identical, we regain a tautological sequent of form

$$\Xi^*, \mathbf{s}^{k_1}(u_1) \neq \mathbf{s}^{k_2}(u_2) \vdash A(\mathbf{s}^{k_3}(z), \overline{t_1^*}), \ldots, A(\mathbf{s}^{k_3}(z), \overline{t_n^*}) \tag{3}$$

where each of u_1, u_2 and z is a either variable or the constant 0. Moreover, s is an instance of $\mathbf{s}^{k_1}(u_1)$, t an instance of $\mathbf{s}^{k_2}(u_2)$ and the parameter $\mathbf{s}^p(0)$ an instance of $\mathbf{s}^{k_3}(z)$. Observe that — if we use the most general unifier to obtain (3) from (2) — k_1, k_2 and k_3 are independent of p. W.l.o.g. we assume that $k_1 \leq k_2$. We can also assume that z is a variable, since otherwise $\mathbf{s}^{k_3}(z) \equiv \mathbf{s}^p(0)$ and therefore nothing is left to prove.

To reconstruct the parameter at the appropriate places — i.e., to regain the required Herbrand disjunction of $\exists \overline{x}(A(\mathbf{s}^p(0), \overline{x}))$ at the right hand side of the sequent — we define a variable substitution σ that is to be applied to the whole sequent (3).

Case 2.1: $u_1 \not\equiv z$ and $u_2 \not\equiv z$. For $i = 1, 2$, if u_i is a variable then set either $\sigma(u_i) = 0$ or $\sigma(u_i) = \mathbf{s}(0)$ in such a way that $\sigma(\mathbf{s}^{k_1}(u_1)) \not\equiv \sigma(\mathbf{s}^{k_2}(u_2))$.

Case 2.2: $u_1 \equiv z$ and $u_2 \equiv z$. σ remains empty. (Observe that in this case $k_1 \neq k_2$).

Case 2.3: $u_1 \equiv z$ and $u_2 \equiv 0$. We set $\sigma(z) = \mathbf{s}^{k_2 - k_1 + 1}(z)$.

Case 2.4: $u_1 \equiv z$ and u_2 is a (different) variable. We set $\sigma(z) = \mathbf{s}^{k_2 - k_1 + 1}(u_2)$.

Case 2.5: $u_2 \equiv z$ and $u_1 \equiv 0$. We set $\sigma(z) = \mathbf{s}(z)$.

Case 2.6: $u_2 \equiv z$ and u_1 is a (different) variable. We set $\sigma(z) = \mathbf{s}(u_1)$.

Note that, in all cases, the depth of $\sigma(\mathbf{s}^{k_1}(u_1) \neq \mathbf{s}^{k_2}(u_2))$ is independent of p. Moreover $\sigma(\mathbf{s}^{k_1}(u_1)) \not\equiv \sigma(\mathbf{s}^{k_2}(u_2))$ which implies that a sequent

$$\Sigma \vdash \sigma(\mathbf{s}^{k_1}(u_1) \neq \mathbf{s}^{k_2}(u_2)) \tag{4}$$

is derivable where Σ consists of a number of instances of axioms in $\mathsf{NFP_s}$ and identity axioms that is independent of p. Using (4) and the result of applying σ to (3) we obtain

$$\Sigma \vdash A(\mathbf{s}^{p'}(z), \overline{t_1}), \ldots, A(\mathbf{s}^{p'}(z), \overline{t_n})$$

where Σ consists of instances from $\mathsf{NFP_s}$ and identity axioms and $|\Sigma|$ is independent from p. Since $p' \leq p$ it remains to instantiate $\mathbf{s}^{p-p'}(0)$ for z to obtain the required Herbrand disjunction (H°).

Case 3: Either $s \not\equiv \mathbf{s}^\ell(0)$ or $t \not\equiv \mathbf{s}^\ell(0)$ for some $\ell \geq 0$. W.l.o.g., we assume t is not of this form.

Since

$$s \neq t \vdash s = t \supset r_1(s) = r_1(t), \ldots, s = t \supset r_k(s) = r_k(t)$$

is a tautological sequent also

$$s \neq t, \Xi \vdash A(\mathbf{s}^p(0), \overline{t_1}), \ldots, A(\mathbf{s}^p(0), \overline{t_n}) \tag{5}$$

is tautological. It thus remains to show that we can also derive a sequent

$$s = t, \Xi' \vdash A(\mathbf{s}^p(0), \overline{t'_1}), \ldots, A(\mathbf{s}^p(0), \overline{t'_n}) \tag{6}$$

where Ξ' consists of axiom instances and $|\Xi'|$ does not depend on p. We can then join (5) and (6) — by applying the cut rule — to derive the required Herbrand disjunction (H°) (which will have $2n$ disjuncts).
Observe that possibly either $s \equiv t[s]$ or $t \equiv s[t]$. However, w.l.o.g., we may assume that $t \not\equiv s[t]$ (i.e., t does not occur in s, but possibly *vice versa*). We begin by replacing in (H^{id}) all occurrences of t by s. Since new occurrences of t might arise we repeat this step until no occurrences of t are left in the sequent. Let us denote the result by

$$\{\Xi\}_s^{t*}, s = s \supset \{r_1(s)\}_s^{t*} = \{r_1(t)\}_s^{t*}, \ldots, s = s \supset \{r_k(s)\}_s^{t*} = \{r_k(t)\}_s^{t*} \vdash \{A(\mathbf{s}^p(0), \overline{t_1})\}_s^{t*}, \ldots, \{A(\mathbf{s}^p(0), \overline{t_n})\}_s^{t*}$$

Since $\{r_1(s)\}_s^{t*} \equiv \{r_1(t)\}_s^{t*}$ the instances of $\mathrm{ID}^=$ can be replaced by k instances of the reflexivity axiom. We denote this sequence of k axiom instances by Ξ' and obtain

$$\{\Xi\}_s^{t*}, \Xi' \vdash \{A(\mathbf{s}^p(0), \overline{t_1})\}_s^{t*}, \ldots, \{A(\mathbf{s}^p(0), \overline{t_n})\}_s^{t*} \tag{7}$$

Observe that although, by assumption, t is not of form $\mathbf{s}^\ell(0)$ it might properly contain the parameter $\mathbf{s}^p(0)$ as subterm. Thus in (7) the parameter might have been replaced; moreover $A(.,.) \not\equiv \{A(.,.)\}_s^{t*}$ in general. Similarly the iterated replacement of t by s might result in members of $\{\Xi\}_s^{t*}$ that are not longer instances of identity axioms. It thus remains to reconstruct the required Herbrand disjunction relative to axioms by re-substituting t for *some* occurrences of s in (7). For this we have to use $s = t$.
We can derive in **LK** any propositional sequent of form

$$s = t, \Pi, B \vdash B_t^s \tag{8}$$

where Π consists of instances of identity axioms and B_t^s is like B except for replacing a single occurrence of s in B — say at position π in B — by t. Clearly, we can bound $|\Pi|$ by $|\pi|$, i.e. the depth of the position π. By setting $\{A(\mathbf{s}^p(0), \overline{t_i})\}_s^{t*}$ or a formula in $\{\Xi\}_s^{t*}$ for B and (repeatedly) applying the cut rule we can transform the derivation of (5) and derivations of appropriate instances of (6) into a derivation of

$$s = t, \Xi', \Pi' \vdash A(\mathbf{s}^p(0), \{\overline{t_1}\}_s^{t*}), \ldots, A(\mathbf{s}^p(0), \{\overline{t_n}\}_s^{t*}) \tag{9}$$

where Ξ' and Π' consist of instances of identity axioms and $|\Xi'| = |\Xi|$. Since we replaced only occurrences of s in positions that are already present in $A(.,.)$ and in the identity axioms, respectively, $|\Pi'|$ does not depend on the size of the parameter p. Therefore (9) is the sequent (6) that we have been looking for. ∎

The above proof is easily adapted to the case where the parameter is composed using a binary function symbol.

Definition 2. *We call a binary function symbol* $\diamond$ fanning out *in presence of a set of axioms* $\mathsf{NFP}_\diamond$ *such that for all* $m \neq n$ *the formula* $[x]^m_\diamond \neq [x]^n_\diamond$ *is provable from* $\mathsf{NFP}_\diamond$ *and the identity axioms.*

Corollary 1. *Let the propositional sequent*

$$\Xi, s = t \supset r_1(s) = r_1(t), \ldots, s = t \supset r_k(s) = r_k(t) \vdash A([c]^p_\diamond, \overline{t_1}), \ldots, A([c]^p_\diamond, \overline{t_n})$$

by provable in **LK**, *where* Ξ *consists of instances of identity axioms, and* k *as well as* $|\Xi|$ *are independent of the size* p *of the parameter* $[c]^p_\diamond$.

If $\diamond$ *is fanning out then a sequent*

$$\Xi', \Omega \vdash A([c]^p_\diamond, \overline{t'_1}), \ldots, A([c]^p_\diamond, \overline{t'_r})$$

is provable in **LK**, *where* $r \leq 2n$, Ω *is a (possibly empty) sequence of instances of the axioms* $\mathsf{NFP}_\diamond$ *and* Ξ *is a sequence of instances of identity axioms and both* $|\Xi|$ *and* $|\Omega|$ *are independent of the size* p *of the parameter.*

6 Final Remark

Two instances of the schema of identity (ID) are necessary indeed to obtain the "short proofs" of Theorem 2. This is because in presence of arbitrary instances of $\mathbf{s}(x) = x$ the condition of "fanning out" in Theorem 3 can be replaced by the following:

> For every term s and every n there are terms $t_1, \ldots, t_n$ such that $s \neq t_i$ and $t_i \neq t_j$ is provable for all $i, j \in \{1, \ldots, n\}$, $i \neq j$.

References

[Baaz, M. and Leitsch, A., 1994] On Skolemization and Proof Complexity. *Fundamenta Informaticae* **20**(4), pages 353–379.

[Baaz, M. and Leitsch, A., 1999] Cut normal forms and proof complexity. *Ann. Pure Appl. Logic* **97**(1-3), pages 127–177.

[Baaz, M. and Pudlák, P., 1993] Kreisel's conjecture for L$\exists_1$. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 29–59. Oxford University Press, 1993. With a postscript by G. Kreisel.

[Leitsch, A., 1997] The Resolution Calculus. Springer-Verlag, Berlin, Heidelberg, New York.

[Takeuti, G., 1980] Proof Theory. North-Holland, Amsterdam, 2nd edition.

[Yukami, T., 1984] Some results on speed-up. *Ann. Japan Assoc. Philos. Sci.*, **6**, pages 195–205.

Comparing the Complexity of Cut-Elimination Methods

Matthias Baaz[1] and Alexander Leitsch[2]

[1] Institut für Computermathematik (E-118),
TU-Vienna, Wiedner Hauptstraße 8-10,
1040 Vienna, Austria
baaz@logic.at
[2] Institut für Computersprachen (E-185),
TU-Vienna, Favoritenstraße 9,
1040 Vienna, Austria
leitsch@logic.at

Abstract. We investigate the relative complexity of two different methods of cut-elimination in classical first-order logic, namely the methods of Gentzen and Tait. We show that the methods are incomparable, in the sense that both can give a nonelementary speed-up of the other one. More precisely we construct two different sequences of LK-proofs with cuts where cut-elimination for one method is elementary and nonelementary for the other one. Moreover we show that there is also a nonelementary difference in complexity for different deterministic versions of Gentzen's method.

1 Introduction

Gentzen's fundamental paper introduced cut-elimination as a fundamental procedure to extract proof theoretic information from given derivations such as Herbrand's Theorem, called Mid-Sequent Theorem in this context. In traditional proof theory, the general *possibility* to extract such informations is stressed, but there is less interest in applying the procedures in concrete cases. This, however, becomes essential if proof theory is considered as a basis for an *automated analysis of proofs*, which becomes important in connection with the development of effective program solving software for mathematical applications such as MATHEMATICA.

In this paper we compare the two most prominent cut-elimination procedures for *classical* logic: Gentzen's procedure and Tait's procedure; we avoid to call them "algorithms" because of their highly indeterministic aspects. From a procedural point of view, they are characterized by their different *cut-selection rule*: Gentzen's procedure selects a highest cut, while Tait's procedure selects a largest one (w.r.t. the number of connectives and quantifiers). The most important logical feature of Gentzen's procedure is, that – contrary to Tait's method – it transforms intuitionistic proofs into intuitionistic proofs (within **LK**) and there is no possibility to take into account classical logic when intended. Tait's

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 49–67, 2001.

procedure, on the other hand, *does not change the inner connections of the derivation*, it replaces cuts by smaller ones without reordering them.

In this paper, we use the sequence γ_n of LK-proofs corresponding to Statman's worst-case sequence to compare Gentzen's and Tait's procedure. The sequence γ_n is transformed twice: first into a sequence ψ_n where Tait's method speeds up Gentzen's nonelementarily, and second into a sequence ϕ_n giving the converse effect. As a complexity measure we take the total number of symbol ocurrences in reduction sequences of cut-elimination (i.e. all symbol occurrences in all proofs occurring during the cut-elimination procedure are measured). Both methods are nondeterministic in nature. But also different deterministic versions of one and the same method may differ quite strongly: we show that even two different deterministic versions of Gentzen's method differ nonelementarily (w.r.t. the total lengths of the corresponding reduction sequences).

Finally we would like to emphasize that the main goal of this paper is to give a comparison of different *cut-elimination methods*. It is not our intention to investigate, at the same time, the efficiency of calculi; for this reason we do not work with improved or computationally optimized versions of **LK**, but rather take a version of **LK** which is quite close to the original one.

2 Definitions and Notation

Definition 1 (complexity of formulas). *If F is a formula in* PL *then the complexity $comp(F)$ is the number of logical symbols occurring in F. Formally we define*

$comp(F) = 0$ *if F is an atom formula,*
$comp(F) = 1 + comp(A) + comp(B)$ *if $F \equiv A \circ B$ for $\circ \in \{\wedge, \vee, \rightarrow\}$,*
$comp(F) = 1 + comp(A)$ *if $F \equiv \neg A$ or $F \equiv (Qx)A$ for $Q \in \{\forall, \exists\}$.*

Definition 2 (sequent). *A sequent is an expression of the form $\Gamma \vdash \Delta$ where Γ and Δ are finite multisets of* PL*-formulas (i.e. two sequents $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$ are considered equal if the multisets represented by Γ_1 and by Γ_2 are equal and those represented by Δ_1, Δ_2 are also equal).*

Definition 3 (the calculus LK). *The initial sequents are $A \vdash A$ for PL-formulas A. In the rules of* **LK** *we always mark the auxiliary formulas (i.e. the formulas in the premis(ses) used for the inference) and the principal (i.e. the infered) formula using different marking symbols. Thus, in our definition, $\wedge$-introduction to the right takes the form*

$$\frac{\Gamma_1 \vdash A^+, \Delta \qquad \Gamma_2 \vdash \Delta_2, B^+}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \wedge B^*, \Delta_2}$$

We usually avoid markings by putting the auxiliary formulas at the leftmost position in the antecedent of sequents and in the rightmost position in the consequent of sequents. The principal formula mostly is identifiable by the context.

Thus the rule above will be written as

$$\frac{\Gamma_1 \vdash \Delta_1, A \qquad \Gamma_2 \vdash \Delta_2, B}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B}$$

Unlike Gentzen's version of **LK** *(see [5]) ours does not contain any "automatic" contractions (in this paper we do not consider intuitionistic logic). Instead we use the additive version of* **LK** *as in the book of Girard [6], combined with multiset structure for the sequents (this is exactly the version of* **LK** *used in [3]) By the definition of sequents over multisets we don't need the exchange rules. In our notation Γ, Δ, Π and Λ serve as metavariables for multisets of formulas; $\vdash$ is the separation symbol. For a complete list of the rules we refer to [3]; we only give three logical and three structural rules here.*

– *The logical rule $\vee$-introduction left:*

$$\frac{A, \Gamma \vdash \Delta \qquad B, \Pi \vdash \Lambda}{A \vee B, \Gamma, \Pi \vdash \Delta, \Lambda} \vee : l$$

– *The logical rules for $\vee$-introduction right:*

$$\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A \vee B} \vee : r1$$

$$\frac{\Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \vee B} \vee : r2$$

– *The structural rules weakening left and right:*

$$\frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A} \; w : r \qquad\qquad \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \; w : l$$

– *The cut rule:*

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda} \; cut$$

An **LK**-*derivation* is defined as a directed tree where the nodes are occurrences of sequents and the edges are defined according to the rule applications in **LK**. Let $\mathcal{A}$ be the set of sequents occurring at the leaf nodes of an **LK**-derivation ψ and S be the sequent occurring at the root (called the *end-sequent*). Then we say that ψ is an **LK**-derivation of S from $\mathcal{A}$ (notation $\mathcal{A} \vdash_{LK} S$). If $\mathcal{A}$ is a set of initial sequents then we call ψ an **LK**-proof of S. Note that, in general, cut-elimination is only possible in **LK**-proofs.

We write

$$\begin{matrix}(\psi)\\ S\end{matrix}$$

to express that ψ is a proof with end sequent S.

Paths in an **LK**-derivation ψ, connecting sequent occurrences in ψ, are defined in the traditional way; a *branch* in ψ is a path starting in the end sequent.

We use the terms "predecessor" and "successor" in the intuitive sense (i.e. contrary to the direction of edges in the tree): If there exists a path from S_1 to S_2 then S_2 is called a *predecessor* of S_1. The successor relation is defined in a analogous way. E.g. every initial sequent is a predecessor of the end sequent.

Definition 4. *The length of a proof ω is defined by the number of symbol occurrences in ω and is denoted by $l(\omega)$.*

The famous proof of the cut-elimination property of LK is based on a double induction on rank and grade of a modified form of cut, namely the mix.

Definition 5 (mix). *Let $\Gamma \vdash \Pi$ and $\Delta \vdash \Lambda$ two sequents and A be a formula which occurs in Π and in Δ; let Π^*, Δ^* be Π, Δ without occurrences of A. Then the rule*

$$\frac{\Gamma \vdash \Pi \quad \Delta \vdash \Lambda}{\Gamma, \Delta^* \vdash \Pi^*, \Lambda}\ mix$$

is called a mix on A. Frequently we label the rule by $mix(A)$ to indicate that the mix is on A.

Definition 6. *Let ϕ be an* **LK***-proof and ψ be a subderivation of the form*

$$\frac{\begin{matrix}(\psi_1) & (\psi_2)\\ \Gamma_1 \vdash \Delta_1 & \Gamma_2 \vdash \Delta_2\end{matrix}}{\Gamma_1, \Gamma_2^* \vdash \Delta_1^*, \Delta_2}\ mix(A)$$

Then we call ψ a mix-derivation in ϕ; if the mix is a cut we speak about a cut-derivation. We define the grade *of ψ as $comp(A)$; the* left-rank *of ψ is the maximal number of nodes in a branch in ψ_1 s.t. A occurs in the consequent of a predecessor of $\Gamma_1 \vdash \Delta_1$. If A is "produced" in the last inference of ψ_1 then the left-rank of ψ is* 1*. The* right-rank *is defined in an analogous way. The* rank *of ψ is the sum of right-rank and left-rank.*

The cut-elimination method of Gentzen can be formalized as a reduction method consisting of rank- and grade reductions on **LK**-proofs. But also Tait's method can be defined in this way, but with another selection of a cut-derivation in the proof. In a slight abuse of language we speak about cut-reduction, even if the cuts are actually mixes.

Definition 7 (cut-reduction rule). *In Gentzen's proof a mix-derivation ψ is selected in an* **LK***-proof ϕ and replaced by a derivation ψ' (with the same end-sequent) s.t. the corresponding mix-derivation(s) in ψ' has either lower grade or lower rank than ψ. These replacements can be interpreted as a reduction relation on* **LK***-proofs. Following the lines of Gentzen's proof of the cut-elimination property in [5] we give a formal definition of the relation $>$ on* **LK***-proofs in the Appendix.*

Using $>$ we can define two proof reduction relations, $>_G$ for Gentzen reduction *and $>_T$ for* Tait reduction*. Let ϕ be a proof and let ψ be a mix-derivation in ϕ occurring at position λ (we write $\phi = \phi[\psi]_\lambda$); assume that $\psi > \psi'$.*

If λ is an occurrence of an uppermost mix *in ϕ then we define $\phi[\psi]_\lambda >_G \phi[\psi']_\lambda$. If λ is an occurrence of a* mix with maximal grade *in ϕ then we define $\phi[\psi]_\lambda >_T \phi[\psi']_\lambda$.*

Definition 8 (cut-reduction sequence). *Let $>_x$ be one of the reduction relations $>_T, >_G$ and ϕ be an* **LK***-proof. Then a sequence $\eta : \phi_1, \ldots, \phi_n$ is called a* cut-reduction sequence *on ϕ w.r.t. $>_x$ if the following conditions are fulfilled*

- *$\phi_1 = \phi$ and*
- *$\phi_k >_x \phi_{k+1}$ for $k = 1, ..n-1$.*

If ϕ_n is cut-free then η is called a cut-elimination sequence *on ϕ w.r.t. $>_x$.*

Note that $>_G$ is more liberal than the direct intepretation of Gentzen's induction proof as a nondeterministic algorithm. But in the speed-up by Gentzen's over Tait's procedure we use the "traditional" Gentzen procedure (where one uppermost cut is eliminated before other uppermost cuts are transformed); this makes our results even stronger. $>_T$ had to be adapted anyway, as the calculus in Tait's paper [9] is not **LK**.

Definition 9. *Let $e : \mathbb{N}^2 \to \mathbb{N}$ be the following function*

$$e(0, m) = m$$
$$e(n+1, m) = 2^{e(n,m)}.$$

A function $f : \mathbb{N}^k \to \mathbb{N}^m$ for $k, m \geq 1$ is called elementary *if there exists an $n \in \mathbb{N}$ and a Turing machine T computing f s.t. the computing time of T on input $(l_1, \ldots, l_n)$ is less or equal $e(n, |(l_1, \ldots, l_k)|)$ where $|\ |$ denotes the maximum norm on $\mathbb{N}^k$.*

The function $s : \mathbb{N} \to \mathbb{N}$ is defined as $s(n) = e(n, 1)$ for $n \in \mathbb{N}$.

Note that the functions e and s are nonelementary.

Definition 10 (NE-improvement). *Let η be a cut-elimination sequence. We denote by $\|\eta\|$ the number of all symbol occurrences in η (i.e. the symbolic length of η). Let $>_x$ and $>_y$ be two cut-reduction relations (e.g. $>_T$ and $>_G$). We say that $>_x$ NE-improves $>_y$ (NE stands for nonelementarily) if there exists a sequence of LK-proofs $(\gamma_n)_{n\in\mathbb{N}}$ with the following properties:*

1. *There exists an elementary function f s.t. for all n there exists a cut-elimination sequence η_n on γ_n w.r.t. $>_x$ with $\|\eta_n\| < f(n)$,*
2. *For all elementary functions g there exists an $m \in \mathbb{N}$ s.t. for all n with $n > m$ and for all cut-elimination sequences θ on γ_n w.r.t. $>_y$: $\|\theta\| > g(n)$.*

3 The Proof Sequence of R. Statman

In [8] Richard Statman proved the remarkable result, that there are sequences of formulas having short proofs, but a nonelementarily increasing Herbrand complexity. More formally, there exists a sequence $(S_n)_{n\in\mathbb{N}}$ of sequents having **LK**-proofs $(\gamma_n)_{n\in\mathbb{N}}$ with $l(\gamma_n) \leq n2^{an}$ for some constant a, but the Herbrand complexity of S_n is $> s(n)/2$ (see Definition 9). As S_n consists of prenex formulas (in fact of universal closures), Herbrand complexity is a lower bound on the length of a cut-free proof. Putting things together we obtain a sequence $(\gamma_n)_{n\in\mathbb{N}}$ of **LK**-proofs of $(S_n)_{n\in\mathbb{N}}$ with the following properties:

- There exist a constant a with $l(\gamma_n) \leq n2^{an}$,
- for all cut-free proofs ψ of S_n $l(\psi) > \frac{1}{2}s(n)$.

This yields that *every* method of cut-elimination on $(\gamma_n)_{n\in\mathbb{N}}$ must be of nonelementary expense. Our aim is to use γ_n in constructing new proofs ϕ_n in which the γ_n are in some sense "redundant"; this redundancy may be "detected" by one cut-elimination method (behaving elementarily on ϕ_n), but not by the other one (having thus nonelementary expense on ϕ_n).

For his result on Herbrand complexity Statman defines a sequence of problems in combinatory logic (expressing iterated exponentiation) together with an elegant sequence of short proofs. The detailed formalization of the short proofs in **LK** can be found in [1]. What we need here is the overall structure of the proofs γ_n, in particular the positions of the cuts; on the other hand we do not need atomic initial sequents like in [1].

The sequence S_n is of the form $\Delta_n \vdash D_n$ where Δ_n consists of a fixed sequence of closed equations + a linear number of equality axioms of at most exponential length. Instances of the formulas H_i defined below form the cut formulas in γ_n, where p is a constant symbol:

$$H_1(y) \equiv (\forall x_1)px_1 = p(yx_1),$$
$$H_{i+1}(y) \equiv (\forall x_{i+1})(H_i(x_{i+1}) \to H_i(yx_{i+1})).$$

From the definition of the H_i it is easy to derive $l(H_i) \leq 2^{bi}$ for all i and some constant b; moreover $comp(H_i) < 2^{i+1}$.

The sequence γ_n is of the form

$$\dfrac{H_1(q) \vdash H_1(q) \qquad \dfrac{\delta_n \quad H_2(\mathbf{T}_n), H_1(q) \vdash H_1(\mathbf{T}_n q)}{\Gamma_{2n+1}, H_1(q) \vdash H_1(\mathbf{T}_n q)}\,(\chi_1)\ cut}{\Delta_n \vdash H_1(\mathbf{T}_n q)}\ cut$$

where δ_n is

$$\dfrac{\dfrac{(\psi_2)}{\Gamma_{2n-1} \vdash H_2(\mathbf{T})} \qquad \dfrac{\dfrac{(\delta_{n-1})}{\Gamma_{2(n-1)} \vdash H_3(\mathbf{T}_{n-1})} \quad \dfrac{(\chi_2)}{H_3(\mathbf{T}_{n-1}), H_2(\mathbf{T}) \vdash H_2(\mathbf{T}_n)}\ cut}{\Gamma_{2(n-1)}, H_2(\mathbf{T}) \vdash H_2(\mathbf{T}_n)}\ cut}{\Gamma_{2n} \vdash H_2(\mathbf{T}_n)}\ cut$$

$\mathbf{T}$ is a combinator defined from the basic combinators $\mathbf{S}, \mathbf{B}, \mathbf{C}, \mathbf{I}$ by $\mathbf{T} \equiv (\mathbf{SB})((\mathbf{CB})\mathbf{I})$ and $\mathbf{T}_{i+1} = \mathbf{T}_i\mathbf{T}$; $\mathbf{T}$ fulfils the equation $(\mathbf{T}y)x = y(yx)$. q is a constant symbol and $\Gamma_1, \Gamma_2, \Gamma_3$ are subsequences of Δ_n. For details we refer to [1]. The proofs ψ_i and χ_i are cutfree proofs with a fixed number of sequents; their length, however, is exponential as the formulas H_i grow in size exponentially. The cuts are hierarchically located in a linear structure: there is only one uppermost cut, i.e. all cuts occur below this cut; every cut has all other cuts as his predecessor or as his successor.

We give some informal description of the proofs ψ_i and χ_i, for greater detail we refer to [1].

First we describe the proof ψ_{i+2} for $i \geq 1$; the case ψ_2 is quite similar. By definition of the formulas H_i we have to show

$$(\forall x_{i+2})(H_{i+1}(x_{i+2}) \to H_{i+1}(\mathbf{T}x_{i+2})).$$

Using the definition of $H_{i+1}(y)$ for some variable y we first prove (in a fixed number of steps) from $H_{i+1}(y)$ that

$$(\forall x_{i+1})(H_i(x_{i+1}) \to H_i(y(yx_{i+1})))$$

holds. Then use an equational proof of $y(yx_{i+1}) = (\mathbf{T}y)x_{i+1}$ and insert it into the consequent of the upper implication via appropriate equational axioms. This way we obtain the formula

$$(\forall x_{i+1})(H_i(x_{i+1}) \to H_i((\mathbf{T}y)x_{i+1}))$$

which is just $H_{i+1}(Ty)$. So we have a derivation of

$$H_{i+1}(y) \to H_{i+1}(\mathbf{T}y).$$

By using y as eigenvariable and applying $\forall$-introduction we obtain

$$(\forall x_{i+2})(H_{i+1}(x_{i+2}) \to H_{i+1}(\mathbf{T}x_{i+2}))$$

which is $H_{i+2}(\mathbf{T})$. This eventually gives the proof ψ_{i+2}.

The proofs χ_i:
We prove the sequent

$$H_{i+1}(\mathbf{T}_{n-i+1}), H_i(\mathbf{T}) \vdash H_i(\mathbf{T}_{n-i+2})$$

by

$$\dfrac{\dfrac{H_i(\mathbf{T}) \vdash H_i(\mathbf{T}) \quad H_i(\mathbf{T}_{n-i+2}) \vdash H_i(\mathbf{T}_{n-i+2})}{H_i(\mathbf{T}) \to H_i(\mathbf{T}_{n-i+2}), H_i(\mathbf{T}) \vdash H_i(\mathbf{T}_{n-i+2})} \to : l}{(\forall x_{i+1})(H_i(x_{i+1}) \to H_i(\mathbf{T}_{n-i+1}x_{i+1})), H_i(\mathbf{T}) \vdash H_i(\mathbf{T}_{n-i+2})} \forall : l$$

which consists of 4 sequents only; the length of χ_i is exponential in i.

Roughly described, the elimination of the uppermost cut is only exponential and the elimination of a further cut below increases the degree of exponentiation. As there are linearly many cut formulas

$$H_1(q), H_2(\mathbf{T}), \ldots, H_{n+2}(\mathbf{T}) \text{ and } H_n(\mathbf{T}_2), \ldots, H_2(\mathbf{T}_n)$$

the resulting total expense is nonelementary.

4 Comparing the Methods of Gentzen and Tait

Our first result expresses the fact that a cut-elimination method selecting maximal cuts can be nonelementarily faster than methods selecting an uppermost cut.

Theorem 1. *Tait's method can give a nonelementary speed-up of Gentzen's method or more formally:* $>_T$ *NE-improves* $>_G$.

Proof. Let γ_n be Statman's sequence defined in Section 3. We know that the maximal complexity of cut formulas in γ_n is less than 2^{n+3}. Let $g(n) = 2^{n+3}$ and the formulas A_i be defined as

$$A_0 = A \text{ for an atom formula } A$$
$$A_{i+1} = \neg A_i \text{ for } i \in \mathbb{N}.$$

For every $n \in \mathbb{N}$ we set $E_n \equiv A_{g(n)}$. Then clearly $comp(E_n) = g(n)$ and thus is greater than the cut-complexity of γ_n. We will build E_n into a more complex formula, making this formula the main formula of a cut. For every $n \in \mathbb{N}$ let ψ_n be the **LK**-proof:

$$\dfrac{\dfrac{\dfrac{(\gamma_n)}{E_n \vdash E_n \quad \Delta_n \vdash D_n}\wedge : r}{E_n, \Delta_n \vdash D_n \wedge E_n} \quad \dfrac{\dfrac{E_n \vdash E_n \quad A \vdash A}{E_n \to A, E_n \vdash A}\to : l}{D_n \wedge E_n, E_n \to A \vdash A}\wedge : l}{E_n, \Delta_n, E_n \to A \vdash A}\, cut$$

By definition of γ_n the proofs γ_n and ψ_n contain only a linear number of sequents, where the size of each sequent is less or equal than $n2^{cn}$ for some constant independent of n. Consequently there exists a constant d s.t. $l(\psi_n) \leq n^2 2^{dn}$ for all n.

We now construct a cut-elimination sequence on ψ_n based on Tait's cut-reduction $>_T$. As $comp(E_n)$ is greater than the cut-complexity of γ_n, and $comp(D_n \wedge E_n) > comp(E_n)$, the most complex cut formula in ψ_n is $D_n \wedge E_n$. This formula is selected by Tait's method and we obtain $\psi_n >_T \psi'_n$ (via rule 3.113.31 in the appendix) for the proof ψ'_n below

$$\dfrac{\dfrac{E_n \vdash E_n \quad \dfrac{E_n \vdash E_n \quad A \vdash A}{E_n, E_n \to A \vdash A}\to : l}{E_n, E_n \to A \vdash A}\, cut}{E_n, \Delta_n, E_n \to A \vdash A}\, w : l^*$$

ψ'_n contains only one single cut with cut formula E_n. Now the left hand side of the cut consists of an atomic sequent only making rule 3.111. applicable. Moreover the cut with formula E_n is the only one in ψ'_n. So reduction can be applied via $>_T$ and we obtain $\psi'_n >_T \psi''_n$ for ψ''_n:

$$\dfrac{\dfrac{\dfrac{E_n \vdash E_n \quad A \vdash A}{E_n, E_n \to A \vdash A}\to : l}{E_n, \Delta_n, E_n \to A \vdash A}\, w : l^*$$

Therefore $\eta_n : \psi_n, \psi'_n, \psi''_n$ is a cut-elimination sequence based on $>_T$. It is easy to see that the lengths of the proofs decrease in every reduction step. So we obtain

$$\|\eta_n\| \leq n^2 2^{dn+2}.$$

In the second part of the proof we show that *every* cut-elimination sequence on ψ_n based on the relation $>_G$ is of nonelementary length in n.

Note that every cut in γ_n lies *above* the cut with cut formula $D_n \wedge E_n$. Therefore, in Gentzen's method, we have to eliminate all cuts in γ_n before eliminating the cut with $D_n \wedge E_n$. So every cut-elimination sequence on ψ_n based on $>_G$ must contain a proof of the form

$$\dfrac{\dfrac{\overset{(\gamma_n^*)}{E_n \vdash E_n} \quad \Delta_n \vdash D_n}{E_n, \Delta_n \vdash D_n \wedge E_n}\, \wedge : r \qquad \dfrac{\dfrac{E_n \vdash E_n \quad A \vdash A}{E_n \to A, E_n \vdash A}\, \to : l}{D_n \wedge E_n, E_n \to A \vdash A}\, \wedge : l}{E_n, \Delta_n, E_n \to A \vdash A}\, cut$$

where γ_n^* is a cut-free proof of $\Delta_n \vdash D_n$. But according to Statman's result we have $l(\gamma_n^*) > \frac{s(n)}{2}$. Clearly the length of γ_n^* is a lower bound on the length of every cut-elimination sequence on ψ_n based on $>_G$. Thus for all cut-elimination sequences θ on ψ_n w.r.t. $>_G$ we obtain

$$\|\theta\| > \frac{s(n)}{2}.$$

◇

A nonelementary speed-up is possible also the other way around. In this case it is an advantage to select the cuts from upwards instead by formula complexity.

Theorem 2. *Gentzen's method can give a nonelementary speed-up of Tait's method or more formally: $>_G$ NE-improves $>_T$.*

Proof. Consider Statman's sequence γ_n defined in Section 3. Locate the uppermost proof δ_1 in γ_n; note that δ_1 is identical to ψ_{n+1}. In γ_n we first replace the proof δ_1 (or ψ_{n+1}) of $\Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})$ by the proof $\hat{\delta}_1$ below:

$$\dfrac{\dfrac{\overset{(\omega)}{P \wedge \neg P}}{P \wedge \neg P \vdash Q}\, w : r \qquad \dfrac{\overset{(\psi_{n+1})}{\Gamma_1 \vdash H_{n+1}(\mathbf{T})}}{Q, \Gamma_1 \vdash H_{n+1}(\mathbf{T})}\, w : l}{P \wedge \neg P, \Gamma_1 \vdash H_{n+1}(\mathbf{T})}\, cut$$

The subproof ω is a proof of $P \wedge \neg P \vdash$ of constant length. Furthermore we use the same inductive definition in defining $\hat{\delta}_k$ as that of δ_k in Section 3. Finally we obtain a proof ϕ_n in place of γ_n. Note that ϕ_n differs from γ_n only by an additional (atomic) cut and the formula $P \wedge \neg P$ in the antecedents of sequents. Clearly

$$l(\phi_n) \leq l(\gamma_n) + cn$$

for some constant c.

Our aim is to define a cut-elimination sequence on ϕ_n w.r.t. $>_G$ which is of elementary complexity. Let S_k be the end sequent of the proof $\hat{\delta}_k$. We first investigate cut-elimination on the proof $\hat{\delta}_n$; the remaining two cuts are eliminated in a similar way. To this aim we prove by induction on k:

($*$) There exists a cut-elimination sequence $\hat{\delta}_{k,1}, \ldots, \hat{\delta}_{k,m}$ of $\hat{\delta}_k$ w.r.t. $>_G$ with the following properties:
(1) $m \leq l(\hat{\delta}_k)$,
(2) $l(\hat{\delta}_{k,i}) \leq l(\hat{\delta}_k)$ for $i = 1, \ldots, m$,
(3) $\hat{\delta}_{k,m}$ is of the form

$$\frac{\begin{array}{c}(\omega)\\ P \wedge \neg P \vdash\end{array}}{S_k}\ w :^*$$

Induction basis $k = 1$:
In $\hat{\delta}_1$ there is only one cut (with the formula Q) where the cut formula is introduced by weakening. Thus by definition of $>_G$, using the rule 3.113.1, we get $\hat{\delta}_1 >_G \hat{\delta}_{1,2}$ where $\hat{\delta}_{1,2}$ is the proof

$$\frac{\begin{array}{c}(\omega)\\ P \wedge \neg P \vdash\end{array}}{P \wedge \neg P, \Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\ w :^*$$

Clearly $2 \leq l(\hat{\delta}_1)$ and $l(\hat{\delta}_{1,2}) \leq l(\hat{\delta}_1)$. Moreover $\hat{\delta}_{1,2}$ is of the form (3). This gives ($*$) for $k = 1$.

(IH) Assume that ($*$) holds for k.

By definition, $\hat{\delta}_{k+1}$ is of the form

$$\frac{\begin{array}{c}(\psi_{n-k+1})\\ \Gamma_{2k+1} \vdash H_{n-k+1}(\mathbf{T}) \quad \rho_k\end{array}}{P \wedge \neg P, \Gamma_{2(k+1)} \vdash H_{n-k+1}(\mathbf{T}_{k+1})}\ cut$$

for ρ_k:

$$\frac{\begin{array}{cc}(\hat{\delta}_k) & (\chi_{n-k+1})\\ P \wedge \neg P, \Gamma_{2k} \vdash H_{n-k+2}(\mathbf{T}_k) & H_{n-k+2}(\mathbf{T}_k), H_{n-k+1}(\mathbf{T}) \vdash H_{n-k+1}(\mathbf{T}_{k+1})\end{array}}{P \wedge \neg P, \Gamma_{2k}, H_{n-k+1}(\mathbf{T}) \vdash H_{n-k+1}(\mathbf{T}_{k+1})}\ cut$$

By (IH) there exists a cut-elimination sequence $\hat{\delta}_{k,1}, \ldots, \hat{\delta}_{k,m}$ on $\hat{\delta}_k$ w.r.t. $>_G$ fulfilling (1), (2) and (3). In particular we have $l(\hat{\delta}_{k,m}) \leq l(\hat{\delta}_k)$ and $\hat{\delta}_{k,m}$ is of the form

$$\frac{\begin{array}{c}(\omega)\\ P \wedge \neg P\end{array}}{S_k}\ w :^*$$

All formulas in S_k, except $P \wedge \neg P$, are introduced by weakening in $\hat{\delta}_{k,m}$. In particular this holds for the formula $H_{n-k+2}(\mathbf{T}_k)$ which is a cut formula in

$\hat{\delta}_{k+1}$. After cut-elimination on $\hat{\delta}_k$ the proof ρ_k is transformed (via $>_G$) into a proof $\hat{\rho}_k$:

$$\frac{\begin{array}{cc}(\hat{\delta}_{k,m}) & (\chi_{n-k+1})\\ P\wedge\neg P, \Gamma_{2k} \vdash H_{n-k+2}(\mathbf{T}_k) & H_{n-k+2}(\mathbf{T}_k), H_{n-k+1}(\mathbf{T}) \vdash H_{n-k+1}(\mathbf{T}_{k+1})\end{array}}{P\wedge\neg P, \Gamma_{2k}, H_{n-k+1}(\mathbf{T}) \vdash H_{n-k+1}(\mathbf{T}_{k+1})}\ cut$$

Now the (only) cut in $\hat{\rho}_k$ is with the cut formula $H_{n-k+2}(\mathbf{T}_k)$ which is introduced by $w : r$ in $\hat{\delta}_{k,m}$. By using iterated reduction of left-rank via the symmetric versions of 3.121.21 and 3.121.22 in the appendix, the cut is eliminated and the proof χ_{n-k+1} "disappears" and the result is again of the form

$$\frac{\begin{array}{c}(\omega)\\ P\wedge\neg P\end{array}}{P\wedge\neg P, \Gamma_{2k}, H_{n-k+1}(\mathbf{T}) \vdash H_{n-k+1}(\mathbf{T}_{k+1})}\ w :^*$$

The proof above is the result of a cut-elimination sequence $\hat{\rho}_{k,1}, \ldots, \hat{\rho}_{k,p}$ on $\hat{\rho}_k$ w.r.t. $>_G$. But then also δ_{k+1} is further reduced to a proof where $\hat{\rho}_k$ is replaced by $\hat{\rho}_{k,p}$; in this proof there is only one cut left (with the formula $H_{n-k+1}(\mathbf{T})$ and we may play the "weakening game" once more. Finally we obtain a proof $\hat{\delta}_{k,r}$ of the form

$$\frac{\begin{array}{c}(\omega)\\ P\wedge\neg P\end{array}}{P\wedge\neg P, \Gamma_{2(k+1)} \vdash H_{n-k+1}(\mathbf{T}_{k+1})}\ w :^*$$

The conditions (1) and (2) are obviously fulfilled. This eventually gives $(*)$.

After the reduction of ϕ_n to $\phi_n[\hat{\delta}_{n,s}]_\lambda$, where λ is the position of $\hat{\delta}_n$ and $\hat{\delta}_{n,s}$ is the result of a Gentzen cut-elimination sequence on $\hat{\delta}_n$, there are only two cuts left. Again these cuts are swallowed by the proofs beginning with ω and followed by a sequence of weakenings. Putting things together we obtain a cut-elimination sequence

$$\eta_n : \ \phi_{n,1}, \ldots, \phi_{n,q}$$

on ϕ_n w.r.t. $>_G$ with the properties:

(1) $l(\phi_{n,i}) \leq l(\phi_n)$ and
(2) $q \leq l(\phi_n)$.

But then

$$\|\eta_n\| \leq l(\phi_n)^2 \leq n^4 2^{cn}.$$

for an appropriate constant c. Therefore η_n is a Gentzen cut-elimination sequence on ϕ_n of elementary complexity.

For the other direction consider Tait's reduction method on the sequence ϕ_n. The cut formulas in ϕ_n fall into two categories;

- the new cut formula Q with $comp(Q) = 0$ and
- the old cut formulas from γ_n.

Now let η be an arbitrary cut-elimination sequence on ϕ_n w.r.t. $>_T$. By definition of $>_T$ only cuts with maximal cut formulas can be selected in a reduction step w.r.t. $>_T$. Therefore, there exists a constant k s.t. $k \geq 1$ and η contains a proof ψ with cut-complexity k. As the new cut in ϕ_n with cut formula Q is of complexity 0, it is still present in ψ.

A straightforward proof transformation gives a proof χ s.t. $\gamma_n >_T^* \chi$, the cut-complexity of χ is k, and $l(\chi) < l(\psi)$ (in some sense the Tait procedure does not "notice" the new atomic cut). But every cut-free proof of γ_n has a length $> \frac{s(n)}{2}$ and cut-elimination of cuts with (fixed) complexity k is elementary [7].

More precisely there exists an elementary function f and a cut-elimination sequence θ on χ w.r.t. $>_T$ s.t. $\|\theta\| \leq f(l(\chi))$. This is only possible if there is no elementary bound on $l(\chi)$ in terms of n (otherwise we would get cut-free proofs of γ_n of length elementarily in n). But then there is no elementary bound on $l(\psi)$ in terms of n. Putting things together we obtain that for every elementary function f and for *every* cut-elimination sequence η on ϕ_n

$$\|\eta\| > f(n) \text{ almost everywhere .}$$

$\diamond$

Theorem 2 shows that there exist cut elimination sequences η_n on ϕ_n w.r.t. $>_G$ s.t. $\|\eta_n\|$ is elementarily bounded in n; however this does not mean that *every* cut-elimination sequence on ϕ_n w.r.t. $>_G$ is elementary. In fact $>_G$ is highly "unstable" in its different deterministic versions. Consider the subproof $\hat{\delta}_1$ in the proof of Theorem 2:

$$\dfrac{\dfrac{\dfrac{}{P \wedge \neg P}\,(\omega)}{P \wedge \neg P \vdash Q}\; w:r \qquad \dfrac{\dfrac{}{\Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\,(\psi_{n+1})}{Q, \Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\; w:l}{P \wedge \neg P, \Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\; cut$$

If, in $>_G$, we focus on the weakening $(w:l)$ in the right part of the cut and apply rule 3.113.2 (appendix) we obtain $\hat{\delta}_1 >_G \mu$, where μ is the proof

$$\dfrac{\dfrac{}{\Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\,(\psi_{n+1})}{P \wedge \neg P, \Gamma_{n+1} \vdash H_{n+1}(\mathbf{T})}\; w:l$$

But μ contains the whole proof ψ_{n+1}. In the course of cut-elimination ψ_{n+1} is built into the produced proofs exactly as in the cut-elimination procedure on γ_n itself. The resulting cut-free proof is in fact longer than γ_n^* (the corresponding cut-free proof of the n-th element of Statman's sequence) and thus is of nonelementary length! This tells us that there are different deterministic versions α_1 and α_2 of $>_G$ s.t. α_1 gives a nonelementary speed-up of α_2 on the input set $(\phi_n)_{n \in \mathbb{N}}$.

In the introduction of additional cuts into Statman's proof sequence we use the weakening rule. Similar constructions can be carried out in versions of the

Gentzen calculus without weakening. What we need is just a sequence of short **LK**-proofs of valid sequents containing "simple" redundant (in our case atomic) formulas on both sides serving as cut formulas. Note that **LK** without any redundancy (working with minimally valid sequents only) is not complete.

5 Conclusion

The main results of this paper hint to a more general theory of algorithmic cut elimination encompassing not only algorithmic specifications of Gentzen's and Tait's procedures but also approaches as cut projection [2] and the resolution based method CERES [4] . From a more proof theoretic point of view, the sequences of proofs arising from the different stages of cut-elimination can be considered as a specific analysis of the proof which extends the information obtainable from the cut-free final stage. Different cut-elimination algorithms stress different aspects of the proof, e.g. constructive content (Gentzen's procedure) or connectivity (Tait's procedure).

References

1. M. Baaz, A. Leitsch: On skolemization and proof complexity, *Fundamenta Informaticae*, 20(4), pp. 353–379, 1994.
2. M. Baaz, A. Leitsch: Fast Cut-Elimination by Projection, Proc. CSL'96, *Lecture Notes in Computer Science* 1258, pp. 18–33, Springer, Berlin, 1997.
3. M. Baaz, A. Leitsch: Cut normal forms and proof complexity, *Annals of Pure and Applied Logic*, 97, pp. 127-177, 1999.
4. M. Baaz, A. Leitsch: Cut-Elimination and Redundancy-Elimination by Resolution, *Journal of Symbolic Computation*, 29, pp. 149-176, 2000.
5. G. Gentzen: Untersuchungen über das logische Schließen, *Mathematische Zeitschrift* 39, pp. 405–431, 1934–1935.
6. J.Y. Girard: Proof Theory and Logical Complexity, in *Studies in Proof Theory*, Bibliopolis, Napoli, 1987.
7. H. Schwichtenberg: Proof Theory: Some Applications of Cut-Elimination, in *Handbook of Mathematical Logic*, ed. by J. Barwise, North Holland, Amsterdam, pp. 867–895, 1989.
8. R. Statman: Lower bounds on Herbrand's theorem, in *Proc. of the Amer. Math. Soc.* 75, pp. 104–107, 1979.
9. W.W. Tait: Normal derivability in classical logic, in *The Syntax and Semantics of Infinitary Languages*, ed. by J. Barwise, Springer, Berlin, pp. 204–236, 1968.
10. G. Takeuti: Proof Theory, North-Holland, Amsterdam, 2nd edition, 1987.

Appendix

Below we list the transformation rules used in Gentzen's proof of cut-elimination in [5]. Thereby we use the same numbers for labelling the subcases. Note that our rules slightly differ from that of Gentzen as we use the purely additive version of **LK**. If a mix-derivation ψ is transformed to ψ' then we define $\psi > \psi'$; remember

that the relation $>$ is the crucial tool in defining Gentzen- and Tait reduction. In all reductions below ψ is a mix-derivation of the form

$$\dfrac{\begin{matrix}(\psi_1) & (\psi_2)\\ \Gamma_1 \vdash \Delta_1 & \Gamma_2 \vdash \Delta_2\end{matrix}}{\Gamma_1, \Gamma_2^* \vdash \Delta_1^*, \Delta_2}\ mix$$

3.11. rank $= 2$.

3.111. $\psi_1 = A \vdash A$:

$$\dfrac{\begin{matrix}A \vdash A & \begin{matrix}(\psi_2)\\ \Delta \vdash \Lambda\end{matrix}\end{matrix}}{A, \Delta^* \vdash \Lambda}\ mix(A)$$

transforms to

$$\dfrac{\begin{matrix}(\psi_2)\\ \Delta \vdash \Lambda\end{matrix}}{A, \Delta^* \vdash \Lambda}\ c : l^*$$

3.112. $\psi_2 = A \vdash A$: analogous to 3.111.

3.113.1. the last inference in ψ_1 is $w : r$:

$$\dfrac{\begin{matrix}\begin{matrix}(\chi_1)\\ \Gamma \vdash \Delta\end{matrix} & \begin{matrix}(\psi_2)\\ \Pi \vdash \Lambda\end{matrix}\\ \dfrac{}{\Gamma \vdash \Delta, A}\ w : r & \end{matrix}}{\Gamma, \Pi^* \vdash \Delta, \Lambda}\ mix(A)$$

transforms to

$$\dfrac{\begin{matrix}(\chi_1)\\ \Gamma \vdash \Delta\end{matrix}}{\Gamma, \Pi^* \vdash \Delta, \Lambda}\ w : l^*$$

3.113.2. the last inference in ψ_2 is $w : l$: symmetric to 3.113.1.

The last inferences in ψ_1, ψ_2 are logical ones and the mix-formula is the principal formula of these inferences:

3.113.31.

$$\dfrac{\begin{matrix}\dfrac{\begin{matrix}(\chi_1) & (\chi_2)\\ \Gamma_1 \vdash \Theta_1, A & \Gamma_2 \vdash \Theta_2, B\end{matrix}}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2, A \wedge B}\ \wedge : r & \dfrac{\begin{matrix}(\chi_3)\\ A, \Gamma_3 \vdash \Theta_3\end{matrix}}{A \wedge B, \Gamma_3 \vdash \Theta_3}\ \wedge : l\end{matrix}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Theta_1, \Theta_2, \Theta_3}\ mix(A \wedge B)$$

transforms to

$$\dfrac{\dfrac{\begin{matrix}(\chi_1) & (\chi_3)\\ \Gamma_1 \vdash \Theta_1, A & A, \Gamma_3 \vdash \Theta_3\end{matrix}}{\Gamma_1, \Gamma_3^* \vdash \Theta_1^*, \Theta_3}\ mix(A)}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Theta_1, \Theta_2, \Theta_3}\ w :^*$$

For the other form of $\wedge : l$ the transformation is straightforward.

3.113.32. The last inferences of ψ_1, ψ_2 are $\vee : r, \vee : l$: symmetric to 3.113.31.

3.113.33.

$$\dfrac{\dfrac{\begin{matrix}(\chi_1[\alpha])\\ \Gamma_1 \vdash \Theta_1, B^x_\alpha\end{matrix}}{\Gamma_1 \vdash \Theta_1, (\forall x)B}\ \forall : r \quad \dfrac{\begin{matrix}(\chi_2)\\ B^x_t, \Gamma_2 \vdash \Theta_2\end{matrix}}{(\forall x)B, \Gamma_2 \vdash \Theta_2}\ \forall : l}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2}\ mix((\forall x)B)$$

transforms to

$$\dfrac{\begin{matrix}(\chi_1[t])\\ \Gamma_1 \vdash \Theta_1, B^x_t\end{matrix} \quad \begin{matrix}(\chi_2)\\ B^x_t, \Gamma_2 \vdash \Theta_2\end{matrix}}{\dfrac{\Gamma_1, \Gamma_2^* \vdash \Theta_1^*, \Theta_2}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2}\ w :^*}\ mix(B^x_t)$$

3.113.34. The last inferences in ψ_1, ψ_2 are $\exists : r, \exists : l$: symmetric to 3.113.33.

3.113.35

$$\dfrac{\dfrac{\begin{matrix}(\chi_1)\\ A, \Gamma_1 \vdash \Theta_1\end{matrix}}{\Gamma_1 \vdash \Theta_1, \neg A}\ \neg : r \quad \dfrac{\begin{matrix}(\chi_2)\\ \Gamma_2 \vdash \Theta_2, A\end{matrix}}{\neg A, \Gamma_2 \vdash \Theta_2}\ \neg : l}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2}\ mix(\neg A)$$

reduces to

$$\dfrac{\begin{matrix}(\chi_2)\\ \Gamma_2 \vdash \Theta_2, A\end{matrix} \quad \begin{matrix}(\chi_1)\\ A, \Gamma_1 \vdash \Theta_1\end{matrix}}{\dfrac{\Gamma_1^*, \Gamma_2 \vdash \Theta_1, \Theta_2^*}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2}\ w :^*}\ mix(A)$$

3.113.36.

$$\dfrac{\dfrac{\begin{matrix}(\chi_1)\\ A, \Gamma_1 \vdash \Theta_1, B\end{matrix}}{\Gamma_1 \vdash \Theta_1, A \to B}\ \to : r \quad \dfrac{\begin{matrix}(\chi_2)\\ \Gamma \vdash \Theta, A\end{matrix} \quad \begin{matrix}(\chi_3)\\ B, \Delta \vdash \Lambda\end{matrix}}{A \to B, \Gamma, \Delta \vdash \Theta, \Lambda}\ \to : l}{\Gamma_1, \Gamma, \Delta \vdash \Theta_1, \Theta, \Lambda}\ mix(A \to B)$$

reduces to

$$\dfrac{\dfrac{\begin{matrix}(\chi_2)\\ \Gamma \vdash \Theta, A\end{matrix} \quad \dfrac{\begin{matrix}(\chi_1)\\ A, \Gamma_1 \vdash \Theta_1, B\end{matrix} \quad \begin{matrix}(\chi_3)\\ B, \Delta \vdash \Lambda\end{matrix}}{A, \Gamma_1, \Delta^* \vdash \Theta_1^*, \Lambda}\ mix(B)}{\Gamma_1^+, \Gamma, \Delta^{*+} \vdash \Theta_1^*, \Theta^+, \Lambda}\ mix(A)}{\Gamma_1, \Gamma, \Delta \vdash \Theta_1, \Theta, \Lambda}\ w :^*$$

3.12. rank > 2:

3.121. right-rank > 1:

3.121.1. The mix formula occurs in the antecedent of the end-sequent of ψ_1.

$$\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\psi_2)\\ \Delta \vdash \Lambda\end{matrix}}{\Pi, \Delta^* \vdash \Sigma^*, \Lambda}\ mix(A)$$

transforms to

$$\dfrac{\dfrac{(\psi_2)}{\delta \vdash \Lambda}}{\Pi, \Delta^* \vdash \Sigma^*, \Lambda}\ w :^* c : *$$

3.121.2. The mix formula does not occur in the antecedent of the end-sequent of ψ_1.

3.121.21. Let r be one of the rules $w : l$ or $c : l$; then

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{\dfrac{(\chi_1)}{\Delta \vdash \Lambda}}{\Theta \vdash \Lambda}\ r}{\Pi, \Theta^* \vdash \Sigma^*, \Lambda}\ mix(A)$$

transforms to

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_1)}{\Delta \vdash \Lambda}}{\Pi, \Delta^* \vdash \Sigma^*, \Lambda}\ mix(A)$$
$$\dfrac{}{\Pi, \Theta^* \vdash \Sigma^*, \Lambda}\ r$$

Note that r may be "degenerated", i.e. it can be skipped if the sequent does not change.

3.121.22. Let r be an arbitrary unary rule (different from $c : l, w : l$) and let C^* be empty if $C = A$ and C otherwise. The formulas B and C may be equal or different or simply nonexisting. Let us assume that ψ is of the form

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{\dfrac{(\chi_1)}{B, \Gamma \vdash \Omega_1}}{C, \Gamma \vdash \Omega_2}\ r}{\Pi, C^*, \Gamma^* \vdash \Sigma^*, \Omega_2}\ mix(A)$$

Let τ be the proof

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_1)}{B, \Gamma \vdash \Omega_1}}{\Pi, B^*, \Gamma^* \vdash \Sigma^*, \Omega_2}\ mix(A)}{\Pi, B, \Gamma^* \vdash \Sigma^*, \Omega_2}\ w :^*$$
$$\dfrac{}{\Pi, C, \Gamma^* \vdash \Sigma^*, \Omega_2}\ r$$

3.121.221. $A \neq C$: then ψ transforms to τ.

3.121.222. $A = C$ and $A \neq B$: in this case C is the principal formula of r. Then ψ transforms to

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\tau)}{\Pi, A, \Gamma^* \vdash \Sigma^*, \Omega_2}}{\Pi, \Pi^*, \Gamma^* \vdash \Sigma^*, \Sigma^*, \Omega_2}\ mix(A)$$
$$\dfrac{}{\Pi, \Gamma^* \vdash \Sigma^*, \Omega_2}\ c :^*$$

3.121.223 $A = B = C$. Then $\Omega_1 \neq \Omega_2$ and ψ transforms to

$$\dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ A, \Gamma \vdash \Omega_1\end{matrix}}{\Pi, \Gamma^* \vdash \Sigma^*, \Omega_1}\, mix(A)}{\Pi, \Gamma^* \vdash \Sigma^*, \Omega_2}\, r$$

3.121.23. The last inference in ψ_2 is binary:

3.121.231. The case $\wedge : r$. Here

$$\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \dfrac{\begin{matrix}(\chi_1)\\ \Gamma_1 \vdash \Theta_1, B\end{matrix} \quad \begin{matrix}(\chi_2)\\ \Gamma_2 \vdash \Theta_2, C\end{matrix}}{\Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2, B \wedge C}\, \wedge : r}{\Pi, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Theta_1, \Theta_2, B \wedge C}\, mix(A)$$

transforms to

$$\dfrac{\dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ \Gamma_1 \vdash \Theta_1, B\end{matrix}}{\Pi, \Gamma_1^* \vdash \Sigma^*, \Theta_1, B}\, mix(A) \quad \dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ \Gamma_2 \vdash \Theta_2, C\end{matrix}}{\Pi, \Gamma_2^* \vdash \Sigma^*, \Theta_2, C}\, mix(A)}{\Pi, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Sigma^*, \Theta_1, \Theta_2, B \wedge C}\, \wedge : r}{\Pi, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Theta_1, \Theta_2, B \wedge C}\, c : r^*$$

3.121.232. The case $\vee : l$. Then ψ is of the form

$$\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \dfrac{\begin{matrix}(\chi_1)\\ B, \Gamma_1 \vdash \Theta_1\end{matrix} \quad \begin{matrix}(\chi_2)\\ C, \Gamma_2 \vdash \Theta_2\end{matrix}}{B \vee C, \Gamma_1, \Gamma_2 \vdash \Theta_1, \Theta_2}\, \vee : l}{\Pi, (B \vee C)^*, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Theta_1, \Theta_2}\, mix(A)$$

Again $(B \vee C)^*$ is empty if $A = B \vee C$ and $B \vee C$ otherwise.
We first define the proof τ:

$$\dfrac{\dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ B, \Gamma_1 \vdash \Theta_1\end{matrix}}{B^*, \Pi, \Gamma_1^* \vdash \Sigma^*, \Theta_1}\, mix(A)}{B, \Pi, \Gamma_1^* \vdash \Sigma^*, \Theta_1}\, x \quad \dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ C, \Gamma_2 \vdash \Theta_2\end{matrix}}{C^*, \Pi, \Gamma_2^* \vdash \Sigma^*, \Theta_2}\, mix(A)}{C, \Pi, \Gamma_2^* \vdash \Sigma^*, \Theta_2}\, x}{B \vee C, \Pi, \Pi^*, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Sigma^*, \Theta_1, \Theta_2}\, \vee : l$$

Note that, in case $A = B$ or $A = C$, the inference x is $w : l$; otherwise x is the identical transformation and can be dropped.
If $(B \vee C)^* = B \vee C$ then ψ transforms to

$$\dfrac{\tau}{\Pi, B \vee C, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Theta_1, \Theta_2}\, c :^*$$

If, on the other hand, $(B \vee C)^*$ is empty (i.e. $B \vee C = A$) then we transform ψ to

$$\dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \tau}{\Pi, \Pi^*, \Pi^*, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Sigma^*, \Sigma^*, \Theta_1, \Theta_2}\, mix(A)}{\Pi, \Gamma_1^*, \Gamma_2^* \vdash \Sigma^*, \Theta_1, \Theta_2}\, c :^*$$

3.121.233. The last inference in ψ_2 is $\rightarrow: l$. Then ψ is of the form:

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{\dfrac{(\chi_1)}{\Gamma \vdash \Theta, B} \quad \dfrac{(\chi_2)}{C, \Delta \vdash \Lambda}}{B \to C, \Gamma, \Delta \vdash \Theta, \Lambda} \rightarrow: l}{\Pi, (B \to C)^*, \Gamma^*, \Delta^* \vdash \Sigma^*, \Theta, \Lambda} \; mix(A)$$

As in 3.121.232 $(B \to C)^* = B \to C$ for $B \to C \neq A$ and $(B \to C)^*$ empty otherwise.

3.121.233.1. A occurs in Γ and in Δ. Again we define a proof τ:

$$\dfrac{\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_1)}{\Gamma \vdash \Theta, B}}{\Pi, \Gamma^* \vdash \Sigma^*, \Theta, B} \; mix(A) \quad \dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_2)}{C, \Delta \vdash \Lambda}}{C^*, \Pi, \Delta^* \vdash \Sigma^*, \Lambda} \; mix(A)}{C, \Pi, \Delta^* \vdash \Sigma^*, \Lambda} \; x}{B \to C, \Pi, \Gamma^*, \Pi, \Delta^* \vdash \Sigma^*, \Theta, \Sigma^*, \Lambda} \rightarrow: l$$

If $(B \to C)^* = B \to C$ then, as in 3.121.232, ψ is transformed to τ + some additional contractions. Otherwise an additional mix with mix formula A is appended.

3.121.233.2 A occurs in Δ, but not in Γ. As in 3.121.233.1 we define a proof τ:

$$\dfrac{\dfrac{\dfrac{(\chi_1)}{\Gamma \vdash \Theta, B} \quad \dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_2)}{C, \Delta \vdash \Lambda}}{C^*, \Pi, \Delta^* \vdash \Sigma^*, \Lambda} \; mix(A)}{C, \Pi, \Delta^* \vdash \Sigma^*, \Lambda} \; x}{B \to C, \Gamma, \Pi, \Delta^* \vdash \Theta, \Sigma^*, \Lambda} \rightarrow: l$$

Again we distinguish the cases $B \to C = A$ and $B \to C \neq A$ and define the transformation of ψ exactly like in 3.121.233.1.

3.121.233.3 A occurs in Γ, but not in Δ: analogous to 3.121.233.2.

3.121.234. The last inference in ψ_2 is $mix(B)$ for some formula B. Then ψ is of the form

$$\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{\dfrac{(\chi_1)}{\Gamma_1 \vdash \Theta_1} \quad \dfrac{(\chi_2)}{\Gamma_2 \vdash \Theta_2}}{\Gamma_1, \Gamma_2^+ \vdash \Theta_1^+, \Theta_2} \; mix(B)}{\Pi, \Gamma_1^*, \Gamma_2^{+*} \vdash \Sigma^*, \Theta_1^+, \Theta_2} \; mix(A)$$

3.121.234.1 A occurs in Γ_1 and in Γ_2. Then ψ transforms to

$$\dfrac{\dfrac{\dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_1)}{\Gamma_1 \vdash \Theta_1}}{\Pi, \Gamma_1^* \vdash \Sigma^*, \Theta_1} \; mix(A) \quad \dfrac{\dfrac{(\psi_1)}{\Pi \vdash \Sigma} \quad \dfrac{(\chi_2)}{\Gamma_2 \vdash \Theta_2}}{\Pi, \Gamma_2^* \vdash \Sigma^*, \Theta_2} \; mix(A)}{\Pi, \Pi^+, \Gamma_1^*, \Gamma_2^{+*} \vdash \Sigma^{*+}, \Sigma^{*+}, \Theta_1^+, \Theta_2} \; mix(B)}{\Pi, \Gamma_1^*, \Gamma_2^{+*} \vdash \Sigma^*, \Theta_1^+, \Theta_2} \; c:^*, w:^*$$

Note that, for $A = B$, we have $\Gamma_2^{*+} = \Gamma_2^*$ and $\Sigma^{*+} = \Sigma^*$; $\Gamma_2^{*+} = \Gamma_2^{+*}$ holds in all cases.

3.121.234.2 A occurs in Γ_1, but not in Γ_2. In this case we have $\Gamma_2^{+*} = \Gamma_2$ and we transform ψ to

$$\dfrac{\dfrac{\begin{matrix}(\psi_1)\\ \Pi \vdash \Sigma\end{matrix} \quad \begin{matrix}(\chi_1)\\ \Gamma_1 \vdash \Theta_1\end{matrix}}{\Pi, \Gamma_1^* \vdash \Sigma^*, \Theta_1}\, mix(A) \quad \begin{matrix}(\chi_2)\\ \Gamma_2 \vdash \Theta_2\end{matrix}}{\Pi, \Gamma_1^*, \Gamma_2^+ \vdash \Sigma^*, \Theta_1^+, \Theta_2}\, mix(B)$$

3.121.234.3 A is in Γ_2, but not in Γ_1: symmetric to 3.121.234.2.

3.122. right-rank = 1 and left-rank > 1: symmetric to 3.121.

Program Extraction from Gentzen's Proof of Transfinite Induction up to ε_0

Ulrich Berger

Department of Computer Science, University of Wales Swansea,
Singleton Park, Swansea SA2 8PP, UK.
u.berger@swan.ac.uk

Abstract. We discuss higher type constructions inherent to intuitionistic proofs. As an example we consider Gentzen's proof of transfinite induction up to the ordinal ε_0. From the constructive content of this proof we derive higher type algorithms for some ordinal recursive hierarchies of number theoretic functions as well as simple higher type primitive recursive definitions of tree ordinals of all heights $< \varepsilon_0$.

1 Introduction

According to the Brouwer-Heyting-Kolmogorov interpretation [19] a proof of an implication $A \to B$ is a construction transforming a (hypothetical) proof of A into a proof of B. Applying this interpretation to nested implication one is led to some kind of 'higher type' constructions. For example a proof of $(A \to B) \to C$ is a construction transforming a construction from proofs of A to proofs of B to a proof of C. Technically the constructive content of an intuitionistic proof can be made explicit by extracting from it a program 'realizing' the proven formula. This program may involve primitive recursion if the proof involves induction, and it may involve higher type functionals if the proof contains subproofs of formulas containing nested implications.

We will illustrate the role of higher types by means of a theorem in proof theory, namely Gentzen's proof of transfinite induction up to the ordinal ε_0 [5] (see also e.g. [11]). Gentzen's proof involves nesting of implications of unbounded depth, and consequently the program extracted from it contains functionals of arbitrarily high types. When applying Gentzen's theorem to predicates expressing the totality of certain hierarchies of number theoretic functions defined by transfinite recursion along ε_0 one obtains higher type algorithms for these hierarchies which do not use transfinite recursion along the wellordering ε_0, but primitive recursion in higher types instead. In the case of the fast growing hierarchy (the transfinite extension of the Ackermann function) one obtains an algorithm due to Schwichtenberg. Surprisingly, the higher type algorithms are more efficient than those using transfinite recursion along the well-ordering ε_0.

The trade-off between transfinite recursion and higher types has been analyzed in detail by Schwichtenberg [12,14] and Terlouw [16,17]. What seems to be new in our presentation is the fact that the functionals obtained by passing

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 68–77, 2001.

from transfinite recursion to higher types are precisely the constructive content of Gentzen's proof.

A similar remark applies to our second application, the extraction of higher type primitive recursive definitions of tree ordinals $< \varepsilon_0$. The method of using iteration and higher type functionals to define (notations for) ordinals goes back to Neumer [9,10] and Feferman [4], and has been pursued recently by Danner [3] and Hancock [7]. Our constructions are closely related to Aczel's work in [1], the difference being that Aczel uses transfinite types (and thus is able to construct larger ordinals).

The plan of this paper is follows. In part 2 we review Gentzen's proof reorganizing it such that it can be smoothly formalized. Using a variant of Kreisel's modified realizability interpretation we then extract, in part 3, from the formalized proof an abstract higher type algorithm, which, in part 4, is specialized to concrete primitive recursive higher type algorithms for the fast growing hierarchy and tree ordinals of all heights $< \varepsilon_0$.

2 Transfinite Induction up to ε_0

2.1 Ordinals $< \varepsilon_0$

Let us begin with a brief review of some basic facts on ordinals $< \varepsilon_0$. Each such ordinal can be uniquely written in Cantor normal form with base ω:

$$x =_{\mathrm{CNF}} \omega^{x_1} + \ldots + \omega^{x_n},$$

which means that $x = \omega^{x_1} + \ldots + \omega^{x_n}$ with $n > 0$ and $x > x_1 \geq \ldots \geq x_n$. If furthermore $y =_{\mathrm{CNF}} \omega^{y_1} + \ldots + \omega^{y_m}$, then $x < y$ iff either the sequence $(x_1, \ldots, x_n)$ is a proper initial segment of $(y_1, \ldots, y_m)$ or else there is some $i < \min(n, m)$ such that $(x_1, \ldots, x_i) = (y_1, \ldots, y_i)$ and $x_{i+1} < y_{i+1}$. We write $\mathsf{NF}(x, y)$ if $x =_{\mathrm{CNF}} \omega^{x_1} + \ldots + \omega^{x_n}$ and $x_n \geq y$. Hence in that case $x + \omega^y =_{\mathrm{CNF}} \omega^{x_1} + \ldots + \omega^{x_n} + \omega^y$. If $\mathsf{NF}(x, y)$ and k is a natural number then we also set $x + \omega^y k := x + \omega^y + \ldots + \omega^y$ (k-times ω^y) (then clearly $x + \omega^y k =_{\mathrm{CNF}} x + \omega^y + \ldots + \omega^y$). Every ordinal $< \varepsilon_0$ is either 0 or else of the form $x + 1 := x + \omega^0$, i.e. a successor ordinal, or else of the form $x + \omega^y$ where $y > 0$ and $\mathsf{NF}(x, y)$, i.e. a limit ordinal. For every limit ordinal z and every natural number k we define the k-th member of the fundamental sequence of z, denoted $z[k]$, recursively as follows: $(x + \omega^{y+1})[k] := x + \omega^x k$, if $\mathsf{NF}(x, y + 1)$. $(x + \omega^y)[k] := x + \omega^{y[k]}$, if $\mathsf{NF}(x, y)$ and y is a limit ordinal. Clearly $x < x + 1$ and $z[k] < z$ if z is a limit ordinal. Moreover $x + 1$ is the least ordinal above x, and z is the least ordinal above all $z[k]$.

Clearly for every ordinal $0 < z < \varepsilon_0$ there exist unique ordinals $x, y < z$ such that $\mathsf{NF}(x, y)$ and $z = x + \omega^y$. This gives rise to an obvious coding of ordinals $< \varepsilon_0$ by finite binary trees: 0 is coded by the empty tree, and if $\mathsf{NF}(x, y)$ then $x + \omega^y$ is coded by the tree composed by the trees coding x and y respectively. Henceforth we will identify ordinals $< \varepsilon_0$ with their codes. Obviously all functions and relations defined above are primitive recursive with respect to this coding.

2.2 The Principle of Transfinite Induction

We state the principle of *transfinite induction up to* ε_0 in such a form that Gentzen's proof can be smoothly formalized. In this form the principle roughly says that a property holds for all ordinals $< \varepsilon_0$ provided it holds for 0 and is closed under successors and limits of fundamental sequences. In order to make this precise we define for a property P of ordinals $< \varepsilon_0$

$$\mathsf{Prog_S}(P) :\leftrightarrow \forall x.P(x) \to P(x+1),$$
$$\mathsf{Prog_L}(P) :\leftrightarrow \forall x.\mathsf{Lim}(x) \to \forall k\, P(x[k]) \to P(x),$$

where $\mathsf{Lim}(x)$ means that x is a limit ordinal. Transfinite induction up to ε_0 is now expressed by the scheme

$$P(0) \to \mathsf{Prog_S}(P) \to \mathsf{Prog_L}(P) \to \forall x\, P(x). \tag{1}$$

which is equivalent to the more familiar scheme $\forall x\,(\forall y < x\, P(y) \to P(x)) \to \forall x\, P(x)$.

Transfinite induction up to ε_0 must not be confused with induction along the built up of ordinals $< \varepsilon_0$ viewed as binary finite binary trees. We will refer to the latter principle, which is equivalent to ordinary induction on the natural numbers, as *structural induction*.

As an example of how to apply (1) we consider the following recursively defined hierarchy of number theoretic functions, which transfinitely extends the Ackermann hierarchy and is usually called the fast growing hierarchy:

$$F_0(n) = 2^n, \quad F_{x+1}(n) = F_x^{(n)}(n), \quad F_y(n) = F_{y[n]}(n)$$

where y is a limit ordinal. Applying (1) to the predicate

$$P(x) :\leftrightarrow \forall n\, \exists k\, F(x,n,k)$$

where F is a suitably axiomatized predicate such that $F(x,n,k)$ expresses "$F_x(n) = k$" we obtain from (1) $\forall x\, P(x)$, i.e. the totality of all F_x $(x < \varepsilon_0)$, since $P(0)$, $\mathsf{Prog_S}(P)$, and $\mathsf{Prog_L}(P)$ obviously hold.

2.3 Gentzen's Proof in Heyting's Arithmetic

We now give a rather detailed proof of Gentzen's result which can be easily formalized in Heyting Arithmetic, HA. It will allow us to see its constructive content in a rather direct way. Of course, since according to Gödel's and Gentzen's results transfinite induction up to ε_0 is not provable in first order arithmetic, the proof cannot be fully formalized in HA, but some inductions have to be done on the meta level. However this won't be an obstacle for our purposes.

Let P be a property of ordinals $< \varepsilon_0$. In order to prove (1) we assume that P is progressive, i.e. satisfies $P(0)$, $\mathsf{Prog_S}(P)$ and $\mathsf{Prog_L}(P)$. We have to show

$\forall x\, P(x)$. Gentzen's crucial idea to prove this was to introduce the following predicates P_n.

$$P_0(y) :\leftrightarrow P(y)$$
$$P_{n+1}(y) :\leftrightarrow \forall x.\mathsf{NF}(x,y) \to P_n(x) \to P_n(x+\omega^y).$$

Note that for every n the predicate P_n is defined in the language of first order arithmetic. However the binary predicate $Q(n,x) :\leftrightarrow P_n(x)$ is not arithmetical. This is one reason why the following proof is not formalizable in first order arithmetic.

Lemma. $P_n(0)$ for every n.

Proof. We show first $\mathsf{Prog}_{\mathsf{L}}(P_n)$ by induction on n, then $\mathsf{Prog}_{\mathsf{S}}(P_n)$ and finally $P_n(0)$. For all three statements the case $n=0$ holds by assumption.

1. $\mathsf{Prog}_{\mathsf{L}}(P_{n+1})$. Assume $\mathsf{Lim}(y)$ and $\forall k\, P_{n+1}(y[k])$. We have to show $P_{n+1}(y)$. So assume $\mathsf{NF}(x,y)$ and $P_n(x)$ and show $P_n(x+\omega^y)$. Since by induction hypothesis $\mathsf{Prog}_{\mathsf{L}}(P_n)$ holds and $x+\omega^y$ is a limit ordinal, it suffices to show $P_n((x+\omega^y)[k])$ for all k. We have $\mathsf{NF}(x,y[k])$ and $(x+\omega^y)[k] = x+\omega^{y[k]}$. Hence $P_n(x+\omega^{y[k]})$ since by assumption $P_{n+1}(y[k])$.

2. $\mathsf{Prog}_{\mathsf{S}}(P_{n+1})$. Assume $P_{n+1}(y)$. We have to show $P_{n+1}(y+1)$. Assume $\mathsf{NF}(x,y+1)$ and $P_n(x)$. We have to show $P_n(x+\omega^{y+1})$. Since $x+\omega^{y+1}$ is a limit ordinal and by 1. we have $\mathsf{Prog}_{\mathsf{L}}(P_n)$ it suffices to show $P_n(x+\omega^{y+1}[k])$ for all k. We have $(x+\omega^{y+1})[k] = x+\omega^y k$. Therefore, since $\mathsf{NF}(x,y)$ and $P_n(x)$, by k-fold application of $P_{n+1}(y)$ we get $P_n(x+\omega^y k)$.

3. $P_{n+1}(0)$. Assume $P_n(x)$. Then $P_n(x+\omega^0)$ holds since we have $\mathsf{Prog}_{\mathsf{S}}(P_n)$ by 2.

Theorem. $\forall x\, P(x)$.

Proof. We show by structural induction on x that $P_n(x)$ holds for all n. In particular then $P_0(x)$ i.e. $P(x)$ holds. $P_n(0)$ holds for all n by the lemma If $x>0$ then there exist x_0, x_1 such that $\mathsf{NF}(x_0,x_1)$ and $x = x_0+\omega^{x_1}$ (i.e. x_0, x_1 are the immediate subtrees of x). By induction hypothesis we have $P_{n+1}(x_1)$ and $P_n(x_0)$. Hence $P_n(x)$.

Remark. The reader may notice that this proof has a striking similarity with the Tait/Troelstra proof of strong normalization for Gödel's system T of primitive recursive functionals of finite types [18]. The predicate P_n corresponds to the strong computability predicate for terms of type level n.

3 The Constructive Content of Gentzen's Proof

3.1 Formal Proof Terms

In order to extract a program from the proof above we write down the formal proof terms corresponding to the lemma and the theorem in the previous section.

Since we are only interested in the extracted programs we omit some parts of the proof which do not have computational content. In particular *subderivations proving quantifier free formulas like* $\mathsf{NF}(x,y)$, $\mathsf{Lim}(y)$ *etc. are omitted*. We formalize the proof in a natural deduction calculus and write derivations as typed λ-terms. We hope that the terms are understandable although we did not give a definition of the syntax of proof terms. Let $u_Z^{P(0)}$, $u_S^{\mathsf{Prog_S}(P)}$ and $u_L^{\mathsf{Prog_L}(P)}$ be assumption variables representing the general assumptions in 2.3. Formalizing the lemma we define derivations $d_{L,n}^{\mathsf{Prog_L}(P_n)}$ $d_{S,n}^{\mathsf{Prog_S}(P_n)}$, and $d_{Z,n}^{P_n(0)}$. $d_{L,n}$ is defined by recursion on n which means that the inductive argument is done on the meta level.

$$
\begin{aligned}
&d_{L,0} := u_L,\ d_{S,0} := u_S,\ d_{Z,0} := u_Z,\\
&d_{L,n+1} := \lambda y \lambda u^{\forall k\, P_{n+1}(y[k])} \lambda x \lambda v^{P_n(x)}.d_{L,n}(x+\omega^y)(\lambda k.ukxv).\\
&d_{S,n+1} := \lambda y \lambda u^{P_{n+1}(y)} \lambda x \lambda v^{P_n(x)}.\\
&\qquad d_{L,n}(x+\omega^{y+1})\mathsf{Ind}[v, \lambda k \lambda w^{P_n(x+\omega^y k)}.u(x+\omega^y k)w].\\
&d_{Z,n+1} := \lambda x \lambda v^{P_n(x)}.d_{S,n}xv.
\end{aligned}
$$

Note that in the derivation $d_{L,n}$ the operator, $\mathsf{Ind}[.,.]$, corresponds to the induction rule that allows us to conclude $\forall k\, A(k)$ from $A(0)$ and $\forall k(A(k) \to A(k{+}1))$. Here we used induction for the formula $A(k) := P_n(x+\omega^y k)$.

Finally, formalizing the theorem, we define derivation terms $e_{n,x}^{P_n(x)}$ by structural recursion on x (hence also here induction is done on the meta level).

$$e_{n,0} := d_{Z,n}, \quad e_{n,x+\omega^y} := e_{n+1,y}e_{n,x}.$$

3.2 Program Extraction

In order to obtain a program from the derivation terms we apply a formalized version of Kreisel's modified realizability interpretation [8]. This interpretation maps each derivation d^A to a term $\mathsf{ep}(d)^{\tau(A)}$ in Gödel's system T [6] *realizing* the formula A. $\tau(A)$, the type of the 'potential realizers' of A, is a finite type built from ground types like the type $\mathbb{N}$ of natural numbers by cartesian products, $\times$, and function spaces, $\to$. For details see [18]. For example we have

$$
\begin{aligned}
&\tau(P_{n+1}) = \varepsilon_0 \to \tau(P_n) \to \tau(P_n),\\
&\tau(\mathsf{Prog_S}(P_n)) = \varepsilon_0 \to \tau(P_n) \to \tau(P_n),\\
&\tau(\mathsf{Prog_L}(P_n)) = \varepsilon_0 \to (\mathbb{N} \to \tau(P_n)) \to \tau(P_n).
\end{aligned}
$$

Since the predicate P has not been specified yet, we neither know the type $\tau(P) = \tau(P_0) := \tau(P(x))$ ($\tau(P(x))$ does not depend on x) nor do we know what it means that a term $t^{\tau(P)}$ realizes the formula $P(x)$. For a moment we regard these entities as parameters (which we can instantiate as we wish later on) and extract 'abstract' programs depending on free variables u_Z, u_S, and u_L, standing for unknown realizers of $P(0)$, $\mathsf{Prog_S}(P)$, and $\mathsf{Prog_L}(P)$, respectively. In this abstract form the extracted programs look very similar to the derivations. Just the types have changed, and induction is replaced by primitive recursion.

$$\begin{aligned}
&\mathsf{ep}(d_{L,0})^{\tau(\mathsf{Prog_L}(P_n))} := u_L,\\
&\mathsf{ep}(d_{S,0})^{\tau(\mathsf{Prog_S}(P_n))} := u_S,\\
&\mathsf{ep}(d_{Z,0})^{\tau(P_n)} := u_Z,\\
&\mathsf{ep}(d_{L,n+1})^{\tau(\mathsf{Prog_L}(P_{n+1}))} := \lambda y\lambda u^{\mathbb{N}\to\tau(P_{n+1})}\lambda x\lambda v^{\tau(P_n)}.\\
&\qquad \mathsf{ep}(d_{L,n})(x+\omega^y)(\lambda k.ukxv).\\
&\mathsf{ep}(d_{S,n+1})^{\tau(\mathsf{Prog_S}(P_{n+1}))} := \lambda y\lambda u^{\tau(P_{n+1})}\lambda x\lambda v^{\tau(P_n)}.\\
&\qquad \mathsf{ep}(d_{L,n})(x+\omega^{y+1})\mathsf{Rec}[v,\lambda k\lambda w^{\tau(P_n)}.u(x+\omega^y k)w].\\
&\mathsf{ep}(d_{Z,n+1})^{\tau(P_{n+1})} := \lambda x\lambda v^{\tau(P_n)}.\mathsf{ep}(d_{S,n})xv.
\end{aligned}$$

Finally

$$\mathsf{ep}(e_{n,0}) := d_{Z,n}, \quad \mathsf{ep}(e_{n+1,x+\omega^y}) := \mathsf{ep}(e_{n+1,y})\mathsf{ep}(e_{n,x}).$$

Recall that $e_{0,x}$ derives $P(x)$, and hence the program $\mathsf{ep}(e_{0,x})$ realizes $P(x)$.

4 Primitive Recursive Programs for Ordinal Recursive Functions

Consider a family of objects $(H_x)_{x<\varepsilon_0}$ defined by transfinite recursion on x.

$$H_0 = \Phi_0, \quad H_{x+1} = \Phi_{\mathsf{S}}(x, H_x), \quad H_y = \Phi_{\mathsf{L}}(y, \lambda k H_{y[k]}) \quad (\text{if } \mathsf{Lim}(y)),$$

where the (common) type τ of the H_x is arbitrary and $\Phi_0 : \tau$, $\Phi_{\mathsf{S}} : \varepsilon_0 \to \tau \to \tau$, $\Phi_{\mathsf{L}} : \varepsilon_0 \to (\mathbb{N} \to \tau) \to \tau$ are some unspecified functionals given ahead. If, for example, we set $\tau := \mathbb{N} \to \mathbb{N}$ and

$$\Phi_0(k) = 2^k\ , \quad \Phi_{\mathsf{S}}(x,f)(k) = f^{(k)}(k)\ , \quad \Phi_{\mathsf{L}}(y,g)(k) = g(k,k)$$

then $(H_x)_{x<\varepsilon_0}$ will be the fast growing hierarchy, defined in 2.2.

In order to obtain via program extraction, as sketched in the previous section, algorithms computing the H_x, we extend our formal system HA by a new predicate symbol G_H. The intended meaning of $\mathsf{G}_H(x,a)$ is '$H_x = a$'. G_H could be defined inductively, namely as the least predicate satisfying

$$\begin{aligned}
&\mathsf{G}_H(0,\Phi_0)\\
&\mathsf{G}_H(x,a) \to \mathsf{G}_H(x+1,\Phi_{\mathsf{S}}(a))\\
&\mathsf{Lim}(y) \to \forall k\mathsf{G}_H(y[k],f(k)) \to \mathsf{G}_H(y,\Phi_{\mathsf{L}}(y,f))
\end{aligned}$$

We will add these three formulas as axioms to our system (without, however, stating the minimality of G_H). Since these new axioms do not contain existential quantifiers or disjunctions the program extraction method is not affected by this extension.

We now use transfinite induction on x to prove the existence of H_x, i.e. we set

$$P(x) :\leftrightarrow \exists a \mathsf{G}_H(x,a).$$

Note that now $\tau(P) = \tau$, and $a\,\mathbf{mr}\,P(x) \leftrightarrow \mathsf{G}_H(x,a)$. Furthermore

$$\begin{aligned}
&u_S\,\mathbf{mr}\,\mathsf{Prog_S}(P) \leftrightarrow [\forall x\,.\,\mathsf{G}_H(x+1,a) \to \mathsf{G}_H(x+1,u_S(a))],\\
&u_L\,\mathbf{mr}\,\mathsf{Prog_L}(P) \leftrightarrow [\forall y\forall f\,.\,\mathsf{Lim}(y) \to \forall k\mathsf{G}_H(y[k],f(k)) \to \mathsf{G}_H(y,u_L(y,f))]
\end{aligned}$$

Hence we may set $u_Z := \Phi_0$, $u_S := \Phi_{\mathsf{S}}$, $u_L := \Phi_{\mathsf{L}}$ and obtain a program $\mathsf{ep}(e_{0,x})^\tau$ realizing $P(x)$, i.e. $\mathsf{G}_H(x, \mathsf{ep}(e_{0,x}))$, which means that $\mathsf{ep}(e_{0,x}) = H_x$. Therefore we have obtained a higher type primitive recursive algorithm for H_x (relative to $\Phi_0, \Phi_{\mathsf{S}}, \Phi_{\mathsf{L}}$).

If we are even more specific and require in addition that Φ_{S} and Φ_{L} do not depend on their first argument – as it is the case for the fast growing hierarchy – then, by normalization, all ordinal arguments disappear, and we obtain simpler programs. Hence let us now consider the more restricted schema

$$H_0 = \Phi_0, \quad H_{x+1} = \Phi_{\mathsf{S}}(H_x), \quad H_y = \Phi_{\mathsf{L}}(\lambda k H_{y[k]}), \quad (\text{if } \mathsf{Lim}(y)).$$

where $\Phi_0 \colon \tau$, $\Phi_{\mathsf{S}} \colon \tau \to \tau$, and $\Phi_{\mathsf{L}} \colon (\mathbb{N} \to \tau) \to \tau$. Setting $\tau_0 = \mathbb{N} \to \mathbb{N}$, $\tau_{n+1} = \tau_n \to \tau_n$ we obtain the following programs $r_n \colon (\mathbb{N} \to \tau_n) \to \tau_n$, $t_n \colon \tau_n$, $t_{n,x} \colon \tau_n$ corresponding to $\mathsf{ep}(d_{L,n}), \mathsf{ep}(d_{Z,n}), \mathsf{ep}(e_{n,x})$ respectively (we use the variables $u_n \colon \mathbb{N} \to \tau_n$ and $v_n \colon \tau_n$).

$$\begin{aligned}
&r_0 = \Phi_L,\\
&r_{n+1} = \lambda u_{n+1} \lambda v_n . r_n(\lambda k . u_{n+1} k v_n),\\
&t_0 = \Phi_0,\\
&t_1 = \Phi_{\mathsf{S}},\\
&t_{n+2} = \lambda v_{n+1} \lambda v_n . r_n(\lambda k \,.\, v_{n+1}^{(k)} v_n),\\
&t_{n,0} := t_n,\\
&t_{n,x+\omega^y} := t_{n+1,y} t_{n,x}
\end{aligned}$$

with the property that $\mathsf{G}_H(x, t_{0,x})$ is provable, that is, $t_{0,x}$ defines H_x.

The terms t_{n+2} can β-equivalently be written in the following, slightly less formal, but more readable way.

$$t_{n+2} v_{n+1} v_n \dots v_0 = \Phi_{\mathsf{L}}(\lambda k \,.\, v_{n+1}^{(k)} v_n \dots v_0).$$

Note that $t_{0,x}$ realizes $P(x)$, which, by definition of P means that $t_{0,x}$ is a program computing H_x.

4.1 The Fast Growing Hierarchy

Recall that in order to obtain the fast growing hierarchy as an instance of the family $(H_x)_{x<\varepsilon_0}$ we have to set $\tau := \mathbb{N} \to \mathbb{N}$ and

$$\Phi_0(k) = 2^k \ , \quad \Phi_{\mathsf{S}}(x, f)(k) = f^{(k)}(k) \ , \quad \Phi_{\mathsf{L}}(y, g)(k) = g(k, k)$$

Now the terms t_n become

$$\begin{aligned}
&t_0 k = 2^k,\\
&t_{n+1} v_n \dots v_0 k = v_n^{(k)} v_{n-1} \dots v_0 k.
\end{aligned}$$

Hence we see that t_n is precisely F_0^{n+1} in Schwichtenberg's notation [13].

4.2 Wellfounded Trees of Heights $< \varepsilon_0$

Finally, we use our extracted program to obtain higher type primitive recursive definitions of wellfounded trees of all heights $< \varepsilon_0$. To this end we add to our formal system a sort tree and constants

$$0\colon \mathsf{tree}, \quad \mathsf{succ}\colon \mathsf{tree} \to \mathsf{tree}\ , \quad \mathsf{lim}\colon (\mathbb{N} \to \mathsf{tree}) \to \mathsf{tree}$$

acting as constructors for countable wellfounded trees. Our family $(H_x)_{x<\varepsilon_0}$ now simply interprets each ordinal notation $x < \varepsilon_0$ as a countable wellfounded tree in the obvious way:

$$H_0 = 0, \quad H_{x+1} = \mathsf{succ}(H_x), \quad H_y = \mathsf{lim}(\lambda k H_{y[k]}), \quad (\text{if } \mathsf{Lim}(y))$$

Hence we have $\tau = \mathsf{tree}$, $\Phi_0 = 0$, $\Phi_{\mathsf{S}} = \mathsf{succ}$, $\Phi_{\mathsf{L}} = \mathsf{lim}$ and we get

$$\begin{aligned}
&t_0 = 0,\\
&t_1 v_0 = \mathsf{succ}(v_0),\\
&t_{n+2} v_{n+1} v_n \dots v_0 = \mathsf{lim}(\lambda k\,.\, v_{n+1}^{(k)} v_n \dots v_0).\\
&t_{n,0} := t_n,\\
&t_{n,x+\omega^y} := t_{n+1,y} t_{n,x}.
\end{aligned}$$

with the property that $t_{0,x}$ defines H_x.

For example the ordinals ω_n ($\omega_0 := \mathsf{succ}(0)$, $\omega_{n+1} := w^{\omega_n}$) which approximate ε_0, are for $n > 0$ mapped to the trees

$$H_{\omega_n} = \mathsf{lim}(\lambda k\,.\, t_n^{(k)} t_{n-1} \dots t_0)$$

This is more or less the same iterative construction of ω_n as given by Aczel in [1], definition (1.8) (except that Aczel works with set-theoretic ordinals instead of tree ordinals).

Remark. Conversely every closed term of type tree denotes a wellfounded tree of height $< \varepsilon_0$. This is so, because every such tree is provably (in HA) wellfounded and hence, again according to Gentzen [5], has height $< \varepsilon_0$.

The concept of a provably wellfounded tree can be made precise as follows. Let P be a predicate on trees. Set

$$\begin{aligned}
\mathsf{Prog}(P) :\leftrightarrow\ & P(0) \wedge\\
& \forall a^{\mathsf{tree}}(P(a) \to P(\mathsf{succ}(a))) \wedge\\
& \forall f^{\mathbb{N}\to\mathsf{tree}}(\forall k P(f(k)) \to P(\mathsf{lim}(f)))
\end{aligned}$$

$$\mathsf{TI}(P, a^{\mathsf{tree}}) :\leftrightarrow \mathsf{Prog}(P) \to P(a)$$

For a closed term t of type tree we say that (the tree defined by) t is provably wellfounded if $\mathsf{TI}(P, t)$ is provable (in HA) for every predicate P.

In order to see that every closed term t^{tree} is indeed provably wellfounded take a predicate P and define predicates P_ρ on ρ as follows:

$$\begin{array}{lll} P_{\mathbb{N}}(k) & :\leftrightarrow & 0 = 0 \\ P_{\mathsf{tree}}(a^{\mathsf{tree}}) & :\leftrightarrow & P(a^{\mathsf{tree}}) \\ P_{\rho\to\sigma}(f^{\rho\to\sigma}) & :\leftrightarrow & \forall x^{\rho}(P_{\rho}(x) \to P_{\sigma}(f(x))) \end{array}$$

Now one proves for terms r^{ρ} with free variables $x_i^{\sigma_i}$ the formulas

$$\mathsf{Prog}(P) \land \bigwedge_i P_{\sigma_i}(x_i) \to P_{\rho}(r)$$

by term induction on r. The hypothesis $\mathsf{Prog}(P)$ is used in the cases $r = 0, \mathsf{succ}, \mathsf{lim}$.

5 Concluding Remarks

In this paper we analysed the constructive content of Gentzen's proof of transfinite induction up to ε_0 and obtained simple higher type primitive recursive definitions of the fast growing hierarchy and trees corresponding to ordinals $< \varepsilon_0$.

I compared the efficiency of the ordinal recursive programs and the higher type programs, not for the fast growing hierarchy, but for the Hardy hierarchy

$$H_0(n) = n, \quad H_{x+1}(n) = H_x(n+1), \quad H_y(n) = F_{y[n]}(n)$$

(because for the Hardy hierarchy it turned out to be easier to find interesting inputs where the programs terminate in reasonable time). The result was that the extracted higher type program was about 4 times faster than the ordinal recursive program above [1]. A more significant speed up has been observed in an implementation of a higher type program for normalizing simply typed lambda terms. This program has been shown to be the constructive content of the Tait/Troelstra strong normalization proof using logical predicates whose similarity with Gentzen's proof we already mentioned [2].

It would be interesting to see whether a similarly smooth computational analysis of proofs of transfinite induction up to higher ordinals such as the Schütte-Feferman ordinal, Γ_0, or the Bachmann-Howard ordinal, $\phi_{\varepsilon_{\Omega+1}}$, [11], is possible. A nice formal proof of transfinite induction up to Γ_0 is given in [7]. It should be possible to extract from this proof higher type definitions of ordinals $< \Gamma_0$ using some kind of predicative polymorphism.

Acknowledgments. I am grateful to Peter Hancock for stimulating discussions and important pointers to the literature.

[1] `http://www-compsci.swan.ac.uk/~csulrich/recent-papers.html`

References

1. P. Aczel. Describing ordinals by functionals of transfinite type. *Journal of Symbolic Logic* **37**(1), 35–47, 1972.
2. U. Berger. Program extraction from normalization proofs. *Typed Lambda Calculi and Applications (TLCA'93)*, LNCS 664, 91–106, 1993.
3. N. Danner. Ordinals and ordinal functions representable in the simply typed lambda calculus. *Annals of Pure and Applied Logic* **97**, 179–201, 1999.
4. S. Feferman. Hereditarily replete functionals over the ordinals. *Intuitionism and Proof theory*, J. Myhill, editor, North-Holland, 289–301, 1970.
5. G. Gentzen. Beweisbarkeit und Unbeweisbarkeit von Anfangsfällen der transfiniten Induktion in der reinen Zahlentheorie. *Mathematische Annalen* **119**, 140–161, 1943.
6. K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* **12**, 280–287, 1958.
7. P. Hancock. Ordinals and Interactive Programs. Unpublished, 2001.
8. G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. *Constructivity in Mathematics*, A. Heyting, editor, North–Holland, 101–128, 1959.
9. W. Neumer. Zur Konstruktion von Ordnungszahlen. *Mathematische Zeitschrift*. I, vol. **58**, 319–413, 1953; II, vol. **59**, 434–454, 1954; III, vol. **60**, 1–16, 1954; IV, vol. **61**, 47–69, 1954; V, vol. **64**, 435–456, 1956.
10. W. Neumer. Algorithmen für Ordnungzahlen und Normalfunktionen. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*. I, vol. **3**, 108–150, 1957; II, vol. **6**, 1–65, 1960.
11. W. Pohlers. Proof Theory. SLNM 1407, 1989.
12. H. Schwichtenberg. Eine Klassifikation der ε_0–rekursiven Funktionen. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* **17**, 61–74, 1971.
13. H. Schwichtenberg. Einige Anwendungen von unendlichen Termen und Wertfunktionalen. Mathematisches Institut der Universität Münster, Habilitationsschrift, 1973.
14. H. Schwichtenberg. Elimination of higher type levels in definitions of primitive recursive functionals by means of transfinite recursion. *Logic Colloquium 73* North–Holland, 279–303, 1975.
15. H. Schwichtenberg. Classifying Recursive Functions. In: E. Griffor, editor, *Handbook of Recursion Theory*, 533-586, North-Holland, 1999.
16. J. Terlouw. On definition trees of ordinal recursive functionals: reduction of the recursion orders by means of type level raising. *Journal of Symbolic Logic* **47**, 395–402, 1982.
17. J. Terlouw. Reduction of higher type levels by means of an ordinal analysis of finite terms. *Annals of Pure and Applied Logic* **28**, 73–102, 1985.
18. A. S. Troelstra. Metamathematical Investigations of Intuitionistic Arithmetic and Analysis. SLNM 344, 1973.
19. A. S. Troelstra and Dirk van Dalen. *Constructivism in Mathematics. An Introduction.* Volumes 121 and 123 of *Studies in Logic and the Foundations of Mathematics*, North–Holland, 1988.

Coherent Bicartesian and Sesquicartesian Categories

Kosta Došen and Zoran Petrić

Mathematical Institute, SANU, Knez Mihailova 35, p.f. 367,
11001 Belgrade, Yugoslavia,
phone: 381 11 630 170, fax: 381 11 186 105,
{kosta,zpetric}@mi.sanu.ac.yu

Abstract. Sesquicartesian categories are categories with nonempty finite products and arbitrary finite sums, including the empty sum. Coherence is here demonstrated for sesquicartesian categories in which the first and the second projection from the product of the initial object with itself are the same. (Every bicartesian closed category, and, in particular, the category **Set**, is such a category.) This coherence amounts to the existence of a faithful functor from categories of this sort freely generated by sets of objects to the category of relations on finite ordinals. Coherence also holds for bicartesian categories where, in addition to this equality for projections, we have that the first and the second injection to the sum of the terminal object with itself are the same. These coherences yield a very easy decision procedure for equality of arrows.

Keywords: categorial proof theory, conjunction and disjunction, decidability of equality of deductions

Mathematics Subject Classification (2000): 18A30, 18A15, 03G30, 03F05

1 Introduction

The connectives of conjunction and disjunction in classical and intuitionistic logic make a structure corresponding to a distributive lattice. Among nonclassical logics, and, in particular, among substructural logics, one finds also conjunction and disjunction that make a structure corresponding to a free lattice, which is not distributive.

In proof theory, however, we are not concerned only with a consequence *relation*, which in the case of conjunction and disjunction gives rise to a lattice order, but we may distinguish certain deductions with the same premise and the same conclusion. For example, there are two different deductions from $A \wedge A$ to A, one corresponding to the first projection and the other to the second projection, and two different deductions from A to $A \vee A$, one corresponding to the first injection and the other to the second injection.

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 78–92, 2001.

If we identify deductions guided by normalization, or cut elimination, we will indeed distinguish the members in each of these two pairs of deductions, and we will end up with natural sorts of categories, whose arrows stand for deductions. Instead of nondistributive lattices, we obtain then categories with binary products and sums (i.e. coproducts), where the product $\times$ corresponds to conjunction and the sum $+$ to disjunction. If, in order to have all finite products and sums, we add also the empty product, i.e. the terminal object, which corresponds to the constant true proposition, and the empty sum, i.e. the initial object, which corresponds to the constant absurd proposition, we obtain *bicartesian* categories, namely categories that are at the same time *cartesian*, namely, with all finite products, and *cocartesian*, with all finite sums. Bicartesian categories need not be distributive.

One may then enquire how useful is the representation of deductions involving conjunction and disjunction in bicartesian categories for determining which deductions are equal and which are not. A drawback is that here cut-free, i.e. composition-free, form is not unique. For example, for $f : A \to C$, $g : A \to D$, $h : B \to C$ and $j : B \to D$ we have the following two composition-free arrow terms

$$\frac{\langle f, g\rangle : A \to C \times D \qquad \langle h, j\rangle : B \to C \times D}{[\langle f, g\rangle, \langle h, j\rangle] : A + B \to C \times D}$$

$$\frac{[f, h] : A + B \to C \qquad [g, j] : A + B \to D}{\langle [f, h], [g, j]\rangle : A + B \to C \times D}$$

which designate the same arrow, but it is not clear which one of them is to be considered in normal form. The same problem arises also for distributive bicartesian categories, i.e. for classical and intuitionistic conjunction and disjunction. (This problem is related to questions that arise in natural deduction with Prawitz's reductions tied to disjunction elimination, sometimes called "commuting conversions".)

Cut elimination may then be supplemented with a *coherence* result. The notion of coherence is understood here in the following sense. We say that coherence holds for some sort of category iff for a category of this sort freely generated by a set of objects there is a faithful functor from it to a *graphical* category, whose arrows are sets of links between the generating objects. These links can be drawn, and that is why we call our category "graphical". Intuitively, these links connect the objects that must remain the same after "generalizing" the arrows. We shall define precisely below the graphical category we shall use for our coherence results, and the intuitive notion of generalizing (which originates in [6] and [7]) will become clear. This category will be the category of relations on finite ordinals. In other cases, however, we might have a different, but similar, category. It is desirable that the graphical category be such that through coherence we obtain a decision procedure for equality of arrows, and in good cases, such as those investigated in [3] and here, this happens indeed.

Although this understanding of coherence need not be the most standard one, the paradigmatic results on coherence of [10] and [5] can be understood as faithfulness results of the type mentioned above, and this is how we understand

coherence here. We refer to [3], and papers cited therein, for further background motivation on coherence.

This paper is a companion to [3], where it was shown that coherence holds for categories with binary, i.e. nonempty finite, products and sums, but without the terminal and the initial object, and without distribution. One obtains thereby a very easy decision procedure for equality of arrows. With the help of this coherence, it was also demonstrated that the categories in question are maximal, in the sense that in any such category that is not a preorder all the equations between arrows inherited from the free categories with binary products and sums are the same. An analogous maximality can be established for cartesian categories (see [1]) and cartesian closed categories (see [11] and [2]).

In this paper we shall be concerned with categories with nonempty finite products and arbitrary finite sums, including the empty sum, i.e. the initial object. We call such categories *sesquicartesian*. As a matter of fact, *sesquicocartesian* would be a more appropriate label for these categories, since they are *cocartesian* categories—namely, categories with arbitrary finite sums—to which we add a part of the cartesian structure—namely, nonempty finite products. *Sesquicartesian* is more appropriate as a label for cartesian categories—namely, categories with arbitrary finite products—to which we add nonempty finite sums, which are a part of the cocartesian structure. However, sesquicocartesian and sesquicartesian categories so understood are dual to each other, and in a context where both types of categories are not considered, there is no necessity to burden oneself with the distinction, and make a strange new name even stranger. So we call here *sesquicartesian* categories with nonempty finite products and arbitrary finite sums.

We shall show that coherence holds for sesquicartesian categories in which the first and the second projection arrow from the product of the initial object with itself are equal. Every bicartesian closed category, and, in particular, the sesquicartesian category **Set**, is such a category. It is not true, however, that sesquicartesian categories that satisfy this condition for projections are maximal, in the sense in which cartesian categories, cartesian closed categories and categories with binary products and sums are maximal.

Bicartesian categories are also not maximal. Coherence does not hold for bicartesian categories in general, but we shall see that it holds for bicartesian categories where besides the equality of the projection arrows mentioned above we also have that the first and the second injection arrow to the sum of the terminal object with itself are equal. The bicartesian category **Set** is not such a category, but a few natural examples of such categories may be found in Section 7. Such categories are also not maximal.

As in [3], our coherences for the special sesquicartesian and special bicartesian categories yield a very easy decision procedure for equality of arrows in the categories of this kind freely generated by sets of objects. Without maximality, however, the application of this decision procedure to an arbitrary category of the appropriate kind is limited. We can use this decision procedure only to show

that two arrows inherited from the free category are equal, and we cannot use it to show that they are not equal.

We said that this paper is a companion to [3], but except for further motivation, for which we refer to [3], we shall strive to make the present paper self-contained. So we have included here definitions and proofs that are just versions of material that may be found also in [3], but which for the sake of clarity it is better to adapt to the new context.

2 Free Coherent Bicartesian and Sesquicartesian Categories

The propositional language $\mathcal{P}$ is generated from a set of *propositional letters* $\mathcal{L}$ with the nullary connectives, i.e. propositional constants, I and O, and the binary connectives $\times$ and $+$. The fragment $\mathcal{P}_{\times,+,\mathrm{O}}$ of $\mathcal{P}$ is obtained by omitting all formulae that contain I. For the propositional letters of $\mathcal{P}$, i.e. for the members of $\mathcal{L}$, we use the schematic letters $p, q, \ldots$, and for the formulae of $\mathcal{P}$, or of its fragments, we use the schematic letters $A, B, \ldots, A_1, \ldots$ The propositional letters and the constants I and O are *atomic* formulae. The formulae of $\mathcal{P}$ in which no propositional letter occurs will be called *constant objects*.

Next we define inductively the *terms* that will stand for the arrows of the free *coherent bicartesian category* $\mathcal{B}$ generated by $\mathcal{L}$. Every term has a *type*, which is a pair (A, B) of formulae of $\mathcal{P}$. That a term f is of type (A, B) is written $f : A \to B$. The *atomic* terms are for every A and every B of $\mathcal{P}$

$$\mathbf{1}_A : A \to A,$$

$$\begin{array}{ll} k_A : A \to \mathrm{I}, & l_A : \mathrm{O} \to A, \\ k^1_{A,B} : A \times B \to A, & l^1_{A,B} : A \to A + B, \\ k^2_{A,B} : A \times B \to B, & l^2_{A,B} : B \to A + B, \\ w_A : A \to A \times A, & m_A : A + A \to A. \end{array}$$

The terms $\mathbf{1}_A$ are called *identities*. The other terms of $\mathcal{B}$ are generated with the following operations on terms, which we present by rules so that from the terms in the premises we obtain the terms in the conclusion:

$$\frac{f : A \to B \qquad g : B \to C}{g \circ f : A \to C}$$

$$\frac{f : A \to B \qquad g : C \to D}{f \times g : A \times C \to B \times D} \qquad \frac{f : A \to B \qquad g : C \to D}{f + g : A + C \to B + D}$$

We use $f, g, \ldots, f_1, \ldots$ as schematic letters for terms of $\mathcal{B}$.

The category $\mathcal{B}$ has as objects the formulae of $\mathcal{P}$ and as arrows equivalence classes of terms so that the following equations are satisfied for $i \in \{1, 2\}$:

$$\begin{array}{ll} (cat\ 1) & \mathbf{1}_B \circ f = f \circ \mathbf{1}_A = f, \\ (cat\ 2) & h \circ (g \circ f) = (h \circ g) \circ f, \end{array}$$

$$
\begin{array}{ll}
(\times 1) & \mathbf{1}_A \times \mathbf{1}_B = \mathbf{1}_{A\times B},\\
(\times 2) & (g_1 \circ g_2) \times (f_1 \circ f_2) = (g_1 \times f_1) \circ (g_2 \times f_2),\\
(k^i) & k^i_{B_1,B_2} \circ (f_1 \times f_2) = f_i \circ k^i_{A_1,A_2},\\
(w) & w_B \circ f = (f \times f) \circ w_A,\\
(kw1) & k^i_{A,A} \circ w_A = \mathbf{1}_A,\\
(kw2) & (k^1_{A,B} \times k^2_{A,B}) \circ w_{A\times B} = \mathbf{1}_{A\times B},\\
(k) & \text{for } f : A \to \mathrm{I},\ f = k_A,\\
(k\mathrm{O}) & k^1_{\mathrm{O},\mathrm{O}} = k^2_{\mathrm{O},\mathrm{O}},
\end{array}
$$

$$
\begin{array}{ll}
(+1) & \mathbf{1}_A + \mathbf{1}_B = \mathbf{1}_{A+B},\\
(+2) & (g_1 \circ g_2) + (f_1 \circ f_2) = (g_1 + f_1) \circ (g_2 + f_2),\\
(l^i) & (f_1 + f_2) \circ l^i_{A_1,A_2} = l^i_{B_1,B_2} \circ f_i,\\
(m) & f \circ m_A = m_B \circ (f + f),\\
(lm1) & m_A \circ l^i_{A,A} = \mathbf{1}_A,\\
(lm2) & m_{A+B} \circ (l^1_{A,B} + l^2_{A,B}) = \mathbf{1}_{A+B},\\
(l) & \text{for } f : \mathrm{O} \to A,\ f = l_A,\\
(l\mathrm{I}) & l^1_{\mathrm{I},\mathrm{I}} = l^2_{\mathrm{I},\mathrm{I}}.
\end{array}
$$

If we omit the equations $(k\mathrm{O})$ and $(l\mathrm{I})$, we obtain the free bicartesian category generated by $\mathcal{L}$.

The free *coherent sesquicartesian category* $\mathcal{S}$ generated by $\mathcal{L}$ has as objects the formulae of $\mathcal{P}_{\times,+,\mathrm{O}}$. In that case, terms in which k_A occurs are absent, and the equations (k) and $(l\mathrm{I})$ are missing. The remaining terms and equations are as in $\mathcal{B}$. If from the equations of $\mathcal{S}$ we omit the equation $(k\mathrm{O})$, we obtain the free sesquicartesian category generated by $\mathcal{L}$.

We shall call terms of $\mathcal{B}$ in which the letters l and m don't occur *K-terms*. (That means there are no subterms of K-terms of the form l_A, $l^i_{A,B}$ and m_A.) Terms of $\mathcal{B}$ in which the letters k and w don't occur will be called L-terms.

3 Cut Elimination

For inductive proofs on the length of objects or terms, it is useful to be able to name the arrows of a category by terms that contain no composition. In this section we shall prove for $\mathcal{B}$ and $\mathcal{S}$ a theorem analogous to cut elimination theorems of logic, which stem from Gentzen [4]. This theorem says that every term of these categories is equal to a term of a special sort, which in an appropriate language would be translated by a composition-free term. We shall call terms of this special sort *cut-free Gentzen terms*.

We define first the following operations on terms of $\mathcal{B}$, which we call *Gentzen operations*:

$$
\begin{array}{ll}
K^1_B f =_{def.} f \circ k^1_{A,B}, & L^1_B f =_{def.} l^1_{A,B} \circ f,\\
K^2_A f =_{def.} f \circ k^2_{A,B}, & L^2_A f =_{def.} l^2_{A,B} \circ f,
\end{array}
$$

$$
\langle f, g\rangle =_{def.} (f \times g) \circ w_C,\ [f, g] =_{def.} m_C \circ (f + g).
$$

Starting from the identities and the terms k_A and l_A, for every A of $\mathcal{P}$, and closing under the Gentzen operations, we obtain the set of *cut-free Gentzen terms* of $\mathcal{B}$. The *Gentzen terms of* $\mathcal{B}$ are obtained by closing cut-free Gentzen terms under composition.

It is easy to show that every term of $\mathcal{B}$ is equal in $\mathcal{B}$ to a Gentzen term, since we have the following equations:

$$\begin{array}{ll} k^1_{A,B} = K^1_B \mathbf{1}_A, & l^1_{A,B} = L^1_B \mathbf{1}_A, \\ k^2_{A,B} = K^2_A \mathbf{1}_B, & l^2_{A,B} = L^2_A \mathbf{1}_B, \\ w_A = \langle \mathbf{1}_A, \mathbf{1}_A \rangle, & m_A = [\mathbf{1}_A, \mathbf{1}_A], \\ f \times g = \langle K^1_C f, K^2_A g \rangle, & f + g = [L^1_D f, L^2_B g]. \end{array}$$

We need the following equations of $\mathcal{B}$:

$$\begin{array}{llll} (K1) & g \circ K^i_A f = K^i_A (g \circ f), & (L1) & L^i_A g \circ f = L^i_A (g \circ f), \\ (K2) & K^i_A g \circ \langle f_1, f_2 \rangle = g \circ f_i, & (L2) & [g_1, g_2] \circ L^i_A f = g_i \circ f, \\ (K3) & \langle g_1, g_2 \rangle \circ f = \langle g_1 \circ f, g_2 \circ f \rangle, & (L3) & g \circ [f_1, f_2] = [g \circ f_1, g \circ f_2] \end{array}$$

in order to prove the following theorem for $\mathcal{B}$.

CUT ELIMINATION. *Every term is equal to a cut-free Gentzen term.*

Proof. We first find for an arbitrary term of $\mathcal{B}$ a Gentzen term h equal to it. Let the *degree* of a Gentzen term be the number of occurrences of Gentzen operations in this term. Take a subterm $g \circ f$ of h such that both f and g are cut-free Gentzen terms. We call such a term a *topmost cut*. We show that $g \circ f$ is either equal to a cut-free Gentzen term, or it is equal to a Gentzen term whose topmost cuts are of strictly smaller degree than the degree of $g \circ f$. The possibility of eliminating the main compositions of topmost cuts, and hence of finding for h a cut-free Gentzen term, follows by induction on degree.

The cases where f or g is $\mathbf{1}_A$, or f is l_A, or g is k_A, are taken care of by (*cat* 1), (l) and (k). The cases where f is $K^i_A f'$ or g is $L^i_A g'$ are taken care of by ($K1$) and ($L1$). And the cases where f is $[f_1, f_2]$ or g is $\langle g_1, g_2 \rangle$ are taken care of by ($L3$) and ($K3$).

The following cases remain. If f is k_A, then g is of a form covered by cases we dealt with above.

If f is $\langle f_1, f_2 \rangle$, then g is either of a form covered by cases above, or g is $K^i_A g'$, in which case we apply ($K2$).

If f is $L^i_A f'$, then g is either of a form covered by cases above, or g is $[g_1, g_2]$, in which case we apply ($L2$). This covers all possible cases.

This proof, with the cases involving k_A omitted, suffices to demonstrate Cut Elimination for $\mathcal{S}$.

Let the *cut-free Gentzen K-terms* of $\mathcal{B}$ be obtained from the identities and the terms k_A by closing under $+$ and the Gentzen operations K^i and $\langle , \rangle$. The *Gentzen K-terms* of $\mathcal{B}$ are obtained by closing the cut-free Gentzen K-terms under composition. Let, dually, the *cut-free Gentzen L-terms* of $\mathcal{B}$ be obtained from

the identities and the terms l_A by closing under $\times$ and the Gentzen operations L^i and $[,]$. The *Gentzen L-terms* of $\mathcal{B}$ are obtained by closing the cut-free Gentzen L-terms under composition.

Then we can prove the following version of Cut Elimination for the K-terms and the L-terms of $\mathcal{B}$.

CUT ELIMINATION FOR K-TERMS AND L-TERMS. *Every K-term is equal to a cut-free Gentzen K-term, and every L-term is equal to a cut-free Gentzen L-term.*

Proof. It is easy to see that every K-term is equal in $\mathcal{B}$ to a Gentzen K-term. That this Gentzen K-term is equal to a cut-free Gentzen K-term is demonstrated as in the proof of Cut Elimination above by induction on the degree of topmost cuts. We have to consider the following additional cases.

If f is k_A or $\langle f_1, f_2\rangle$, then g cannot be of the form $g_1 + g_2$. If f is $f_1 + f_2$, and g is not of a form already covered by cases in the proof above, then g is of the form $g_1 + g_2$, in which case we apply $(+2)$. This covers all possible cases.

Cut Elimination for L-terms follows by duality.

Let the *cut-free Gentzen K-terms* of $\mathcal{S}$ be terms of $\mathcal{S}$ obtained from the identities by closing under $+$ and the Gentzen operations K^i and $\langle,\rangle$. The *Gentzen K-terms* of $\mathcal{S}$ are obtained by closing the cut-free Gentzen K-terms of $\mathcal{S}$ under composition. Let the *cut-free Gentzen L-terms* of $\mathcal{S}$ be terms of $\mathcal{S}$ obtained from the identities and the terms l_A by closing under $\times$ and the Gentzen operations L^i and $[,]$. The *Gentzen L-terms* of $\mathcal{S}$ are obtained by closing the cut-free Gentzen L-terms of $\mathcal{S}$ under composition.

Then we can establish Cut Elimination for K-terms and L-terms of $\mathcal{S}$, where equality in $\mathcal{B}$ is replaced by equality in $\mathcal{S}$. We just proceed as in the proof above, with inapplicable cases involving k_A omitted.

4 The Graphical Category

We shall now define a graphical category $\mathcal{G}$ into which $\mathcal{B}$ and $\mathcal{S}$ can be mapped. The objects of $\mathcal{G}$ are finite ordinals. An arrow $f : n \to m$ of $\mathcal{G}$ will be a binary relation from n to m, i.e. a subset of $n \times m$ with domain n and codomain m. The identity $\mathbf{1}_n : n \to n$ of $\mathcal{G}$ is the identity relation on n, and composition of arrows is composition of relations.

For an object A of $\mathcal{B}$, let $|A|$ be the number of occurrences of propositional letters in A. For example, $|(p \times (q + p)) + (\mathrm{I} \times p)|$ is 4.

We now define a functor G from $\mathcal{B}$ to $\mathcal{G}$ such that $G(A) = |A|$. It is clear that $G(A \times B) = G(A + B) = |A| + |B|$. We define G on arrows inductively:

$$
\begin{aligned}
G(\mathbf{1}_A) &= \{(x, x) : x \in |A|\} = \mathbf{1}_{|A|},\\
\\
G(k^1_{A,B}) &= \{(x, x) : x \in |A|\},\\
G(k^2_{A,B}) &= \{(x + |A|, x) : x \in |B|\},\\
G(w_A) &= \{(x, x) : x \in |A|\} \cup \{(x, x + |A|) : x \in |A|\},\\
G(k_A) &= \emptyset,
\end{aligned}
$$

$$
\begin{aligned}
G(l^1_{A,B}) &= \{(x,x) : x \in |A|\},\\
G(l^2_{A,B}) &= \{(x,x+|A|) : x \in |B|\},\\
G(m_A) &= \{(x,x) : x \in |A|\} \cup \{(x+|A|,x) : x \in |A|\},\\
G(l_A) &= \emptyset,
\end{aligned}
$$

$$G(g \circ f) = G(g) \circ G(f),$$

and for $f : A \to B$ and $g : C \to D$,

$$G(f \times g) = G(f + g) = G(f) \cup \{(x+|A|, y+|B|) : (x,y) \in G(g)\}.$$

Though $G(\mathbf{1}_A)$, $G(k^1_{A,B})$ and $G(l^1_{A,B})$ are the same as sets of ordered pairs, in general they have different domains and codomains, the first being a subset of $|A| \times |A|$, the second a subset of $(|A| + |B|) \times |A|$, and the third a subset of $|A| \times (|A| + |B|)$. We have an analogous situation in some other cases.

The arrows $G(f)$ of $\mathcal{G}$ can easily be represented graphically, by drawings linking propositional letters, as it is illustrated in [3]. This is why we call this category "graphical".

It is easy to check that G is a functor from $\mathcal{B}$ to $\mathcal{G}$. We show by induction on the length of derivation that if $f = g$ in $\mathcal{B}$, then $G(f) = G(g)$ in $\mathcal{G}$. (Of course, G preserves identities and composition.)

For the bicartesian structure of $\mathcal{G}$ we have that the operations $\times$ and $+$ on objects are both addition of ordinals, the operations $\times$ and $+$ on arrows coincide and are defined by the clauses for $G(f \times g)$ and $G(f + g)$, and the terminal and the initial object also coincide: they are both the ordinal zero. The category $\mathcal{G}$ has zero arrows—namely, the empty relation. The bicartesian category $\mathcal{G}$ is a linear category in the sense of [9] (see p. 279). The functor G from $\mathcal{B}$ to $\mathcal{G}$ is not just a functor, but a bicartesian functor; namely, a functor that preserves the bicartesian structure of $\mathcal{B}$.

We also have a functor defined analogously to G, which we call G too, from $\mathcal{S}$ to $\mathcal{G}$. It is obtained from the definition of G above by just rejecting clauses that are no longer applicable.

Our aim is to show that the functors G from $\mathcal{B}$ and $\mathcal{S}$ are faithful.

5 Auxiliary Results

We shall say for a term of $\mathcal{B}$ of the form $f_n \circ \ldots \circ f_1$, for some $n \geq 1$, where f_i is composition-free, that it is *factorized*. By using $(\times 2)$, $(+2)$ and $(cat\,1)$ it is easy to show that every term of $\mathcal{B}$ is equal to a factorized term of $\mathcal{B}$. A subterm f_i in a factorized term $f_n \circ \ldots \circ f_1$ is called a *factor*.

A term of $\mathcal{B}$ where all the atomic terms are identities will be called a *complex identity*. According to $(\times 1)$, $(+1)$ and $(cat\,1)$, every complex identity is equal to an identity. A factor which is a complex identity will be called an *identity factor*. It is clear that if $n > 1$, we can omit in a factorized term every identity factor, and obtain a factorized term equal to the original one.

A term of $\mathcal{B}$ is said to be in K-L *normal form* iff it is of the form $g \circ f : A \to B$ for f a K-term and g an L-term. Note that K-L normal forms are not unique,

since $(m_A \times m_A) \circ w_{A+A}$ and $m_{A\times A} \circ (w_A + w_A)$, which are both equal to $w_A \circ m_A$, are both in K-L normal form.

We can prove the following proposition for $\mathcal{B}$.

K-L NORMALIZATION. *Every term is equal to a term in K-L normal form.*

Proof. Suppose $f : B \to C$ is a composition-free K-term that is not a complex identity, and $g : A \to B$ is a composition-free L-term that is not a complex identity. We show by induction on the length of $f \circ g$ that

$$(*) \quad f \circ g = g' \circ f' \text{ or } f \circ g = f' \text{ or } f \circ g = g'$$

for f' a composition-free K-term and g' a composition-free L-term.

We shall not consider below cases where g is m_B, which are easily taken care of by (m). Cases where f is k_B or g is l_B are easily taken care of by (k) and (l). The following cases remain.

If f is $k^i_{C,E}$ and g is $g_1 \times g_2$, then we use (k^i). If f is w_B, then we use (w). If f is $f_1 \times f_2$ and g is $g_1 \times g_2$, then we use $(\times 2)$, the induction hypothesis, and perhaps $(cat\ 1)$.

Finally, if f is $f_1 + f_2$, then we have the following cases. If g is $l^i_{B_1,B_2}$, then we use (l^i). If g is $g_1 + g_2$, then we use $(+2)$, the induction hypothesis, and perhaps $(cat\ 1)$. This proves $(*)$.

Every term of $\mathcal{B}$ is equal to an identity or to a factorized term $f_n \circ \ldots \circ f_1$ without identity factors. Every factor f_i of $f_n \circ \ldots \circ f_1$ is either a K-term or an L-term or, by $(cat\ 1)$, $(\times 2)$ and $(+2)$, it is equal to $f''_i \circ f'_i$ where f'_i is a K-term and f''_i is an L-term. For example, $(k^1_{A,B} \times l^1_{C,D}) + (w_E + l_F)$ is equal to

$$((\mathbf{1}_A \times l^1_{C,D}) + (\mathbf{1}_{E\times E}) + l_F) \circ ((k^1_{A,B} \times \mathbf{1}_C) + (w_E + \mathbf{1}_{\mathrm{O}})).$$

Then it is clear that by applying $(*)$ repeatedly, and by applying perhaps $(cat\,1)$, we obtain a term in K-L normal form.

Note that to reduce a term of $\mathcal{B}$ to K-L normal form we have used in this proof all the equations of $\mathcal{B}$ except $(kw1)$, $(kw2)$, $(lm1)$, $(lm2)$, $(k\mathrm{O})$ and $(l\mathrm{I})$.

The definition of K-L normal form for $\mathcal{S}$ is the same. Then the proof above, with some parts omitted, establishes K-L Normalization also for $\mathcal{S}$.

We can then prove the following lemma for $\mathcal{B}$ and $\mathcal{S}$.

LEMMA 5.1. *Let $f : A_1 \times A_2 \to B$ be a K-term and let $G(f)$ be nonempty. If for every $(x, y) \in G(f)$ we have $x \in |A_1|$, then f is equal to a term of the form $f' \circ k^1_{A_1,A_2}$, and if for every $(x, y) \in G(f)$ we have $x - |A_1| \in |A_2|$, then f is equal to a term of the form $f' \circ k^2_{A_1,A_2}$.*

Proof. We proceed by induction on the length of B. If B is atomic, it can only be a propositional letter, since $G(f)$ is nonempty. If B is a propositional letter or $B_1 + B_2$, then, by Cut Elimination for K-terms and L-terms, f is equal to a cut-free Gentzen K-term, which must be equal to a term of the form $f' \circ k^i_{A_1,A_2}$. The condition on $G(f)$ dictates whether i here is 1 or 2.

If B is $B_1 \times B_2$, and for every $(x, y) \in G(f)$ we have $x \in |A_1|$, then for $k^i_{B_1,B_2} \circ f : A_1 \times A_2 \to B_i$, for every $(x, z) \in G(k^i_{B_1,B_2} \circ f)$ we have $x \in |A_1|$. So, by the induction hypothesis,

$$k^i_{B_1,B_2} \circ f = f_i \circ k^1_{A_1,A_2},$$

and hence

$$\begin{aligned} f &= \langle k^1_{B_1,B_2} \circ f, k^2_{B_1,B_2} \circ f \rangle \\ &= \langle f_1, f_2 \rangle \circ k^1_{A_1,A_2}. \end{aligned}$$

We reason analogously if for every $(x, y) \in G(f)$ we have $x - |A_1| \in |A_2|$.

We can prove analogously the following dual lemma for $\mathcal{B}$ and $\mathcal{S}$.

LEMMA 5.2. *Let $f : A \to B_1 + B_2$ be an L-term and let $G(F)$ be nonempty. If for every $(x, y) \in G(f)$ we have $y \in |B_1|$, then g is equal to a term of the form $l^1_{B_1,B_2} \circ g'$, and if for every $(x, y) \in G(f)$ we have $y - |B_1| \in |B_2|$, then g is equal to a term of the form $l^2_{B_1,B_2} \circ g'$.*

6 Coherence for Coherent Sesquicartesian Categories

We shall prove in this section that the functor G from $\mathcal{S}$ to $\mathcal{G}$ is faithful, i.e. we shall show that we have coherence for coherent sesquicartesian categories. These categories are interesting because the category **Set** of sets with functions, where cartesian product is $\times$, disjoint union is $+$, and the empty set is O, is such a category. As a matter of fact, every bicartesian closed category is a coherent sesquicartesian category. A bicartesian closed category is a bicartesian category that is cartesian closed, i.e., every functor $A \times ...$ has a right adjoint $...^A$. And in every cartesian closed category with a terminal object O we have $(k\mathrm{O})$, because $Hom(\mathrm{O} \times \mathrm{O}, \mathrm{O}) \cong Hom(\mathrm{O}, \mathrm{O}^{\mathrm{O}})$. In every cartesian category in which $(k\mathrm{O})$ holds we have that $Hom(A, \mathrm{O})$ is a singleton or empty, because for $f, g : A \to \mathrm{O}$ we have $k^1_{\mathrm{O},\mathrm{O}} \circ \langle f, g \rangle = k^2_{\mathrm{O},\mathrm{O}} \circ \langle f, g \rangle$ (cf. [8], Proposition 8.3, p. 67).

The category **Set** shows that coherent sesquicartesian categories are not maximal in the following sense. In the category $\mathcal{S}$ the equations $l_{\mathrm{O}\times A} \circ k^1_{\mathrm{O},A} = \mathbf{1}_{\mathrm{O}\times A}$ and $l_{A\times\mathrm{O}} \circ k^2_{A,\mathrm{O}} = \mathbf{1}_{A\times\mathrm{O}}$ (in which only terms of $\mathcal{S}$ occur) don't hold, but they hold in **Set**, and **Set** is not a preorder. That these two equations don't hold in $\mathcal{S}$ follows from the fact that G is a functor from $\mathcal{S}$ to $\mathcal{G}$, but $G(l_{\mathrm{O}\times p} \circ k^1_{\mathrm{O},p})$ and $G(l_{p\times\mathrm{O}} \circ k^2_{p,\mathrm{O}})$ are empty, whereas $G(\mathbf{1}_{\mathrm{O}\times p})$ and $G(\mathbf{1}_{p\times\mathrm{O}})$ contain $(0, 0)$. In the case of cartesian categories and categories with binary products and sums, we had coherence, and used this coherence to prove maximality in [1] and [3]. With coherent sesquicartesian categories, however, coherence and maximality don't go hand in hand any more.

First we prove the following lemmata.

LEMMA 6.1. *A constant object of $\mathcal{P}_{\times,+,\mathrm{O}}$ is isomorphic in $\mathcal{S}$ to* O.

Proof. We have in $\mathcal{S}$ the isomorphisms

$$\begin{array}{ll} k^1_{\mathrm{O},\mathrm{O}} = k^2_{\mathrm{O},\mathrm{O}} : \mathrm{O} \times \mathrm{O} \to \mathrm{O}, & l_{\mathrm{O}} = w_{\mathrm{O}} : \mathrm{O} \to \mathrm{O} \times \mathrm{O}, \\ m_A \circ (\mathbf{1}_A + l_A) : A + \mathrm{O} \to A, & l^1_{A,\mathrm{O}} : A \to A + \mathrm{O}, \\ m_A \circ (l_A + \mathbf{1}_A) : \mathrm{O} + A \to A, & l^2_{\mathrm{O},A} : A \to \mathrm{O} + A. \end{array}$$

LEMMA 6.2. *If $f, g : A \to B$ are terms of $\mathcal{S}$ and either A or B is isomorphic in $\mathcal{S}$ to* O, *then $f = g$ in $\mathcal{S}$.*

Proof. Suppose $i : A \to \mathrm{O}$ is an isomorphism in $\mathcal{S}$. Then from

$$f \circ i^{-1} = g \circ i^{-1} = l_B$$

we obtain $f = g$.

Suppose $i : B \to \mathrm{O}$ is an isomorphism in $\mathcal{S}$. Then from

$$k^1_{\mathrm{O},\mathrm{O}} \circ \langle i \circ f, i \circ g \rangle = k^2_{\mathrm{O},\mathrm{O}} \circ \langle i \circ f, i \circ g \rangle$$

we obtain $i \circ f = i \circ g$, which yields $f = g$.

LEMMA 6.3. *If $f, g : A \to B$ are terms of $\mathcal{S}$ and $G(f) = G(g) = \emptyset$ in $\mathcal{G}$, then $f = g$ in $\mathcal{S}$.*

Proof. By *K*-*L* Normalization, $f = f_2 \circ f_1$ for $f_1 : A \to C$ a *K*-term and $f_2 : C \to B$ an *L*-term. Since for every $z \in |C|$ there is a $y \in |B|$ such that $(z, y) \in G(f_2)$, we must have $G(f_1)$ empty; otherwise, $G(f)$ would not be empty. On the other hand, if there is a propositional letter in C at the z-th place, there must be for some $x \in |A|$ a pair (x, z) in $G(f_1)$. So C is a constant object of $\mathcal{P}_{\times,+,\mathrm{O}}$.

Analogously, $g = g_2 \circ g_1$ for $g_1 : A \to D$ a *K*-term, and $g_2 : D \to B$ an *L*-term. As before, D is a constant object of $\mathcal{P}_{\times,+,\mathrm{O}}$. By Lemma 6.1, there is an isomorphism $i : C \to D$ of $\mathcal{S}$, and $f = f_2 \circ i^{-1} \circ i \circ f_1$. By Lemmata 6.1 and 6.2, we obtain $i \circ f_1 = g_1$ and $f_2 \circ i^{-1} = g_2$, from which $f = g$ follows.

We shall next prove the following coherence proposition.

FAITHFULNESS OF G FROM $\mathcal{S}$. *If $f, g : A \to B$ are terms of $\mathcal{S}$ and $G(f) = G(g)$ in $\mathcal{G}$, then $f = g$ in $\mathcal{S}$.*

Proof. Lemma 6.3 covers the case when $G(f) = G(g) = \emptyset$. So we assume $G(f) = G(g) \neq \emptyset$, and proceed by induction on the length of A and B. In this induction we need not consider the cases when either A or B is a constant object; otherwise, $G(f)$ and $G(g)$ would be empty. So in the basis of the induction, when both of A and B are atomic, we consider only the case when both of A and B are propositional letters. In this case we conclude by Cut Elimination that f and g exist iff A and B are the same propositional letter p, and $f = g = \mathbf{1}_p$ in $\mathcal{S}$. (We could conclude the same thing by interpreting $\mathcal{S}$ in conjunctive-disjunctive logic.) Note that we didn't need here the assumption $G(f) = G(g)$.

If A is $A_1 + A_2$, then $f \circ l^1_{A_1,A_2}$ and $g \circ l^1_{A_1,A_2}$ are of type $A_1 \to B$, while $f \circ l^2_{A_1,A_2}$ and $g \circ l^2_{A_1,A_2}$ are of type $A_2 \to B$. We also have

$$\begin{aligned} G(f \circ l^i_{A_1,A_2}) &= G(f) \circ G(l^i_{A_1,A_2}) \\ &= G(g) \circ G(l^i_{A_1,A_2}) \\ &= G(g \circ l^i_{A_1,A_2}), \end{aligned}$$

whence, by the induction hypothesis,

$$f \circ l^i_{A_1,A_2} = g \circ l^i_{A_1,A_2}$$

in $\mathcal{S}$. Then we infer that

$$[f \circ l^1_{A_1,A_2}, f \circ l^2_{A_1,A_2}] = [g \circ l^1_{A_1,A_2}, g \circ l^2_{A_1,A_2}],$$

from which it follows that $f = g$ in $\mathcal{S}$. We proceed analogously if B is $B_1 \times B_2$.

Suppose now A is $A_1 \times A_2$ or a propositional letter, and B is $B_1 + B_2$ or a propositional letter, but A and B are not both propositional letters. Then, by Cut Elimination, f is equal in $\mathcal{S}$ either to a term of the form $f' \circ k^i_{A_1,A_2}$, or to a term of the form $l^i_{B_1,B_2} \circ f'$. Suppose $f = f' \circ k^1_{A_1,A_2}$. Then for every $(x, y) \in G(f)$ we have $x \in |A_1|$. (We reason analogously when $f = f' \circ k^2_{A_1,A_2}$.)

By K-L normalization, $g = g_2 \circ g_1$ in $\mathcal{S}$ for $g_1 : A_1 \times A_2 \to C$ a K-term and g_2 an L-term. Since $g_2 : C \to B$ is an L-term, and hence k^i and k don't occur in it, for every $z \in |C|$ we have a $y \in |B|$ such that $(z, y) \in G(g_2)$. If for some $(x, z) \in G(g_1)$ we had $x \notin |A_1|$, then for some $(x, y) \in G(g_2 \circ g_1)$ we would have $x \notin |A_1|$, but this is impossible since $G(g_2 \circ g_1) = G(g) = G(f)$. So for every $(x, z) \in G(g_1)$ we have $x \in |A_1|$. Moreover, $G(g_1)$ is not empty, since otherwise $G(g)$ would be empty. Then, by Lemma 5.1, we have that $g_1 = g'_1 \circ k^1_{A_1,A_2}$ and $g = g_2 \circ g'_1 \circ k^1_{A_1,A_2}$ hold in $\mathcal{S}$.

Because of the particular form of $G(k^1_{A_1,A_2})$, we can infer from $G(f) = G(g)$ that $G(f') = G(g_2 \circ g'_1)$, but since f' and $g_2 \circ g'_1$ are of type $A_1 \to B$, by the induction hypothesis we have $f' = g_2 \circ g'_1$ in $\mathcal{S}$, and hence $f = g$. When $f = l^i_{B_1,B_2} \circ f'$, we reason analogously and apply Lemma 5.2.

To verify whether for $f, g : A \to B$ in the language of $\mathcal{S}$ we have $f = g$ in $\mathcal{S}$ it is enough to draw $G(f)$ and $G(g)$, and check whether they are equal, which is clearly a finite task. So we have here an easy decision procedure for the equations of $\mathcal{S}$.

It is clear that we also have coherence for coherent dual sesquicartesian categories, which are categories with arbitrary finite products and nonempty finite sums where (lI) holds.

As a consequence of Cut Elimination, of the functoriality of G from $\mathcal{B}$, and of the Faithfulness of G from $\mathcal{S}$, we obtain that $\mathcal{S}$ is a full subcategory of $\mathcal{B}$.

7 Coherence for Coherent Bicartesian Categories

We shall finally prove in this section that the functor G from $\mathcal{B}$ to $\mathcal{G}$ is faithful, i.e. we shall show that we have coherence for coherent bicartesian categories. But before that we shall give a few examples of coherent bicartesian categories.

Note first that the bicartesian category **Set**, where product, sum and the initial object are taken as usual (see the beginning of the previous section), and a singleton is the terminal object I, is not a coherent bicartesian category. The equation (lI) does not hold in **Set**.

Every bicartesian category in which the terminal and the initial object are isomorphic is a coherent bicartesian category. Such is, for instance, the category **Set*** of pointed sets, i.e. sets with a distinguished element $*$ and $*$-preserving maps, which is isomorphic to the category of sets with partial maps. In **Set*** the objects I and O are both $\{*\}$, the product $\times$ is cartesian product, the sum $A+B$ of the objects A and B is $\{(a,*) : a \in A\} \cup \{(*,b) : b \in B\}$, with $(*,*)$ being the $*$ of the product and of the sum, while

$$l^1_{A,B}(a) = (a,*), \quad l^2_{B,A}(a) = (*,a),$$
$$m_A(a,b) = \begin{cases} a \text{ if } a \neq * \\ b \text{ if } b \neq * \\ * \text{ if } a = b = * \end{cases}$$
$$(f+g)(a,b) = (f(a), g(b)).$$

In **Set*** we have that I+I is isomorphic to I.

A fortiori, every bicartesian category in which all finite products and sums are isomorphic (i.e. every linear category in the sense of [9], p. 279), is a coherent bicartesian category. Such are, for example, the category of commutative monoids with monoid homomorphisms, and its subcategory of vector spaces over a fixed field with linear transformations. We shall next present a concrete coherent bicartesian category in which the terminal and the initial object, as well as finite products and sums in general, are not isomorphic.

As coherent sesquicartesian categories were not maximal, so coherent bicartesian categories are not maximal either. This is shown by the category **Set***+$\emptyset$, which is the category of pointed sets extended with the empty set $\emptyset$ as an additional object. The arrows of **Set***+$\emptyset$ are the $*$-preserving maps plus the empty maps with domain $\emptyset$. Let I be $\{*\}$, let O be $\emptyset$, let $\times$ be cartesian product, and let the sum $A+B$ be defined as in **Set*** if either A or B is not $\emptyset$. If both A and B are $\emptyset$, then $A+B$ is $\emptyset$. If $A \times B$ and $A+B$ are not $\emptyset$, then their $*$ is $(*,*)$, as in **Set***. Next, let l_B be $\emptyset : \emptyset \to B$, and if A is not $\emptyset$, let $l^1_{A,B}$, $l^2_{B,A}$ and m_A be defined as in **Set***. If A is $\emptyset$, then $l^1_{A,B}$ is $\emptyset : \emptyset \to \emptyset + B$, $l^2_{B,A}$ is $\emptyset : \emptyset \to B + \emptyset$ and m_A is $\emptyset : \emptyset \to \emptyset$. We also have

$$(f+g)(a,b) = \begin{cases} (f(a), g(b)) & \text{if neither } f \text{ nor } g \text{ is an empty map} \\ (f(a), *) & \text{if } f \text{ is not an empty map and } g \text{ is} \\ (*, g(b)) & \text{if } g \text{ is not an empty map and } f \text{ is.} \end{cases}$$

If f and g are both empty maps, then $f+g$ is the empty map with appropriate codomain.

With other arrows and operations on arrows defined in the obvious way, we can check that **Set***+$\emptyset$ is a coherent bicartesian category. In this category we have that $\emptyset \times A = A \times \emptyset = \emptyset$. In the category $\mathcal{B}$ the equations $l_{\mathrm{O}\times A} \circ k^1_{\mathrm{O},A} = \mathbf{1}_{\mathrm{O}\times A}$

and $l_{A\times \mathrm{O}} \circ k^2_{A,\mathrm{O}} = \mathbf{1}_{A\times \mathrm{O}}$ (in which only terms of $\mathcal{B}$ occur) don't hold, as we explained at the beginning of the previous section, but they hold in $\mathbf{Set}^*{+}\emptyset$, and $\mathbf{Set}^*{+}\emptyset$ is not a preorder.

Note that a coherent bicartesian category $\mathcal{C}$ is cartesian closed only if $\mathcal{C}$ is a preorder. We have $Hom_{\mathcal{C}}(A, B) \cong Hom_{\mathcal{C}}(\mathrm{I}, B^A)$, and for $f, g : \mathrm{I} \to D$ with ($l\mathrm{I}$) we obtain $[f, g] \circ l^1_{\mathrm{I},\mathrm{I}} = [f, g] \circ l^2_{\mathrm{I},\mathrm{I}}$, which gives $f = g$. The category $\mathbf{Set}$ is cartesian closed, whereas $\mathbf{Set}^*$ and $\mathbf{Set}^*{+}\emptyset$ are not.

We can prove the following lemmata, which extend Lemmata 6.1-6.3.

LEMMA 7.1. *A constant object of $\mathcal{P}$ is isomorphic in $\mathcal{B}$ to either* O *or* I.

Proof. In addition to the isomorphisms of the proof of Lemma 6.1, we have in $\mathcal{B}$ the isomorphisms

$$\begin{array}{ll} k_{\mathrm{I}} = m_{\mathrm{I}} : \mathrm{I} + \mathrm{I} \to \mathrm{I}, & l^1_{\mathrm{I},\mathrm{I}} = l^2_{\mathrm{I},\mathrm{I}} : \mathrm{I} \to \mathrm{I} + \mathrm{I}, \\ k^1_{A,\mathrm{I}} : A \times \mathrm{I} \to A, & (\mathbf{1}_A \times k_A) \circ w_A : A \to A \times \mathrm{I}, \\ k^2_{\mathrm{I},A} : \mathrm{I} \times A \to A, & (k_A \times \mathbf{1}_A) \circ w_A : A \to \mathrm{I} \times A. \end{array}$$

LEMMA 7.2. *If $f, g : A \to B$ are terms of $\mathcal{B}$ and either A or B is isomorphic in $\mathcal{B}$ to* O *or* I, *then $f = g$ in $\mathcal{B}$.*

Proof. We repeat what we had in the proof of Lemma 6.2, and reason dually when A or B is isomorphic to I.

LEMMA 7.3. *If $f, g : A \to B$ are terms of $\mathcal{B}$ and $G(f) = G(g) = \emptyset$ in $\mathcal{G}$, then $f = g$ in $\mathcal{B}$.*

Proof. As in the proof of Lemma 6.3, by K-L Normalization, we have $f = f_2 \circ f_1$ for $f_1 : A \to C$ a K-term, $f_2 : C \to B$ an L-term, and C a constant object of $\mathcal{P}$; we also have $g = g_2 \circ g_1$ for $g_1 : A \to D$ a K-term, $g_2 : D \to B$ an L-term, and D a constant object of $\mathcal{P}$. Next we apply Lemma 7.1. If C and D are both isomorphic to O, we reason as in the proof of Lemma 6.3, and we reason analogously when they are both isomorphic to I. If $i : C \to \mathrm{O}$ and $j : \mathrm{I} \to D$ are isomorphisms of $\mathcal{B}$, then we have

$$\begin{aligned} f_2 \circ f_1 &= g_2 \circ j \circ k_{\mathrm{O}} \circ i \circ f_1, \text{ by Lemma 7.2,} \\ &= g_2 \circ g_1, \text{ by Lemma 7.2,} \end{aligned}$$

and so $f = g$ in $\mathcal{B}$. (Note that $k_{\mathrm{O}} = l_{\mathrm{I}}$ in $\mathcal{B}$.)

We can then prove the following coherence proposition.

FAITHFULNESS OF G FROM $\mathcal{B}$. *If $f, g : A \to B$ are terms of $\mathcal{B}$ and $G(f) = G(g)$ in $\mathcal{G}$, then $f = g$ in $\mathcal{B}$.*

Proof. Lemma 7.3 covers the case when $G(f) = G(g) = \emptyset$, and when $G(f) = G(g) \neq \emptyset$ we imitate the proof of the Faithfulness of G from $\mathcal{S}$ in the previous section.

As before, this proposition provides an easy decision procedure for the equations of $\mathcal{B}$.

References

1. Došen, K., Petrić, Z.: The Maximality of Cartesian Categories. Math. Logic Quart. **47** (2001) 137-144 (available at: http:// xxx.lanl.gov/ math.CT/ 9911059)
2. Došen, K., Petrić, Z.: The Maximality of the Typed Lambda Calculus and of Cartesian Closed Categories. Publ. Inst. Math. (N.S.) **68**(**82**) (2000) 1-19 (available at: http:// xxx.lanl.gov/ math.CT/ 9911073)
3. Došen, K., Petrić, Z.: Bicartesian Coherence. To appear in Studia Logica (available at: http:// xxx.lanl.gov/ math.CT/ 0006052)
4. Gentzen, G.: Untersuchungen über das logische Schließen. Math. Z. **39** (1935) 176-210, 405-431. English translation in: The Collected Papers of Gerhard Gentzen. Szabo, M. E. (ed.), North-Holland, Amsterdam (1969)
5. Kelly, G. M., Mac Lane, S.: Coherence in Closed Categories. J. Pure Appl. Algebra **1** (1971) 97-140, 219
6. Lambek, J.: Deductive Systems and Categories I: Syntactic Calculus and Residuated Categories. Math. Systems Theory **2** (1968) 287-318
7. Lambek, J.: Deductive Systems and Categories II: Standard Constructions and Closed Categories. In: Category Theory, Homology Theory and their Applications I. Lecture Notes in Mathematics, Vol. 86. Springer, Berlin (1969) 76-122
8. Lambek, J., Scott, P.J.: Introduction to Higher-Order Categorical Logic. Cambridge University Press, Cambridge (1986)
9. Lawvere, F. W., Schanuel, S. H.: Conceptual Mathematics: A First Introduction to Categories. Cambridge University Press, Cambridge (1997). First edn. Buffalo Workshop Press, Buffalo (1991)
10. Mac Lane, S.: Natural Associativity and Commutativity. Rice University Studies, Papers in Mathematics **49** (1963) 28-46
11. Simpson, A. K.: Categorical Completeness Results for the Simply-Typed Lambda-Calculus. In: Dezani-Ciancaglini, M., Plotkin, G. (eds), Typed Lambda Calculi and Applications (Edinburgh, 1995). Lecture Notes in Computer Science, Vol. 902. Springer, Berlin (1995) 414-427

Indexed Induction-Recursion

Peter Dybjer[1] and Anton Setzer[2]

[1] Department of Mathematics and Computing Science,
Chalmers University of Technology.
Fax: +46 31 165655.
peterd@cs.chalmers.se
[2] Department of Computer Science, University of Wales Swansea,
Singleton Park, Swansea SA2 8PP, UK.
Fax: +44 1792 295651.
a.g.setzer@swan.ac.uk

Abstract. We give two finite axiomatizations of indexed inductive-recursive definitions in intuitionistic type theory. They extend our previous finite axiomatizations of inductive-recursive definitions of sets to *indexed families* of sets and encompass virtually all definitions of sets which have been used in intuitionistic type theory. The more restricted of the two axiomatization arises naturally by considering indexed inductive-recursive definitions as initial algebras in slice categories, whereas the other admits a more general and convenient form of an introduction rule.
The class of indexed inductive-recursive definitions properly contains the class of indexed inductive definitions (so called "inductive families"). Such definitions are ubiquitous when using intuitionistic type theory for formalizing mathematics and program correctness. A side effect of the present paper is to get compact finite axiomatizations of indexed inductive definitions in intuitionistic type theory as special cases.
Proper indexed inductive-recursive definitions (those which do not correspond to indexed inductive definitions) are useful in intuitionistic metamathematics, and as an example we formalize Tait-style computability predicates for dependent types. We also show that Palmgren's proof-theoretically strong construction of higher-order universes is an example of a proper indexed inductive-recursive definition. A third interesting example is provided by Bove and Capretta's definition of termination predicates for functions defined by nested recursion.
Our axiomatizations form a powerful foundation for generic programming with dependent types by introducing a type of codes for indexed inductive-recursive definitions and making it possible to define generic functions by recursion on this type.

Keywords: Dependent type theory, Martin-Löf Type Theory, inductive definitions, inductive-recursive definitions, inductive families, initial algebras, normalization proofs, generic programming.

1 Introduction

In [8], the first author introduced the concept of inductive-recursive definitions, an extension of ordinary inductive definitions. By inductive-recursive definitions

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 93–113, 2001.

one can inductively define a set U while simultaneously recursively defining a function $T : U \to D$, where D is an arbitrary type.

The prime example of an inductive-recursive definition is a universe, that is, a set U of codes for sets together with a decoding function $T : U \to \text{set}$, which assigns to a code in U the set it denotes. As an example consider the constructor $\widehat{\Sigma}$ introducing codes for the Σ-type. We have $\widehat{\Sigma} : (a : U, b : T(a) \to U) \to U$ (the reader not familiar with this notation, which will be explained in Sec. 2, might temporarily substitute $\Pi a : U.\Pi b : (T(a) \to U).U$ for the type of $\widehat{\Sigma}$). Note that the second argument b of $\widehat{\Sigma}(a, b)$ refers to $T(a)$. This makes sense since before $\widehat{\Sigma}(a, b)$ is introduced, $a : U$ has to be established, and when $a : U$ is established, one can recursively define $T(a)$. We then define recursively $T(\widehat{\Sigma}(a, b)) = \Sigma(T(a), (x)T(b(x)))$ (or in more traditional notation $T(\widehat{\Sigma}(a, b)) = \Sigma x : T(a).T(b(x))$). This is possible, since $T(a)$ and $T(b(x))$ are known before we introduce $\widehat{\Sigma}(a, b)$. Here we see the novelty of inductive-recursive definitions relative to inductive definitions: an introduction rule for U may refer to the function T which is simultaneously defined. Note that T even can appear negatively in the type of a constructor, as in the type of $\widehat{\Sigma}$. However, the constructor refers only strictly positively to elements of the inductively defined set U.

In the special case where $T : U \to \mathbf{1}$, where $\mathbf{1}$ is the type with only one element, so that T does not contain any information, inductive-recursive definitions specialize to inductive definitions. For a detailed explanation of the concept of inductive-recursive definitions, the reader is advised to read the aforementioned article [8].

In [8] the case of *indexed* inductive-recursive definitions (**IIR**) is also considered. In an IIR one defines a *family* of sets $U(i)$, indexed over $i : I$, inductively, while simultaneously recursively defining a family of functions $T(i) : U(i) \to D[i]$ for some collection of types $D[i]$ $(i : I)$. (We have to write $D[i]$, the result of substituting i for a fixed variable in D here, since $D[i]$ is a type and we cannot write $D : I \to \text{type}$.) Again constructors of $U(i)$ can refer to $T(j)$ applied to elements of $U(j)$.

In [9] and [10] we presented closed axiomatizations of the theory of non-indexed inductive-recursive definitions. It is the objective of this article to extend this to indexed inductive-recursive definitions. We will see that the resulting rules are not much more complicated than the rules for the non-indexed case. We will look at two alternatives: In **restricted IIR** (introduced by T. Coquand for use in the Half and Agda [4] systems) we can determine for each index i the set of arguments of the constructors introducing elements of the set U_i. In **unrestricted IIR** [8], for a constructor C with arguments $\boldsymbol{a}$, the index i s.t. $C(\boldsymbol{a}) : \mathrm{U}_i$ depends on the arguments $\boldsymbol{a}$ of C.

Indexed inductive definitions **(IID)** subsume inductively defined relations, such as the identity relation understood as the least reflexive relation. The identity relation was indeed the only example of an IID in the early versions of Martin-Löf type theory [13]. Theoretically, this is not very limiting since one can define many other families of sets (predicates) using the identity together

with the other set formers of type theory. But from a more practical point of view it became desirable to extend Martin-Löf type theory with a general notion of inductive definition of an indexed family of sets (often called "inductive family" for short) [6,7]. Such a general mechanism is also a key part of the Calculus of Inductive Constructions [21], the impredicative type theory underlying the Coq system. Indexed inductive definitions are ubiquitous in formalizations carried out in the different proof systems for Martin-Löf type theory and the Calculus of Inductive Constructions.

No proper IIR were part of the original versions of Martin-Löf type theory. However, when proving normalization for an early version of this theory, Martin-Löf made use of an informal construction on the metalevel which has an implicit indexed inductive-recursive character.

Plan. In Section 2 we introduce the logical framework of dependent types which is the basis for our theories of IIR. This section also explains the notation used in the paper. In Section 3 we begin by reviewing a few examples of IID, such as finitely branching trees, the even and odd numbers, the accessible part of a relation, the identity set, and context free grammars. Then we give some examples of proper IIR: Martin-Löf's computability predicates for dependent types, Palmgren's higher order universes, and Bove and Capretta's analysis of the termination of nested recursive definitions of functional programs. We also introduce the restricted form of an inductive-recursive definition which has an easier syntax and is easier to implement. In Section 4 we first explain how the restricted version of indexed inductive-recursive definitions arises from the existence of initial algebras of strictly positive endofunctors on slice categories over the category of indexed families of types. Then we show how to formalize the general notion of an indexed inductive-recursive definition and give a uniform theory for both the general and the restricted form by giving a type of codes for IIR and then derive the formation, introduction, elimination and equality rules for a particular code. In Section 5 we show how to instantiate the general theory to some of the examples given in Section 3.

2 The Logical Framework

Before giving the rules for IIR we need to introduce the basic Logical Framework of dependent types. This is essentially Martin-Löf's Logical Framework [18] extended with rules for the types $\mathbf{0}$, $\mathbf{1}$, and $\mathbf{2}$. A more detailed presentation can be found in [10].

The Logcial Framework has the following forms of judgements: Γ context, and $A : \text{type}$, $A = B : \text{type}$, $a : A$, $a = b : A$, depending on contexts Γ (written as $\Gamma \Rightarrow A : \text{type}$, etc.). We have $\text{set} : \text{type}$ and if $A : \text{set}$, then $A : \text{type}$. The collection of types is closed under the formation of dependent function types written as $(x : A) \to B$ (which is often written as $\Pi x : A.B$ in the literature – we prefer to reserve $\Pi x : A.B$ for the inductive-recursively defined set with constructor $\lambda : ((x : A) \to B) \to (\Pi x : A.B)$). The elements of $(x : A) \to B$ are

denoted by $(x : A)a$ (abstraction of x in a; this is often denoted by $\lambda x : A.a$ in the literature) and application is written in the form $a(b)$. We have $\beta-$ and η-rules. Types are also closed under the formation of dependent products written as $(x : A) \times B$ (often denoted by $\Sigma x : A.B$ which is here reserved for the inductively defined set with introduction rule $\mathrm{p} : ((x : A) \times B) \to (\Sigma x : A.B)$). The elements of $(x : A) \times B$ are denoted by $\langle a, b \rangle$, the projection functions by π_0 and π_1 and again we have β and η-rule (surjective pairing). There is the type $\mathbf{1}$, with unique element $\star : \mathbf{1}$ and η-rule expressing that, if $a : \mathbf{1}$, then $a = \star : \mathbf{1}$. Further we have the empty type $\mathbf{0}$ with elimination rule Efq (ex falsum quodlibet).

Moreover, we include in our logical framework the type $\mathbf{2}$ with two elements $\star_0 : \mathbf{2}$ and $\star_1 : \mathbf{2}$, ordinary elimination rule $\mathrm{C}_2 : (a : \mathbf{2}, A[\star_0], A[\star_1]) \to A(a)$ (where $x : \mathbf{2} \Rightarrow A[x] : \mathrm{type}$) and the strong elimination rule

$$\frac{a : \mathbf{2} \qquad A : \mathrm{type} \qquad B : \mathrm{type}}{\mathrm{C}_2^{\mathrm{type}}(a, A, B) : \mathrm{type}}$$

with equality rules

$$\mathrm{C}_2^{\mathrm{type}}(\star_0, A, B) = A \ , \qquad \mathrm{C}_2^{\mathrm{type}}(\star_1, A, B) = B \ .$$

It is necessary to have the strong elimination rule, since we want to inductively define indexed families of sets $U : I \to \mathrm{set}$ and functions $T : (i : I) \to U(i) \to D[i]$ where $D[i]$ depends non-trivially on i, as in the definition of Palmgren's higher-order universe (where for instance $D[0] = \mathrm{set}$, $D[1] = \mathrm{Fam(set)} \to \mathrm{Fam(set)}$, see 3.2 for more explanation).

We will also add a level between set and type, which we call stype for small types: stype : type. (The reason for the need for stype is discussed in [8].) If $a : \mathrm{set}$ then $a : \mathrm{stype}$. Moreover, stype is also closed under dependent function types, dependent products and includes $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$. However, set itself will not be in stype. The logical framework does not have any rules for introducing elements of set, they will be introduced by IIR later and set will therefore consist exactly of the sets introduced by IIR.

We also use some abbreviations, such as omitting the type in an abstraction, that is, writing $(x)a$ instead of $(x : A)a$, and writing repeated application as $a(b_1, \ldots, b_n)$ instead of $a(b_1) \cdots (b_n)$ and repeated abstraction as $(x_1 : A_1, \ldots, x_n : A_n)a$ instead of $(x_1 : A_1) \cdots (x_n : A_n)a$.

In the following we will sometimes refer to a type depending on a variable x. We want to use the notation $D[t]$ for $D[x := t]$ for some fixed variable x and D for $(x)D[x]$. Note that we can't simply introduce $D : I \to \mathrm{type}$, since this goes beyond the logical framework. Instead we introduce the notion of an abstracted expression, which is an expression together with one or several designated free variables. For an abstracted expression E, $E[t_1, \ldots, t_n]$ means the substitution of the variables by $t_1, \ldots, t_n$. If we take for D above an abstracted expression of the form $(x)E$, then $D[t]$ denotes $D[x := t]$ and we can write D as parameter for $(x)E$. More formally:

Definition 1. *(a) An n-times abstracted expression is an expression* $(x_1, \ldots, x_n)E$ *where* $x_1, \ldots, x_n$ *are distinct variables and* E *an expression of the language of type theory. An abstracted expression is a* 1*-times abstracted expression.*
(b) $((x_1, \ldots, x_n)E)[t_1, \ldots, t_n] := E[x_1 := t_1, \ldots, x_n := t_n]$.
(c) Whenever we write $s[a_1, \ldots, a_n]$*,* s *is to be understood as an n-times abstracted expression.*
(d) If $U : A \to B$*, we identify* U *with the abstracted expression* $(a)U(a)$.

3 Some Examples

3.1 Indexed Inductive Definitions

Trees and forests. Many IID occur as the simultaneous inductive definition of finitely many sets, each of which has a different name. One example is the set of well-founded trees Tree with finite branching degrees, which is defined together with the set Forest of finite lists of such trees. The constructors are:

$$\begin{aligned} \text{tree} &: \text{Forest} \to \text{Tree} \ , \\ \text{nil} &: \text{Forest} \ , \\ \text{cons} &: \text{Tree} \to \text{Forest} \to \text{Forest} \ . \end{aligned}$$

If we replace Tree by $\text{Tree}'(\star_0)$ and Forest by $\text{Tree}'(\star_1)$, where $x : \mathbf{2} \Rightarrow \text{Tree}'(x) : \text{set}$, we obtain an IID with index set $\mathbf{2}$. Since for every index we know the set of constructors introducing elements of it in advance, we have an example of restricted IID.

The even number predicate. Another simple example of an IID is the predicate $\text{Even} : \text{N} \to \text{set}$. This is inductively generated by the two rules

$$\begin{aligned} &\text{C}_0 : \text{Even}(0) \ , \\ &\text{C}_1 : (n : \text{N}) \to \text{Even}(n) \to \text{Even}(\text{S}(\text{S}(n))) \ . \end{aligned}$$

In this form we have unrestricted IID, since in this form the constructors introducing an element of $\text{Even}(n)$ is not given in advance.

The accessible part of a relation. Let I be a set and $<: I \to I \to \text{set}$ be a binary relation on it. We define the accessible part (the largest well-founded initial segment) of $<$ as a predicate $\text{Acc} : I \to \text{set}$ by a generalized indexed inductive definition with one introduction rule:

$$\text{acc} : (i : I) \to ((x : I) \to (x < i) \to \text{Acc}(x)) \to \text{Acc}(i) \ .$$

Note that acc introduces elements of $\text{Acc}(i)$ while referring to possibly infinitely many elements of the sets $\text{Acc}(x)$ (for each $x : I$ and each proof of $x < i$). Acc is an example of a restricted IID.

The identity relation. The only example of an IID in Martin-Löf's original formulation of type theory, which is not an ordinary inductive-recursive definition is the identity type ("identity type" is only used for historic reason – a more appropriate name would be "identity set"). Assume A : set. The identity on A is given as the least reflexive relation on $A \times A$, and is the intensional equality type on A. We have as formation rule $a : A, b : A \Rightarrow \mathrm{I}(A, a, b) : \mathrm{set}$. The introduction rule expresses that it is reflexive:

$$\mathrm{r} : (a : A) \to \mathrm{I}(A, a, a) \ .$$

The elimination rule inverts the introduction rule. Assume we have a type which is a reflexive relation on $A \times A$ and a subrelation of the identity type on A. So assume $a : A, b : A, p : \mathrm{I}(A, a, b) \Rightarrow C[a, b, p] : \mathrm{type}$ and for every $a : A$ we have $s[a] : C[a, a, \mathrm{r}(a)]$ (so s is the step-function corresponding to the constructor r). Then for every $a, b : A$ and $p : \mathrm{I}(A, a, b)$ we have $\mathrm{J}(a, b, p, (x)s[x]) : C[a, b, p]$. The equality rule now uses the step function in case an element is introduced by a constructor: $\mathrm{J}(a, a, \mathrm{r}(a), (x)s[x]) = s[a]$.

If we write $\mathrm{I}'_A(\langle a, b\rangle)$ instead of $\mathrm{I}(A, a, b)$, we obtain for every A : set an unrestricted IID I'_A with index set $A \times A$.

Context free grammars. IID occur very frequently in applications in computer science. For example a context free grammars over a finite alphabet Σ and a finite set of nonterminals NT can be seen as an $\mathrm{NT} \times \Sigma^*$-indexed inductive definition L, where each production corresponds to an introduction rule.

As an example consider the context free grammar with $\Sigma = \{a, b\}$, $\mathrm{NT} = \{A, B\}$ and productions $A \longrightarrow a$, $A \longrightarrow BB$, $B \longrightarrow AA$, $B \longrightarrow b$. This corresponds to an inductive definition of an indexed family L, where $L(A, \alpha)$ is the set of derivation trees of the string α from the start symbol A. So α is in the language generated by the grammar with start symbol A iff $L(A, \alpha)$ is inhabited. L has one constructor for each production: $C_0 : L(A, a)$, $C_1 : L(B, \alpha) \to L(B, \beta) \to L(A, \alpha\beta)$, $C_2 : L(A, \alpha) \to L(A, \beta) \to L(B, \alpha\beta)$, $C_3 : L(B, b)$.

Alternatively, we can inductively define an NT-indexed set D of "abstract syntax trees" for the grammar, and then recursively define the string $d(A, p)$ ("concrete syntax") corresponding to the abstract syntax tree $p : D(A)$. In the example given before we get $C_0 : D(A)$, $C_1 : D(B) \to D(B) \to D(A)$, $C_2 : D(A) \to D(A) \to D(B)$, $C_3 : D(B)$. Further $d(A, C_0) = a$, $d(A, C_1(p, q)) = d(B, p) * d(B, q)$, $d(B, C_2(p, q)) = d(A, p) * d(B, q)$, $d(B, C_3) = b$.

More examples. There are many more examples (eg. the set of formulas derivable in a formal system, computation rules of the operational semantics of a programming language) of similar nature. If Formula is a set of formulas of the formal system, then to be a theorem is given by a Formula-indexed inductive definition Theorem : Formula $\to$ set, where the axioms and inference rules correspond to introduction rules. An element d : Theorem(ϕ) is a notation for a derivation (or proof tree) with conclusion ϕ. Yet more examples are provided by the computation rules in the definition of the *operational semantics* of a programming language.

Proofs by induction on the structure of an indexed inductive definition of these kinds are often called proofs by "rule induction". Thus the general form of rule induction is captured by the elimination rule for unrestricted IIR which we will give later.

3.2 Indexed Inductive-Recursive Definitions

Martin-Löf's computability predicates for dependent types. We shall now turn to proper IIR. As a first example we shall formalize the Tait-style computability predicates for dependent types introduced by Martin-Löf [17]. This example was crucial for the historical development of IIR, since it may be viewed as an early occurrence of the informal notion of an IIR. In [17] Martin-Löf presents an early version of his intuitionistic type theory and proves a normalization theorem using such Tait-style computability predicates. He works in an informal intuitionistic metalanguage but gives no explicit justification for the meaningfulness of these computability predicates. (Later Aczel [1] has shown how to model a similar construction in classical set theory.) Since the metalanguage is informal the inductive-recursive nature of this definition is implicit. One of the objectives of the current work is indeed to formalize an extension of Martin-Löf type theory where the inductive-recursive nature of this and other definitions is formalized. In this way we hope to help clarify the reason why it is an acceptable notion from the point of view of intuitionistic meaning explanations in the sense of Martin-Löf [14,16,15].

First recall that for the case of the simply typed lambda calculus the Tait-computability predicates ϕ_A are predicates on terms of type A which are defined by *recursion* on the structure of A. We read $\phi_A(a)$ as "a is a computable term of type A". To match Martin-Löf's definition [17] we consider here a version where the clause for function types is

– If $\phi_B(b[a])$ for all closed terms a such that $\phi_A(a)$ then $\phi_{A\to B}(\lambda x.b[x])$.

(Here $b[x]$ denotes an expression with a possible occurrence of the free variable x and $b[a]$ denotes the expression which is obtained by substituting x by a in $b[x]$.)

How can we generalize this to dependent types? First we must assume that we have introduced the syntax of expressions for dependent types including Π-types, with lambda abstraction and application. Now we cannot define ϕ_A for all (type) expressions A but only for those which are "computable types". The definition of ϕ_A has several clauses, such as the following one for Π [17, p. 161]:

> 4.1.1.2. Suppose that ϕ_A has been defined and that $\phi_{B[a]}$ has been defined for all closed terms a of type A such that $\phi_A(a)$. We then define $\phi_{\Pi x:A.B[x]}$ by the following three clauses.
>
> 4.1.1.2.1. If $\lambda x.b[x]$ is a closed term of type $\Pi x : A.B[x]$ and $\phi_{B[a]}(b[a])$ for all closed terms a of type A such that $\phi_A(a)$, then $\phi_{\Pi x:A.B[x]}(\lambda x.b[x])$.
>
> 4.1.1.2.2. ...
>
> 4.1.1.2.3. ...

(We omit the cases 4.1.1.2.2 and 4.1.1.2.3, which express closure under reduction, since they are not relevant for the present discussion. Note also that the complete definition of the computability predicate also has one case for each of the other type formers of type theory.)

We also note that Martin-Löf does not use the term "A is a computable type" but only states "that ϕ_A has been defined". We can understand Martin-Löf's definition as an indexed inductive-recursive definition by introducing a predicate Φ on expressions, where $\Phi(A)$ stands for "ϕ_A is defined" or "A is a *computable type*". Moreover, we add a second argument to ϕ so that $\phi_A(p, a)$ means that "a is a computable term of the computable type A, where p is a proof that A is computable. Now we observe that we define Φ inductively while we simultaneously recursively define ϕ.

It would be possible to formalize Martin-Löf's definition verbatim, but for simplicity we shall follow a slightly different version due to C. Coquand [5]. Assume that we have inductively defined the set Exp of expressions and have an operation $\mathrm{Apl} : \mathrm{Exp} \to \mathrm{Exp} \to \mathrm{Exp}$ for the application of one expression to another. Apl is a constructor of Exp, and there will be additional reduction rules for expressions, like reduction of β-redexes. We will write in the following $A\ b$ for $\mathrm{Apl}(A, b)$.

Now define an Exp-indexed IIR

$$\begin{aligned} &\Psi : \mathrm{Exp} \to \mathrm{set}\ , \\ &\psi : (A : \mathrm{Exp}) \to \Psi(A) \to \mathrm{Exp} \to \mathrm{set}\ . \end{aligned}$$

So the index set is Exp, Ψ plays the rôle of U, ψ the rôle of $T : (a : U) \to D[a]$, where $D[a] = \mathrm{Exp} \to \mathrm{set}$ for $a : \mathrm{Exp}$. Note that ψ depends negatively on Ψ, so this is not a simultaneous inductive definition. The introduction rule for Ψ (corresponding to Martin-Löf's 4.1.1.2) is:

$$\begin{aligned} \pi : &(A : \mathrm{Exp}) \to (p : \Psi(A)) \to (B : \mathrm{Exp}) \to \\ &(q : (a : \mathrm{Exp}) \to \psi(A, p, a) \to \Psi(B\,a)) \to \\ &\Psi(\Pi(A, B))\ . \end{aligned}$$

Note that q refers to $\psi(A, p, a)$ which is short notation for $\psi(A, p)(a)$, where $\psi(A, p)$ is the result of the recursively defined function for the second argument p. The corresponding equality rule is

$$\psi(\Pi(A, B), \pi(A, p, B, q), b) = \forall a : \mathrm{Exp}.\forall x : \psi(A, p, a).\psi(B\,a, q(a, x), b\,a)\ .$$

Again, the reader should be aware that we have presented only one crucial case of the complete IIR in [5]. For instance there are clauses corresponding to closure under reductions.

Palmgren's higher-order universes [20]. This construction generalizes Palmgren's super universe [19], that is, a universe which for any family of sets in it contains a universe containing this family.

It is outside the scope of this paper to give a full explanation of higher-order universes, and the interested reader is referred to Palmgren [20]. They are included here as an example of a proof-theoretically strong construction which is subsumed by our theory of IIR: they are conjectured to reach (without elimination rules into arbitrary types) the strength of Kripke-Platek set theory with one recursive Mahlo ordinal. They also provide an example where we have decoding functions $\mathrm{T}_k : \mathrm{U}_k \to D[k]$ with $D[k] = \mathrm{OP}^k(\mathrm{set})$ which depend non-trivially on k and, for $k > 0$, is a higher type which goes beyond set.

The higher-order universes of level n is a family of universes $\mathrm{U}_k, \mathrm{T}_k$, indexed by $k < n$, where U_k is a set of codes for operators on families of sets of level k, and $\mathrm{T}_k : \mathrm{U}_k \to \mathrm{OP}^k(\mathrm{set})$ is the decoding function. The set $\mathrm{OP}^k(\mathrm{set})$ of such operators is defined by introducing more generally for A : type

$$\mathrm{Fam}(A) := (X : \mathrm{set}) \times (X \to A) \ , \qquad \mathrm{Op}(A) := \mathrm{Fam}(A) \to \mathrm{Fam}(A) \ ,$$
$$\mathrm{Op}^n(A) := \underbrace{\mathrm{Op}(\cdots(\mathrm{Op}}_{n \text{ times}}(A))\cdots) \ ,$$

Let

$$n : \mathrm{N}, \qquad A_k : \mathrm{set} \ , \qquad B_k : A_k \to \mathrm{Op}^k(\mathrm{set}) \quad (k = 0, \ldots, n) \ .$$

In the following all sets and constructors are parameterized with respect to n, A_k, B_k, but for simplicity we omit those parameters.

U_0, T_0 have the standard closure properties of a universe. Additionally we have for $k = 0, \ldots, n$ the two constructors (with decodings)

$$\widehat{\mathrm{A}}_k : \mathrm{U}_0 \ , \qquad \mathrm{T}_0(\widehat{\mathrm{A}}_k) = A_k \ ,$$
$$\widehat{\mathrm{B}}_k : A_k \to \mathrm{U}_k \ , \mathrm{T}_k(\widehat{\mathrm{B}}_k(a)) = B_k(a) \ .$$

Furthermore, under the additional assumptions $i \in \{0, \ldots, n-1\}$, $f : \mathrm{U}_{i+1}$, we have two more constructors (with decodings)

$$\mathrm{ap}_i^0(f) : (u : \mathrm{U}_0, \mathrm{T}_0(u) \to \mathrm{U}_i) \to \mathrm{U}_0 \ ,$$
$$\mathrm{T}_0(\mathrm{ap}_i^0(f, u, v)) = \pi_0(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle)) \ ,$$
$$\mathrm{ap}_i^1(f) : (u : \mathrm{U}_0, v : \mathrm{T}_0(u) \to \mathrm{U}_i, \pi_0(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle))) \to \mathrm{U}_i \ ,$$
$$\mathrm{T}_i(\mathrm{ap}_i^1(f, u, v, a)) = \pi_1(\mathrm{T}_{i+1}(f)(\langle \mathrm{T}_0(u), \mathrm{T}_i \circ v \rangle))(a) \ .$$

If we let $I := \{0, \ldots, n\}$ (which more formally should be replaced by N_{n+1}), then we observe that we have introduced $\mathrm{U} : I \to \mathrm{set}$ together with $\mathrm{T} : (i : I, \mathrm{U}(i)) \to D_i$ where $D_i = \mathrm{Op}^i(\mathrm{set})$. For each $i : I$ we can determine a collection of constructors, each of which refers strictly positively to U and (positively and negatively) to T applied to the arguments of U referred to.

Bove and Capretta's analysis of the termination of nested recursive definitions of functional programs. In a recent paper [3], Bove and Capretta use indexed inductive-recursive definitions in their analysis of the termination of functions

defined by nested general recursion. Given such a function f the idea is to simultaneously define a predicate $D(x)$ expressing that $f(x)$ terminates, and a function $f'(x,p)$ which returns the same value as $f(x)$ but has as second argument a proof $p : D(x)$ that $f(x)$ terminates.

Assume for instance the rewrite rules $f(0) \longrightarrow f(f(1))$, $f(1) \longrightarrow 2$, $f(2) \longrightarrow f(1)$ on the domain $\{0,1,2\}$. We now inductive-recursively define the termination predicate D for for f. We get one constructor for each rewrite rule: $C_0 : (p : D(1), q : D(f'(1,p))) \to D(0)$, $C_1 : D(1)$, $C_2 : (p : D(1)) \to D(2)$. Furthermore, the equality rules for f' are $f'(0, C_0(p,q)) = f'(f'(1,p),q)$, $f'(1,C_1) = 2$, $f'(2,C_2(p)) = f'(1,p)$. This is a proper IIR, since in the type of C_0 the second argument depends on $f'(1,p)$, where p is the first argument.

3.3 Restricted Indexed Inductive-Recursive Definitions

There are reasons for focussing attention on IIR where we can determine for every index i the set of constructors introducing elements of U_i. More precisely this means that if C is a constructor of the IIR, and we write its type in uncurried form, then its type is of the form $((i : I) \times A(i)) \to \mathrm{U}_i$, so the first argument determines the index i. An example which satisfies this restriction is the accessible part of a relation.

We can also determine the set of constructors for U_i, where a constructor has type $B \to \mathrm{U}_i$ and i does not depend on B, provided the equality on I is decidable. In this case the type of C can be replaced by $(i' : I) \to \mathrm{C}_2^{\mathrm{type}}(i =_{\mathrm{dec}} i', B, \mathbf{0}) \to \mathrm{U}_{i'}$, where $=_{\mathrm{dec}}$ is the decidable equality on I, with result in $\mathbf{2}$). In this way for example finitely branching trees and forests can be seen to be captured by restricted IIR.

In the implementation of the Half proof system of type theory Thierry Coquand enforced this restriction, and it was kept in the Agda system [4], the successor of Half. One reason is that both the introduction and elimination rules can be specified more simply: for introducing an IIR, the constructors C_i of U_j are just listed in the form data$\{C_1(\boldsymbol{a}_1) \mid \cdots \mid C_n(\boldsymbol{a}_n)\}$, and one doesn't have to include the index j in the arguments of $\boldsymbol{a}_i$ of $C_i(\boldsymbol{a}_i)$ – this simplifies the syntax of the system substantially. If one wants to define $g(x)$ for $x : U_i$, one can write it in the form

$$\begin{aligned} \text{case } x \text{ of } \{C_1(\boldsymbol{a}_1) &\to f_1(\boldsymbol{a}_n); \\ &\cdots \\ C_n(\boldsymbol{a}_n) &\to f_n(\boldsymbol{a}); \} \end{aligned}$$

For defining unrestricted IIR, a more complicated syntax has to be used, especially the elimination rules have to make use of a more general form of pattern matching. See for instance the proof assistant Alf ([12], [2]), where unrestricted IIR can be implemented.

Another reason for this restriction is that it is easier to construct mathematical models: below we will see that restricted IIR can be modelled as initial algebras in an I-indexed slice category. Furthermore, domain-theoretic models of restricted IIR can be given more easily. That complications arise when modelling

unrestricted IIR is one of the reasons why many believe that a fully satisfactory understanding of the identity type has not yet been achieved.

It is often possible to replace general IIR by restricted IIR, especially if we have a decidable equality on the index set. For example, we can define a function by recursion which tests equality of natural numbers. Using this equality we can write the introduction rules for the even number predicate in an alternative way: $\mathrm{Even}(n)$ holds iff $n = 0$ or there exists m such that $n = \mathrm{S}(\mathrm{S}(m))$ and $\mathrm{Even}(m)$. That is, we have two constructors: $\mathrm{C}_0(n) : (n = 0) \to \mathrm{Even}(n)$, and $\mathrm{C}_1(n) : (m : \mathrm{N}) \times (n = \mathrm{S}(\mathrm{S}(m))) \times (\mathrm{Even}(m)) \to (\mathrm{Even}(n))$. (It is also worth mentioning here that alternatively, we can directly define the even numbers by primitive recursion on the natural numbers: $\mathrm{Even}(0)$ is true and $\mathrm{Even}(\mathrm{S}(n))$ is the negation of $\mathrm{Even}(n)$, using the two element universe $\mathbf{2}$ with decoding $T : \mathbf{2} \to \mathrm{set}$.)

However, it is not always possible to transform a definition into the restricted form. The prime example is that of the identity relation. In [11] we however show that if we have extensional equality, we can simulate general IIR by restricted IIR. The definitions are quite complicated though, and the resulting programs may be computationally inefficient.

4 Formalizing the Theory of Indexed Inductive-Recursive Definitions

4.1 The Category of Indexed Families of Types

As for the non-indexed case, we shall derive a formalization of IIR by modeling them as initial algebra constructions in slice categories. Let R be a set of rules for the language of type theory (where each rule is given by a finite set of judgements as premisses and one judgement as conclusion) which includes the logical framework used in this article and an equality. For such R let $\mathrm{TT}(R)$ be the resulting type theory. The category $\mathbf{Type}(R)$ is the category, the objects of which are A s.t. $\mathrm{TT}(R)$ proves $A : \mathrm{type}$, and the morphisms from A to B of which are terms f s.t. $\mathrm{TT}(R)$ proves $f : A \to B$. Objects A, A' such that $\mathrm{TT}(R)$ proves $A = A' : \mathrm{type}$ and functions f, f' s.t. $\mathrm{TT}(R)$ proves $f = f' : B \to C$ are identified. In order to model I-indexed inductive-recursive definitions, where I is an arbitrary stype, we will use the category $\mathbf{Fam}(R, I)$ of I-indexed families of types. An object of $\mathbf{Fam}(R, I)$ is an I-indexed family of types, that is, an abstracted expression A for which we can prove $i : I \Rightarrow A[i] : \mathrm{type}$ in $\mathrm{TT}(R)$. An arrow from A to B is a I-indexed function, that is, an abstracted expression f for which we can prove (in $\mathrm{TT}(R)$) $i : I \Rightarrow f[i] : A[i] \to B[i]$. Again we identify A, A' s.t. we can prove $i : I \Rightarrow A[i] = A'[i] : \mathrm{type}$ and f, f' s.t. we can can prove $i : I \Rightarrow f[i] = f'[i] : B[i] \to C[i]$. We will usually omit the argument R in $\mathbf{Fam}(R, I)$.

If $\mathbf{C}$ is a category, D an object of it, then $\mathbf{C}/D$ is the slice category with objects pairs (A, f) where A is an object of $\mathbf{C}$ and f an arrow $A \to D$, and as morphisms from (A, f) to (B, g) morphisms $h : A \to B$ s.t. $g \circ h = f$. Note

that we write pairs with round brackets on this level. This is different from the notation $\langle a, b\rangle$ for the pair of a and b in the logical framework.

General assumption 4.11 *In the following we assume in all rules* I : stype, $i : I \Rightarrow D[i]$: type *(*D *an abstracted expression).*

4.2 Coding Several Constructors by One

We can code several constructors of an IIR into one as follows: let J be a finite index set for all constructors and A_j be the type of the jth constructor. Then replace all constructors by one constructor of type $(j : J) \to A_j$, which is definable using case distinction on **2**. In case of restricted IIR we can obtain one constructor in restricted form with type $(i : I, j : J) \to A_{ij} \to C_i$, if the type of the jth constructor is $i : I \to A_{ij} \to C_i$.

In this way it will suffice to consider only IIR with one constructor in the sequel.

4.3 Restricted IIR as Initial Algebras in Slice Categories

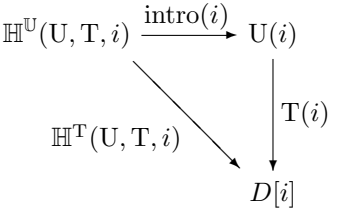
Assume we have a restricted IIR with one constructor

$$\text{intro} : (i : I) \to \mathbb{H}^{\mathrm{U}}(\mathrm{U}, \mathrm{T}, i) \to \mathrm{U}(i)$$

(with all arguments of intro except i in uncurried form), which introduces $\mathrm{U} : I \to \text{set}$ and $\mathrm{T} : (i : I, \mathrm{U}(i)) \to D[i]$. Here $\mathbb{H}^{\mathrm{U}} : (U : I \to \text{set}, T : (i : I, U(i)) \to D[i], I) \to \text{stype}$ with no free occurrences of U, T and i. Let further the equality rule for T be

$$\mathrm{T}(i, \text{intro}(i, a)) = \mathbb{H}^{\mathrm{T}}(\mathrm{U}, \mathrm{T}, i, a) \ ,$$

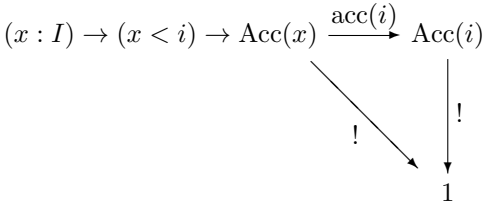
where $\mathbb{H}^{\mathrm{T}} : (U : I \to \text{set}, T : (i : I, U(i)) \to D[i], i : I, \mathbb{H}^{\mathrm{U}}(U, T, i)) \to D[i]$ with no free occurrences of U, T, i or a. Now one observes that the introduction rule and equality rule are captured as an I-indexed family of algebras for $\mathbb{H}$ in $\mathbf{Fam}(I)/D$, where $\mathbb{H}(U, T) = (\mathbb{H}^{\mathrm{U}}(U, T), \mathbb{H}^{\mathrm{T}}(U, T))$.[1]



Note that the situation generalizes the situation for non-indexed induction-recursion [10], where the rules for U and T are captured as an algebra of an appropriate endofunctor $\mathbb{F}$ on the slice category $\mathbf{Type}/D$.

[1] To be pedantic: one has to replace $\mathbb{H}(U, T)$, $\mathbb{H}^{\mathrm{U}}(U, T)$, $\mathbb{H}^{\mathrm{T}}(U, T)$ by uncurried variants $\mathbb{H}'((U, T))$, $\mathbb{H}'^{\mathrm{U}}((U, T))$, $\mathbb{H}'^{\mathrm{T}}((U, T))$

If $D[i] = \mathbf{1}$ and therefore $\mathbb{H}(U, T, i)$ does not depend on T, we have the important special case of a restricted indexed inductive definition. We show the example of the accessible part of a relation:

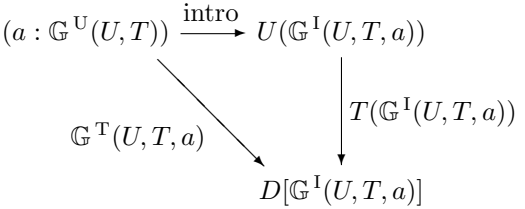


i.e. $\mathbb{H}^{\mathrm{U}}(U, T, i) = (x : I) \to (x < i) \to U(x)$.

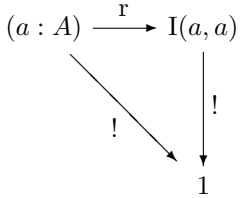
4.4 A Diagram for General IIR

If a particular IIR does not have the restricted form then we do not prima facie know the set of constructors for a particular expression. On the other hand, given an argument to the constructor, we can determine the index of the constructed element.

To capture this situation the categorical representation must be changed. Depending on $U : I \to \mathrm{set}$ and $T : (i : I, U(i)) \to D[i]$ we have an stype $\mathbb{G}^{\mathrm{U}}(U, T)$ of the arguments of the constructor intro, and, depending on $a : \mathbb{G}^{\mathrm{U}}(U, T)$, we have $\mathbb{G}^{\mathrm{I}}(U, T, a) : I$, which provides the i s.t. $\mathrm{intro}(a) : \mathrm{U}_i$, and $\mathbb{G}^{\mathrm{T}}(U, T, a) : D[\mathbb{G}^{\mathrm{I}}(U, T, a)]$, which determines the value of $\mathrm{T}(i, \mathrm{intro}(a))$. The diagram is:



As a first simple instance we look at the identity relation on A. It has index set $I := A \times A$, $D[i] = \mathbf{1}$, and $\mathbb{G}^{\mathrm{U}}(U, T) = A$, $\mathbb{G}^{\mathrm{I}}(U, T, a) = \langle a, a \rangle$, and $\mathrm{U}(\langle a, a \rangle) = \mathrm{I}_A(a, a)$:



As a second illustration we show how to obtain the rules for computability predicates for dependent types. (As in Section 3 we only give a definition containing one case, but the complete definition [5] can be obtained by expanding the definition corresponding to the additional constructors).

$$\begin{aligned}
\mathbb{G}^{\mathrm{U}}(\Psi,\psi) &= (A : \mathrm{Exp}) \times (p : \Psi(A)) \times \\
&\quad (B : \mathrm{Exp}) \times ((a : \mathrm{Exp}) \to \psi(A,p,a) \to \psi(B\ a)) \ , \\
\mathbb{G}^{\mathrm{I}}(\Psi,\psi,\langle A,p,B,q\rangle) &= \Pi(A,B) \ , \\
\mathbb{G}^{\mathrm{T}}(\Psi,\psi,\langle A,p,B,q\rangle) &= (b)\forall a : \mathrm{Exp}.\forall x : \psi(A,p,a).\psi(B\ a, q(a,x), b\ a) \ .
\end{aligned}$$

Note that in the general case we no longer have an endofunctor $\mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D$.

4.5 A Uniform Theory for Restricted and General IIR

If we look at the functors arising in general IIR we observe that we have obtained one stype $\mathbb{G}^{\mathrm{U}}(U,T)$ and two functions

$$\begin{aligned}
&\mathbb{G}^{\mathrm{I}}(U,T) : \mathbb{G}^{\mathrm{U}}(U,T) \to I \ , \\
&\mathbb{G}^{\mathrm{T}}(U,T) : (a : \mathbb{G}^{\mathrm{U}}(U,T)) \to D[\mathbb{G}^{\mathrm{I}}(U,T,a)] \ .
\end{aligned}$$

where (U,T) is an element of $\mathbf{Fam}(I)/D$, $\mathbb{G}^{\mathrm{IT}}(U,T) := \langle \mathbb{G}^{\mathrm{I}}(U,T), \mathbb{G}^{\mathrm{T}}(U,T)\rangle$ an element of $E := (i : I) \times D[i]$, so $(\mathbb{G}^{\mathrm{U}}(U,T), \mathbb{G}^{\mathrm{IT}}(U,T))$ is an element of $\mathbf{Type}/E$. This can be expanded to a functor

$$\mathbb{G} : \mathbf{Fam}(I)/D \to \mathbf{Type}/E \ .$$

In restricted IIR we have for $i : I$

$$\begin{aligned}
&\mathbb{H}^{\mathrm{U}}(U,T,i) : \mathrm{stype} \ , \\
&\mathbb{H}^{\mathrm{T}}(U,T,i) : (a : \mathbb{H}^{\mathrm{U}}(U,T,i)) \to D[i] \ ,
\end{aligned}$$

which can be combined to an endofunctor (with obvious arrow part and extension to U s.t. $i : I \Rightarrow U[i] : \mathrm{type}$)

$$\mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D \ .$$

Consider the functor $\pi_i : \mathbf{Fam}(I)/D \to \mathbf{Type}/D[i]$ with object part $\pi_i(U,T) := (U[i], T[i])$ and obvious arrow part.

Every element $(U,T) : \mathbf{Fam}(I)/D$ is uniquely determined by its projections $\pi_i(U,T)$. One also notes that for every sequence of functors $\mathbb{H}_i : \mathbf{Fam}(I)/D \to \mathbf{Type}/D[i]$ there exists a unique functor $\mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Fam}(I)/D$ s.t. $\pi_i \circ \mathbb{H} = \mathbb{H}_i$. $\mathbb{H}$ and $\mathbb{G}$ will be strictly positive functors in much the same way as $\mathbb{F}$ in [10] . But since $\mathbb{H}$ is determined by $\pi_i \circ \mathbb{H}$ and both $\pi_i \circ \mathbb{H}$ and $\mathbb{G}$ are functors $\mathbf{Fam}(I)/D \longrightarrow \mathbf{Type}/E$, (where $E = (i : I) \times D[i]$ in the general case and $E = D[i]$ in the restricted case), it is more economical to more generally introduce the notion of a strictly positive functor

$$\mathbb{F} : \mathbf{Fam}(I)/D \longrightarrow \mathbf{Type}/E$$

for an arbitrary type E. From this we can derive the functors $\mathbb{G}$ and $\mathbb{H}$. As in [10], we define for $E : \mathrm{type}$ the type of indices for strictly positive functors

$$\frac{E : \mathrm{type}}{\mathrm{OP}_{I,D,E} : \mathrm{type}} \ ,$$

together with, for $\gamma : \mathrm{OP}_{I,D,E}$ (we omit I, D, E, if the parameter γ is given)

$$\mathbb{F}^{\mathrm{U}}_{\gamma} : (U : I \to \mathrm{set}, T : (i : I, U(i)) \to D[i]) \to \mathrm{stype} \ ,$$
$$\mathbb{F}^{\mathrm{T}}_{\gamma} : (U : I \to \mathrm{set}, T : (i : I, U(i)) \to D[i], a : \mathbb{F}^{\mathrm{U}}_{\gamma}(U, T)) \to E \ .$$

(It is straightforward to define arrow parts of these functor and the extension to arguments (U, T) s.t. $i : I \Rightarrow U[i] : \mathrm{type}$.)

We construct elements of $\mathrm{OP}_{I,D,E}$ in a similar way as in the non-indexed case:

- Base Case: This corresponds to having IIR with no arguments of the constructor and one only has to determine the result of E:

$$\iota \ : \ E \to \mathrm{OP}_{I,D,E} \ ,$$
$$\mathbb{F}^{\mathrm{U}}_{\iota(e)}(U, T) = \mathbf{1} \ ,$$
$$\mathbb{F}^{\mathrm{T}}_{\iota(e)}(U, T, \star) = e \ .$$

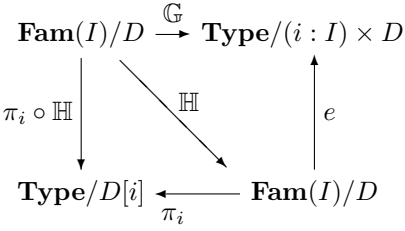
- Nondependent union of functors: This corresponds to the situation where the constructor has an argument A and (depending on $a : A$) further arguments coded as $\gamma(a)$.

$$\sigma \ : \ (A : \mathrm{stype}, \gamma : A \to \mathrm{OP}_{I,D,E}) \to \mathrm{OP}_{I,D,E} \ ,$$
$$\mathbb{F}^{\mathrm{U}}_{\sigma(A,\gamma)}(U, T) = (a : A) \times \mathbb{F}^{\mathrm{U}}_{\gamma(a)}(U, T) \ ,$$
$$\mathbb{F}^{\mathrm{T}}_{\sigma(A,\gamma)}(U, T, \langle a, b\rangle) = \mathbb{F}^{\mathrm{T}}_{\gamma(a)}(U, T, b) \ .$$

- Dependent union of functors: This corresponds to the situation where the constructor has one inductive argument indexed over A. For each element $a : A$ we have to determine an index $i(a) : I$, which indicates the set $U(i(a))$, the inductive argument is chosen from. The result of T for this argument is an element of $(a : A) \to D[i(a)]$ and the other arguments depend on this function. Let $T \circ [i, f] := (x)T(i(x), f(x))$.

$$\delta : (A : \mathrm{stype}, i : A \to I, \gamma : ((a : A) \to D[i(a)]) \to \mathrm{OP}_{I,D,E}) \to \mathrm{OP}_{I,D,E} \ ,$$
$$\mathbb{F}^{\mathrm{U}}_{\delta(A,i,\gamma)}(U, T) = (f : (a : A) \to U(i(a))) \times \mathbb{F}^{\mathrm{U}}_{\gamma(T\circ[i,f])}(U, T) \ ,$$
$$\mathbb{F}^{\mathrm{T}}_{\delta(A,i,\gamma)}(U, T, \langle f, b\rangle) = \mathbb{F}^{\mathrm{T}}_{\gamma(T\circ[i,f])}(U, T, b) \ .$$

We finish this subsection by drawing a diagram which shows the relationship between the functors $\mathbb{G}$ and $\mathbb{H}$ for general and restricted IIR:



where $e(U,T) = ((i : I) \times U(i), (a)\langle \pi_0(a), T(\pi_0(a), \pi_1(a))\rangle)$. So the functors in the restricted case are those which factor through an endofunctor $\mathbb{H}$, which itself is determined by the "fibers" $\pi_i \circ \mathbb{H}$.

The functors for the restricted case have codes of the form

$$\sigma(I, (i) \cdots \delta(A, f, (a) \cdots \delta(B, g, (b) \cdots \sigma(C, (c) \cdots \delta(D, (d) \cdots \iota(\langle i, e\rangle)))))) \ ,$$

ie. the codes always start with $\sigma(I, (i) \ldots$ and the innermost parts are of the form $\iota(\langle i, e\rangle)$ where i is value of the first non-inductive argument. (Note that the order of the constructors in an element of OP might depend on arguments preceding them). That a functor $\mathbb{G}$ has such a code corresponds exactly to the fact that it is obtained by the above diagram from strictly positive functors $\pi_i \circ \mathbb{H}$.

4.6 Formation and Introduction Rules for Restricted IIR

Restricted IIR (indicated by a superscript r) are given by strictly positive endofunctors $\mathbb{H}$ in the category $\mathbf{Fam}(I)/D$, which can be given by their (strictly positive) projections $\pi_i \circ \mathbb{H} : \mathbf{Fam}(I)/D \to \mathbf{Type}/D[i]$. So the set of codes for these functors is given as a family of codes for $\pi_i \circ \mathbb{H}$, and the type of codes is given as

$$\mathrm{OP}^{\mathrm{r}}_{I,D} : \mathrm{type} \ , \qquad \mathrm{OP}^{\mathrm{r}}_{I,D} = (i : I) \to \mathrm{OP}_{I,D,D[i]} \ .$$

Assume now $\gamma : \mathrm{OP}^{\mathrm{r}}_{I,D}$, $U : I \to \mathrm{set}$, $T : (i : I, U(i)) \to D[i]$, $i : I$. The object part of $\mathbb{H}$ (restricted to $U : I \to \mathrm{set}$) is defined as

$$\mathbb{H}^{\mathrm{U}}_{\gamma}(U,T,i) := \mathbb{F}^{\mathrm{U}}_{\gamma(i)}(U,T) : \mathrm{stype}$$
$$\mathbb{H}^{\mathrm{T}}_{\gamma}(U,T,i,a) := \mathbb{F}^{\mathrm{T}}_{\gamma(i)}(U,T,a) : D[i]$$

for $a : \mathbb{H}^{\mathrm{U}}_{\gamma}(U,T,i)$.

We have the following formation rules for $\mathrm{U}^{\mathrm{r}}_{\gamma}$ and $\mathrm{T}^{\mathrm{r}}_{\gamma}$:

$$\mathrm{U}^{\mathrm{r}}_{\gamma}(i) : \mathrm{set} \ , \qquad \mathrm{T}^{\mathrm{r}}_{\gamma}(i) : \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \to D[i] \ .$$

$\mathrm{U}^{\mathrm{r}}_{\gamma}(i)$ has constructor

$$\mathrm{intro}^{\mathrm{r}}_{\gamma}(i) : \mathbb{H}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i) \to \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \ ,$$

and the equality rule for $\mathrm{T}^{\mathrm{r}}_{\gamma}(i)$ is:

$$\mathrm{T}^{\mathrm{r}}_{\gamma}(i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)) = \mathbb{H}^{\mathrm{T}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i, a) \ .$$

4.7 Formation and Introduction Rules for General IIR

In general IIR (as indicated by superscript g) we have to consider strictly positive functors in $\mathbb{G} : \mathbf{Fam}(I)/D \to \mathbf{Type}/(i : I) \times D[i]$. The type of codes is given as

$$\mathrm{OP}^{\mathrm{g}}_{I,D} : \mathrm{type} \ , \qquad \mathrm{OP}^{\mathrm{g}}_{I,D} = \mathrm{OP}_{I,D,(i:I)\times D[i]} \ .$$

Assume now $\gamma : \mathrm{OP}^{\mathrm{g}}_{I,D}$, $U : I \to \mathrm{set}$, $T : (i : I, U(i)) \to D[i]$. The components of $\mathbb{G}$ needed in the following are

$$\begin{aligned}
\mathbb{G}^{\mathrm{U}}_{\gamma}(U,T) &:= \mathbb{F}^{\mathrm{U}}_{\gamma}(U,T) : \mathrm{stype} \\
\mathbb{G}^{\mathrm{I}}_{\gamma}(U,T,a) &:= \pi_0(\mathbb{F}^{\mathrm{T}}_{\gamma}(U,T,a)) : I \\
\mathbb{G}^{\mathrm{T}}_{\gamma}(U,T,a) &:= \pi_1(\mathbb{F}^{\mathrm{T}}_{\gamma}(U,T,a)) : D[\mathbb{G}^{\mathrm{I}}_{\gamma}(U,T,a)]
\end{aligned}$$

for $a : \mathbb{G}^{\mathrm{U}}_{\gamma}(U,T)$.

We have essentially the same formation rules for $\mathrm{U}^{\mathrm{g}}_{\gamma}$ and $\mathrm{T}^{\mathrm{g}}_{\gamma}$

$$\mathrm{U}^{\mathrm{g}}_{\gamma} : I \to \mathrm{set} \ , \qquad \mathrm{T}^{\mathrm{g}}_{\gamma} : (i : I, \mathrm{U}^{\mathrm{g}}_{\gamma}(i)) \to D[i] \ .$$

There is one constructor for all $\mathrm{U}^{\mathrm{g}}_{\gamma}(i)$. Depending on its arguments $\mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a)$ determines the index it belongs to. So the introduction rule is:

$$\mathrm{intro}^{\mathrm{g}}_{\gamma} : (a : \mathbb{G}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma})) \to \mathrm{U}^{\mathrm{g}}_{\gamma}(\mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a)) \ .$$

The equality rule for $\mathrm{T}^{\mathrm{g}}_{\gamma}$ is:

$$\mathrm{T}^{\mathrm{g}}(\mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)) = \mathbb{G}^{\mathrm{T}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a) \ .$$

4.8 Elimination Rules for IIR

We now define the induction principle both for the restricted and the general case. We define first more generally $\mathbb{F}^{\mathrm{IH}}_{\gamma}$ and $\mathbb{F}^{\mathrm{map}}_{\gamma}$ for $\gamma : \mathrm{OP}_{I,D,E}$. Assume F is a twice abstracted expression and

$$\begin{aligned}
&\gamma : \mathrm{OP}_{I,D,E} \ , \quad U : I \to \mathrm{set} \ , \quad T : (i : I, U(i)) \to D[i] \ , \\
&i : I, u : U(i) \Rightarrow F[i,u] : \mathrm{type} \ .
\end{aligned}$$

The rules for $\mathbb{F}^{\mathrm{IH}}$ and $\mathbb{F}^{\mathrm{map}}$ are as follows:

$$\frac{a : \mathbb{F}^{\mathrm{U}}_{\gamma}(U,T)}{\mathbb{F}^{\mathrm{IH}}_{\gamma}(U,T,F,a) : \mathrm{type}} \qquad \frac{h : (i : I, a : U(i)) \to F[i,u]}{\mathbb{F}^{\mathrm{map}}_{\gamma}(U,T,F,h) : (a : \mathbb{F}^{\mathrm{U}}_{\gamma}(U,T)) \to \mathbb{F}^{\mathrm{IH}}_{\gamma}(U,T,F,a)}$$

$$\mathbb{F}^{\mathrm{IH}}_{\iota(e)}(U,T,F,\star) = \mathbf{1} \ ,$$

$$\mathbb{F}^{\mathrm{map}}_{\iota(e)}(U,T,F,h,\star) = \star \ ,$$

$$\mathbb{F}^{\mathrm{IH}}_{\sigma(A,\gamma)}(U,T,F,\langle a,b\rangle) = \mathbb{F}^{\mathrm{IH}}_{\gamma(a)}(U,T,F,b) \ ,$$

$$\mathbb{F}^{\mathrm{map}}_{\sigma(A,\gamma)}(U,T,F,h,\langle a,b\rangle) = \mathbb{F}^{\mathrm{map}}_{\gamma(a)}(U,T,F,h,b) \ ,$$

$$\mathbb{F}^{\mathrm{IH}}_{\delta(A,i,\gamma)}(U,T,F,\langle f,b\rangle) = ((a : A) \to F[i(a), f(a)]) \times \mathbb{F}^{\mathrm{IH}}_{\gamma(T\circ[i,f])}(U,T,F,b) \ ,$$

$$\mathbb{F}^{\mathrm{map}}_{\delta(A,i,\gamma)}(U,T,F,h,\langle f,b\rangle) = \langle h \circ [i,f], \mathbb{F}^{\mathrm{map}}_{\gamma(T\circ[i,f])}(U,T,F,h,b)\rangle \ .$$

In the restricted case we have now under the assumptions

$$\begin{array}{l}\gamma : \mathrm{OP}^{\mathrm{r}}_{I,D} \ , \\ i : I, a : \mathrm{U}^{\mathrm{r}}_{\gamma}(i) \Rightarrow F[i,a] : \mathrm{type} \ , \\ h : (i : I, a : \mathbb{H}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, i), \mathbb{F}^{\mathrm{IH}}_{\gamma(i)}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, F, a)) \to F[i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)] \ ,\end{array}$$

$$\begin{array}{l}\mathrm{R}^{\mathrm{r}}_{\gamma,F}(h) : (i : I, a : \mathrm{U}^{\mathrm{r}}_{\gamma}(i)) \to F[i,a] \ , \\ \mathrm{R}^{\mathrm{r}}_{\gamma,F}(h, i, \mathrm{intro}^{\mathrm{r}}_{\gamma}(i,a)) = h(i, a, \mathbb{F}^{\mathrm{map}}_{\gamma(i)}(\mathrm{U}^{\mathrm{r}}_{\gamma}, \mathrm{T}^{\mathrm{r}}_{\gamma}, F, h, a)) \ .\end{array}$$

And for the general case we have under the assumptions

$$\begin{array}{l}\gamma : \mathrm{OP}^{\mathrm{g}}_{I,D} \ , \\ i : I, a : \mathrm{U}^{\mathrm{g}}_{\gamma}(i) \Rightarrow F[i,a] : \mathrm{type} \ , \\ h : (a : \mathbb{G}^{\mathrm{U}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}), \mathbb{F}^{\mathrm{IH}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, F, a)) \to F[\mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)] \ ,\end{array}$$

$$\begin{array}{l}\mathrm{R}^{\mathrm{g}}_{\gamma,F}(h) : (i : I, a : \mathrm{U}^{\mathrm{g}}_{\gamma}(i)) \to F[i,a] \ , \\ \mathrm{R}^{\mathrm{g}}_{\gamma,F}(h, \mathbb{G}^{\mathrm{I}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, a), \mathrm{intro}^{\mathrm{g}}_{\gamma}(a)) = h(a, \mathbb{F}^{\mathrm{map}}_{\gamma}(\mathrm{U}^{\mathrm{g}}_{\gamma}, \mathrm{T}^{\mathrm{g}}_{\gamma}, F, h, a)) \ .\end{array}$$

In the restricted case one can show that the above rules express that the family $(\mathrm{U}^{\mathrm{r}}_{\gamma}(i), \mathrm{T}^{\mathrm{r}}_{\gamma}(i))$ for $i : I$ is (the carrier of) an initial algebra for the endofunctor $\mathbb{H}$ on $\mathbf{Fam}(I)/D$. This generalizes a theorem in [10] about the connection between inductive-recursive definitions of a set U and a function $\mathrm{T} : \mathrm{U} \to D$ and initial algebras in the slice category $\mathbf{Type}/D$.

Definition 2. *(a) The basic theory of indexed inductive recursive definitions (***Bas-IIR***) consists of the rules for the logical framework as introduced in this article, the formation and introduction rules for* OP *and the defining rules for* $\mathbb{F}^{\mathrm{U}}$, $\mathbb{F}^{\mathrm{T}}$, $\mathbb{F}^{\mathrm{IH}}$ *and* $\mathbb{F}^{\mathrm{map}}$.

(b) The theory $\mathbf{IIR^r}$ *of restricted indexed inductive recursive definitions consists of* **Bas-IIR**, *the defining rules for* OP^{r}, $\mathbb{H}^{\mathrm{U}}$, $\mathbb{H}^{\mathrm{T}}$, *and the formation/introduction/elimination/equality rules for* U^{r} *and* T^{r}.

(c) The theory $\mathbf{IIR^g}$ *of general indexed inductive recursive definitions consists of* **Bas-IIR**, *the defining rules for* OP^{g}, $\mathbb{G}^{\mathrm{U}}$, $\mathbb{G}^{\mathrm{I}}$, $\mathbb{G}^{\mathrm{T}}$, *and the formation/introduction/elimination/equality rules for* U^{g} *and* T^{g}.

(d) $\mathbf{IID^r}$ *and* $\mathbf{IID^g}$ *are the restrictions of* $\mathbf{IIR^r}$ *and* $\mathbf{IIR^g}$, *where in all rules in this article* $D[i] = \mathbf{1}$. *These are the theories of restricted and general indexed inductive definitions.*

5 The Examples Revisited

We first introduce the following abbreviations:

$$\gamma +_{\mathrm{OP}} \gamma' := \sigma(\mathbf{2}, (x)\mathrm{C}_2(x, \gamma, \gamma'))$$

and

$$\gamma_1 +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n := (\cdots((\gamma_1 +_{\mathrm{OP}} \gamma_2) +_{\mathrm{OP}} \gamma_3) +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n)$$

if $n \geq 3$.

So if $\gamma_1, \ldots, \gamma_n$ are codes for constructors C_i then $\gamma_1 +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_n$ is a code for a constructor C. The first argument of C codes an element i of $\{1, \ldots, n\}$. The later arguments of C are the arguments of the constructor C_i.

In the following $(-)$ stands for an abstraction (x) for a variable x, which is not used later. Let $\iota^{\mathrm{g}}_\star(a) := \iota(\langle a, \star\rangle)$, $\iota^{\mathrm{r}}_\star := \iota(\star)$.

- The trees and forests have code $\gamma : \mathrm{OP}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1}}$ $(= \mathbf{2} \to \mathrm{OP}_{\mathbf{2},(-)\mathbf{1},\mathbf{1}})$, where
  $\gamma(\star_0) = \delta(\mathbf{1}, (-)\star_1, (-)\iota^{\mathrm{r}}_\star)$,
  $\gamma(\star_1) = \iota^{\mathrm{r}}_\star +_{\mathrm{OP}} \delta(\mathbf{1}, (-)\star_0, (-)\delta(\mathbf{1}, (-)\star_1, (-)\iota^{\mathrm{r}}_\star))$.
  Then $\mathrm{Tree} = \mathrm{U}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1},\gamma}(\star_0)$, $\mathrm{Forest} = \mathrm{U}^{\mathrm{r}}_{\mathbf{2},(-)\mathbf{1},\gamma}(\star_1)$.
- The even number predicate has code
  $\iota^{\mathrm{g}}_\star(0) +_{\mathrm{OP}} \sigma(\mathrm{N}, (n)\delta(\mathbf{1}, (-)n, (-)\iota^{\mathrm{g}}_\star(\mathrm{S}(\mathrm{S}(n))))) : \mathrm{OP}^{\mathrm{g}}_{\mathrm{N},(-)\mathbf{1}}$
  $(= \mathrm{OP}_{\mathrm{N},(-)\mathbf{1},\mathrm{N}\times\mathbf{1}})$.
- The accessible part of a relation has code
  $(i)\delta((x : I) \times (x < i), (z)\pi_0(z), (-)\iota^{\mathrm{r}}_\star) : \mathrm{OP}^{\mathrm{r}}_{I,(-)\mathbf{1}}$
  $(= (i : I) \to \mathrm{OP}_{I,(-)\mathbf{1},\mathbf{1}})$.
  As a general IIR it has code
  $\sigma(I, (i)\delta((x : I) \times (x < i), (z)\pi_0(z), (-)\iota^{\mathrm{g}}_\star(i))) : \mathrm{OP}^{\mathrm{g}}_{I,(-)\mathbf{1}}$
  $(= \mathrm{OP}_{I,(-)\mathbf{1},I\times\mathbf{1}})$.
- The identity set has code
  $\sigma(A, (a)\iota^{\mathrm{g}}_\star(\langle a, a\rangle)) : \mathrm{OP}^{\mathrm{g}}_{A\times A,(-)\mathbf{1}}$ $(= \mathrm{OP}_{A\times A,(-)\mathbf{1},(A\times A)\times\mathbf{1}})$.
- For the Tait-style computability predicates for dependent types we have $I = \mathrm{Exp}$, $D[i] = \mathrm{Exp} \to \mathrm{set}$. The rules given in Subsection 3.2 are incomplete, additional constructors have to be added by using $+_{\mathrm{OP}}$ (the current definition actually defines the empty set). The code for the constructor given in Subsection 3.2 is

$$\begin{array}{l}\sigma(\mathrm{Exp}, (A)\\ \ \delta(\mathbf{1}, (-)A, (\psi_A)\\ \ \ \sigma(\mathrm{Exp}, (B)\\ \ \ \ \delta((a : \mathrm{Exp}) \times \psi_A(\star, a), (y)(B\,\pi_0(y)), (\psi_B)\\ \ \ \ \ \iota(\langle \Pi(A, B),\ (b)\forall a : \mathrm{Exp}.\forall x : \psi_A(\star, a).\psi_B(\langle a, x\rangle, (b\,a))\rangle)))))\\ : \mathrm{OP}^{\mathrm{g}}_{I,(i)D[i]}\ \ (= \mathrm{OP}_{I,(i)D[i],(i:I)\times D[i]})\ .\end{array}$$

- E. Palmgren's higher order universe has code
  $\gamma : \mathrm{OP}^{\mathrm{r}}_{\{0,\ldots,n\},(l)\mathrm{Op}^l(\mathrm{set})}$
  $(= (k : \{0, \ldots, n\}) \to \mathrm{OP}_{\{0,\ldots,n\},(l)\mathrm{Op}^l(\mathrm{set}),\mathrm{Op}^k(\mathrm{set})})$,
  where

$$\begin{array}{rl}\gamma(0) := & \gamma_{\mathrm{Univ}} +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{A}}_0} +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{A}}_n} +_{\mathrm{OP}} \gamma_{\widehat{\mathrm{B}}_0} +_{\mathrm{OP}}\\ & \gamma_{\mathrm{ap}^0_0} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^0_1} +_{\mathrm{OP}} \cdots +_{\mathrm{OP}} \gamma_{\mathrm{ap}^0_n} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^1_0}\ ,\\ \gamma(i) := & \gamma_{\widehat{\mathrm{B}}_i} +_{\mathrm{OP}} \gamma_{\mathrm{ap}^1_i} \qquad (i = 1, \ldots, n-1)\ ,\\ \gamma(n) := & \gamma_{\widehat{\mathrm{B}}_n}\ .\end{array}$$

and

$$
\begin{aligned}
&\gamma_{\mathrm{Univ}} \text{ is a code for all standard universe constructors,}\\
&\gamma_{\widehat{\mathrm{A}}_k} := \iota(A_k)\ ,\\
&\gamma_{\widehat{\mathrm{B}}_k} := \sigma(A_k, (a)\iota(B_k(a)))\ ,\\
&\gamma_{\mathrm{ap}_i^0} := \delta(\mathbf{1}, (-)(i+1), (T_f)\\
&\qquad \delta(\mathbf{1}, (-)0, (T_u)\\
&\qquad\ \delta(T_u(\star), (-)i, (T_v)\\
&\qquad\quad \iota(\pi_0(T_f(\star, \langle T_u(\star), T_v\rangle)))))))\ ,\\
&\gamma_{\mathrm{ap}_i^1} := \delta(\mathbf{1}, (-)(i+1), (T_f)\\
&\qquad \delta(\mathbf{1}, (-)0, (T_u)\\
&\qquad\ \delta(T_u(\star), (-)i, (T_v)\\
&\qquad\quad \sigma(\pi_0(T_f(\star, \langle T_u(\star), T_v\rangle))), (a)\\
&\qquad\quad\ \iota(\pi_1(T_f(\star), \langle T_u(\star), T_v\rangle)(a))))))\ .
\end{aligned}
$$

(To be formally correct we would have to work with N_{n+1} instead of $\{0, \ldots, n\}$).

6 Further Results

There is an extended version [11] of this article which contains some additional topics.

In particular we show the consistency of our theories by constructing a model in classical set theory where type-theoretic function spaces are interpreted as full set-theoretic function spaces, and some axioms for large cardinals are used for interpreting the type of sets closed under IIR and also to interpret the logical framework. This construction is a generalization of the model for non-indexed induction-recursion given in [9].

Furthermore, we show that the restricted and general form of an IIR are intertranslatable under certain conditions. We also show how to simulate certain forms of IIR by non-indexed inductive-recursive definitions (IR).

References

1. P. Aczel. *Frege Structures and the Notions of Proposition, Truth, and Set*, pages 31–59. North-Holland, 1980.
2. T. Altenkirch, V. Gaspes, B. Nordström, and B. von Sydow. A user's guide to ALF. http://www.www.cs.chalmers.se/ComputingScience/Research/Logic/alf/guide.html, 1996.
3. A. Bove and V. Capretta. Nested general recursion and partiality in type theory. To appear in the Proceedings of TPHOL 2001.
4. C. Coquand. Agda. 2000. http://www.cs.chalmers.se/ catarina/agda/.
5. C. Coquand. A realizability interpretation of Martin-Löf's type theory. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998.

6. P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.
7. P. Dybjer. Inductive families. *Formal Aspects of Computing*, 6:440–465, 1994.
8. P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65(2), June 2000.
9. P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. In J.-Y. Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, April 1999.
10. P. Dybjer and A. Setzer. Induction-recursion and initial algebras. Submitted for publication, April 2000.
11. P. Dybjer and A. Setzer. Indexed induction-recursion. To appear as a Report of Institut Mittag-Leffler, 2001.
12. L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In *Types for proofs and programs*, volume 806 of *Lecture Notes in Computer Science*, pages 213–317. Springer, 1994.
13. P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium '73*, pages 73–118. North-Holland, 1975.
14. P. Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science, VI, 1979*, pages 153–175. North-Holland, 1982.
15. P. Martin-Löf. On the meaning of the logical constants and the justification of the logical laws, 1983. Notes from a series of lectures given in Siena.
16. P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
17. P. Martin-Löf. An intuitionistic theory of types. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998. Reprinted version of an unpublished report from 1972.
18. B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory: an Introduction*. Oxford University Press, 1990.
19. E. Palmgren. *On Fixed Point Operators, Inductive Definitions and Universes in Martin-Löf's Type Theory*. PhD thesis, Uppsala University, 1991.
20. E. Palmgren. On universes in type theory. In G. Sambin and J. Smith, editors, *Twenty-Five Years of Constructive Type Theory*. Oxford University Press, 1998.
21. C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings Typed λ-Calculus and Applications*, pages 328–245. Springer-Verlag, LNCS, March 1993.

Modeling Meta-logical Features in a Calculus with Frozen Variables

Birgit Elbl

Institut für Theoretische Informatik und Mathematik,
Fakultät für Informatik, UniBw München, 85577 Neubiberg, Germany
birgit@informatik.unibw-muenchen.de

Abstract. We consider logic programming computations involving meta-logical predicates and connectives. The meaning of these elements depends on structural properties of the arguments, e.g. being an uninstantiated variable or a ground term when the goal is called, or involve success / failure conditions for the components which relates them to control. To model these effects we use a substructural calculus and introduce a binding mechanism at the level of sequents called freezing.

Keywords: logic programming, substructural logic, control

1 Introduction

In logic programming the basic data structure is the algebra of terms where terms need not be ground but may contain variables. A predicate takes terms as arguments and in case of a successful evaluation substitutions for their variables are returned. The substituted terms usually contain still variables which may be instantiated by further subcomputations. In this setting tests for the property of being a variable or ground or a procedure to produce a 'fresh' variant of a term seem quite natural primitives. They are, however, usually classified as 'meta-logical', as variables are no longer 'logical' in the sense that they are just representing all (ground) instances but turn into objects in their own right. An axiomatization should take this into account. A calculus for deriving statements concerning the results which enjoys the substitution property can not treat these expressions as variables, as e.g. $\texttt{var}(x)$ succeeds, although no ground instance does, $\texttt{ground}(x)$ fails, although no ground instance does etc.

A similar phenomenon occurs when considering additional connectives. The arguments of the connectives are goals, usually with variables. It is quite natural to accept here conditions on success or failure of the goal "as it is" but then variables change their status: the success of a goal does not imply success of all its instances, the failure of the test for failing of a goal does not imply that this is the case for all instances etc. Note that, although these features belong to the 'control' group, their meaning can be easily explained as a function on results without referring to operational details. Hence it seems appropriate to ask for a proof-theoretic description.

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 114–129, 2001.

A possible solution is a deductive system in the style of [11,9]. There `not` and `cut` have already been considered. Program variables just are no longer 'logical' in the sense above. Here we try a different route: we want to keep free variables in general, serving to describe unification and value passing in terms of equality axioms and substitution and to further compositional descriptions of the ordinary connectives, as these do involve judgments about all instances. The compromise investigated here is to introduce a binding mechanism at the level of sequents, not connected with introducing a logical symbol and turning a variable in a state between 'free' and 'special constant'. We do not request the substitution property for these variables but with respect to rule application they are treated as variables. They can be turned into regular bound variables by quantification. The term 'frozen' is chosen to emphasize that they loose their ability to vary and also to resemble the 'freeze' mechanism sometimes considered in extensions (see [12] for a discussion of this feature and meta-logical predicates in general) where also variables are turned into true objects. In contrast to this approach, however, the frozen variables here do not behave like constants.

In the sequel we proceed as follows: first we introduce the calculus with frozen variables and formulate axioms and rules for meta-logical features, then we present a model based on streams of answers and show the soundness of the calculus w.r.t. this interpretation. In order to relate it to a procedural interpretation we present an operational semantics of the language with meta-logical features and show how successful and failing computations can be mapped to derivations of the corresponding judgment.

2 A Calculus with Frozen Variables

2.1 Introducing Frozen Variables

We presuppose fixed sets VAR of variables, and PRED_n of n-ary predicate symbols. Further basic symbols are given by a signature Σ. The expressions are either (object) terms or goals, also referred to as expressions of sort ι or o respectively. The signature consists of constructor symbols $c\colon \iota^n \longrightarrow \iota$ to fix the term algebra, symbols $b\colon \iota^n \longrightarrow o$ for 'built-in' predicates and connectives $\circ\colon o^n \longrightarrow o$. The latter include $\mathbf{1}, \mathsf{F}, \mathbf{0}\colon o$ and $\otimes, *\colon o \times o \longrightarrow o$ for which we use infix notation. Terms (of type ι) are built from variables and constructor symbols as usual.

Definition 1. *The set of goals is inductively defined by:*

- $\mathbf{1}, \mathsf{F}, \mathbf{0}$ *are goals, and* $p(\bar{t})$ *is a goal if* p *is an* n*-ary predicate symbol or built-in* $p\colon \iota^n \longrightarrow o$ *and* $\bar{t} = t_1, \ldots, t_n$ *are terms.*
- *If* $G_1, \ldots, G_n$ *are goals and* $\circ\colon o^n \longrightarrow o$ *then* $\circ(G_1, \ldots, G_n)$ *is a goal. If* G *is a goal and* x *is a variable then* $\mathsf{E}xG$ *is a goal.*

A goal $\mathsf{E}\bar{u}(\bar{x} = \bar{t})$ *where* $\bar{t} = t_1, \ldots, t_n$*, and* $\bar{x} = x_1, \ldots, x_n$ *are pairwise distinct variables that do not occur in* $\bar{t}$ *or* $\bar{u}$ *is called* positive answer.

If Γ *and* Δ *are finite lists of goals and* d *is a finite set of variables then* $\Gamma \Rightarrow^d \Delta$ *is a sequent.*

The quantifier $\mathsf{E}x$ binds the mentioned variable. In the sequel goals that are equal up to bound renaming are identified. The notation FV is used to refer to the set of free variables, and $\bar{t} = \bar{s}$ is a shorthand for $t_1 = s_1 \otimes \ldots \otimes t_n = s_n$ if $\bar{t} = t_1, \ldots, t_n$ and $\bar{s} = s_1, \ldots, s_n$. In a sequent $\Gamma \Rightarrow^d \Delta$ the variables in d are *frozen*. Although there is no binding construction in the formulas, these variables are not treated as regular *free* variables: if x is frozen, we do not expect that the derivability of $\Gamma \Rightarrow^d \Delta$ implies the derivability of $\Gamma\{t/x\} \Rightarrow^d \Delta\{t/x\}$.

Equality is used for unification (with occur check), $\otimes$ for goal conjunction, $*$ for disjunction. The constant F corresponds to *fail*, $\mathbf{1}$ to a prolog atom defined by $\mathbf{1}.$, $\mathbf{0}$ to a prolog atom defined by $\mathbf{0} :- \mathbf{0}$. The name "positive answer" reflects the fact that these goals represent answer substitutions.

Sequents $\Gamma \Rightarrow^d \Delta$ are derived in a substructural calculus $\mathrm{K}_{\mathrm{frz}}$. Similar to linear logic [7] the calculus $\mathrm{K}_{\mathrm{frz}}$ has no weakening and contraction rules. All connectives used here are multiplicatives and non-commutative. In contrast to linear logic the multiplicatives are not dual to each other. Furthermore, the quantifier is not standard. The calculus $\mathrm{K}_{\mathrm{frz}}[\emptyset]$ is obtained from a fragment of the calculus LPC^+ used in [4] to define a declarative semantics for pure prolog+negation which is sufficient for definite programs by adding frozen variables. There is no universal quantification but it is understood that whenever we formulate an axiom we want to accept every instance resulting by substitution for non-frozen free variables and renaming frozen ones as an axiom. There is no implication but the "top-level" implication corresponding to the sequent arrow. The notation $G \Leftrightarrow^d H$ is used to refer to the sequents $G \Rightarrow^d H$ and $H \Rightarrow^d G$. An inference rule with premises $S_1, \ldots, S_n$ $(n \geq 0)$ and conclusion S is written $S_1, \ldots, S_n \vdash S$.

Definition 2. *The axioms and rules of the calculus $K_{frz}[\emptyset]$ are as follows:*

1. *identity and structural:* $\vdash G \Rightarrow^d G$, *and* $(\Gamma \Rightarrow^d G), (G \Rightarrow^e \Delta) \vdash \Gamma \Rightarrow^{d \cup e} \Delta$, *and* $\Gamma \Rightarrow^d \Delta \vdash \Gamma \Rightarrow^{d \cup e} \Delta$
2. *propositional connectives:*
   $\vdash \mathbf{0}, \Gamma \Rightarrow^d \Delta$
   $\Gamma, G_1, G_2, \Pi \Rightarrow^d \Delta \vdash \Gamma, G_1 \otimes G_2, \Pi \Rightarrow^d \Delta$
   $(G_1 \Rightarrow^d \Delta), (G_2 \Rightarrow^e \Sigma) \vdash G_1 * G_2 \Rightarrow^{d \cup e} \Delta, \Sigma$
   $(\Gamma \Rightarrow^d G_1), (\Pi \Rightarrow^e G_2) \vdash \Gamma, \Pi \Rightarrow^{d \cup e} G_1 \otimes G_2$ *if* $e \cap \mathrm{FV}(\Gamma) = \emptyset$
   $\Gamma \Rightarrow^d \Delta, G_1, G_2, \Sigma \vdash \Gamma \Rightarrow^d \Delta, G_1 * G_2, \Sigma$
   $\Gamma, \Pi \Rightarrow^d \Delta \vdash \Gamma, \mathbf{1}, \Pi \Rightarrow^d \Delta$, *and* $\vdash \Rightarrow^d \mathbf{1}$
   $\vdash \mathsf{F} \Rightarrow^d$, *and* $\Gamma \Rightarrow^d \Delta, \Sigma \vdash \Gamma \Rightarrow^d \Delta, \mathsf{F}, \Sigma$
   $\vdash \mathsf{F} \Rightarrow^d \mathsf{F} \otimes G$, *and* $\vdash (G_1 \otimes H) * (G_2 \otimes H) \Rightarrow^d (G_1 * G_2) \otimes H$
3. *quantification*
   $\Gamma, G_1, \Pi \Rightarrow^d \Delta \vdash \Gamma, \mathsf{E}xG_1, \Pi \Rightarrow^{d \setminus \{x\}} \Delta$ *!(x)*
   $\Gamma, G_1, \Pi \Rightarrow^d G_2 \vdash \Gamma, \mathsf{E}xG_1, \Pi \Rightarrow^{d \setminus \{x\}} \mathsf{E}xG_2$ *!(x)*
   $\Gamma, \Pi \Rightarrow^d G_2 \vdash \Gamma, \Pi \Rightarrow^{d \setminus \{x\}} \mathsf{E}xG_2$ *!(x)*
   $\vdash \mathsf{E}x\mathsf{E}yG \Rightarrow^d \mathsf{E}y\mathsf{E}xG$
4. *mixed:*
   $\vdash \mathsf{E}xG_1 * \mathsf{E}xG_2 \Rightarrow^d \mathsf{E}x(G_1 * G_2)$
   $\vdash \mathsf{E}x(G_1 \otimes G_2) \Rightarrow^d \mathsf{E}xG_1 \otimes G_2$ *!(x)*
   $\vdash \mathsf{E}x(G_1 \otimes G_2) \Rightarrow^d G_1 \otimes \mathsf{E}xG_2$ *!(x)*

5. *equality:*

$(=_1)$ $t = t \Leftrightarrow^d \mathbf{1}$
$(=_2)$ $t = s \Leftrightarrow^d s = t$
$(=_3)$ $x = t \Leftrightarrow^d \mathsf{F}$ *if* x *occurs in* t *but* $x \not\equiv t$
$(=_4)$ $f(\bar{t}) = g(\bar{s}) \Leftrightarrow^d \mathsf{F}$ *if* $f \not\equiv g$
$(=_5)$ $\bar{t} = \bar{s} \Leftrightarrow^d f(\bar{t}) = f(\bar{s})$
$(=_6)$ $x = t \otimes A \Leftrightarrow^d x = t \otimes A\{t/x\}$
$(C_=)$ $t = s \otimes t' = s' \Leftrightarrow^d t' = s' \otimes t = s$
$(D_=)$ $t = s \otimes (A * B) \Leftrightarrow^d (t = s \otimes A) * (t = s \otimes B)$
$(\mathsf{F}_=)$ $t = s \otimes \mathsf{F} \Leftrightarrow^d \mathsf{F}$
$(\mathsf{E}/=)$ $\mathsf{E}x(x = t) \Leftrightarrow^d \mathbf{1}$ *if* x *does not occur in* t *or* $t \equiv x$
$(\otimes/=)$ $G \Rightarrow^d H \vdash x = t, G \Rightarrow^{d\cup\{x\}} x = t \otimes H$ *if* $x \notin d$

Here G_1, G_2, G, H *are arbitrary formulas,* $\Gamma, \Pi, \Delta, \Sigma$ *finite lists of formulas,* d, e *finite sets of variables, and* $!(x)$ *stands for the eigenvariable condition:* x *must not occur free in the conclusion.*

All axioms and rules are correct w.r.t. a classical interpretation, ignoring the variable set attached to the sequent arrow. The only rules subject to a condition on the additional set of variables are $(\Rightarrow \otimes)$ and $(\otimes/=)$. Variables are removed from this set ('melted') only in the (E) rules. Note that in all cases the melted variable occurs only bound in the conclusion. Hence frozen variables are not turned into free variables. The axioms $(=_1) - (=_6)$ are appropriate versions of Clark's equality axioms. In $(C_=), (D_=), (\mathsf{F}_=)$ laws are formulated for equality which hold no longer in general. These axioms have been used already in the version without frozen variables. In a similar way we added now $(\otimes/=)$: without frozen variables it is derivable using the $\otimes$-rule, now it is no longer admissible for arbitrary formulas but we add it for equations. In this presentation the general axioms and rules are separated from those determined by a specific extension but obviously the frozen variables are superfluous if nothing further is added: then $\vdash \Gamma \Rightarrow^d \Delta$ for *some* d implies $\vdash \Gamma \Rightarrow^e \Delta$ for *every* e.

2.2 Modeling Meta-logical Features

First we consider some predicates for structure inspection and manipulation.

Definition 3. *Let* Σ_{struct} *be the signature which contains additional predicate symbols* $\mathbf{var}, \mathbf{ground}\colon \iota \longrightarrow o$ *and* $\mathbf{copy}\colon \iota \times \iota \longrightarrow o$. *The set* T_{struct} *consists of:*

$\Rightarrow^{d\cup\{x\}} \mathbf{var}(x)$
$\mathsf{F} \Rightarrow^d \mathbf{var}(c(\bar{x}))$ *for* $c\colon \iota^n \longrightarrow \iota$

$\mathbf{ground}(x_1), \ldots, \mathbf{ground}(x_n) \Rightarrow^d \mathbf{ground}(c(x_1, \ldots, x_n))$ *for* $c\colon \iota^n \longrightarrow \iota$
$\mathsf{F} \Rightarrow^{d\cup\{x\}} \mathbf{ground}(x)$

$\mathsf{E}\bar{u}(s = t\{\bar{u}/\bar{x}\}) \Rightarrow^{d\cup\{\bar{x}\}} \mathbf{copy}(t, s)$ *where* $\mathrm{FV}(t) = \{\bar{x}\}, \{\bar{u}/\bar{x}\}$ *a renaming, and* $\mathrm{FV}(s) \cap \{\bar{u}\} = \emptyset$

To demonstrate the effect of copy consider the following way to define a test for the variable property in presence of two distinct constants c, d:

$$v(t) :\equiv \mathsf{E}u\mathsf{E}v(\mathsf{copy}(t,u) \otimes \mathsf{copy}(t,v) \otimes u = c \otimes v = d)$$

Then $\mathrm{K}_{\mathrm{frz}}[T_{\mathrm{struct}}] \vdash \Rightarrow^{\{x\}} v(x)$ and $\mathrm{K}_{\mathrm{frz}}[T_{\mathrm{struct}}] \vdash \mathsf{F} \Rightarrow^{\{\bar{x}\}} v(f(\bar{x}))$.

The second group consists of connectives which depend on a simple condition on the result for their arguments.

Definition 4. *Let Σ_{cond} be the signature which contains additional connectives* fail_if, succeed_if, once, *and $(\cdot \to \cdot;\cdot)$. The set T_{cond} consists of:*

$\mathsf{F} \Rightarrow^d G \vdash \Rightarrow^d \mathit{fail\_if}(G)$
$A * \mathbf{0} \Rightarrow^d G \vdash \mathsf{F} \Rightarrow^{d\cup e} \mathit{fail\_if}(G)$ *where* $\mathrm{FV}(G, A) = e$ *and A is a positive answer*
$\mathsf{F} \Rightarrow^d G \vdash \mathsf{F} \Rightarrow^d \mathit{succeed\_if}(G)$
$A * \mathbf{0} \Rightarrow^d G \vdash \Rightarrow^{d\cup e} \mathit{succeed\_if}(G)$ *where* $\mathrm{FV}(G, A) = e$ *and A is a positive answer*
$\mathsf{F} \Rightarrow^d G \vdash \mathsf{F} \Rightarrow^d \mathit{once}(G)$
$A * \mathbf{0} \Rightarrow^d G \vdash A \Rightarrow^{d\cup e} \mathit{once}(G)$ *where* $\mathrm{FV}(G, A) = e$ *and A is a positive answer*
$G \Rightarrow^d H \vdash \mathit{fail\_if}(G) \Rightarrow^d \mathit{fail\_if}(H)$
$G \Rightarrow^d H \vdash \mathit{succeed\_if}(G) \Rightarrow^d \mathit{succeed\_if}(H)$
$G \Rightarrow^d H \vdash \mathit{once}(G) \Rightarrow^d \mathit{once}(H)$
$(\mathsf{F} \Rightarrow^d G), (\Gamma \Rightarrow^e H_2) \vdash \Gamma \Rightarrow^{d\cup e} (G \to H_1; H_2)$
$(A * \mathbf{0} \Rightarrow^{d_1} G), (\Gamma \Rightarrow^{d_2} H_1) \vdash A, \Gamma \Rightarrow^{d_1 \cup d_2 \cup e} (G \to H_1; H_2)$
where A is a positive answer, $\mathrm{FV}(A) \cap d_2 = \emptyset$, *and* $\mathrm{FV}(G, A) = e$
$(G \Rightarrow^d G'), (H_1 \Rightarrow^{e_1} H_1'), (H_2 \Rightarrow^{e_2} H_2') \vdash (G \to H_1; H_2) \Rightarrow^{d\cup e_1\cup e_2} (G' \to H_1'; H_2')$
if $\mathrm{FV}(G) \cap e_1 = \emptyset$.

The signature Σ and the theory T_Σ determine a language. Programs are sets of recursive predicate definitions in this language.

Definition 5. *A predicate definition is a sequent $\mathit{Def}_p^P(\bar{x}) \Rightarrow p(\bar{x})$ where $\mathit{Def}_p^P(\bar{x})$ is a goal, p is in* PRED_n, *and $\bar{x} = x_1, \ldots, x_n$ are pairwise distinct variables containing the free variables in $\mathit{Def}_p^P(\bar{x})$. A program is a finite set of predicate definitions containing for every predicate name at most one definition.*

A sequent is derivable in $K_{frz}[T]$ ($T = \emptyset$ or $T = T_{struct}$ or $T = T_{cond}$ or $T = T_{struct} \cup T_{cond}$) from P if it is deducible using the axioms and rules in $K_{frz}[\emptyset]$ and T and axioms $\mathit{Def}_p^P(\bar{t}) \Rightarrow^d p(\bar{t})$ which are obtained by substitution (and bound renaming if necessary) from elements in P. Derivability of $\Gamma \Rightarrow^d \Delta$ in $K_{frz}[T]$ from P is denoted by $(K_{frz}[T], P) \vdash \Gamma \Rightarrow^d \Delta$.

The notation $\mathrm{K}_{\mathrm{frz}}[T] \vdash \Gamma \Rightarrow^d \Delta$ is used for $(\mathrm{K}_{\mathrm{frz}}[T], \emptyset) \vdash \Gamma \Rightarrow^d \Delta$.

Lemma 6. *Let $T = \emptyset$ or $T = T_{struct}$ or $T = T_{cond}$ or $T = T_{struct} \cup T_{cond}$. Let $\Gamma \Rightarrow^{\{\bar{x}\}} \Delta$ be a sequent and θ a substitution so that θ restricted to $\{\bar{x}\}$ is a renaming and no variable $x_i\theta$ occurs in any term $y\theta$ where $y \not\equiv x_i$.*

If $(K_{frz}[T], P) \vdash \Gamma \Rightarrow^{\{\bar{x}\}} \Delta$ then $(K_{frz}[T], P) \vdash \Gamma\theta \Rightarrow^{\{\bar{x}\theta\}} \Delta\theta$.

Proof. Induction on the height of the derivation.

Here we assume that bound variables in Γ, Δ are renamed if necessary for the application of θ. The second condition on θ serves a similar purpose: the unintended freezing of variables in substituted terms is avoided. For example, if c is a constant then $\mathrm{K}_{\mathrm{frz}}[T_{\mathrm{struct}}] \vdash y = c \Rightarrow^{\{x\}} y = c \otimes \mathtt{var}(x)$ but $\mathrm{K}_{\mathrm{frz}}[T_{\mathrm{struct}}] \nvdash x = c \Rightarrow^{\{x\}} x = c \otimes \mathtt{var}(x)$.

For defined predicates we added only an implication as axiom. This is sufficient for deriving success statements, also including failure. Here failure is formalized by $\mathsf{F} \Rightarrow^d G$, hence by an expression in which G occurs *positive*. In order to strengthen the power concerning general statements one could add further axioms $p(\bar{t}) \Rightarrow^d \mathrm{Def}_p^P(\bar{t})$. An alternative axiomatization could be based on rules $(\Gamma \Rightarrow^d \mathrm{Def}_p^P(\bar{t})) \vdash (\Gamma \Rightarrow^d p(\bar{t}))$ and $(\mathrm{Def}_p^P(\bar{t}) \Rightarrow^d \Delta) \vdash (p(\bar{t}) \Rightarrow^d \Delta)$, or for prolog-style definitions $\mathrm{Def}_p^P(\bar{x}) \equiv *_{i=1}^m \mathsf{E}\bar{u}^{(i)}(\bar{x} = \bar{s}^{(i)} \otimes (\otimes \Pi_i))$ (where $\{\bar{x}\} \cap \{\bar{u}^{(i)}\} = \emptyset$, $\mathrm{FV}(\bar{s}^{(i)}, \Pi_i) = \{\bar{u}\}^{(i)}$, and Π_i contains only atomic formulas):

$$(\Gamma \Rightarrow^d \mathsf{E}\bar{u}^{(1)}(\bar{t} = \bar{s}^{(1)} \otimes (\otimes \Pi_1)), \ldots, \mathsf{E}\bar{u}^{(m)}(\bar{t} = \bar{s}^{(m)} \otimes (\otimes \Pi_m))) \vdash \Gamma \Rightarrow^d p(\bar{t})$$
$$(\mathsf{E}\bar{u}^{(i)}(\bar{t} = \bar{s}^{(i)} \otimes (\otimes \Pi_i)) \Rightarrow \Delta_i)_{i=1,\ldots,m} \vdash p(\bar{t}) \Rightarrow \Delta_1, \ldots, \Delta_m$$

with the usual condition on variables. More in the style of the calculus with definitional reflection in [10] is the version

$$(\Gamma \Rightarrow^d \mathsf{E}\bar{u}^{(i_1)}(\hat{\sigma}_1 \otimes (\otimes \Pi_{i_1}\sigma_1)), \ldots, \mathsf{E}\bar{u}^{(i_k)}(\hat{\sigma}_k \otimes (\otimes \Pi_{i_k}\sigma_k))) \vdash \Gamma \Rightarrow^d p(\bar{t})$$
$$(\mathsf{E}\bar{u}^{(i_\nu)}(\hat{\sigma}_\nu \otimes (\otimes \Pi\sigma_\nu)) \Rightarrow^d \Delta_\nu)_{\nu=1,\ldots,k} \vdash p(\bar{t}) \Rightarrow^d \Delta_1, \ldots, \Delta_k$$

where $i_1, \ldots, i_k$ are the numbers of clauses with a head that unifies with $p(\bar{t})$, σ_ν the mgu and $\hat{\sigma}_\nu$ is a representing formula for σ_ν. Note, however, that it is necessary to include $\hat{\sigma}_\nu$ even in the second rule to avoid unsound inferences. To see this, consider as an example the clauses $((p(c) :- q(d)); q(d).)$. Using the rules above one could derive $q(d) \Rightarrow \mathbf{1}$, hence $x = c \otimes q(d) \Rightarrow x = c$, and finally $p(x) \Rightarrow x = c$. To infer $p(x) \Rightarrow \mathbf{1}$ would be wrong as $p(d) \Rightarrow \mathbf{1}$ is no consequence. Also for the basic connectives as well as for the conditional group in Σ_{cond} one could add further axioms. The test they would have to pass is the semantics in Sec.3.

In the version above we only have some rules concerning results and for each connective in Σ_{cond} a rule expressing monotonicity. The rule for $(\cdot \to \cdot; \cdot)$ and one of the rules for $\otimes$, however, are subject to conditions on the free and frozen variables. Hence, although all connectives are positive, the derivability of $G \Rightarrow^d H$ does not imply derivability of $\mathcal{C}[G] \Rightarrow^{d \cup e(\mathcal{C})} \mathcal{C}[H]$ (for $e(\cdot)$ still to be defined) for *arbitrary* contexts $\mathcal{C}$. The rule $(G \Rightarrow^d H) \vdash (A \otimes G \Rightarrow^d A \otimes H)$ would even lead to unsound inferences as can be exemplified by $G \equiv \mathbf{1}$, $H \equiv \mathtt{var}(x)$, $d = \{x\}$, $A \equiv (x = c)$. For a specific class of contexts this monotonicity property does hold and this will be proved and used in Sec. 4.2 where the calculus is related to an operational semantics.

3 Semantics

3.1 Preliminaries

For the semantics, domains and operations are used which have been introduced in [4] and used in a denotational semantics for pure prolog. Fixed point semantics have been suggested before, see [8],[3],[1]. In the sequel all necessary ingredients are presented briefly (without any proofs). For more details the reader is referred to [4], Sections 2 and 5.1, and [6] (for $\mathbb{G}, \mathbb{P}$ and the extension to control).

The original application is a semantics for pure prolog. Here we turn things round, consider the domains as given and ask for ways to describe their elements and operations on them. Nevertheless, although the programming language prolog is not used here, some general concepts of it motivate the definitions. The arguments for a predicate are terms $\bar{t}$ and we assume an evaluation that produces substitutions θ for the variables in $\bar{t}$. A finite set $\{t_1/x_1, \ldots, t_n/x_n\}$, where $x_1, \ldots, x_n$ are distinct variables and $t_1, \ldots, t_n$ are terms, is called *finite substitution*. Similar to [3], this notion of finite substitution takes into account the domain: We allow bindings x/x in a substitution and distinguish between θ and $\theta \cup \{x/x\}$. The presence of a variable x in a substitution θ implies that x is not just an auxiliary variable. The choice of *auxiliary* variables should be irrelevant, hence $\{f(x)/y\}$ equivalent to $\{f(u)/y\}$ if considered as answers for a goal $?- p(v, y)$. But considered as answers for a goal $?- p(x, y)$ they are essentially different: the first states a relation between the arguments, the second doesn't. This difference is reflected in the domain of variables: If we want to state the relation between x and y we use the substitution $\{f(x)/y, x/x\}$, equivalently $\{f(u)/y, u/x\}$, otherwise $\{f(u)/y\}$, equivalently $\{f(x)/y\}$. Using the relation $\sim$, we abstract further from syntactic properties, concentrating on the fact whether a condition involving a variable x is produced or not, identifying the situations where this is due to the fact that x does not *occur* in the goal or the specific predicate definition.

Let FSubst be the set of finite substitutions. Elements of FSubst are denoted by $\theta, \sigma, \delta, \rho$, possibly indexed, or explicitly by $\{\bar{t}/\bar{x}\}$. The domain and the set of variables in the range are defined by $\mathrm{Dom}(\{\bar{t}/\bar{x}\}) = \{\bar{x}\}$ and $\mathrm{Rg}(\{\bar{t}/\bar{x}\}) = \mathrm{VAR}(t_1) \cup \ldots \cup \mathrm{VAR}(t_n)$. For every finite set d of variables, FSubst_d denotes the set of finite substitutions with domain d. If $\bar{y} = y_1, \ldots, y_n$ are distinct variables, we call $\delta = \{\bar{y}/\bar{x}\}$ a *renaming* and use δ^{-1} for $\{\bar{x}/\bar{y}\}$. Furthermore $\mathrm{id}_{\{\bar{x}\}}$ stands for $\{\bar{x}/\bar{x}\}$. We use the following notations for operations on FSubst:

$$\begin{aligned}
&\theta \downarrow d := \{t/x \mid (t/x) \in \theta \text{ and } x \in d\} \quad (d \subset \mathrm{VAR}) \\
&\theta - x := \theta \downarrow (\mathrm{Dom}(\theta) \setminus \{x\}) \text{ for variables } x \\
&\theta \cdot \sigma := \{t\sigma/x \mid (t/x) \in \theta\}
\end{aligned}$$

The operation $\cdot$ can be defined as $(\theta \circ \sigma) \downarrow \mathrm{Dom}(\theta)$ where $\circ$ stands for the usual composition, and vice versa $\theta \circ \sigma = (\theta \cdot \sigma) \cup (\sigma \downarrow (\mathrm{Dom}(\sigma) \setminus \mathrm{Dom}(\theta)))$ (if bindings x/x are not removed from $\theta \circ \sigma$). For $\theta, \sigma \in \mathrm{FSubst}$, $\theta \approx \sigma$ iff $\theta \cdot \delta = \sigma$ for a renaming $\delta \in \mathrm{FSubst}_{\mathrm{Rg}(\theta)}$. If $\mathrm{Dom}(\theta) \cap \mathrm{Dom}(\sigma) = \emptyset$ and also $\mathrm{Rg}(\theta) \cap \mathrm{Rg}(\sigma) = \emptyset$,

we call θ and σ *disjoint* and use the notation $\theta \cup \sigma$ for $\theta \cup \sigma$. Let $D(\theta) := \bigcap\{d \subset \mathrm{Dom}(\theta) \mid (\theta \downarrow d) \cup \delta = \theta \text{ for some renaming } \delta\}$ for all $\theta \in \mathrm{FSubst}$ and $\sigma \sim \theta$ iff $\sigma \downarrow D(\sigma) \approx \theta \downarrow D(\theta)$.

The set of *answers* is $\mathrm{AS} := \mathrm{FSubst}/\sim$, $\tilde{\theta} = \{\sigma \in \mathrm{FSubst} \mid \sigma \sim \theta\}$ ($\theta \in \mathrm{FSubst}$), and $\mathrm{AS}_d := \{\tilde{\theta} \mid \theta \in \mathrm{FSubst}_d\}$ ($d \subset \mathrm{VAR}$ finite).

For every $\theta \in \mathrm{FSubst}$, $\alpha \in \mathrm{AS}_{\mathrm{Rg}(\theta)}$, $d \subset \mathrm{VAR}$, let $\theta \cdot \alpha := \widetilde{\theta \cdot \sigma}$ for some $\sigma \in \alpha$ that satisfies $\mathrm{Dom}(\sigma) = \mathrm{Rg}(\theta)$, and $\alpha \downarrow d := \widetilde{\sigma \downarrow d}$ for some $\sigma \in \alpha$.

In prolog the user can decide for backtracking as long as there has been no termination symbol, hence the evaluation of a goal may produce several, also infinitely many answers. As results we get finite terminating, finite non-terminating and infinite sequences. Gathering these three kinds of sequences in a set and adding the appropriate approximation ordering, we arrive at a domain of *streams*. This structure has been used before for the semantics of logic programs (see [8], [3], [1]). Let us recall some basic facts about streams and specialize to the case of streams of substitutions. Let M be a set. The set of *streams* over M is

$$\mathrm{Stream}(M) := \bigcup_{n\in\omega} M^n \cup \bigcup_{n\in\omega} (M^n \times \{\bot\}) \cup M^\omega$$

where $\bot \notin M$. We use $[m_1, \ldots, m_n]$ for elements in M^n, $[m_1, \ldots, m_n, \bot]$ for those in $M^n \times \{\bot\}$, $(m_i)_{i\in\omega}$ for elements in M^ω. S is called *total* if $S \in \bigcup_{n\in\omega} M^n \cup M^\omega$, *partial* otherwise. S is called *finite* if $S \in \bigcup_{n\in\omega} M^n \cup (\bigcup_{n\in\omega} M^n \times \{\bot\})$. Prefixing is denoted by ::, i.e. $m_0 :: [m_1, \ldots, m_n, \bot] = [m_0, m_1, \ldots, m_n, \bot]$, and $m_0 :: [m_1, \ldots, m_n] = [m_0, m_1, \ldots, m_n]$, and $m_0 :: (m_{i+1})_{i\in\omega} = (m_i)_{i\in\omega}$. The symbol $*$ stands for stream concatenation, i.e. $S' * S = S'$ if S' is partial or infinite, $[m_1, \ldots, m_n] * S = m_1 :: \ldots :: m_n :: S$, and $\sqsubseteq$ denotes the usual ordering on streams, i.e. $S_1 \sqsubseteq S_2$ iff $S_1 = S_2$ or $S_1 = [m_1, \ldots, m_n, \bot]$ and $S_2 = [m_1, \ldots, m_n] * S'$ for some $m_1, \ldots, m_n \in M$ and a stream S'. The least element is denoted by $\bot$.

If $f\colon M \longrightarrow M$, we use $\mathrm{map}(f)$ for the corresponding function on $\mathrm{Stream}(M)$ whose application amounts to applying f to every element:

$$\begin{array}{ll} \mathrm{map}(f)([\,])=[\,] & \mathrm{map}(f)(m :: S) = f(m) :: (\mathrm{map}(f)(S)) \\ \mathrm{map}(f)(\bot)=\bot & \mathrm{map}(f)((m_i)_{i\in\omega})=(f(m_i))_{i\in\omega}. \end{array}$$

It is convenient to associate with every stream S a *type* $|S|$:

$$|[m_0, \ldots, m_n]| := \{0, \ldots, n\}, |[m_0, \ldots, m_n, \bot]| := \{0, \ldots, n, \bot\}, |(m_i)_{i\in\omega}| := \omega.$$

We use $S(i)$ for the $(i+1)$th component of S if $i \in |S| \cap \mathbb{N}$, and extend this by $S(\bot) := \bot$. Given a stream S and a function $F\colon |S| \cap \mathbb{N} \longrightarrow \mathrm{Stream}(M)$, let

$$\mathop{*}_{i\in|S|} F(i) := \begin{cases} F(0) * \ldots * F(n) & \text{if } |S| = \{0, \ldots, n\} \\ F(0) * \ldots * F(n) * \bot & \text{if } |S| = \{0, \ldots, n, \bot\} \\ \sup_{n\in\omega}(F(0) * \ldots * F(n) * \bot) & \text{if } |S| = \omega. \end{cases}$$

For every $S \in \mathrm{Stream}(M)$ and $F : M \longrightarrow \mathrm{Stream}(M)$ we use $*_{m\in S} F(m)$ for $*_{i\in|S|} F(S(i))$. Note that $(\mathrm{Stream}(M), \sqsubseteq)$ is an algebraic cpo with least element

$\bot$. Furthermore, every $(S_i)_{i\in I}$ that is bounded above is a chain, and every finite element is compact. The operation $*$ is continuous.

For $d \subset \text{VAR}$ we define $\text{AStream}_d := \text{Stream}(\text{AS}_d)$. The set of answer streams is $\text{AStream} := \bigcup_{d\subset \text{Var finite}} \text{AStream}_d$. For $S \in \text{Stream}(\text{FSubst}_d)$ we use $\tilde{S}$ for $*_{\theta\in S}[\tilde{\theta}]$.

The symbols $\cdot$ and $\downarrow$ are also used for the "mapped" operations where one argument is a stream of finite substitutions or an answer stream.

The value of a goal includes the results of evaluation of all instances. Hence a function $\text{FSubst} \longrightarrow \text{AStream}$ is assigned to every goal as its semantic value. Let $F\colon \text{FSubst} \longrightarrow \text{Stream(AS)}$, $d \subset \text{VAR}$. Then $F \downarrow d$ is the function defined by $(F \downarrow d)(\theta) := F(\theta \downarrow d)$ and $D(F) := \bigcap\{d \subset \text{Var} \mid F = F \downarrow d\}$.

Assume informally a semantics for prolog goals that assigns to every G the function $[\![G]\!]$ that maps every finite substitution θ, $\text{Dom}(\theta) \supset \text{FV}(G)$ to the stream of answers produced when evaluating $?- G\theta$. If $\text{Dom}(\theta) \not\supset \text{FV}(G)$ then $\theta\cup\, \delta$ for an appropriate renaming δ is considered instead and the result is restricted to the variables in $\text{Rg}(\theta)$. Then it is easy to see that the following conditions are satisfied:

1. $D(F)$ is finite.
2. $F(\theta) \in \text{AStream}_{\text{Rg}(\theta)}$ for all $\theta \in \text{FSubst}$
3. $F(\theta \cdot \delta) = \delta^{-1} \cdot F(\theta)$ for all renamings $\delta \in \text{FSubst}_{\text{Rg}(\theta)}$ for all $\theta \in \text{FSubst}$
4. $F(\theta\cup\, \delta) \downarrow \text{Rg}(\theta) = F(\theta)$ for all $\theta \in \text{FSubst}$ and renaming δ so that δ,θ are disjoint.

In [4] these conditions have been presented as an alternative characterization of SF which is used there as the domain of values of goals. Here (as in [6]) we use them for the definition of $\mathbb{G}$:

Definition 7. *The set $\mathbb{G}$ is the set of functions $F\colon \text{FSubst} \longrightarrow \text{Stream}(AS)$ that satisfy the conditions 1–4. Furthermore $\mathbb{G}(d) := \{F \in \mathbb{G} \mid D(F) \subset d\}$.*

If $A[\theta]$ is any expression so that $\lambda\theta \in \text{FSubst}_d.A[\theta]$ is a function $\text{FSubst}_d \longrightarrow \text{Stream(AS)}$ so that $f(\theta) \in \text{AStream}_{\text{Rg}(\theta)}$ and $f(\theta \cdot \delta) = \delta^{-1} \cdot f(\theta)$ for every $\theta \in \text{FSubst}_d$ and renamings $\delta \in \text{FSubst}_{\text{Rg}(\theta)}$, then there is a uniquely determined extension F of f in $\mathbb{G}(d)$ which is denoted by $\lambda\theta : d.A[\theta]$.

For term tuples $\bar{t}$, $\bar{s}$ of equal length let $\overline{\text{mgu}}(\bar{t};\bar{s}) := [\tilde{\theta}]$ if $(\bar{t})$ and $(\bar{s})$ are unifiable with computed mgu $\theta = \{\bar{r}/\bar{x}\}$, $\text{FV}(\bar{t},\bar{s}) = \{\bar{x}\}$, and $\overline{\text{mgu}}(\bar{t};\bar{s}) = [\,]$ if $(\bar{t})$ and $(\bar{s})$ are not unifiable. The function $m_{\bar{t};\bar{s}}$ is defined by $m_{\bar{t};\bar{s}} := \lambda\theta : \text{VAR}(\bar{t},\bar{s}).\overline{\text{mgu}}(\bar{t}\theta;\bar{s}\theta)$.

Definition 8. *Let $F, G \in \mathbb{G}$. The functions $F-x$, $F*G$, F_θ, $F\otimes G : \text{FSubst} \longrightarrow \text{Stream}(AS)$ are defined by*

1. *$F - x := F \downarrow (D(F) \setminus \{x\})$ for all variables x*
2. *$(F * G)(\theta) = F(\theta) * G(\theta)$ for all $\theta \in \text{FSubst}$*
3. *$F_\theta := \lambda\sigma : Rg(\theta).F(\theta \cdot \sigma)$ for $\theta \in \text{FSubst}$*
4. *$F\otimes G := \lambda\theta : D(F)\cup D(G).(F\times G)(\theta)$ where $(F\times G)(\theta) := *_{\sigma\in S_\theta}(\sigma\cdot G(\theta\cdot\sigma))$ for streams S_θ in $\text{Stream}(\text{FSubst}_{Rg(\theta)})$ satisfying $\tilde{S}_\theta = F(\theta)$.*

The functions $F - x$, F_θ, $F * G$, $F \otimes G$ are in $\mathbb{G}$ if F, G are. We assume the usual ordering $\sqsubseteq$ on functions for $\mathbb{G}$. Then $(\mathbb{G}, \sqsubseteq)$ is obviously a partially ordered set with least element $\lambda\theta : \emptyset.\bot$. Furthermore we will make use of the following properties of stream functions:

1. Functions F, G in $\mathbb{G}$ are equal if they coincide on some FSubst_d where $D(F) \cup D(G) \subset d \subset \mathrm{VAR}$, d finite.
2. If $F, G \in \mathbb{G}$ and $\mathrm{Dom}(\theta) \supset D(F) \cup D(G)$, then $(F \otimes G)(\theta) = (F \times G)(\theta)$ for $\times$ in the definition above.
3. $(\mathbb{G}, \otimes, \mathbb{1})$ and $(\mathbb{G}, *, \mathbb{F})$ are monoids where $\mathbb{1} := \lambda\theta : \emptyset.[\tilde{\emptyset}]$, and $\mathbb{F} := \lambda\theta : \emptyset.[\,]$.
4. For every finite $d \subset \mathrm{VAR}$, $(\mathbb{G}(d), \sqsubseteq)$ is a cpo and $\otimes, *, \cdot_\theta, \downarrow d, -x$ are continuous on $\mathbb{G}(d)$.

The domains for the terms and predicates are defined next.

Definition 9. *The set $\mathbb{T}$ is the set of terms (of type ι) and $\mathbb{P}_n$ is the set of functions $f : \mathbb{T}^n \longrightarrow AStream$ satisfying:*

- $f(\bar{t}) \in AStream(\mathrm{VAR}(\bar{t}))$ *for every* $(\bar{t}) \in \mathbb{T}^n$
- $f(\bar{t}\delta) = \delta^{-1} \cdot f(\bar{t})$ *for every renaming* $\delta \in \mathrm{FSubst}_{\mathrm{VAR}(\bar{t})}$

Furthermore $\mathbb{T}^n \twoheadrightarrow \mathbb{G}$ is the set of functions $f \in \mathbb{T}^n \longrightarrow \mathbb{G}$ so that $f(\bar{t})(\theta) = f(\bar{s})(\sigma)$ if $\bar{t}\theta = \bar{s}\sigma$ for every $(\bar{t}), (\bar{s}) \in \mathbb{T}^n$ and $\theta, \sigma \in \mathrm{FSubst}$ satisfying $Dom(\theta) \supset \mathrm{VAR}(\bar{t})$, $Dom(\sigma) \supset \mathrm{VAR}(\bar{s})$.

The function $p2tg_n : \mathbb{P}_n \longrightarrow (\mathbb{T}^n \twoheadrightarrow \mathbb{G})$ is defined by $p2tg_n(f)(\bar{t}) := \lambda\theta : \mathrm{VAR}(\bar{t}).f(\bar{t}\theta)$ for $\theta \in \mathrm{FSubst}_{\mathrm{VAR}(\bar{t})}$.

Ordered with the usual ordering on functions $\mathbb{P}_n$ and $\mathbb{T}^n \twoheadrightarrow \mathbb{G}$ are domains. The function $\mathrm{p2tg}_n$ is a monotonic, continuous bijection. The elements of $\mathbb{P}$ are used as interpretations of predicates. The interpretation of application must ensure that applying the value of a predicate to terms yields an element of $\mathbb{G}$. Hence we will combine $\mathrm{p2tg}_n$ with ordinary application. The restriction to $\mathbb{P}$ as domain for predicates implies that the models considered below yield a standard structure in the sense of [4], Sec. 5.2.

3.2 Semantics for $\mathbf{K_{frz}}$

First we define a general notion of model for $\mathrm{K}_{\mathrm{frz}}[T]$.

Definition 10. *A $\mathbb{P}$-$\mathbb{G}$-interpretation I of Σ is an assignment of elements $[\![b]\!] \in \mathbb{P}_n$ to the built-in predicates $b\colon \iota^n \longrightarrow o$ and of mappings $[\![\circ]\!]\colon \mathbb{G}^n \longrightarrow \mathbb{G}$ to the connectives $\circ\colon o^n \longrightarrow o$ so that*

1. *$[\![=]\!] = \lambda t, s.\overline{mgu}(t; s)$, $[\![\mathbf{1}]\!] = \lambda\theta : \emptyset.[\tilde{\emptyset}]$, $[\![\mathsf{F}]\!] = \lambda\theta : \emptyset.[\,]$, $[\![\mathbf{0}]\!] = \lambda\theta : \emptyset.\bot$, and the connectives $\otimes$ and $*$ are interpreted by the corresponding operations on $\mathbb{G}$,*
2. *every $[\![\circ]\!]$ is monotonic and continuous on all $\mathbb{G}(d)$, $d \subset \mathrm{VAR}$ finite, and*

3. *for every connective* $\circ\colon o^n \longrightarrow o$, $G_1, \ldots, G_n \in \mathbb{G}$ *and finite substitution* θ *so that* $Dom(\theta) \supset D(G_1) \cup \ldots \cup D(G_n)$ *the equation* $(\llbracket\circ\rrbracket(G_1, \ldots, G_n))_\theta = \llbracket\circ\rrbracket((G_1)_\theta, \ldots, (G_n)_\theta)$ *holds.*

A $\mathbb{P}$-$\mathbb{G}$*-structure* $\mathfrak{M}$ *is a pair* $(I_{\mathrm{PR}}, I_\Sigma)$ *where* I_Σ *is a* $\mathbb{P}$-$\mathbb{G}$*-interpretation of* Σ *and* I_{PR} *assigns elements* $\llbracket p\rrbracket$ *of* $\mathbb{P}_n$ *to all predicate names* p *in* PRED_n*. The mapping* id_V *is given by* $id_V(x) = x$ *for every variable* x*.*

The value of a goal in a $\mathbb{P}$-$\mathbb{G}$*-structure* $\mathfrak{M} = (I_{\mathrm{PR}}, I_\Sigma)$ *w.r.t. a variable assignment* $\varphi\colon \mathrm{VAR} \longrightarrow \mathbb{T}$ *is defined recursively by:*
$\llbracket p(\bar{t})\rrbracket^{\mathfrak{M},\varphi} = p2tg_n(\llbracket p\rrbracket))(\bar{t}\{\varphi(x)/x \mid x \text{ in } \bar{t}\})$ *if* $p \in \mathrm{PRED}_n$ *or* $b\colon \iota^n \longrightarrow \iota$
$\llbracket\circ(G_1, \ldots, G_n)\rrbracket^{\mathfrak{M},\varphi} = \llbracket\circ\rrbracket(\llbracket G_1\rrbracket^{\mathfrak{M},\varphi}, \ldots, \llbracket G_n\rrbracket^{\mathfrak{M},\varphi})$ *if* $\circ\colon o^n \longrightarrow o$
$\llbracket \mathsf{E}xG\rrbracket^{\mathfrak{M},\varphi} = \llbracket G\rrbracket^{\mathfrak{M},\varphi_{x\leftarrow v}} - v$ *for some variable* v *not in any* $\varphi(y)$, $y \in \mathrm{FV}(G)$*, and* $\llbracket G\rrbracket^{\mathfrak{M}} := \llbracket G\rrbracket^{\mathfrak{M},id_V}$*.*

A sequent $\Gamma \Rightarrow^d \Delta$ *is valid in* $\mathfrak{M}$ *iff* $\llbracket\otimes\Gamma\rrbracket^{\mathfrak{M}}(\theta) \sqsubseteq \llbracket*\Delta\rrbracket^{\mathfrak{M}}(\theta)$ *for every finite substitution* θ *satisfying* $D(\theta) \cap d = \emptyset$*.*

For a program P*, a* $\mathbb{P}$-$\mathbb{G}$*-structure is a model iff every sequent* $Def_p^P(\bar{t}) \Rightarrow^d p(\bar{t})$ *which is obtained by substitution (and bound renaming if necessary) from an element in* P *is valid in* $\mathfrak{M}$*.*

We can read the definition above as follows: The carrier of a $\mathbb{P}$-$\mathbb{G}$-structure is $\mathbb{T}$. Every symbol $c\colon \iota^n \longrightarrow \iota$ is interpreted as $\lambda\bar{t}.c(\bar{t})$. The interpretation of the quantifier is determined by $\cdot - x$. While the interpretation of the fixed symbols is given by the operations defined in subsection 3.1, the remaining symbols in Σ are interpreted by some I_Σ.

Validity in $\mathfrak{M}$ refers only to the assignment id_V but it is equivalent to a notion of validity w.r.t. all assignments as stated below.

Lemma 11. *Let* $\mathfrak{M}$ *be a* $\mathbb{P}$*-model. For every assignment* φ*, let* φ_e *denote the finite substitution* $\{\varphi(x)/x \mid x \in e\}$ *for finite* $e \subset \mathrm{VAR}$*, and, for every finite* $d \subset \mathrm{VAR}$*,* $\mathfrak{M} \models \Gamma \Rightarrow^d \Delta[\varphi]$ *iff* $\llbracket\otimes\Gamma\rrbracket^{\mathfrak{M},\varphi}(\theta) \sqsubseteq \llbracket*\Delta\rrbracket^{\mathfrak{M},\varphi}(\theta)$ *for all* $\theta \in \mathrm{FSubst}$ *satisfying* $Dom(\theta) = Rg(\varphi_{\mathrm{FV}(\Gamma,\Delta)})$ *and* $D(\varphi_{\mathrm{FV}(\Gamma,\Delta)} \cdot \theta) \cap d = \emptyset$*. Then*

1. $\llbracket G\rrbracket^{\mathfrak{M},\varphi} = (\llbracket G\rrbracket^{\mathfrak{M}})_{\varphi_{\mathrm{FV}(G)}}$ *for all goals* G *and*
2. $\mathfrak{M} \models \Gamma \Rightarrow^d \Delta$ *iff* $\mathfrak{M} \models \Gamma \Rightarrow^d \Delta[\varphi]$ *for all* φ*.*

Proof. The second statement follows using the first, which in turn is proved by induction on G, using the definition of p2tg and the conditions on the interpretations of connectives.

Definition 12. *For* Σ_{struct} *we fix the following standard interpretation:*

$$\llbracket \mathtt{var}\rrbracket(t) = \begin{cases} [\tilde{\emptyset}] & \text{if } t \text{ is a variable} \\ [\,] & \text{otherwise} \end{cases}$$

$$\llbracket \mathtt{ground}\rrbracket(t) = \begin{cases} [\tilde{\emptyset}] & \text{if } t \text{ is ground} \\ [\,] & \text{otherwise} \end{cases}$$

$\llbracket \mathtt{copy}\rrbracket(t, s) = \overline{mgu}(t\{\bar{u}/\bar{x}\}, s) \downarrow \mathrm{VAR}(s)$ *where* $\mathrm{VAR}(t) = \{\bar{x}\}$*,* $\{\bar{u}/\bar{x}\}$ *is a renaming,* $\{\bar{u}\} \cap \mathrm{VAR}(s) = \emptyset$*.*

Definition 13. *For Σ_{cond} we fix the following standard interpretation:*
$[\![(\cdot \to \cdot;\cdot)]\!](G, H_1, H_2) = \lambda\theta : D(G) \cup D(H_1) \cup D(H_2).F_{G,H_1,H_2}(\theta)$
where $F_{G,H_1,H_2}(\theta) = \begin{cases} \sigma \cdot H_1(\theta \cdot \sigma) & \text{if } G(\theta) \sqsupseteq [\tilde{\sigma}, \bot], Dom(\sigma) = Rg(\theta) \\ H_2(\theta) & \text{if } G(\theta) = [\,] \\ \bot & \text{otherwise} \end{cases}$
Furthermore $[\![\texttt{fail\_if}]\!](G) = [\![(\cdot \to \cdot;\cdot)]\!](G, \lambda\theta : \emptyset.[\,], \lambda\theta : \emptyset.[\tilde{\emptyset}])$,
$[\![\texttt{once}]\!](G) = [\![(\cdot \to \cdot;\cdot)]\!](G, \lambda\theta : \emptyset.[\tilde{\emptyset}], \lambda\theta : \emptyset.[\,])$, *and*
$[\![\texttt{succeed\_if}]\!](G) = [\![\texttt{fail\_if}]\!]([\![\texttt{fail\_if}]\!](G))$.

4 Properties of $\mathbf{K_{frz}}$

For this subsection we fix $\Sigma = \Sigma_{\text{struct}} \cup \Sigma_{\text{cond}}$ and the standard interpretation as just defined. Hence $\models$ refers to this interpretation and $\cdot \vdash \cdot$ to derivability in $\mathrm{K}_{\text{frz}}[T_{\text{struct}} \cup T_{\text{cond}}]$.

4.1 Soundness of $\mathbf{K_{frz}}$

Theorem 14. *Let P be a program and $\Gamma \Rightarrow^d \Delta$ a sequent. If $P \vdash \Gamma \Rightarrow^d \Delta$ then $\Gamma \Rightarrow^d \Delta$ holds in every model of P.*

Proof. By induction on the height of the derivation, distinguishing cases according to the last rule. We use the notation $F \sqsubseteq^d G$ for $F(\theta) \sqsubseteq G(\theta)$ for all $\theta \in \mathrm{FSubst}$ so that $D(\theta) \cap d = \emptyset$. We consider only the cases where frozen variables are relevant:

$(\Gamma \Rightarrow^d G_1), (\Pi \Rightarrow^e G_2) \vdash (\Gamma, \Pi \Rightarrow^{d \cup e} G_1 \otimes G_2)$, $\mathrm{FV}(\Gamma) \cap e = \emptyset$: Let $F_1 := [\![\otimes \Gamma]\!]^{\mathfrak{M}}$ and $F_2 := [\![\otimes \Pi]\!]^{\mathfrak{M}}$. Then by IH $F_1 \sqsubseteq^d [\![G_1]\!]^{\mathfrak{M}}$ and $F_2 \sqsubseteq^e [\![G_2]\!]^{\mathfrak{M}}$, and $D(F_1) \cap e = \emptyset$. Let $\theta \in \mathrm{FSubst}$ so that $D(\theta) \cap (d \cup e) = \emptyset$ and $\mathrm{Dom}(\theta) = D(F_1) \cup D(F_2) \cup D([\![G_1]\!]^{\mathfrak{M}}) \cup D([\![G_2]\!]^{\mathfrak{M}})$, and $S \in \mathrm{Stream}(\mathrm{FSubst}_{\mathrm{Rg}(\theta)})$ so that $\tilde{S} = F_1(\theta)$. Then $D(\theta \cdot \sigma) \cap e = \emptyset$ for every σ in S, as $D(\theta) \cap e = \emptyset$ and $D(F_1) \cap e = \emptyset$. As a consequence, $(F_1 \otimes F_2)(\theta) = \underset{\sigma \in S}{*} \sigma \cdot F_2(\theta \cdot \sigma) \sqsubseteq \underset{\sigma \in S}{*} \sigma \cdot [\![G_2]\!]^{\mathfrak{M}}(\theta \cdot \sigma) \sqsubseteq ([\![G_1]\!]^{\mathfrak{M}} \otimes [\![G_2]\!]^{\mathfrak{M}})(\theta)$.

$(G \Rightarrow^d H) \vdash (\mathsf{E}xG \Rightarrow^d \mathsf{E}xH)$: By IH $[\![G]\!]^{\mathfrak{M}} \sqsubseteq^d [\![H]\!]^{\mathfrak{M}}$. Let $\theta \in \mathrm{FSubst}$ so that $D(\theta) \cap (d \setminus \{x\}) = \emptyset$. Then $([\![G]\!]^{\mathfrak{M}} - x)(\theta) = [\![G]\!]^{\mathfrak{M}}(\theta - x) \sqsubseteq [\![H]\!]^{\mathfrak{M}}(\theta - x) = ([\![H]\!]^{\mathfrak{M}} - x)(\theta)$. Similar reasoning applies to the remaining cases of quantifier rules.

$G \Rightarrow^d H \vdash x = t, G \Rightarrow^{d \cup \{x\}} x = t \otimes H$ and $x \notin d$: The case that x occurs in t is immediate. Hence assume that x does not occur in t. Let $F_1 := [\![G]\!]^{\mathfrak{M}}$ and $F_2 := [\![H]\!]^{\mathfrak{M}}$. By IH $F_1 \sqsubseteq^d F_2$. Let $\theta \in \mathrm{FSubst}$ so that $D(\theta) \cap (d \cup \{x\}) = \emptyset$, $\mathrm{Dom}(\theta) = D(F_1) \cup D(F_2) \cup \{x\} \cup \mathrm{FV}(t)$. As $x \in \mathrm{Dom}(\theta) \setminus D(\theta)$ there are $\theta_0 \in \mathrm{FSubst}$ and a variable u so that $\theta = \theta_0 \cup \{u/x\}$. For the variables $\bar{v}$ in $\mathrm{Rg}(\theta_0)$ we have $(m_{x;t} \otimes F_1)(\theta) = (\{t\theta_0/u\} \cup \{\bar{v}/\bar{v}\}) \cdot F_1(\theta \cdot (\{t\theta_0/u\} \cup \{\bar{v}/\bar{v}\})) = (\{t\theta_0/u\} \cup \{\bar{v}/\bar{v}\}) \cdot F_1(\theta_0 \cup \{t\theta_0/x\})$.As $x \notin d$, the set $D(\theta_0 \cup \{t\theta_0/x\}) \cap d$ is empty. Hence $(m_{x;t} \otimes F_1)(\theta) \sqsubseteq (\{t\theta_0/u\} \cup \{\bar{v}/\bar{v}\}) \cdot F_2(\theta_0 \cup \{t\theta_0/x\}) = (m_{x;t} \otimes F_2)(\theta)$.

The soundness of the rules for Σ_{cond} is immediate from the definition of the interpretation and the soundness of the monotonicity rule $(G \Rightarrow^d G'), (H_1 \Rightarrow^{e_1} H_1'), (H_2 \Rightarrow^{e_2} H_2') \vdash (G \to H_1; H_2) \Rightarrow^{d \cup e_1 \cup e_2} (G' \to H_1'; H_2')$ if $\text{FV}(G) \cap e_1 = \emptyset$ which is proved similar to the first case.

4.2 Relating $\mathbf{K_{frz}}$ to an Operational Semantics

It will be shown that K_{frz} is adequate in the following sense: if S is a computed value for a goal G evaluated w.r.t. a program P then the corresponding sequent is derivable in the calculus. Although values as just defined may be infinite, the *computed* values here are only finite approximations to this obtained in finite time. The latter are represented by expressions in our language.

Definition 15. *Let $m \geq 0$. The goals $A_1 * \ldots * A_m$ and $A_1 * \ldots * A_m * \mathbf{0}$ are called* solved *if each A_i is a positive answer.*

The property we show below is the following: if a solved goal approximates a goal G operationally then $S \Rightarrow^d G$ can be proved for $d = \text{FV}(S, G)$. This can be viewed as soundness of the operational semantics w.r.t. the calculus but does not imply the completeness of the calculus w.r.t. the standard interpretation above. As we are not concerned with the connection to search for SLD refutations here, we assume a different operational semantics, based on rewriting goals.

Definition 16. *Let P be a program. The relation $\mapsto$ is defined by:*

$p(\bar{t}) \mapsto \mathit{Def}_p^P(\bar{t})$
$(G_1 \otimes G_2) \otimes H \mapsto G_1 \otimes (G_2 \otimes H)$
$(G_1 * G_2) \otimes H \mapsto (G_1 \otimes H * G_2 \otimes H)$
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathsf{E}v.G, \Gamma), \Delta)) \mapsto *(\Pi, \mathsf{E}\bar{u}\mathsf{E}v. \otimes (\bar{x} = \bar{t}, G, \Gamma), \Delta)$
if v not free in $\bar{x}, \bar{t}, \Gamma$
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathbf{1}, \Gamma), \Delta) \mapsto *(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \Gamma), \Delta)$
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathsf{F}, \Gamma), \Delta) \mapsto *(\Pi, \Delta)$
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, s_1 = s_2, \Gamma), \Delta) \mapsto *(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \bar{y} = \bar{r}, \Gamma), \Delta)$
if s_1 is no variable, $(\bar{y}; \bar{r})$ and $(s_1; s_2)$ have the same unifiers,
$\text{FV}(s_1, s_2) \subset \text{FV}(\bar{y}, \bar{r})$, and no y_i occurs in some r_j.
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, s_1 = s_2, \Gamma), \Delta) \mapsto *(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathsf{F}, \Gamma), \Delta)$
if s_1 and s_2 are not unifiable
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, y = s, \Gamma), \Delta) \mapsto *(\Pi, \mathsf{E}\bar{u}. \otimes ((\bar{x}, y) = (\bar{t}\{s/y\}, s), \Gamma\{s/y\}), \Delta)$
if y not in $s, \bar{u}, \bar{x}$.
$*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, u_i = s, \Gamma), \Delta) \mapsto *(\Pi, \mathsf{E}\bar{v}. \otimes (\bar{x} = \bar{t}\{s/u_i\}, \Gamma\{s/u_i\}), \Delta)$
if u_i not in $s, \bar{x}$ where $\bar{v}$ is obtained from $\bar{u}$ by deleting u_i.

$$\mathit{var}(t) \mapsto \begin{cases} \mathbf{1} & \text{if } t \text{ is a variable} \\ \mathsf{F} & \text{otherwise} \end{cases}$$
$$\mathit{ground}(t) \mapsto \begin{cases} \mathbf{1} & \text{if } t \text{ is ground} \\ \mathsf{F} & \text{otherwise} \end{cases}$$
$\mathit{copy}(t, s) \mapsto \mathsf{E}\bar{u}(s = t\{\bar{u}/\bar{x}\})$
where $\text{FV}(t) = \{\bar{x}\}$ and $\{\bar{u}/\bar{x}\}$ is a renaming so that $\{\bar{u}\} \cap \text{FV}(s) = \emptyset$.

$\mathit{fail\_if}(\mathsf{F}) \mapsto \mathbf{1}$
$\mathit{fail\_if}(A * G) \mapsto \mathsf{F}$ *if* A *is a positive answer*
$\mathit{succeed\_if}(\mathsf{F}) \mapsto \mathsf{F}$
$\mathit{succeed\_if}(A * G) \mapsto \mathbf{1}$ *if* A *is a positive answer*
$\mathit{once}(\mathsf{F}) \mapsto \mathsf{F}$
$\mathit{once}(A * G) \mapsto A$ *if* A *is a positive answer*
$(\mathsf{F} \to H_1; H_2) \mapsto H_2$
$(A * G \to H_1; H_2) \mapsto A \otimes H_1$ *if* A *is a positive answer*

If necessary we add P as subscript to emphasize the dependency. To avoid circular reductions, e.g. $(x_1 = c_1 \otimes x_2 = c_2) \otimes x_3 = c_3 \mapsto^+ (x_1 = c_1 \otimes x_2 = c_2) \otimes x_3 = c_3$ one may wish to restrict $\mapsto$ further. For the results below, however, this is not necessary.

Lemma 17. *If* $G \mapsto H$ *then* $\mathrm{FV}(H) \subset \mathrm{FV}(G)$ *and* $P \vdash H \Rightarrow^d G$ *for all finite* $\mathrm{VAR} \supset d \supset \mathrm{FV}(G)$.

Proof. Straightforward assignment of $\mathrm{K}_{\mathrm{frz}}$ derivations to reductions $G \mapsto H$.

The relation $\mapsto$ determines reductions. But if this operational semantics shall capture the intended informal meaning they must not be performed in arbitrary contexts. To model the effect of the point of selection correctly the admissible contexts must be restricted. As usual a context $\mathcal{C}$ is an expression with a hole _. The expression $\mathcal{C}[G]$ denotes the result of substituting G for the hole where variables in G may be bound.

Definition 18. *The set of D-contexts ("disjunctive") is inductively defined by:*

- *The empty context* _ *is a D-context.*
- $*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathcal{C}, \Gamma), \Delta)$ *is a D-context if*
  - Π *consists of positive answers and* Δ *of goals,*
  - $\bar{u}$ *are pairwise distinct variables,*
  - $\bar{x}$ *are pairwise distinct variables which do not occur free in* $\Gamma, \bar{t}, \bar{u}, \mathcal{C}$*, and*
  - $\mathcal{C}$ *is a D-context.*
- $\mathit{fail\_if}(\mathcal{C})$, $\mathit{succeed\_if}(\mathcal{C})$, $\mathit{once}(\mathcal{C})$, *and* $(\mathcal{C} \to H_1; H_2)$ *are D-contexts if* $\mathcal{C}$ *is a D-context and* H_1, H_2 *are goals.*

The sets of relevant bound variables $rbv(\mathcal{C})$ *and relevant free variables* $rfv(\mathcal{C})$ *of a D-context* $\mathcal{C}$ *are recursively defined by:* $rbv(\_) = rfv(\_) = \emptyset$
$rbv(\mathit{fail\_if}(\mathcal{C})) = rbv(\mathit{succeed\_if}(\mathcal{C})) = rbv(\mathit{once}(\mathcal{C})) = rbv((\mathcal{C} \to H_1; H_2)) = rbv(\mathcal{C})$
$rfv(\mathit{fail\_if}(\mathcal{C})) = rfv(\mathit{succeed\_if}(\mathcal{C})) = rfv(\mathit{once}(\mathcal{C})) = rfv((\mathcal{C} \to H_1; H_2)) = rfv(\mathcal{C})$
$rbv(*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathcal{C}, \Gamma), \Delta)) = rbv(\mathcal{C}) \cup \{\bar{u}\}$
$rfv(*(\Pi, \mathsf{E}\bar{u}. \otimes (\bar{x} = \bar{t}, \mathcal{C}, \Gamma), \Delta)) = (rfv(\mathcal{C}) \setminus \{\bar{u}\}) \cup \{\bar{x}\}$.

Here $*(G_1, \ldots, G_n)$ and $\otimes(G_1, \ldots, G_n)$ stand for the disjunction or conjunction respectively of the formulas $G_1, \ldots, G_n$.

Definition 19. *The relation* $\rhd$ *on goals is defined by* $\mathcal{C}[G] \rhd \mathcal{C}[H]$ *if* $G \mapsto H$ *and* $\mathcal{C}$ *is a D-context so that* $rfv(\mathcal{C}) \cap \mathrm{FV}(G, H) = \emptyset$.

Lemma 20. *Let* P *be a program,* G, H *goals,* $d \subset \mathrm{VAR}$ *finite, and* $\mathcal{C}$ *a D-context so that* $rfv(\mathcal{C}) \cap d = \emptyset$. *If* $P \vdash G \Rightarrow^d H$ *then* $P \vdash \mathcal{C}[G] \Rightarrow^{(d \setminus rbv(\mathcal{C})) \cup rfv(\mathcal{C})} \mathcal{C}[H]$.

Proof. By induction on $\mathcal{C}$. The case of an empty context is immediate, and for $\texttt{fail\_if}(\mathcal{C})$, $\texttt{succeed\_if}(\mathcal{C})$, $\texttt{once}(\mathcal{C})$, and $(\mathcal{C} \to H_1; H_2)$ we just combine the induction hypothesis with the monotonicity of the connective. So consider $\mathcal{C}' \equiv *(\Pi, \mathsf{E}\bar{u}.\otimes(\bar{x} = \bar{t}, \mathcal{C}, \Gamma), \Delta)$: By IH $P \vdash \mathcal{C}[G] \Rightarrow^{(d \setminus \mathrm{rbv}(\mathcal{C})) \cup \mathrm{rfv}(\mathcal{C})} \mathcal{C}[H]$. As x_i are neither free in $\mathcal{C}$ nor in d, we can deduce $P \vdash \otimes(\bar{x} = \bar{t}, \mathcal{C}[G], \Gamma) \Rightarrow^{(d \setminus \mathrm{rbv}(\mathcal{C})) \cup \mathrm{rfv}(\mathcal{C}) \cup \{\bar{x}\}} \otimes(\bar{x} = \bar{t}, \mathcal{C}[H], \Gamma)$, hence $P \vdash \mathsf{E}\bar{u}.\otimes(\bar{x} = \bar{t}, \mathcal{C}[G], \Gamma) \Rightarrow^{d'} \mathsf{E}\bar{u}.\otimes(\bar{x} = \bar{t}, \mathcal{C}[H], \Gamma)$, where $d' = ((d \setminus \mathrm{rbv}(\mathcal{C})) \cup \mathrm{rfv}(\mathcal{C}) \cup \{\bar{x}\}) \setminus \{\bar{u}\} = (d \setminus (\mathrm{rbv}(\mathcal{C}) \cup \{\bar{u}\})) \cup (\mathrm{rfv}(\mathcal{C}) \setminus \{\bar{u}\}) \cup \{\bar{x}\}$. This implies $P \vdash \mathcal{C}'[G] \Rightarrow^{(d \setminus \mathrm{rbv}(\mathcal{C}')) \cup \mathrm{rfv}(\mathcal{C}')} \mathcal{C}'[H]$.

Theorem 21. *Let* P *be a program,* G *a goal,* $\mathrm{FV}(G) = \{\bar{x}\}$, $\{\bar{u}/\bar{x}\}$ *a renaming so that* $\{\bar{u}\} \cap \{\bar{x}\} = \emptyset$, *and* S *a solved goal. If* $\mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\}) \rhd^* S$ *then* $P \vdash S \Rightarrow^{\{\bar{x}\}} G$. *If* $\mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\}) \rhd^* S * H$ *for some* H *then* $P \vdash S * \mathbf{0} \Rightarrow^{\{\bar{x}\}} G$.

Proof. By induction on the length of the computation we show that $\mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\}) \rhd^* H$ implies $P \vdash H \Rightarrow^{\{\bar{x}\}} \mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\})$. Assume that $\mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\}) \rhd^n H_1 \rhd H_2$. Then $\mathrm{FV}(H_2) \subset \mathrm{FV}(H_1) \subset \{\bar{x}\}$ and there is a D-context $\mathcal{C}$ so that $H_i = \mathcal{C}[H_i']$, $i = 1, 2$, for some H_1', H_2' satisfying $H_1' \mapsto H_2'$ and $\mathrm{rfv}(\mathcal{C}) \cap \mathrm{FV}(H_1', H_2') = \emptyset$. Combining Lem.17 and Lem.20 we obtain that $P \vdash \mathcal{C}(H_2') \Rightarrow^d \mathcal{C}(H_1')$ for $d = (\mathrm{FV}(H_1', H_2') \setminus \mathrm{rbv}(\mathcal{C})) \cup \mathrm{rfv}(\mathcal{C})$, hence $P \vdash H_2 \Rightarrow^{\{\bar{x}\}} H_1$, as $(\mathrm{FV}(H_1', H_2') \setminus \mathrm{rbv}(\mathcal{C})) \cup \mathrm{rfv}(\mathcal{C}) \subset \{\bar{x}\}$. Combining this with the induction hypothesis yields $P \vdash H_2 \Rightarrow^{\{\bar{x}\}} \mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G)\{\bar{u}/\bar{x}\}$. Observing that $\vdash S * \mathbf{0} \Rightarrow^{\emptyset} S * H$ for every H and $\vdash \mathsf{E}\bar{u}(\bar{x} = \bar{u} \otimes G\{\bar{u}/\bar{x}\}) \Rightarrow^{\emptyset} G$ completes the proof.

5 Conclusion

The problem of modeling meta-logical features in a proof-theoretic setting has been addressed. The calculus $\mathrm{K}_{\mathrm{frz}}$ provides a logical semantics of the programming constructs defined by the operational semantics in Sec.4.2. In this respect the approach presented here contrasts to a description of prolog as in [2], which, on the other hand, applies to the full language prolog.

The crucial ingredient here is 'freezing' variables. Frozen variables differ from free ones because we do not have the substitution property for them. They are no constants which is indicated by the fact that distinct frozen variables do unify. Furthermore they may occur in quantification rules where variables are expected. Freezing variables is a binding mechanism on the sequent level in the sense that it turns variables into non-free objects but in contrast to bound variables assignments to frozen variables are not completely irrelevant, and moreover they

are still affected by a regular binding by quantification performed later in the deduction. Hence frozen variables do not belong to any of these groups.

The problem of restricting a judgment to the 'uninstantiated' case occurred before in considering lazy list comprehension in [5] where it was solved by introducing auxiliary predicates ∇_n which corresponds to checking whether the arguments are distinct free variables but looping (not failing) if they are not. Instead of $S \Rightarrow^{\{\bar{x}\}} G$ ($\bar{x}$ the distinct free variables in G) one then derives $\nabla(\bar{x}), S \Rightarrow G$. Although one is interested in the sequents $\nabla(\bar{x}), S \Rightarrow G$ only, the ∇ expressions may occur anywhere in the sequent, and special axioms and rules for ∇ have to be added. Compared to this approach the work presented here can be viewed as further analysis of the utilization of ∇, restricting it to the originally intended result judgments and avoiding extra axioms and rules by absorbing them into the remaining rules.

By modeling the effect of the Σ_{cond} group we obtain an example how a proof-theoretic approach can cope with features with a quite non-logical flavor. The general setting of K_{frz}, however, could well be applied to other connectives, e.g. related to a more advanced selection mechanism, and may even be used as a guide in their design.

References

1. M. Baudinet. Proving termination properties of Prolog programs: A semantic approach. *Journal of Logic Programming*, 14(1 and 2):1–29, October 1992.
2. E. Börger and D. Rosenzweig. A mathematical definition of full Prolog. *Science of Computer Programming*, 24(3):249.
3. S. K. Debray and P. Mishra. Denotational and operational semantics for Prolog. *Journal of Logic Programming*, 5(1):61–91, March 1988.
4. B. Elbl. A declarative semantics for depth-first logic programs. *Journal of Logic Programming*, 41(1):27–66, 1999.
5. B. Elbl. Lazy list comprehension in logic programming. *Journal of Logic and Computation*, (to appear).
6. B. Elbl. A non-definability result for a predicational language with the usual control. *International Journal of Foundations of Computer Science*, 12(3):385–396, 2001.
7. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.
8. N. D. Jones and A. Mycroft. Stepwise development of operational and denotational semantics for Prolog. In *Proceedings of the 1984 International Symposium on Logic Programming*, pages 281 – 288. IEEE, 1984.
9. G. E. Mints. Complete calculus for pure prolog. *Proc. Acad. Sci. Estonian SSR*, 35(4):367–380, 1986.
10. P. Schroeder-Heister. Definitional reflection and the completion. In R. Dyckhoff, editor, *Extensions of Logic Programming. Fourth International Workshop, St. Andrews, Scotland, April 1993, Proceedings*, Springer Lecture Notes in Artificial Intelligence, pages 333–347, 1994.
11. J. Shepherdson. Mints type deductive calculi for logic programming. *Annals of Pure and Applied Logic*, 56:7–17, 1992.
12. L. Sterling and E. Y. Shapiro. *The Art of Prolog.* MIT Press, 1986.

Proof Theory and Post-turing Analysis

Lew Gordeew

Tübingen University
WSI für Informatik, Logik und Sprachtheorie
Sand 13, D-72076 Tübingen, Germany
gordeew@informatik.uni-tuebingen.de

Abstract. Traditional proof theory and Post-Turing computability are parts of discrete mathematics, whereas traditional analysis deals with non-discrete objects like real numbers and related topological notions. On the other hand, familiar mathematical proofs very often use methods of (functional) analysis which are compatible with functional formalizations of Post-Turing computability. We elaborate these connections with regard to the polynomial-time computability.

1 Introduction

Traditional proof theory provides a powerful tool for proving conservation results. That is to say, if we are given three classes $\mathcal{N}$, $\mathcal{M}$, $\mathcal{K}$ such that $\mathcal{N} \subseteq \mathcal{M}$, and if we conjecture that $\mathcal{N} \cap \mathcal{K} \subseteq \mathcal{M} \cap \mathcal{K}$ in fact strengthens to $\mathcal{N} \cap \mathcal{K} = \mathcal{M} \cap \mathcal{K}$ then we can hope to prove this conjecture by the appropriate cut elimination techniques. In particular, having $\mathsf{P} \subseteq \mathsf{NP}$ (i.e. $\mathsf{P} \leq \mathsf{NP}$) one can try to prove the conjecture $\mathsf{P} = \mathsf{NP}$ (?) by using proof theoretical methods. In fact, the study was made in this direction (cf. [T]) by transferring $\mathsf{P} = \mathsf{NP}$ (?) to the analogous proof theoretical conjecture $\mathsf{S}_2^1 = \mathsf{S}_2^2$ (?), where each S_2^i $(i \geq 0)$ denotes a sequent calculus in the 1-order language $\mathcal{L}^b$ with:

- three binary predicates $=, <, \leq$
- two individual constants 0, 1
- five functions $+, \cdot, \lambda x.\,|x|, \lambda x.\,\left[\frac{1}{2}x\right], \lambda xy.x\sharp y$, where:
  - $|x|$ = *the length of the binary expansion of* x *with* $|x| = 0$
  - $\left[\frac{1}{2}x\right] = \max\{y \in \mathbb{N} \mid 2y \leq x\}$
  - $x\sharp y = 2^{|x||y|}$

that extends basic Gentzen sequent calculus LK by adding:

- defining axioms for the underlying predicates, individual constants and functions
- defining rules for bounded quantifiers $(\forall x \leq a)$, $(\exists x \leq a)$
- the following weak induction rule Σ_i^b−PIND

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 130–152, 2001.

$$\Sigma_i^b\text{–PIND}: \frac{A\left[\frac{1}{2}a\right], \Gamma \to \Delta, A(a)}{A(0), \Gamma \to \Delta, A(t)} \text{ if } A(a) \in \Sigma_i^b,\ a \notin \mathtt{FV}(\Gamma, \Delta, A(t))$$

where Σ_i^b are classes of $\mathcal{L}^b$-formulas which are defined along with the dual classes Π_i^b by the recursion on $i \geq 0$, as follows.

1. $\Sigma_0^b = \Pi_0^b =$ *the formulas whose quantifiers are all in the sharply bounded form* $(\forall x \leq |a|)$ *or* $(\exists x \leq |a|)$.
2. Σ_{k+1}^b *and* Π_{k+1}^b *are the smallest classes of formulas satisfying the following eight conditions:*
   a) $\Pi_k^b \subseteq \Sigma_{k+1}^b$
   b) *if* $A \in \Sigma_{k+1}^b$ *then* $(\forall x \leq |a|)\, A \in \Sigma_{k+1}^b$ *and* $(\exists x \leq a)\, A \in \Sigma_{k+1}^b$
   c) *if* $A \in \Sigma_{k+1}^b$ *and* $B \in \Sigma_{k+1}^b$ *then* $A \wedge B \in \Sigma_{k+1}^b$ *and* $A \vee B \in \Sigma_{k+1}^b$
   d) *if* $A \in \Sigma_{k+1}^b$ *and* $B \in \Pi_{k+1}^b$ *then* $\neg B \in \Sigma_{k+1}^b$ *and* $B \to A \in \Sigma_{k+1}^b$
   e) $\Sigma_k^b \subseteq \Pi_{k+1}^b$
   f) *if* $A \in \Pi_{k+1}^b$ *then* $(\forall x \leq |a|)\, A \in \Pi_{k+1}^b$ *and* $(\exists x \leq a)\, A \in \Pi_{k+1}^b$
   g) *if* $A \in \Pi_{k+1}^b$ *and* $B \in \Pi_{k+1}^b$ *then* $A \wedge B \in \Pi_{k+1}^b$ *and* $A \vee B \in \Pi_{k+1}^b$
   h) *if* $A \in \Pi_{k+1}^b$ *and* $B \in \Sigma_{k+1}^b$ *then* $\neg B \in \Pi_{k+1}^b$ *and* $B \to A \in \Pi_{k+1}^b$

By familiar considerations (cf. [B,T]), one can infer $\mathsf{P} = \mathsf{NP}$ from $\mathsf{S}_2^1 = \mathsf{S}_2^2$. Having this, one could hope to prove $\mathsf{S}_2^1 = \mathsf{S}_2^2$ by some "weak" cut elimination techniques showing that S_2^2 is in fact a conservative extension of S_2^1. To this end, one should be able to eliminate or at least weaken the instances Σ_2^b–PIND occurring in a given derivation in S_2^2. The question about such "weak" cut elimination is still open. However, it is known that on the one hand full cut elimination implies full exponentiation, provably in S_2^1, and on the other hand $\forall x \exists y\, (y = 2^x)$ is not provable in S_2^1 (see references and discussion in [T]). This shows that familiar proof theoretical methods might be not fine enough, when dealing with questions like $\mathsf{S}_2^1 = \mathsf{S}_2^2$ (?). On the other hand, it is doubtful that a hierarchy like $\left\{\Sigma_i^b\right\}_{i \geq 0}$ that is based on bounded quantifiers is fine enough for the desired proof theoretical translation of the notions of computational complexity. We ask if there are other, more natural formalizations of Post-Turing computability, which can provide us with functional hierarchies using other parameters?

Now consider the dual conjecture $\mathsf{P} \neq \mathsf{NP}$ (?)*, i.e.* $\mathsf{P} < \mathsf{NP}$ (?). While transferring $\mathsf{P} \neq \mathsf{NP}$ (?) to $\mathsf{S}_2^1 \neq \mathsf{S}_2^2$ (?), one can hope to infer $\mathsf{S}_2^1 \neq \mathsf{S}_2^2$ from the appropriate "weak" incompleteness theorems by encoding metamathematics in some S_2^i (see references and discussion in [T]). However, the bounded quantifier approach might be too rough for this sake, too. Analogous encoding problems arise, when trying to infer $\mathsf{P} \neq \mathsf{NP}$ from the relative consistency of $\mathsf{P}^f \neq \mathsf{NP}^f$ (cf. [T]). On the other hand, if $\mathsf{P} < \mathsf{NP}$ holds then we can try to prove it in a straightforward way. To this end, it would suffice to prove that a chosen NP-complete problem is not decidable by any polynomial time Turing machine. For example, one can take either the consistency (= satisfiability) problem $\mathrm{SAT}_{\mathrm{CNF}}$ with respect to boolean conjunctive normal formulas (abbr.: CNF) or, by duality,

negated tautology problem $\overline{\mathrm{TAU}}_{\mathrm{DNF}}$ with respect to boolean disjunctive normal formulas (abbr.: DNF). So the question arises how to **prove** $\mathrm{SAT}_{\mathrm{CNF}} \notin \mathsf{PTIME}$ a/o $\overline{\mathrm{TAU}}_{\mathrm{DNF}} \notin \mathsf{PTIME}$ should it really be the case? Furthermore, by duality, it will suffice to **prove** that $\mathrm{TAU}_{\mathrm{DNF}} \notin \mathsf{PTIME}$ holds. Consider

$$\mathrm{DNF} \ni \digamma = \bigvee^{i \in I} \bigwedge_{j \in J(i)} \mathfrak{L}^i_j$$

where $\mathfrak{L}^i_j$ are literals, $\#(I), \#(J(i)) < \infty$ for all $i \in I$, $j \in J(i)$. We ask how to **prove** the conjecture that $\digamma \in \mathrm{TAU}_{\mathrm{DNF}}$ cannot be verified in polynomial time? Now $\digamma$ is a tautology iff for every choice function $f : I \ni i \mapsto f(i) \in J(i)$, the correlated diagonal choice sequence $\mathcal{D}_f = \left\{\mathfrak{L}^i_{f(i)} \mid i \in I\right\}$ is inconsistent, i.e. $(\exists i, k \in I)\left(\mathfrak{L}^i_{f(i)} = \neg\mathfrak{L}^k_{f(k)}\right)$. So a hypothetical "naive" PTIME-solution for $\digamma \in \mathrm{TAU}_{\mathrm{DNF}}$ should be able to:

1. PTIME-enumerate a suitable "sufficient" set $f_0, f_1, ..., f_\iota, ...$ of the above choice functions,
2. PTIME-verify if every subsequent diagonal choice sequence $\mathcal{D}_{f_\iota}$ is inconsistent.

Part 2 is unproblematic, but part 1 must fail, since for sufficiently long tautologies $\digamma$, every "sufficient" set of choice functions is exponential in $\#(\digamma)$. The latter follows from the familiar result that for sufficiently long inconsistent $\digamma$, the number of clauses in every resolution proof of $\neg\digamma$ is exponential in $\#(\digamma)$ (see [H]). Arguing *ad absurdum*, this can serve as a proof of the corresponding weak variant of the conjecture $\mathrm{TAU}_{\mathrm{DNF}} \notin \mathsf{PTIME}$. Of course, the general case is more involved, since, in the above description, part 1 might include other components of $\digamma$ than the (literals determined by the) choice functions, while part 2 might be based on other verification ideas. Furthermore, both parts can be interactive. Loosely speaking, our hypothetical contradiction is as follows, where $\mathcal{E}(\digamma)$ denotes the set of all enumerations of the literal-occurrences in $\digamma$.

- Boolean value **1** of a DNF $\digamma$ cannot be PTIME-*determined* by any polynomial subset of $\mathcal{E}(\digamma)$.
- However, every boolean-valued PTIME-computation of a DNF $\digamma$ is PTIME-*determined* by a *certain* polynomial subset of $\mathcal{E}(\digamma)$.

Now the next question is how to clarify the above loose description and formalize 'PTIME-*determined*' and '*certain*' in strict mathematical terms. We assume that these notions are definable by induction on the complexity of computation, which is assumed to be parametric in the total number of states of the correlated Turing machine. Basic analysis and topology provide us with the intuitive background.

The underlying proof idea is based on Baire-Borel-Lebesgue analytic approach, rather than Hilbert-Herbrand-Gentzen finitistic proof theory. To explain the point, let ϕ be any boolean two-valued PTIME-computable evaluation with the domain DNF. Loosely speaking, we ask **how many solutions** F have the equations $\phi(F) = \mathbf{1}$ and $\phi(F) = \mathbf{0}$, rather than trying to redefine a/o analyze ϕ within the traditional number theoretical framework. The latter approach seems to be extremely difficult due to the underlying encoding problems (see above). To put the former more exactly, consider any mapping $\mathfrak{f} : U \to \mathbf{2} = \{\mathbf{0}, \mathbf{1}\}$ and let $X := \mathfrak{f}^{-1}(\{\mathbf{1}\})$ and $Y := \mathfrak{f}^{-1}(\{\mathbf{0}\})$. A desired hypothetical contradiction (see above) turns out to be relevant to

Proposition 1. *There is no covering*

- $U = \bigcup\limits_{i \in I} \mathcal{O}_i = \bigcup\limits_{s \in S} \mathcal{O}_s \cup \bigcup\limits_{l \in L} \mathcal{O}_l$

  such that the following 5 conditions hold.

  1. $S \cup L = I$
  2. $S \cap L = \emptyset$
  3. $(\forall i \in I)\,(\mathcal{O}_i \subseteq X \vee \mathcal{O}_i \subseteq Y)$
  4. $(\forall l \in L)\,(\mathcal{O}_l \cap Y \neq \emptyset)$
  5. $\bigcup\limits_{s \in S} \mathcal{O}_s \cup Y \neq U$

  The main equation and the conditions 1-3 should express that O_s for $s \in S$ and O_l for $l \in L$ are respectively "small" and "large" inverse neighborhoods of the values $\in \mathbf{2}$. One can view O_s and O_l as having measure 0 and > 0, respectively. The conditions 4 and 5 should express that X is both "nowhere dense" and "of the second category", respectively.

Proof. Otherwise, let $u \in U \setminus \left(\bigcup\limits_{s \in S} \mathcal{O}_s \cup Y \right)$ be fixed, by 5. Since $u \notin Y$, i.e. $\mathfrak{f}(u) \neq \mathbf{0}$, we have $\mathfrak{f}(u) = \mathbf{1}$, i.e. $u \in X$. Since $U = \bigcup\limits_{s \in S} \mathcal{O}_s \cup \bigcup\limits_{l \in L} \mathcal{O}_l$, there exists a $l \in L$ such that $u \in \mathcal{O}_l$. Thus $u \in \mathcal{O}_l \cap X$, and hence $\mathcal{O}_l \subseteq X$, by 3. However $\mathcal{O}_l \cap Y \neq \emptyset$, by 4, and hence $X \cap Y \neq \emptyset$ - a contradiction, Q.E.D.

Summary 2 *We wish to infer the desired hypothetical conclusion* $\mathsf{P} < \mathsf{NP}$ *from this proposition for a suitable* $U \subseteq \mathrm{DNF}$ *and* $\mathfrak{f} = \phi$*, while assuming that* $\phi(F) = \mathbf{1} \Leftrightarrow F \in \mathrm{TAU}_{\mathrm{DNF}}$*. The neighborhoods $\mathcal{O}_s$ and $\mathcal{O}_l$ satisfying 1-5 should arise by recursion on the number of states of the Turing machine M correlated with ϕ (abbr.: $\|M\|$). Basic case $\|M\| = 2$ is easy (Section 5.3 below). Formalization of the recursive step is more involved. To this end, we describe a new hierarchy of* PTIME*-computable functions that is parametric in $\|M\|$ (Section 4, especially 4.2.3 below). This work might be of interest regardless of the present context (see also first part of this introduction). The rest is yet to be elaborated.*

2 Basic Definitions

In this section, we revise Post-Turing concept of a human computation (see [P, Tu]).

Notation 3 *As compared to more familiar presentations, we strictly distinguish between tapes and machines proper. We regard a given Turing machine as a pair* $\langle T, M\rangle$ *consisting of tape* T *and machine proper* M*. Moreover,* M *determines the corresponding computation as a partial function* $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$ *where* $\mathbb{T}$ *is the correlated tape space. By* $\mathbb{N}$ *and* $\mathbb{Z}$ *we denote the natural numbers and the integers, respectively, and put* $\mathbb{N}^{\leq}_n := \{x \in \mathbb{N} \mid x \leq n\}$ *and* $\mathbb{N}^{>}_n := \{x \in \mathbb{N} \mid x > n\}$ *for any* $n \in \mathbb{N}$*. Generally, by* $[X \to Y]$ *and* $[X \curvearrowright Y]$ *we denote the set of all total functions and all partial functions from* X *to* Y*, respectively. Moreover, for any* $\mathfrak{f} \in [X \curvearrowright Y]$ *and* $x \in X$*, we denote by* $\mathtt{Dom}(\mathfrak{f})$ *the domain of* $\mathfrak{f}$*, and express by* $\mathfrak{f}(x)\downarrow$ *and* $\mathfrak{f}(x)\uparrow$ *the conditions* $(\exists y \in Y)\,\mathfrak{f}(x) = y$ *and* $(\forall y \in Y)\,\mathfrak{f}(x) \neq y$*, respectively. Thus* $\mathtt{Dom}(\mathfrak{f}) = \{x \in X \mid \mathfrak{f}(x)\downarrow\} \subseteq X$*.*

2.1 The Tape Space $\mathbb{T}$

Consider a fixed finite set $\{0\} \subseteq \mathcal{A} \subset \mathbb{N}$, and let $\mathcal{A}_0 = \mathcal{A} \cap \mathbb{N}^{>}_0$. $\mathcal{A}$ is called the *alphabet*, the elements of $\mathcal{A}_0$ are called the (gödelnumbers of) *letters*, 0 being the canonical number of the empty character $\emptyset$. With every tape T in the alphabet $\mathcal{A}$ (abbr.: $\mathcal{A}$-tape) is correlated its *display function* $\varphi_T \in [\mathbb{Z} \to \mathcal{A}]$ whose values are characters printed on T. That is, $\varphi_T(0)$ is the character printed in the distinguished *start/stop* cell labelled by Ⓢ and, generally, $\varphi_T(z)$ is the character printed in $|z|^{\text{th}}$ cell to the right (left) of Ⓢ if $z > 0$ ($z < 0$). Moreover, we assume that $\#\{z \in \mathbb{Z} \mid \varphi_T(z) \neq 0\} < \infty$. Thus every $\varphi \in [\mathbb{Z} \to \mathcal{A}]$ with $\#\{z \in \mathbb{Z} \mid \varphi(z) \neq 0\} < \infty$ uniquely determines a $\mathcal{A}$-tape T with $\varphi = \varphi_T$. For brevity, in the sequel we drop subscripts '$_{\mathcal{A}}$' and '$_T$'. Now for any $d \in \mathbb{N}$ we set:

- $\mathbb{T}_{(d)} := \{\varphi \in [\mathbb{Z} \to \mathcal{A}] \mid (\forall z \in \mathbb{Z})\,((z \leq -d \vee z \geq d) \to \varphi(z) = 0)\}$
- $\mathbb{T} := \bigcup\limits_{d \in \mathbb{N}} \mathbb{T}_{(d)}$ thus being isomorphic to the space of all $\mathcal{A}$-tapes.[1]

For any $a, b \in \mathbb{N}$, by $\overleftarrow{0}, \varphi(-a), ..., \varphi(0), ..., \varphi(b), \overrightarrow{0}$ we denote a $\varphi \in \mathbb{T}$ such that $(\forall z \in \mathbb{Z})\,((z < -a \vee z > b) \to \varphi(z) = 0)$. The tape $\overleftrightarrow{0} := \lambda z.0$ is called *empty*; thus $\varphi \in \mathbb{T}_{(0)} \Leftrightarrow \varphi = \overleftrightarrow{0}$.

[1] Generally, we adopt traditional constructive realizability-interpretation of indexed sums according to which $x \in \bigcup\limits_{i \in I} Y_i$ is understood as $x \in Y_{f(x)} \wedge f(x) \in I$, for a given fixed algorithm f. This subtle distinction is not important for the contents, though, since we consider only recursive functions a/o functionals.

2.2 The Computation $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$

***M*-syntax.**

1. Consider a fixed finite set $\{0,1\} \subseteq \mathcal{S} \subset \mathbb{N}$, a metasymbol *'fill-in'* $\boxdot \notin \mathbb{N}$, and let $\mathcal{S}^- = \mathcal{S} \setminus \{1\}$ and $\mathcal{A}^+ = \mathcal{A} \cup \{\boxdot\}$. The pair $\langle \mathcal{A}^+, \mathcal{S} \rangle$ is called the *vocabulary* of a Turing machine M under consideration. The elements of $\mathcal{S}$ and $\mathcal{S}^-$ are called *states* and *workstates* (of the machine), while the elements of $\mathcal{A}_0$, $\mathcal{A}$ and $\mathcal{A}^+$ are called *letters*, *characters* and *protocharacters* (of the alphabet), respectively. It is assumed that 0 and 1 correspond to the initial and the final state of M, respectively.
2. Regard M as a partial finite function, as follows
   $M \in [\mathcal{A} \times \mathcal{S}^- \curvearrowright \{-1,0,1\} \times \mathcal{A}^+ \times \mathcal{S}]$
   whose graph is a fixed finite set of *orders* viewed as term-rewrite rules
   $\langle \mathfrak{a}, \alpha \rangle \hookrightarrow M(\mathfrak{a}, \alpha) = \langle \Delta(\mathfrak{a}, \alpha), \Pi(\mathfrak{a}, \alpha), \Sigma(\mathfrak{a}, \alpha) \rangle$
   which split into the three specifications, where $\mathfrak{a}, \mathfrak{b} \in \mathcal{A}$, $\alpha \in \mathcal{S}^-$, $\beta \in \mathcal{S}$:
   a) $\langle \mathfrak{a}, \alpha \rangle \hookrightarrow \langle 0, \mathfrak{b}, \beta \rangle$ *"rewrite* $\mathfrak{a}$ *to* $\mathfrak{b}$ *and change* α *to* β*"*,
   b) $\langle \mathfrak{a}, \alpha \rangle \hookrightarrow \langle 1, \boxdot, \beta \rangle$ *"move one cell to the right, change* α *to* β *and fill* $\boxdot$*"*,
   c) $\langle \mathfrak{a}, \alpha \rangle \hookrightarrow \langle -1, \boxdot, \beta \rangle$ *"move one cell to the left, change* α *to* β *and fill* $\boxdot$*"*.

   The left- and right-hand parts of a term-rewrite rule are called *redex* (= argument) and *contractum* (= value), respectively. Note that, by the functional uniformity of M, equal redexes have equal contracta. Hence $\Delta(\mathfrak{a}, \alpha)$, $\Pi(\mathfrak{a}, \alpha)$ and $\Sigma(\mathfrak{a}, \alpha)$ are uniquely determined by every $\langle \mathfrak{a}, \alpha \rangle \in \mathtt{Dom}(M)$. Thus $\Delta \in [\mathtt{Dom}(M) \to \{-1,0,1\}]$, $\Pi \in [\mathtt{Dom}(M) \to \mathcal{A}^+]$ and $\Sigma \in [\mathtt{Dom}(M) \to \mathcal{S}]$.

***M*-semantics.**

1. A machine M starts when in the state 0 while scanning the cell Ⓢ of the input $\mathcal{A}$-tape φ. Every single computation step n is uniquely determined by the above intended interpretation of the uniquely determined order whose redex $\langle \mathfrak{a}, \alpha \rangle$ coincides with the second component of the scanned position $\langle 0, \langle \mathfrak{a}, \alpha \rangle \rangle$ of the cell Ⓢ, whose gödelnumber is 0, provided that such order exists. To put it more precisely, let φ_n be the $\mathcal{A}$-tape obtained at step n. Consider *printed positions* as pairs $\langle z, \varphi_n(z) \rangle$, where $z \in \mathbb{Z}$, and let the *scanning position* be the triplet $\langle 0, \langle \varphi_n(0), \sigma_n \rangle \rangle$, where $\sigma_n \in \mathcal{S}$ is the state symbol obtained at the computtion step n. Moreover, we always assume $\sigma_0 = 0$. Now if $\varphi_n(0) = \mathfrak{a}$ and $\sigma_n = \alpha$ then the computation step $n+1$ according to the order $\langle \mathfrak{a}, \alpha \rangle \hookrightarrow \langle \Delta(\mathfrak{a}, \alpha), \Pi(\mathfrak{a}, \alpha), \Sigma(\mathfrak{a}, \alpha) \rangle$ rewrites every printed position $\langle z, \varphi(z) \rangle$ to $\langle z, \varphi_{n+1}(z) \rangle$, and the scanning position $\langle 0, \langle \varphi_n(0), \sigma_n \rangle \rangle$ to $\langle 0, \langle \varphi_{n+1}(0), \sigma_{n+1} \rangle \rangle$, such that $\varphi_{n+1}(z) := \Pi(\mathfrak{a}, \alpha)$ *if* $\Delta(\mathfrak{a}, \alpha) = 0$ *and* $z = 0$, *else* $:= \varphi_n(z + \Delta(\mathfrak{a}, \alpha))$, while $\sigma_{n+1} := \Sigma(\mathfrak{a}, \alpha)$. The machine keeps executing these computation steps so long as possible. The output is stipulated by cases, as follows. Either the machine arrives at the last computation step s in the scanning position $\langle 0, \langle \varphi_s(0), \sigma_s \rangle \rangle$ from which no further step

can be made, or else the computation never terminates. In the former case, there are two subcases depending on whether or not σ_s coincided with 1. Namely, if $\sigma_s = 1$ then the computation is over and the output $\mathcal{A}$-tape is φ_s. In the remaining subcase, and in the latter case, the output is undefined.

2. For any given $\mathcal{A}$ and $\mathcal{S}$, as above, every machine M with vocabulary $\langle \mathcal{A}^+, \mathcal{S} \rangle$ thus uniquely determines the required *computing mapping* (or just *computation*) $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$. Namely, for any given input $\varphi \in \mathbb{T}$, we let $\Phi_M(\varphi) \in \mathbb{T}$ be the correlated output, as above, if it is defined. Below we express this definition in more formal terms.
3. Let $\Phi_M(\varphi) := \begin{cases} \varphi_{\min\{n \mid \sigma_n = 1\}}, & \text{if } \exists n\,(\sigma_n = 1) \\ \text{undefined}, & \text{otherwise} \end{cases}$
   where $\sigma_n \in \mathcal{S}$, $\varphi_n \in \mathbb{T}$ arise as follows by simultaneous recursion on $n \in \mathbb{N}$.
   a) $\sigma_0 := 0$ and $\varphi_0 := \varphi$
   b) $\sigma_{n+1} := \Sigma(\varphi_n(0), \sigma_n)$, iff $\langle \varphi_n(0), \sigma_n \rangle \in \mathtt{Dom}(M)$
   c) $\varphi_{n+1}(z) := \begin{cases} \Pi(\varphi_n(0), \sigma_n), & \text{if } z = 0 \\ \varphi_n(z), & \text{if } z \neq 0 \end{cases}$,
      iff $\langle \varphi_n(0), \sigma_n \rangle \in \mathtt{Dom}(M)$ and $\Delta(\varphi_n(0), \sigma_n) = 0$
   d) $\varphi_{n+1}(z) = \varphi_n(z + \Delta(\varphi_n(0), \sigma_n))$,
      iff $\langle \varphi_n(0), \sigma_n \rangle \in \mathtt{Dom}(M)$ and $\Delta(\varphi_n(0), \sigma_n) \neq 0$

First normal form. We revise 2.2.2.3 in functional terms. To this end, we introduce two auxiliary functionals $\Re \in [\mathbb{T}\times\mathcal{S}^- \curvearrowright \mathbb{T}\times\mathcal{S}]$ *(recursion)*, $\Im \in [\mathbb{T} \times \mathcal{S}^- \times \mathbb{N} \curvearrowright \mathbb{T}\times\mathcal{S}]$ *(induction)* and for any $\mathbb{E}_0, ..., \mathbb{E}_k$, $i \leq k$, denote by $\mathsf{C}_i \in [\mathbb{E}_0 \times ... \times \mathbb{E}_k \to \mathbb{E}_i]$ the i^{th} Cartesian component of $\mathbb{E}_0 \times ... \times \mathbb{E}_k$. Thus in particular for any φ, α we have $\mathsf{C}_0(\varphi, \alpha) = \varphi$ and $\mathsf{C}_1(\varphi, \alpha) = \alpha$. The whole definition runs as follows.

1. Define $\Re \in [\mathbb{T}\times\mathcal{S}^- \curvearrowright \mathbb{T}\times\mathcal{S}]$ by setting
   $\mathtt{Dom}(\Re) := \{\langle \varphi, \alpha \rangle \in \mathbb{T}\times\mathcal{S}^- \mid \langle \varphi(0), \alpha \rangle \in \mathtt{Dom}(M)\}$
   and $\Re(\varphi, \alpha) := \Re_{\Delta(\varphi(0),\alpha)}(\varphi, \alpha)$,
   where:
   a) $\Re_0(\varphi, \alpha) := \left\langle \lambda z. \begin{cases} \Pi(\varphi(0), \alpha), & \text{if } z = 0 \\ \varphi(z), & \text{if } z \neq 0 \end{cases}, \Sigma(\varphi(0), \alpha) \right\rangle$,
      provided that $\langle \varphi(0), \alpha \rangle \in \mathtt{Dom}(M)$ and $\Delta(\varphi(0), \alpha) = 0$
   b) $\Re_d(\varphi, \alpha) := \langle \lambda z.\varphi(z + d), \Sigma(\varphi(0), \alpha) \rangle$,
      provided that $\langle \varphi(0), \alpha \rangle \in \mathtt{Dom}(M)$ and $\Delta(\varphi(0), \alpha) = d \neq 0$
2. Define $\Im \in [\mathbb{T} \times \mathcal{S}^- \times \mathbb{N} \curvearrowright \mathbb{T}\times\mathcal{S}]$ by setting
   $\mathtt{Dom}(\Im) :=$
   $$\left\{ \langle \varphi, \alpha, n \rangle \in \mathbb{T} \times \mathcal{S}^- \times \mathbb{N} \mid (\forall m \leq n) \underbrace{\Re(...\Re}_{m}(\varphi, \alpha)...) \in \mathtt{Dom}(\Re) \right\}$$
   and $\begin{cases} \Im(\varphi, \alpha, 0) := \langle \varphi, \alpha \rangle \\ \Im(\varphi, \alpha, n+1) := \Re(\Im(\varphi, \alpha, n)), \text{ iff defined} \end{cases}$
3. Call the following equation the *first normal form* of $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$.
   $\Phi_M(\varphi) = \begin{cases} \mathsf{C}_0\left(\Im\left(\varphi, 0, \min\left\{n \in \mathbb{N} \mid \mathsf{C}_1(\Im(\varphi, 0, n)) = 1\right\}\right)\right), & \text{if defined} \\ \text{undefined}, & \text{otherwise} \end{cases}$
   Hence $\mathtt{Dom}(\Phi_M) :=$
   $\left\{\varphi \in \mathbb{T} \mid (\exists n \in \mathbb{N})\left(\left\langle \varphi, 0, n \right\rangle \in \mathtt{Dom}(\Im) \wedge \mathsf{C}_1(\Im(\varphi, 0, n)) = 1\right)\right\}$

3 The Recursion on the Number of States

In this section, we wish to redefine previous semantic definitions of $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$ *by recursion on the state-rank* $\|M\|$*, where* $\|M\| := \#(\mathcal{S})$ *is the total number of states of the underlying machine* M*.*

3.1 Loose Description

We assume that the initial case refers to M having only two states 0 and 1. The correlated initial mapping $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$ is understood as in 2.2 above - so far no revision is required. Consider the recursion step. Suppose $\#(\mathcal{S}) = p = q+1 > 2$, and let $\tau = \max(\mathcal{S})$. Consider the correlated mapping $\Phi_M \in [\mathbb{T} \curvearrowright \mathbb{T}]$. In the sequel we abbreviate by $(\alpha \hookrightarrow)$ and $(\hookrightarrow \alpha)$, respectively, any order whose redex state is α, but the contractum state in not α, and any order whose contractum state is α, but the redex state is not α, while by (α) and $(\alpha \hookrightarrow \alpha)$, respectively, we'll denote any order whose either redex or contractum state is α and any order whose redex and contractum states are both α. Now consider the (not necessarily disjunct) decomposition

– $M = N_0 \cup N_1 \cup N_2 \cup N_3$

where N_0, N_1, N_2, N_3 arise from M by respectively deleting:

1. all orders (τ) and $(\tau \hookrightarrow \tau)$
2. all orders $(\tau \hookrightarrow)$, $(\tau \hookrightarrow \tau)$ and $(\hookrightarrow 1)$
3. all orders (0) and $(0 \hookrightarrow 0)$
4. all orders $(0 \hookrightarrow)$, $(0 \hookrightarrow 0)$ and $(\hookrightarrow 1)$

We replace each N_i, $i < 4$, by a suitable machine M_i in the same alphabet $\mathcal{A}$, whose set of states is $\mathcal{S}^\tau := \mathcal{S} \setminus \{\tau\}$. Now each M_i arises by renaming some states in the orders of N_i, for the sake of τ-elimination. Loosely speaking, we treat τ as a new 0. Renaming $(\alpha \hookrightarrow)$ (or $(\hookrightarrow \alpha), (\alpha), (\alpha \hookrightarrow \alpha)$) by $(\beta \hookrightarrow)$ (or $(\hookrightarrow \beta), (\beta), (\beta \hookrightarrow \beta)$) is understood literally as rewriting every state-occurrence α under consideration to β. The renaming in question is specified with respect to the above clauses 1-4, i.e. M_i corresponding to the clause $i+1$:

1. no renaming required, hence $M_0 = N_0$
2. rename every $(\hookrightarrow \tau)$ by $(\hookrightarrow 1)$
3. rename every $(\tau) \mid (\tau \hookrightarrow \tau)$ by $(0) \mid (0 \hookrightarrow 0)$, respectively
4. rename every $(\tau) \mid (\tau \hookrightarrow \tau) \mid (\hookrightarrow 0)$ by $(0) \mid (0 \hookrightarrow 0) \mid (\hookrightarrow 1)$, respectively

Obviously $\|M_i\| = \#(\mathcal{S}^\tau) = q < \|M\|$ holds for all $i < 4$. Hence by the recursion, with every M_i, $i < 4$, we can correlate the appropriate functional description of the corresponding mapping $\Phi_i := \Phi_{M_i} \in [\mathbb{T} \curvearrowright \mathbb{T}]$. This enables us to revise, in the functional terms, previous semantic definition of the mapping Φ_M. Namely, it is readily seen that every input $\varphi \in \mathbb{T}$ uniquely determines $\Phi_M(\varphi) \in \mathbb{T}$ as a composition of the either form:

$\Phi_0(\varphi)$,
$\Phi_1 \circ \Phi_2(\varphi)$,
$\Phi_1 \circ \Phi_3 \circ \Phi_0(\varphi)$,
$\Phi_1 \circ \Phi_3 \circ \Phi_1 \circ \Phi_2(\varphi)$,
$\Phi_1 \circ \Phi_3 \circ \Phi_1 \circ \Phi_3 \circ \Phi_0(\varphi) = (\Phi_1 \circ \Phi_3)^2 \circ \Phi_0(\varphi)$,
$\Phi_1 \circ \Phi_3 \circ \Phi_1 \circ \Phi_3 \circ \Phi_1 \circ \Phi_2(\varphi) = (\Phi_1 \circ \Phi_3)^2 \circ \Phi_1 \circ \Phi_2(\varphi)$,

etc., where we put $\Phi_i \circ \Phi_j \circ ... \circ \Phi_k \circ \Phi_l(\varphi) = \Phi_l(\Phi_k(...\Phi_j(\Phi_i(\varphi))...))$ and $(\Phi_i \circ ... \circ \Phi_j)^n(\varphi) = \underbrace{(\Phi_i \circ ... \circ \Phi_j) \circ ... \circ (\Phi_i \circ ... \circ \Phi_j)}_{n \text{ times}}(\varphi)$.

3.2 Explanation

Let φ be any given input tape. Let be a given machine M with a distinguished state symbol $\tau \notin \{0,1\}$. Suppose $\Phi_M(\varphi) \downarrow$, and consider the process of computing the output $\Phi_M(\varphi)$. Recall that for any n we denote by σ_n the state symbol of the scanning position at computation step n. First suppose that we arrived at final step s with $\sigma_s = 1$ such that $(\forall n < s)\, \sigma_n \neq \tau$. Then we have $\Phi_M(\varphi) = \Phi_{M_0}(\varphi)$, i.e. the output can just as well be obtained by using instead of M the machine M_0 that arises by dropping all orders containing state symbol τ, which yields $\|M_0\| < \|M\|$. Otherwise, let $m_0 := \min\{n < s \mid \sigma_n = \tau\}$, and consider partial result φ_{m_0} obtained at step m_0. Since m_0 yields the first occurrence of τ, while in form $(\hookrightarrow \tau)$ and in the absence of $(\hookrightarrow 1)$, we have $\varphi_{m_0} = \Phi_{M_1}(\varphi)$, where M_1 arises by rewriting in $N_1 \subseteq M$ every $(\hookrightarrow \tau)$ to the corresponding $(\hookrightarrow 1)$, which by definition of N_1 yields $\|M_1\| < \|M\|$. Consider the rest of computation. First suppose that in the remainder we don't use state symbol 0, i.e. $(\forall n : m_0 < n < s)\, \sigma_n \neq 0$. Then we can pass from φ_{m_0} to the output $\Phi_M(\varphi)$ by using instead of M the machine M_2 that arises by rewriting in $N_2 \subseteq M$ every state-occurrence τ by 0. This obviously yields $\|M_2\| < \|M\|$, while being admissible by definition of N_2. Summing up, in this case we have $\Phi_M(\varphi) = \Phi_{M_2}(\varphi_{m_0}) = \Phi_{M_2}(\Phi_{M_1}(\varphi)) = \Phi_{M_1} \circ \Phi_{M_2}(\varphi)$. Otherwise, let $m_1 := \min\{m_0 < n < s \mid \sigma_n = 0\}$ and consider partial result φ_{m_1} obtained at step m_1. Since m_1 provides us with the first occurrence of 0, while in form $(\hookrightarrow 0)$ and in the absence of $(\hookrightarrow 1)$ and , we have $\varphi_{m_1} = \Phi_{M_3}(\varphi_{m_0})$, where M_3 arises from M by rewriting in $N_3 \subseteq M$ every state-occurrence τ by 0, while simultaneously replacing $(\hookrightarrow 0)$ by $(\hookrightarrow 1)$. This obviously yields $\|M_3\| < \|M\|$, while being admissible by definition of N_3. Suppose that state symbol τ does not occur in the rest of computation. Then we can safely replace there M by M_0 (see above), which yields $\Phi_M(\varphi) = \Phi_{M_0}(\varphi_{m_1}) = \Phi_{M_0}(\Phi_{M_3}(\varphi_{m_0})) = \Phi_{M_0}(\Phi_{M_3}(\Phi_{M_1}(\varphi))) = \Phi_{M_1} \circ \Phi_{M_3} \circ \Phi_{M_0}(\varphi)$. Otherwise, we take the first remaining occurrence of state symbol τ and ask if state symbol 0 occurs in the remainder. If this is not the case, by previous argument we arrive at $\Phi_M(\varphi) = \Phi_{M_1} \circ \Phi_{M_3} \circ \Phi_{M_1} \circ \Phi_{M_2}(\varphi)$. Otherwise, we take the first remaining occurrence of state symbol 0 and ask if state symbol τ occurs in the remainder, and so on. As a whole, this yields a recursion with entails iterated compositions $(\Phi_{M_1} \circ \Phi_{M_3})^n$ followed either by Φ_{M_0} or $\Phi_{M_1} \circ \Phi_{M_2}$ - as shown in 3.1. In 3.3

below we just formalize previous description - note that $\zeta_M(\varphi, i)$ corresponds to m_i, while $\iota_M(\varphi, i)$ being the index of the correlated machine M_ι.

One can also view it as a finite automaton[2], where 'q' denotes the critical state τ:

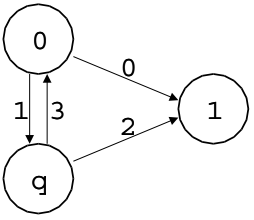


Fig. 1. A graph of the automaton

3.3 More Precise Description

Characteristic functions. For any given machine M, we define the auxiliary partial *state function* $\sigma_M \in [\mathbb{T} \times \mathbb{N} \rightharpoonup \mathcal{S}]$ by setting $\sigma_M(\varphi, n) := \sigma_n$ where σ_n arises as in 2.2.2.3 with respect to φ. Furthermore, for any $i < 4$ and the adjacent M_i, Φ_i, $i < 4$ (see 3.1 above), we define one partial *index function* $\iota_M \in [\mathbb{T} \times \mathbb{N} \rightharpoonup \{0,1,2,3\}]$ and one partial *time function* $\zeta_M \in [\mathbb{T} \times \mathbb{N} \rightharpoonup \mathbb{N}_0^>]$ by simultaneous recursion, as follows, where as above $\tau = \max(\mathcal{S})$, etc. (For brevity we'll often drop the subscript '$_M$' and dummy variable 'φ' in $\sigma_M(\varphi, n)$, $\iota_M(\varphi, n)$ and $\zeta_M(\varphi, n)$.)

1. $\iota_M(\varphi, 0) := \begin{cases} 0, \text{ if } (\exists n > 0)\,(\sigma(n) = 1 \wedge (\forall m : 0 < m < n)\,\sigma(m) \neq \tau) \\ 1, \text{ if } (\exists n > 0)\,\sigma(n) = \tau \\ \text{otherwise undefined} \end{cases}$

2. $\zeta_M(\varphi, 0) := \begin{cases} \min\{n > 0 \mid \sigma(n) = \tau\}, \text{ if } \iota(0) = 1 \\ \text{otherwise undefined} \end{cases}$

3. $\iota_M(\varphi, i+1) := \begin{cases} 0, \text{ if } \left[\begin{array}{l} \iota(i) = 3 \wedge (\exists n > \zeta(i)) \\ (\sigma(n) = 1 \wedge (\forall m : \zeta(i) < m < n)\,\sigma(m) \neq \tau) \end{array}\right] \\ 1, \text{ if } \iota(i) = 3 \wedge (\exists n > \zeta(i))\,\sigma(n) = \tau \\ 2, \text{ if } \left[\begin{array}{l} \iota(i) = 1 \wedge (\exists n > \zeta(i)) \\ (\sigma(n) = 1 \wedge (\forall m : \zeta(i) < m < n)\,\sigma(m) \neq 0) \end{array}\right] \\ 3, \text{ if } \iota(i) = 1 \wedge (\exists n > \zeta(i))\,\sigma(n) = 0 \end{cases}$

4. $\zeta_M(\varphi, i+1) := \begin{cases} \min\{n > \zeta(i) \mid \sigma(n) = \tau\}, \text{ if } \iota(i+1) = 1 \\ \min\{n > \zeta(i) \mid \sigma(n) = 0\}, \text{ if } \iota(i+1) = 3 \\ \text{otherwise undefined} \end{cases}$

[2] This idea was due to Andreas Krebs.

Length function and recursive expansion. With any given M we correlate its partial *length function* $\ell_M \in \left[\mathbb{T} \curvearrowright \mathbb{N}_0^{>}\right]$:

1. $\ell_M(\varphi) := \min\{n \mid \iota_M(\varphi, n) \in \{0, 2\}\}$, iff defined.
   (For brevity, we'll often drop the subscript $\text{`}_M\text{'}$.)
2. Having this, we obtain the following *recursive expansion* of Φ_M.
   $\Phi_M(\varphi) = \Phi_{\iota(\varphi,0)} \circ ... \circ \Phi_{\iota(\varphi,\ell(\varphi))}(\varphi)$, iff defined.
3. To put it in a more transparent form, we have
   $$\Phi_M^{(q+1)}(\varphi) = \begin{cases} \left(\Phi_1^{(q)} \circ \Phi_3^{(q)}\right)^{\left[\frac{1}{2}\ell(\varphi)\right]} \circ \Phi_0^{(q)}(\varphi), & \text{iff } \iota(\varphi, \ell(\varphi)) = 0 \\ \left(\Phi_1^{(q)} \circ \Phi_3^{(q)}\right)^{\left[\frac{1}{2}\ell(\varphi)\right]-1} \circ \Phi_1^{(q)} \circ \Phi_2^{(q)}(\varphi), & \text{iff } \iota(\varphi, \ell(\varphi)) = 2 \end{cases}$$
   where the superscripts $\text{`}^{(q+1)}\text{'}$ and $\text{`}^{(q)}\text{'}$ refer to the number of states of M and M_i, respectively. This expansion will be further specified in the next section.

4 Transitive Normal Forms

We wish to prolong the recursive expansion 3.3.2.3 in order to obtain explicit functional expansion of Φ_M referring to the computations of the smallest state-rank 2.

Notation 4 *Basically, we wish to regard Φ_M as a functional composition whose components are basic computations of the state-rank* 2*, possibly decorated a/o extended by auxiliary combinatorial operations. It is assumed that all these components are effectively determined by M.*

4.1 Transitive Recursive Expansion

We wish to eliminate the index and length functions which figure in the recursive expansion 3.3.2.3. Let $p = \|M\|$. We replace $\Phi_M = \Phi_M^{(p)}$ by the *transitive functional computation* $\Psi_M = \Psi_M^{(p)} \in \left[\mathbb{T} \times \mathbb{N}_p^{\leq} \times \mathbb{N} \to \mathbb{T} \times \mathbb{N}_p^{\leq}\right]$. To this end, we let $\Psi_M^{(p)} := \Psi_M^{(p,p)}$, where $\Psi_M^{(q,r)}$ is defined for all $q \leq r$ such that $\Psi_M^{(2,r)} \in \left[\mathbb{T} \times \mathbb{N}_r^{\leq} \to \mathbb{T} \times \mathbb{N}_r^{\leq}\right]$ and $\Psi_M^{(q,r)} \in \left[\mathbb{T} \times \mathbb{N}_r^{\leq} \times \mathbb{N}^{\leq} \to \mathbb{T} \times \mathbb{N}_r^{\leq}\right]$ if $q > 2$, while also using two auxiliary combinatorial functionals $\mathsf{Q}_0^{(q,r)}, \mathsf{Q}_1^{(q,r)} \in \left[\mathbb{N}_r^{\leq} \to \mathbb{N}_r^{\leq}\right]$ where $q < r$. Furthermore, for the sake of brevity, with any $\mathfrak{F} \in \left\{\Psi_M^{(s,r)}, \mathsf{Q}_\iota^{(s,r)}\right\}$ we correlate $\widehat{\mathfrak{F}} \in \left[\mathbb{T} \times \mathbb{N}_r^{\leq} \times \mathbb{N} \to \mathbb{T} \times \mathbb{N}_r^{\leq} \times \mathbb{N}\right]$ by setting $\widehat{\mathfrak{F}}(\varphi, i, t) := \langle \psi, j, t\rangle$ or $:= \langle \varphi, j, t\rangle$ such that $\langle \psi, j\rangle = \mathfrak{F}(\varphi, i)$, $\langle \psi, j\rangle = \mathfrak{F}(\varphi, i, t)$ or $j = \mathfrak{F}(i)$, respectively. The main definition runs by recursion on $p \geq 2$, as shown below, where the subscripts $\text{`}_i\text{'}$ in $\widehat{\Psi}_i^{(q,r)}$ for $i < 4$ refer to M_i being defined as above via the decomposition $M = N_0 \cup N_1 \cup N_2 \cup N_3$.

1. Let $\Psi_M^{(2,r)} \in \left[\mathbb{T} \times \mathbb{N}_r^{\leq} \to \mathbb{T} \times \mathbb{N}_r^{\leq}\right]$ by setting

$$\Psi_M^{(2,r)}(\varphi, i) := \begin{cases} \langle \varphi, i \rangle, & \text{if } i \neq 1 \\ \begin{cases} \left\langle \Phi_M^{(2)}(\varphi), 2 \right\rangle, & \text{if } \Phi_M^{(2)}(\varphi) \downarrow \\ \langle \varphi, i \rangle, & \text{otherwise} \end{cases}, & \text{if } i = 1 \end{cases}$$ [3]

2. Let $\mathsf{Q}_0^{(q,r)}, \mathsf{Q}_1^{(q,r)} \in \left[\mathbb{N}_r^{\leq} \to \mathbb{N}_r^{\leq}\right]$ by setting:

a) $$\mathsf{Q}_0^{(q,r)}(i) := \begin{cases} i, & \text{if } i \neq q \\ i+1, & \text{if } i = q \end{cases}$$

b) $$\mathsf{Q}_1^{(q,r)}(i) := \begin{cases} i, & \text{if } i \notin \{1, q\} \\ 0, & \text{if } i = 1 \\ 1, & \text{if } i = q \end{cases}$$

3. If $p = q + 1 > 2$ then let $\Psi_M^{(q+1,r)} \in \left[\mathbb{T} \times \mathbb{N}_r^{\leq} \times \mathbb{N} \to \mathbb{T} \times \mathbb{N}_r^{\leq}\right]$ by setting:

a) If $i \neq 1$ then $\Psi_M^{(q+1,r)}(\varphi, i, t) := \langle \varphi, i \rangle$

b) $\Psi_M^{(q+1,r)}(\varphi, 1, 0) := \langle \varphi, 0 \rangle$

c) If $t > 0$ then $\Psi_M^{(q+1,r)}(\varphi, 1, t) :=$

$$\left(\widehat{\Psi}_0^{(q,r)} \circ \widehat{\mathsf{Q}}_0^{(q,r)} \circ \widehat{\Psi}_1^{(q,r)} \circ \widehat{\mathsf{Q}}_1^{(q,r)} \circ \widehat{\Psi}_2^{(q,r)} \circ \widehat{\mathsf{Q}}_0^{(q,r)} \circ \widehat{\Psi}_3^{(q,r)} \circ \widehat{\mathsf{Q}}_1^{(q,r)}\right)^t (\varphi, 1, t)$$

These equations we refer to as the *transitive recursive expansions* of Ψ_M. According to the intended interpretation, t is a "global-time parameter" approximating length functions occurring in all recursive expansions 3.3.2 which participate in the final composite expression of $\Phi_M^{(p)}$.

4. The relationship between $\Psi_M^{(p)}$ and $\Phi_M^{(p)}$ is as follows. Let $t \in \mathbb{N}_0$ and suppose $\Phi_M^{(p)}(\varphi) \downarrow$. Then, by 3.3, we easily arrive at:

a) $$\mathsf{C}_1\left(\Psi_M^{(p)}(\varphi, 1, t)\right) = p \Leftrightarrow \mathsf{C}_0\left(\Psi_M^{(p)}(\varphi, 1, t)\right) = \Phi_M^{(p)}(\varphi)$$

b) $$\Phi_M^{(p)}(\varphi) = \mathsf{C}_0\left(\Psi_M^{(p)}\left(\varphi, 1, \min\left\{t \in \mathbb{N} \mid \mathsf{C}_1\left(\Psi_M^{(p)}(\varphi, 1, t)\right) = p\right\}\right)\right)$$

c) $$\Phi_M^{(p)}(\varphi) = \mathsf{C}_0\left(\Psi_M^{(p)}(\varphi, 1, t)\right), \text{ if } t \geq \min\left\{n \in \mathbb{N} \mid \mathsf{C}_1(\Im(\varphi, 0, n)) = 1\right\}$$

4.2 Explanation

To illuminate the method, consider first non-trivial case $p = 3$. By definition 4.1.3 (c) we have

$$\Psi_M^{(3)}(\varphi, 1, t) = \Psi_M^{(3,3)}(\varphi, 1, t)$$
$$:= \left(\widehat{\Psi}_0^{(2,3)} \circ \widehat{\mathsf{Q}}_0^{(2,3)} \circ \widehat{\Psi}_1^{(2,3)} \circ \widehat{\mathsf{Q}}_1^{(2,3)} \circ \widehat{\Psi}_2^{(2,3)} \circ \widehat{\mathsf{Q}}_0^{(2,3)} \circ \widehat{\Psi}_3^{(2,3)} \circ \widehat{\mathsf{Q}}_1^{(2,3)}\right)^t (\varphi, 1, t)$$

[3] The whole definition-by-cases is decidable, since the underlying M has only two states (see also Remark 7 below).

This results in the following definition by cases.

1. Suppose $\Phi_0^{(2)}(\varphi)\downarrow$. Hence:
   a) $\widehat{\Psi}_0^{(2,3)}(\varphi,1,t)=\left\langle \Phi_0^{(2)}(\varphi),2,t\right\rangle$
   b) $\widehat{\mathsf{Q}}_0^{(2,3)}\left(\Phi_0^{(2)}(\varphi),2,t\right)=\left\langle \Phi_0^{(2)}(\varphi),3,t\right\rangle$
   c) $\widehat{\Psi}_1^{(2,3)}\left(\Phi_0^{(2)}(\varphi),3,t\right)=\left\langle \Phi_0^{(2)}(\varphi),3,t\right\rangle$
   d) $\widehat{\mathsf{Q}}_1^{(2,3)}\left(\Phi_0^{(2)}(\varphi),3,t\right)=\left\langle \Phi_0^{(2)}(\varphi),3,t\right\rangle$
   e) etc., which yields $\Psi_M^{(3,3)}(\varphi,1,t)=\left\langle \Phi_0^{(2)}(\varphi),3,t\right\rangle$
2. Suppose $\Phi_0^{(2)}(\varphi)\uparrow$. Hence:
   a) $\widehat{\Psi}_0^{(2,3)}(\varphi,1,t)=\langle\varphi,1,t\rangle$
   b) $\widehat{\mathsf{Q}}_0^{(2,3)}(\varphi,1,t)=\langle\varphi,1,t\rangle$
   c) Suppose $\Phi_1^{(2)}(\varphi)\uparrow$. Hence:
      i. $\widehat{\Psi}_1^{(2,3)}(\varphi,1,t)=\langle\varphi,1,t\rangle$
      ii. $\widehat{\mathsf{Q}}_1^{(2,3)}(\varphi,1,t)=\langle\varphi,0,t\rangle$
      iii. $\widehat{\Psi}_2^{(2,3)}(\varphi,0,t)=\langle\varphi,0,t\rangle$
      iv. $\widehat{\mathsf{Q}}_0^{(2,3)}(\varphi,0,t)=\langle\varphi,0,t\rangle$
      v. etc., which yields $\Psi_M^{(3,3)}(\varphi,1,t)=\langle\varphi,0,t\rangle$
   d) Suppose $\Phi_1^{(2)}(\varphi)\downarrow$. Hence:
      i. $\widehat{\Psi}_1^{(2,3)}(\varphi,1,t)=\left\langle \Phi_1^{(2)}(\varphi),2,t\right\rangle$
      ii. $\widehat{\mathsf{Q}}_1^{(2,3)}\left(\Phi_1^{(2)}(\varphi),2,t\right)=\left\langle \Phi_1^{(2)}(\varphi),1,t\right\rangle$
      iii. Suppose $\Phi_2^{(2)}\left(\Phi_1^{(2)}(\varphi)\right)\downarrow$. Hence:
         A. $\widehat{\Psi}_2^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle \Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),2,t\right\rangle$
         B. $\widehat{\mathsf{Q}}_0^{(2,3)}\left(\Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),2,t\right)=\left\langle \Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right\rangle$
         C. $\widehat{\Psi}_3^{(2,3)}\left(\Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right)=\left\langle \Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right\rangle$
         D. $\widehat{\mathsf{Q}}_1^{(2,3)}\left(\Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right)=\left\langle \Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right\rangle$
         E. etc., which yields $\Psi_M^{(3,3)}(\varphi,1,t)=\left\langle \Phi_1^{(2)}\circ\Phi_2^{(2)}(\varphi),3,t\right\rangle$
      iv. Suppose $\Phi_2^{(2)}\left(\Phi_1^{(2)}(\varphi)\right)\uparrow$. Hence:
         A. $\widehat{\Psi}_2^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle \Phi_1^{(2)}(\varphi),1,t\right\rangle$
         B. $\widehat{\mathsf{Q}}_0^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle \Phi_1^{(2)}(\varphi),1,t\right\rangle$
         C. Suppose $\Phi_3^{(2)}\left(\Phi_1^{(2)}(\varphi)\right)\uparrow$. Hence:
            1. $\widehat{\Psi}_3^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle \Phi_1^{(2)}(\varphi),1,t\right\rangle$
            2. $\widehat{\mathsf{Q}}_1^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle \Phi_1^{(2)}(\varphi),0,t\right\rangle$
            3. etc., which yields $\Psi_M^{(3,3)}(\varphi,1,t)=\left\langle \Phi_1^{(2)}(\varphi),0,t\right\rangle$

D. Suppose $\Phi_3^{(2)}\left(\Phi_1^{(2)}(\varphi)\right)\downarrow$. Hence:
1. $\widehat{\Psi}_3^{(2,3)}\left(\Phi_1^{(2)}(\varphi),1,t\right)=\left\langle\Phi_1^{(2)}\circ\Phi_3^{(2)}(\varphi),2,t\right\rangle$
2. $\widehat{\mathsf{Q}}_1^{(2,3)}\left(\Phi_1^{(2)}\circ\Phi_3^{(2)}(\varphi),2,t\right)=\left\langle\Phi_1^{(2)}\circ\Phi_3^{(2)}(\varphi),1,t\right\rangle$
3. Suppose $\Phi_0^{(2)}\left(\Phi_1^{(2)}\circ\Phi_3^{(2)}(\varphi)(\varphi)\right)\downarrow$, etc. (see 1 above).
4. Suppose $\Phi_0^{(2)}\left(\Phi_1^{(2)}\circ\Phi_3^{(2)}(\varphi)(\varphi)\right)\uparrow$, etc. (see 2 above).

4.3 PTIME Normal Form

Suppose M is a PTIME*-machine of polynomial degree $m\in\mathbb{N}_0^{>}$.* That is, $\mathtt{Dom}(\Phi_M)=\mathbb{T}$, i.e. $\Phi_M(\varphi)\downarrow$ holds for every $\varphi\in\mathbb{T}$; furthermore, for any $\varphi\in\mathbb{T}_{(d)}$ the computation of $\Phi_M(\varphi)$ requires $\leq(2d-1)^m$ steps. Hence by the first normal form (2.2.3) we have

1. $(\forall d\in\mathbb{N}_0^{>})\left(\forall\varphi\in\mathbb{T}_{(d)}\right)\min\left\{n\in\mathbb{N}\mid\mathsf{C}_1(\Im(\varphi,0,n))=1\right\}\leq(2d-1)^m$
2. From this, by 4.1.4, we arrive at the following PTIME*-characteristic equations*, where as above $p=\|M\|$, $d\in\mathbb{N}_0^{>}$ and $\varphi\in\mathbb{T}_{(d)}$.
   a) $\Phi_M(\varphi)=\mathsf{C}_0(\Psi_M(\varphi,1,(2d-1)^m))$
   b) $p=\mathsf{C}_1(\Psi_M(\varphi,1,(2d-1)^m))$

   In the remainder of this section we use the transitive recursive expansions (4.1.3) in order to obtain the correlated explicit normal form of an arbitrary PTIME-computation Φ_M. This normal form provides us with the desired hierarchy of PTIME-computable recursive functions which is parametric in the number of states $\|M\|$ of the underlying Turing machines. To this end, it will suffice to complete the recursive definition of $\Psi_M=\Psi_M^{(p)}\in\left[\mathbb{T}\times\mathbb{N}_p^{\leq}\times\mathbb{N}\to\mathbb{T}\times\mathbb{N}_p^{\leq}\right]$. With $\mathfrak{F}=\mathsf{Q}_\iota^{(s,p)}$ we correlate $\widetilde{\mathfrak{F}}\in\left[\mathbb{T}\times\mathbb{N}_p^{\leq}\to\mathbb{T}\times\mathbb{N}_p^{\leq}\right]$ by setting $\widetilde{\mathfrak{F}}(\varphi,i):=\langle\varphi,\mathfrak{F}(i)\rangle$. The definition runs as follows, where composed subscripts '$i_1\ldots i_n$' in $\Psi_{i_1\ldots i_n}^{(2,p)}$ refer to the corresponding iterated decompositions of the shape $(\ldots(M_{i_1})\ldots)_{i_n}$; moreover, for the sake of brevity, we drop '$\circ$' in the functional compositions $\widetilde{\mathsf{Q}}_{j_2}^{(2,p)}\circ\ldots\circ\widetilde{\mathsf{Q}}_{j_s}^{(s,p)}$. Moreover, we set $\mathrm{mod}(x):=x-2\left[\frac{1}{2}x\right]$.
3. **Conclusion.** We revise the operator Ψ_M. Suppose $p=\|M\|$, $t\in\mathbb{N}_0^{>}$, $d\in\mathbb{N}_0$, $\varphi\in\mathbb{T}_{(d)}$. It is not difficult to verified, by tedious calculations, that one can rewrite Ψ_M, and hence Φ_M, by 4.3.2, to the desired PTIME *normal form*:
   a) $\Psi_M(\varphi,i,t)=\Upsilon_0^{(p)}\circ\Upsilon_1^{(p)}\circ\ldots\circ\Upsilon_{8\cdot4^{p-3}t^{p-2}-1}^{(p)}(\varphi,i)$
   b) $\Phi_M(\varphi)=\mathsf{C}_0\left(\Upsilon_0^{(p)}\circ\Upsilon_1^{(p)}\circ\ldots\circ\Upsilon_{8\cdot4^{p-3}(2d-1)^{m(p-2)}-1}^{(p)}(\varphi,1)\right)$

   where each $\Upsilon_\nu^{(p)}\in\left[\mathbb{T}\times\mathbb{N}_p^{\leq}\to\mathbb{T}\times\mathbb{N}_p^{\leq}\right]$ is defined as follows.
   i. If $p=3$ then $8t>\nu=8k_1+k_0$ where $k_0<8$, $k_1<t$ and

$$\Upsilon_\nu^{(3)}:=\begin{cases}\Psi_{\frac{1}{2}k_0}^{(2,3)}, & \text{if } k_0\in\{0,2,4,6\}\\ \widetilde{\mathsf{Q}}_{\mathrm{mod}\left(\frac{1}{2}(k_0-1)\right)}^{(2,3)}, & \text{if } k_0\in\{1,3,5,7\}\end{cases}$$

ii. If $p>3$ then $8\cdot 4^{p-3}t^{p-2}>\nu=$
$8\cdot 4^{p-3}t^{p-3}k_{2(p-3)+1}+8\cdot 4^{p-4}t^{p-3}k_{2(p-3)}+...+8\cdot 4tk_3+8tk_2+8k_1+k_0$
where $k_0<8$, $k_1<t$, $k_{2(j+1)}<4$, $k_{2(j+1)+1}<t$. We set $t=r+1$ and

$$\Upsilon^{(p)}_{\nu}:=$$

$$\begin{cases}
\Psi^{(2,p)}_{k_{2(p-3)}...k_2\frac{1}{2}k_0} & :\ k_0\in\{0,2,4,6\}\\
\widetilde{\mathsf{Q}}^{(2,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_0-1)\right)} & :\ k_0\in\{1,3,5\}\\
\widetilde{\mathsf{Q}}^{(2,p)}_{1} & :\begin{bmatrix} k_0=7\wedge\\ k_1,k_3,...,k_{2(p-3)-1}<r\end{bmatrix}\\
\widetilde{\mathsf{Q}}^{(2,p)}_{1}\widetilde{\mathsf{Q}}^{(3,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_2-1)\right)} & :\begin{bmatrix} k_0=7\wedge\\ k_1=r\wedge\\ k_3,...,k_{2(p-3)-1}<r\end{bmatrix}\\
... & ...\\
... & ...\\
... & ...\\
\widetilde{\mathsf{Q}}^{(2,p)}_{1}\widetilde{\mathsf{Q}}^{(3,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_2-1)\right)}...\widetilde{\mathsf{Q}}^{(s,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_{2(s-2)}-1)\right)} & :\begin{bmatrix} k_0=7\wedge\\ k_1=...=k_{2(s-2)-1}=r\wedge\\ k_{2(s-2)+1},...,k_{2(p-3)-1}<r\end{bmatrix}\\
... & ...\\
... & ...\\
... & ...\\
\widetilde{\mathsf{Q}}^{(2,p)}_{1}\widetilde{\mathsf{Q}}^{(3,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_2-1)\right)}...\widetilde{\mathsf{Q}}^{(p-1,p)}_{\mathrm{mod}\left(\frac{1}{2}(k_{2(p-3)}-1)\right)} & :\begin{bmatrix} k_0=7\wedge\\ k_1=...=k_{2(p-3)-1}=r\end{bmatrix}
\end{cases}$$

Example 5. If $p=4$ then $8\cdot 4t^2>\nu=8\cdot 4tk_3+8tk_2+8k_1+k_0$, where $k_0<8, k_1<t, k_2<4, k_3<t$ and

$$\Upsilon^{(4)}_{\nu}:=\begin{cases}
\Psi^{(2,4)}_{k_2\circ\frac{1}{2}k_0}, & \text{if } k_0\in\{0,2,4,6\}\\
\widetilde{\mathsf{Q}}^{(2,4)}_{\mathrm{mod}\left(\frac{1}{2}(k_0-1)\right)}, & \text{if } k_0\in\{1,3,5\}\\
\widetilde{\mathsf{Q}}^{(2,4)}_{1}, & \text{if } k_0=7\wedge k_1\neq t-1\\
\widetilde{\mathsf{Q}}^{(2,4)}_{1}\circ\widetilde{\mathsf{Q}}^{(3,4)}_{\mathrm{mod}\left(\frac{1}{2}(k_2-1)\right)}, & \text{if } k_0=7\wedge k_1=t-1
\end{cases}$$

5 Topological Considerations

As mentioned above (see Introduction), we wish to get more insight into the structure of $\phi^{-1}(\{\mathbf{0}\})$ *and* $\phi^{-1}(\{\mathbf{1}\})$ *for any boolean-valued* PTIME*-computable function* ϕ. To this end, according to 2.2 (see above) we first of all let $\phi:=\Phi_M$. It remains to define boolean values in $\mathbb{T}$. Now suppose $1,2\in\mathcal{A}_0$ and set $\mathbf{0}:=\boxed{1}$, $\mathbf{1}:=\boxed{2}$, where for any $\mathfrak{b}\in\mathcal{A}_0$ we denote by $\boxed{\mathfrak{b}}$ a singleton $\lambda z.\begin{cases}\mathfrak{b}, \text{if } z=0\\ 0, \text{if } z\neq 0\end{cases}\in\mathbb{T}$. Such $\boxed{\mathfrak{b}}$ are called *discrete tapes*. Generally, any Φ_M

such that $(\forall\varphi\in U\subseteq\mathbb{T})\,(\exists\mathfrak{b}\in\mathcal{B}\subseteq\mathcal{A}_0)\,\Phi_M(\varphi)=\boxed{\mathfrak{b}}$ is called *discrete-valued for U in $\mathcal{B}$*. So, loosely speaking, we wish to study topological structure of the sets $\Phi_M^{-1}\left(\left\{\boxed{\mathfrak{b}}\right\}\right)\subseteq\mathbb{T}$ for arbitrary PTIME-discrete-valued computations Φ_M. To this end, the recursion on the number of states of M (see Section 4 above) will provides us with the natural parameter $\|M\|$. Basic case $\|M\|=2$ is easy and straightforward (see 5.3 below), but more complex computations Φ_M where $\|M\|>2$ require special techniques referring to more complicated sets $Z\subseteq\mathbb{T}$ and $\Phi_M^{-1}(Z)\subseteq\mathbb{T}$.

5.1 Basic Notions

1. For any $r\in\mathbb{N}$ and $\varphi,\psi\in\mathbb{T}$, let
   $\varphi\equiv_r\psi:\Leftrightarrow(\forall z\in\mathbb{Z})\,(-r\leq z<r\to\varphi(z)=\psi(z))$.
2. For any $\mu,\nu\in\mathbb{N}$, $\mathfrak{b}_{-\mu},...,\mathfrak{b}_0,...,\mathfrak{b}_\nu\in\mathcal{A}$ and $x_{-\mu},...,x_0,...,x_\nu\in\mathbb{N}_0$, denote by $\boxed{\mathfrak{b}_{-\mu}^{x_{-\mu}}...\widehat{\mathfrak{b}_0}^{x_0}...\mathfrak{b}_\nu^{x_\nu}}$ a tape $\varphi\in\mathbb{T}$ such that:
   a) $\varphi(z)=0$, if $z\geq\sum\limits_{0\leq i\leq\nu}x_i$ or $z<x_0-\sum\limits_{0\leq i\leq\mu}x_i$
   b) $\varphi\left(u-x_j+\sum\limits_{0\leq i\leq j}x_i\right)=\mathfrak{b}_j$, if $0\leq j\leq\nu$ and $0\leq u<x_j$
   c) $\varphi\left(-u+x_0-\sum\limits_{0\leq i\leq j}x_{-i}\right)=\mathfrak{b}_{-j-1}$, if $0\leq j<\mu$ and $0<u\leq x_{-j}$

   That is to say, $\boxed{\mathfrak{b}_{-\mu}^{x_{-\mu}}...\widehat{\mathfrak{b}_0}^{x_0}...\mathfrak{b}_\nu^{x_\nu}}$ has the shape
   $$\overleftarrow{0}\,,\underbrace{\mathfrak{b}_{-\mu},...,\mathfrak{b}_{-\mu}}_{x_{-\mu}},...,\overset{\textcircled{S}}{\underbrace{\mathfrak{b}_0,...,\mathfrak{b}_0}_{x_0}},...,\underbrace{\mathfrak{b}_\nu,...,\mathfrak{b}_\nu}_{x_\nu},\overrightarrow{0}$$
   where the leftmost occurrence of $\mathfrak{b}_0$ is printed in the cell Ⓢ. For brevity, we'll often drop exponents $(\cdot)^{`1'}$, and decorations $\widehat{(\cdot)}$ which are seen from the context; thus $\boxed{\mathfrak{b}}=\boxed{\widehat{\mathfrak{b}}^1}$ for $\mu=\nu=0$, $x_0=1$.
3. For any $\mu,\nu\in\mathbb{N}$ and $\mathfrak{b}_{-\mu},...,\mathfrak{b}_0,...,\mathfrak{b}_\nu\in\mathcal{A}$, call a *$\mu,\nu$-skeleton* a decorated tuplet $\left\langle\mathfrak{b}_{-\mu},...,\widehat{\mathfrak{b}_0},...,\mathfrak{b}_\nu\right\rangle$ such that the following conditions hold:[4]
   a) $(\forall\iota:-\mu\leq\iota<\nu\wedge\iota\neq-1)\,\mathfrak{b}_\iota\neq\mathfrak{b}_{\iota+1}$
   b) $\mathfrak{b}_\nu=0\Rightarrow\nu=0$
   c) $\mathfrak{b}_{-\mu}=0\Rightarrow\mu=0$

   Denote by $\mathbb{S}_{\mu,\nu}$ the set of all μ,ν-skeletons and let $\mathbb{S}:=\bigcup\limits_{\mu,\nu\in\mathbb{N}}\mathbb{S}_{\mu,\nu}$. We say $\mathfrak{S}'=\left\langle\mathfrak{b}'_{-\theta},...,\widehat{\mathfrak{b}'_0},...,\mathfrak{b}'_\vartheta\right\rangle\in\mathbb{S}_{\theta,\vartheta}$ *extends* $\mathfrak{S}=\left\langle\mathfrak{b}_{-\mu},...,\widehat{\mathfrak{b}_0},...,\mathfrak{b}_\nu\right\rangle\in\mathbb{S}_{\mu,\nu}$ iff $\mu\leq\theta$, $\nu\leq\vartheta$ and $(\forall\iota:-\mu\leq\iota<\nu)\,\mathfrak{b}_\iota=\mathfrak{b}'_\iota$.

[4] One can also regard decorated tuplets $\left\langle\mathfrak{b}_{-\mu},...,\widehat{\mathfrak{b}_0},...,\mathfrak{b}_\nu\right\rangle$ as abbreviations of the ordinary tuplets $\langle\mathfrak{b}_{-\mu},...,\langle0,\mathfrak{b}_0\rangle,...,\mathfrak{b}_\nu\rangle$, which uniquely determine the dimension $\langle\mu,\nu\rangle$. We'll often drop decorations $\widehat{(\cdot)}$ if either $\mu=0$ or $\nu=0$ is the case.

4. If $\varphi = \boxed{\mathfrak{b}_{-\mu}^{x_{-\mu}} ... \widehat{\mathfrak{b}_0^{x_0}} ... \mathfrak{b}_{\nu}^{x_{\nu}}} \in \mathbb{T}$ and $\mathfrak{S} = \left\langle \mathfrak{b}_{-\mu}, ..., \widehat{\mathfrak{b}_0}, ..., \mathfrak{b}_{\nu} \right\rangle \in \mathbb{S}_{\mu,\nu}$ then $\mathfrak{S}$ is called the *skeleton* of φ, and $\langle \mu, \nu \rangle$ is called the *dimension* of both φ and $\mathfrak{S}$, while $\langle x_{-\mu}, ..., x_0, ..., x_{\nu} \rangle \in (\mathbb{N}_0)^{\mu+\nu+1}$ is called the *body* of φ. Note that both the dimension and body are uniquely determined for every $\varphi \in \mathbb{T}$. Denote by $\mathbb{T}_{\mu,\nu}$ the set of all $\varphi \in \mathbb{T}$ whose dimension is $\langle \mu, \nu \rangle$. Obviously, $\varphi \in \mathbb{T}_{(d)} \cap \mathbb{T}_{\mu,\nu}$ implies $\max(\mu, \nu) \leq d$.
5. Call a μ, ν-*environment* a pair $\langle \mathfrak{S}, \mathfrak{X} \rangle$ such that $\mathfrak{S} = \left\langle \mathfrak{b}_{-\mu}, ..., \widehat{\mathfrak{b}_0}, ..., \mathfrak{b}_{\nu} \right\rangle \in \mathbb{S}_{\mu,\nu}$ and $\mathfrak{X} \subseteq (\mathbb{N}_0)^{\mu+\nu+1}$. Denote by $\mathbb{E}_{\mu,\nu}$ the set of all μ, ν-environments and let $\mathbb{E} := \bigcup\limits_{\mu,\nu \in \mathbb{N}} \mathbb{E}_{\mu,\nu}$.
6. The correlated *canonical embedding* $\Bbbk \in [\mathbb{E}_{\mu,\nu} \to \wp(\mathbb{T}_{\mu,\nu})]$ is stipulated as follows. For any $\mathfrak{E} = \langle \mathfrak{S}, \mathfrak{X} \rangle = \left\langle \left\langle \mathfrak{b}_{-\mu}, ..., \widehat{\mathfrak{b}_0}, ..., \mathfrak{b}_{\nu} \right\rangle, \mathfrak{X} \right\rangle \in \mathbb{E}_{\mu,\nu}$ we set $\Bbbk(\mathfrak{E}) := \left\{ \boxed{\mathfrak{b}_{-\mu}^{x_{-\mu}} ... \widehat{\mathfrak{b}_0^{x_0}} ... \mathfrak{b}_{\nu}^{x_{\nu}}} \mid \langle x_{-\mu}, ..., x_0, ..., x_{\nu} \rangle \in \mathfrak{X} \right\} \subset \mathbb{T}_{\mu,\nu}$. Note that for every $\Omega \subseteq \mathbb{E}_{\mu,\nu}$, by definition $\Bbbk(\Omega) = \{\Bbbk(\mathfrak{E}) \mid \mathfrak{E} \in \Omega\} \subset \wp(\mathbb{T}_{\mu,\nu})$.
7. For any $r, \mu, \nu \in \mathbb{N}$ and $\mathfrak{E} \in \mathbb{E}$ we set $\Phi_M^{\bigstar}(r, \mu, \nu, \mathfrak{E}) := \{\mathfrak{O} \in \mathbb{E}_{\mu,\nu} \mid (\forall \varphi \in \Bbbk(\mathfrak{O}))(\exists \psi \in \Bbbk(\mathfrak{E}))\, \Phi_M(\varphi) \equiv_r \psi\} \subseteq \mathbb{E}_{\mu,\nu}$. So in particular, if Φ_M is discrete-valued for U in $\mathcal{B} \supseteq \mathcal{A}$, and $\mathfrak{b} \in \mathcal{A}_0$, then $\Phi_M^{-1}\left(\left\{\boxed{\mathfrak{b}}\right\}\right) = \bigcup\limits_{\mu,\nu \in \mathbb{N}} \bigcup \Bbbk\left(\Phi_M^{\bigstar}(0, \mu, \nu, \langle \langle \mathfrak{b} \rangle, \{1\} \rangle)\right)$ where obviously $\Phi_M^{\bigstar}(0, \mu, \nu, \langle \langle \mathfrak{b} \rangle, \{1\} \rangle) = \{\mathfrak{O} \in \mathbb{E}_{\mu,\nu} \mid (\forall \varphi \in \Bbbk(\mathfrak{O}))\, \Phi_M(\varphi)(0) = \mathfrak{b}\}$.

5.2 General Approach

Consider any natural number $p \geq 2$, an alphabet $\mathcal{A} \supseteq \{0, 1, 2\}$ and the correlated tape space $\mathbb{T}$. Let M be a PTIME-machine with the alphabet $\mathcal{A}$ such that $\|M\| = p$. Arguing along the lines of Proposition 1 (see Introduction), we choose a suitable "large" $\varepsilon \in \mathbb{N}_0^{>}$, let $U \subseteq \bigcup\limits_{\mu,\nu \leq \varepsilon} \mathbb{T}_{\mu,\nu}$ and suppose that Φ_M is discrete-valued for U in $\mathcal{B} = \{1, 2\}$. We set:

$$X := \left\{\varphi \in U \mid \Phi_M(\varphi) = \boxed{2}\right\} = \Phi_M^{-1}(\{\mathbf{1}\}) \cap U$$
$$Y := \left\{\varphi \in U \mid \Phi_M(\varphi) = \boxed{1}\right\} = \Phi_M^{-1}(\{\mathbf{0}\}) \cap U$$

We look for a suitable *discrete covering*

– $\Omega^{(p)} \subseteq \bigcup\limits_{\mu,\nu \leq \varepsilon} \mathbb{E}_{\mu,\nu}$ that admits a suitable *partition* $\Omega^{(p)} = \Omega_{\mathrm{S}}^{(p)} \cup \Omega_{\mathrm{L}}^{(p)}$ such that:
  1. $\bigcup \Bbbk\left(\Omega^{(p)}\right) = U$
  2. $\Omega_{\mathrm{S}}^{(p)} \cap \Omega_{\mathrm{L}}^{(p)} = \emptyset$
  3. $\Omega^{(p)} \subseteq \bigcup\limits_{\mu,\nu \leq \varepsilon} \bigcup\limits_{b \in \mathcal{B}} \Phi_M^{\bigstar}(0, \mu, \nu, \langle \langle b \rangle, \{1\} \rangle)$

We set $\Omega^{(p)} = \{\mathfrak{O}_i \neq \emptyset \mid i \in I\}$, $\Omega_{\mathrm{S}}^{(p)} = \{\mathfrak{O}_s \mid s \in S\}$, $\Omega_{\mathrm{L}}^{(p)} = \{\mathfrak{O}_l \mid l \in L\}$ where $S \cup L = I$ and (by 2 above) $S \cap L = \emptyset$. Moreover, for any $\mathfrak{O}_i \in \Omega^{(p)}$ we let $\mathcal{O}_i := \Bbbk(\mathfrak{O}_i)$. Hence

- $\Bbbk\left(\Omega^{(p)}\right) = \{\mathcal{O}_i \mid i \in I\} = \{\mathcal{O}_s \mid s \in S\} \cup \{\mathcal{O}_l \mid l \in L\}$ which by 1-3 above yields
- $U = \bigcup\limits_{i \in I} \mathcal{O}_i = \bigcup\limits_{s \in S} \mathcal{O}_s \cup \bigcup\limits_{l \in L} \mathcal{O}_l$ along with the first 3 conditions of Proposition 1:
  1. $S \cup L = I$
  2. $S \cap L = \emptyset$
  3. $(\forall i \in I)\,(\mathcal{O}_i \subseteq X \vee \mathcal{O}_i \subseteq Y)$

We wish to further specify the desired covering $\Omega^{(p)}$ in order to fulfill the remaining two conditions 4, 5 of Proposition 1, provided that $\Phi_M = \Phi_M^{(p)}$ is the hypothetical boolean evaluation on CNF. To this end, we argue by induction on p. The basis of induction does not require any estimate on the complexity of $\Phi_M = \Phi_M^{(2)}$ - this case is exposed in the next subsection. The induction step relies upon the recursive expansions of a PTIME-computation Φ_M elaborated in Chapters 3, 4. (Note that being discrete-valued is not hereditary, in that if we pass from M to M_i and the adjacent discrete-valued computation $\Phi_M^{(q+1)}$ to $\Phi_i^{(q)}$ (see 3.1, 3.3 above) then $\Phi_i^{(q)}$ must not be discrete-valued.) Loosely speaking, the desired environments of $\Omega^{(q+1)}$ for $q > 1$ arise from the environments of $\Omega^{(q)}$ by a suitable polynomial iteration of the operation $\Phi_M^{\bigstar}$. The resulting pairs $\langle \mathfrak{S}, \mathfrak{X} \rangle \in \Omega^{(q+1)}$ include sets $\mathfrak{X} \subseteq (\mathbb{N}_0)^{\mu+\nu+1}$ which can be loosely referred to as the solutions of the appropriate polynomial systems of polynomial equations determined by $\Omega^{(q)}$, while the desired measurable partition $\Omega^{(q+1)} = \Omega_{\mathrm{S}}^{(q+1)} \cup \Omega_{\mathrm{L}}^{(q+1)}$ is supposed to be determined by a natural Cartesian measure of $\mathfrak{X}$.

5.3 Basic Case

Proposition 6. *Let $\|M\| = 2$, $\mathfrak{b} \in \mathcal{A}_0$, $\mu, \nu \in \mathbb{N}$, $\varphi \in \mathbb{T}_{\mu,\nu}$, and suppose that $\Phi_M(\varphi)(0) = \mathfrak{b}$. There exist $\xi \in \mathbb{N}$ and $\mathfrak{S} \in \mathbb{S}_{0,\xi} \cup \mathbb{S}_{\xi,0}$ such that one of the following conditions 1-6 holds, where for brevity we put $(\mathbb{N}_0^{>})^0 := \emptyset$ in 2 and 5.*

1. *$\xi \leq \nu$, $\mathfrak{S} = \langle \mathfrak{b}_0, ..., \mathfrak{b}_\xi \rangle \in \mathbb{S}_{0,\xi}$ and the following two conditions hold.*
   a) *There exist $\mathfrak{S}' = \left\langle \mathfrak{b}'_{-\mu}, ..., \widehat{\mathfrak{b}'_0}, ..., \mathfrak{b}'_\nu \right\rangle \in \mathbb{S}_{\mu,\nu}$ that extends $\mathfrak{S}$ and $\mathfrak{x} = \langle x_{-\mu}, ..., x_0, ..., x_\nu \rangle \in (\mathbb{N}_0^{>})^{\mu+\nu+1}$ such that $\varphi = \boxed{\mathfrak{b}'^{x_{-\mu}}_{-\mu} ... \widehat{\mathfrak{b}'^{x_0}_0} ... \mathfrak{b}'^{x_\nu}_\nu}$.*
   b) *For any $\mathfrak{S}'' = \left\langle \mathfrak{b}''_{-\theta}, ..., \widehat{\mathfrak{b}''_0}, ..., \mathfrak{b}''_\vartheta \right\rangle \in \mathbb{S}_{\theta,\vartheta}$ extending $\mathfrak{S}$ and any $\mathfrak{x} = \langle x_{-\theta}, ..., x_0, ..., x_\vartheta \rangle \in (\mathbb{N}_0^{>})^{\theta+\vartheta+1}$, it holds $\Phi_M\left(\boxed{\mathfrak{b}''^{x_{-\theta}}_{-\theta} ... \widehat{\mathfrak{b}''^{x_0}_0} ... \mathfrak{b}''^{x_\vartheta}_\vartheta}\right)(0) = \mathfrak{b}$.*
2. *$\xi \leq \nu$, $\mathfrak{S} = \langle \mathfrak{b}_0, ..., \mathfrak{b}_\xi \rangle \in \mathbb{S}_{0,\xi}$ and the following two conditions hold.*

a) There exist $\mathfrak{S}'=\left\langle \mathfrak{b}'_{-\mu},...,\widehat{\mathfrak{b}'_0},...,\mathfrak{b}'_\nu\right\rangle \in \mathbb{S}_{\mu,\nu}$ *that extends* $\mathfrak{S}$ *and*
$\mathfrak{x}=\langle x_{-\mu},...,x_0,...,x_\nu\rangle \in (\mathbb{N}_0^>)^{\mu+\xi}\times \mathbb{N}_1^>\times (\mathbb{N}_0^>)^{\nu-\xi}$ *such that*
$\varphi=\boxed{\mathfrak{b}'^{x_{-\mu}}_{-\mu}...\widehat{\mathfrak{b}'^{x_0}_0}...\mathfrak{b}'^{x_\nu}_\nu}$.

b) For any $\mathfrak{S}''=\left\langle \mathfrak{b}''_{-\theta},...,\widehat{\mathfrak{b}''_0},...,\mathfrak{b}''_\vartheta\right\rangle \in \mathbb{S}_{\theta,\vartheta}$ *extending* $\mathfrak{S}$ *and any*
$\mathfrak{x}=\langle x_{-\theta},...,x_0,...,x_\vartheta\rangle \in (\mathbb{N}_0^>)^{\theta+\xi}\times \mathbb{N}_1^>\times (\mathbb{N}_0^>)^{\vartheta-\xi}$, *it holds*
$\Phi_M\left(\boxed{\mathfrak{b}''^{x_{-\theta}}_{-\theta}...\widehat{\mathfrak{b}''^{x_0}_0}...\mathfrak{b}''^{x_\vartheta}_\vartheta}\right)(0)=\mathfrak{b}$.

3. $0<\xi\le\nu$, $\mathfrak{S}=\langle \mathfrak{b}_0,...,\mathfrak{b}_\xi\rangle \in \mathbb{S}_{0,\xi}$ *and the following two conditions hold.*

a) There exist $\mathfrak{S}'=\left\langle \mathfrak{b}'_{-\mu},...,\widehat{\mathfrak{b}'_0},...,\mathfrak{b}'_\nu\right\rangle \in \mathbb{S}_{\mu,\nu}$ *that extends* $\mathfrak{S}$ *and*
$\mathfrak{x}=\langle x_{-\mu},...,x_0,...,x_\nu\rangle \in (\mathbb{N}_0^>)^{\mu+\xi-1}\times \{1\}\times (\mathbb{N}_0^>)^{\nu-\xi+1}$ *such that*
$\varphi=\boxed{\mathfrak{b}'^{x_{-\mu}}_{-\mu}...\widehat{\mathfrak{b}'^{x_0}_0}...\mathfrak{b}'^{x_\nu}_\nu}$.

b) For any $\mathfrak{S}''=\left\langle \mathfrak{b}''_{-\theta},...,\widehat{\mathfrak{b}''_0},...,\mathfrak{b}''_\vartheta\right\rangle \in \mathbb{S}_{\theta,\vartheta}$ *extending* $\mathfrak{S}$ *and any*
$\mathfrak{x}=\langle x_{-\theta},...,x_0,...,x_\vartheta\rangle \in (\mathbb{N}_0^>)^{\theta+\xi-1}\times \{1\}\times (\mathbb{N}_0^>)^{\vartheta-\xi+1}$, *it holds*
$\Phi_M\left(\boxed{\mathfrak{b}''^{x_{-\theta}}_{-\theta}...\widehat{\mathfrak{b}''^{x_0}_0}...\mathfrak{b}''^{x_\vartheta}_\vartheta}\right)(0)=\mathfrak{b}$.

4. $\xi\le\mu$, $\mathfrak{S}=\langle \mathfrak{b}_{-\xi},...,\mathfrak{b}_0\rangle \in \mathbb{S}_{\xi,0}$ *and the above conditions 1 (a), (b) hold.*

5. $\xi\le\mu$, $\mathfrak{S}=\langle \mathfrak{b}_{-\xi},...,\mathfrak{b}_0\rangle \in \mathbb{S}_{\xi,0}$ *and the following two conditions hold.*

a) There exist $\mathfrak{S}'=\left\langle \mathfrak{b}'_{-\mu},...,\widehat{\mathfrak{b}'_0},...,\mathfrak{b}'_\nu\right\rangle \in \mathbb{S}_{\mu,\nu}$ *that extends* $\mathfrak{S}$ *and*
$\mathfrak{x}=\langle x_{-\mu},...,x_0,...,x_\nu\rangle \in (\mathbb{N}_0^>)^{\mu-\xi}\times \mathbb{N}_1^>\times (\mathbb{N}_0^>)^{\nu+\xi}$ *such that*
$\varphi=\boxed{\mathfrak{b}'^{x_{-\mu}}_{-\mu}...\widehat{\mathfrak{b}'^{x_0}_0}...\mathfrak{b}'^{x_\nu}_\nu}$.

b) For any $\mathfrak{S}''=\left\langle \mathfrak{b}''_{-\theta},...,\widehat{\mathfrak{b}''_0},...,\mathfrak{b}''_\vartheta\right\rangle \in \mathbb{S}_{\theta,\vartheta}$ *extending* $\mathfrak{S}$ *and any*
$\mathfrak{x}=\langle x_{-\theta},...,x_0,...,x_\vartheta\rangle \in (\mathbb{N}_0^>)^{\theta-\xi}\times \mathbb{N}_1^>\times (\mathbb{N}_0^>)^{\vartheta+\xi}$, *it holds*
$\Phi_M\left(\boxed{\mathfrak{b}''^{x_{-\theta}}_{-\theta}...\widehat{\mathfrak{b}''^{x_0}_0}...\mathfrak{b}''^{x_\vartheta}_\vartheta}\right)(0)=\mathfrak{b}$.

6. $0<\xi\le\mu$, $\mathfrak{S}=\langle \mathfrak{b}_{-\xi},...,\mathfrak{b}_0\rangle \in \mathbb{S}_{\xi,0}$ *and the following two conditions hold.*

a) There exist $\mathfrak{S}'=\left\langle \mathfrak{b}'_{-\mu},...,\widehat{\mathfrak{b}'_0},...,\mathfrak{b}'_\nu\right\rangle \in \mathbb{S}_{\mu,\nu}$ *that extends* $\mathfrak{S}$ *and*
$\mathfrak{x}=\langle x_{-\mu},...,x_0,...,x_\nu\rangle \in (\mathbb{N}_0^>)^{\mu-\xi+1}\times \{1\}\times (\mathbb{N}_0^>)^{\nu+\xi-1}$ *such that*
$\varphi=\boxed{\mathfrak{b}'^{x_{-\mu}}_{-\mu}...\widehat{\mathfrak{b}'^{x_0}_0}...\mathfrak{b}'^{x_\nu}_\nu}$.

b) For any $\mathfrak{S}''=\left\langle \mathfrak{b}''_{-\theta},...,\widehat{\mathfrak{b}''_0},...,\mathfrak{b}''_\vartheta\right\rangle \in \mathbb{S}_{\theta,\vartheta}$ *extending* $\mathfrak{S}$ *and any*
$\mathfrak{x}=\langle x_{-\theta},...,x_0,...,x_\vartheta\rangle \in (\mathbb{N}_0^>)^{\theta-\xi+1}\times \{1\}\times (\mathbb{N}_0^>)^{\vartheta+\xi-1}$, *it holds*
$\Phi_M\left(\boxed{\mathfrak{b}''^{x_{-\theta}}_{-\theta}...\widehat{\mathfrak{b}''^{x_0}_0}...\mathfrak{b}''^{x_\vartheta}_\vartheta}\right)(0)=\mathfrak{b}$.

Proof. Since M has only two states 0 and 1, the computation leading to the value $\Phi_M(\varphi)(0) = \varphi_\ell(0) = \mathfrak{b}$ where $\ell := \min\{n \mid \sigma_n = 1\}$ (see 2.2.2.3 above) ends at the first occurrence of a state $\sigma_\ell \neq 0$, which is only possible if $\sigma_\ell = 1$. Hence we can isolate the last computation step ℓ executing the order $\langle \varphi_\ell(0), 0\rangle \hookrightarrow \langle \Delta(\varphi_\ell(0), 0), \Pi(\varphi_\ell(0), 0), 1\rangle$ and ignore the initial state 0 occurring in all preceding scanning positions and the adjacent orders. Let

$$d := \begin{cases} \min\{n \leq \ell \mid \Delta(\varphi_n(0), 0) \neq 0\}, & \text{if } (\exists n \leq \ell)\, \Delta(\varphi_n(0), 0) \neq 0 \\ 0, & \text{otherwise} \end{cases},$$

$e := \Delta(\varphi_\ell(0), 0)$ and $s := \sum_{i<\ell} \Delta(\varphi_i(0), 0)$; note that $d = \Delta(\varphi_d(0), 0)$. In the remainder we argue by cases.

- Suppose $d = 0$. We let $\xi := 0$, $\mathfrak{S} := \langle \varphi(0)\rangle \in \mathbb{S}_{0,0}$ and observe that such $\langle \xi, \mathfrak{S}\rangle$ satisfies the condition 1.
- Suppose $d = 1$. Note that $\Delta(\varphi_n(0), 0) \geq 0$ holds for all $n < \ell$. For otherwise, any $\Delta(\varphi_n(0), 0) = -1$, $n < \ell$, would create a cycle preventing the computation to reach step $n+1$. Hence $d = \#\{n < \ell \mid \Delta(\varphi_n(0), 0) \neq 0\}$. We stipulate that one of the required pair $\langle \xi, \mathfrak{S}\rangle$ shall satisfy the corresponding condition 1, 2 or 3.
  - Suppose $e < 1$. Let $\boxed{\overrightarrow{\varphi}_s} := \overleftarrow{0}, \varphi(0), ..., \varphi(s), \overrightarrow{0}$. This uniquely determines $\xi \leq \nu$ and $\mathfrak{S} = \langle \mathfrak{b}_0, ..., \mathfrak{b}_\xi\rangle \in \mathbb{S}_{0,\xi}$ for which there is a $\mathfrak{y} = \langle y_0, ..., y_\xi\rangle \in (\mathbb{N}_0^>)^{\xi+1}$ such that $\boxed{\overrightarrow{\varphi}_s} = \boxed{\mathfrak{b}_0^{y_0} ... \mathfrak{b}_\xi^{y_\xi}}$. It is readily verified that such $\langle \xi, \mathfrak{S}\rangle$ satisfies the condition 1. Namely, the subcondition (a) holds for $\mathfrak{S}'$ and $\mathfrak{x}$ being the skeleton and the body of φ. Note that $\mathfrak{x}$ coincides with $\mathfrak{y}$ on every coordinate $\iota < \xi$. The only nontrivial observation is related to subcondition (b) - it reads that any computation step $n < \ell$ with $\Delta(\varphi_n(0), 0) = 1$ can be iterated x times for any $x \in \mathbb{N}_0^>$, without changing the final value $\Phi_M(\varphi)(0)$.
  - Suppose $e = 1$ and $\varphi(s) = \mathfrak{b}$. Since $\Phi_M(\varphi)(0) = \mathfrak{b}$, we have $\varphi(s) = \varphi(s+1) = \mathfrak{b}$. Let $\boxed{\overrightarrow{\varphi}_{s+1}} := \overleftarrow{0}, \varphi(0), ..., \varphi(s), \varphi(s+1), \overrightarrow{0}$. Arguing as above, we get $\xi \leq \nu$ and $\mathfrak{S} = \langle \mathfrak{b}_0, ..., \mathfrak{b}_\xi\rangle = \langle \mathfrak{b}_0, ..., \mathfrak{b}\rangle \in \mathbb{S}_{0,\xi}$ and $\mathfrak{y} = \langle y_0, ..., y_\xi\rangle \in (\mathbb{N}_0^>)^\xi$ with $y_\xi > 1$, such that $\boxed{\overrightarrow{\varphi}_{s+1}} = \boxed{\mathfrak{b}_0^{y_0} ... \mathfrak{b}_\xi^{y_\xi}}$. Having this, it is readily verified that such $\langle \xi, \mathfrak{S}\rangle$ satisfies the condition 2 (see analogous passage above).
  - Suppose $e = 1$ and $\varphi(s) \neq \mathfrak{b}$. Since $\Phi_M(\varphi)(0) = \mathfrak{b}$, we have $\varphi(s+1) = \mathfrak{b}$. Define $\boxed{\overrightarrow{\varphi}_{s+1}}$ as above. By the above token, we get $0 < \xi \leq \nu$ and $\mathfrak{S} = \langle \mathfrak{b}_0, ..., \mathfrak{b}_{\xi-1}, \mathfrak{b}_{\varsigma+1}\rangle = \langle \mathfrak{b}_0, ..., \mathfrak{b}_{\xi-1}, \mathfrak{b}\rangle \in \mathbb{S}_{0,\xi}$ and $\mathfrak{y} = \langle y_0, ..., y_{\xi-1}, y_\xi\rangle \in (\mathbb{N}_0^>)^{\xi+1}$ such that $\boxed{\overrightarrow{\varphi}_{s+1}} = \boxed{\mathfrak{b}_0^{y_0} ... \mathfrak{b}_{\xi-1}^{y_{\xi-1}} \mathfrak{b}_\xi^{y_\xi}} = \boxed{\mathfrak{b}_0^{y_0} ... \mathfrak{b}_{\xi-1}^{y_{\xi-1}} \mathfrak{b}}$. Moreover, we observe that $y_{\xi-1} = 1$, since the leftmost occurrence of $\mathfrak{b}_{\xi-1}$ in the string $\mathfrak{b}_{\xi-1}^{y_{\xi-1}}$ is immediately followed by the final occurrence $\mathfrak{b}$, while $\mathfrak{b}_{\xi-1} \neq \mathfrak{b}$. Having this, it is readily verified that such $\langle \xi, \mathfrak{S}\rangle$ satisfies the condition 3.

– Suppose $d=-1$. This case is symmetric to previous case $d=1$. We note that $\Delta(\varphi_n(0),0)\leq 0$ holds for all $n<\ell$, and stipulate by the same token that the required pair $\langle\xi,\mathfrak{S}\rangle$ shall satisfy the corresponding condition 4, 5 or 6 depending on whether $e>-1$, $e=-1\wedge\varphi(s)\neq\mathfrak{b}$ or else $e=-1\wedge\varphi(s)=\mathfrak{b}$.

Corollary 7. *Let* $\|M\|=2$, $\mathcal{B}\subseteq\mathcal{A}_0$, $\mu,\nu\in\mathbb{N}$. *There is a recursive functional* $\mathtt{J}_{\mu,\nu}\in[\mathbb{T}_{\mu,\nu}\times\mathcal{B}\curvearrowright\mathbb{E}_{\mu,\nu}]$, $\mathtt{Dom}(\mathtt{J}_{\mu,\nu})=\{\langle\varphi,\mathfrak{b}\rangle\in\mathbb{T}_{\mu,\nu}\times\mathcal{B}\mid\Phi_M(\varphi)(0)=\mathfrak{b}\}$, *such that for all* $\langle\varphi,\mathfrak{b}\rangle\in\mathtt{Dom}(\mathtt{J}_{\mu,\nu})$, $\mathtt{J}_{\mu,\nu}(\varphi,\mathfrak{b})=\langle\mathfrak{S},\mathfrak{X}\rangle\in\Phi_M^{\bigstar}(0,\mu,\nu,\langle\langle\mathfrak{b}\rangle,\{1\}\rangle)$ *and exactly one of the following conditions 1-5 holds.*

1. $\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu+\nu+1}$
2. $(\exists\xi\leq\nu)\,\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu+\xi}\times\mathbb{N}_1^{>}\times(\mathbb{N}_0^{>})^{\nu-\xi}$
3. $(\exists\xi:0<\xi\leq\nu)\,\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu+\xi-1}\times\{1\}\times(\mathbb{N}_0^{>})^{\nu-\xi+1}$
4. $(\exists\xi\leq\mu)\,\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu-\xi}\times\mathbb{N}_1^{>}\times(\mathbb{N}_0^{>})^{\nu+\xi}$
5. $(\exists\xi:0<\xi\leq\mu)\,\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu-\xi+1}\times\{1\}\times(\mathbb{N}_0^{>})^{\nu+\xi-1}$

Remark 8. Weakening in the corollary the assumption $\mathcal{B}\subseteq\mathcal{A}_0$ to $\mathcal{B}\subseteq\mathcal{A}$ will result in weakening $(\exists\xi:0<\xi\leq\nu)$ and $(\exists\xi:0<\xi\leq\mu)$ to $(\exists\xi:0<\xi\leq\nu+1)$ and $(\exists\xi:0<\xi\leq\mu+1)$ in the conditions 3 and 5, respectively. To put it more exactly, the resulting modification yields the two new solutions $\mathtt{J}_{\mu,\nu}(\varphi,0)=\langle\mathfrak{S},\mathfrak{X}\rangle$ for $\mathfrak{X}=(\mathbb{N}_0^{>})^{\mu+\nu}\times\{1\}$ and $\mathfrak{X}=\{1\}\times(\mathbb{N}_0^{>})^{\mu+\nu}$. The proof is readily obtained by adding the corresponding two new cases in Proposition 5. Now by 5.1.4 (see above), this refinement of the corollary obviously implies the decidability of $\Phi_M(\varphi)\downarrow$, provided that $\|M\|=2$ (cf. Footnote 2 in 4.1.1 above).

Definition 9. *Let* $\|M\|=2$, $\mathcal{B}=\{1,2\}\subseteq\mathcal{A}_0$, $\varepsilon\in\mathbb{N}_0^{>}$ *and* $U\subseteq\bigcup\limits_{\mu,\nu\leq\varepsilon}\mathbb{T}_{\mu,\nu}$. *Furthermore, suppose that* Φ_M *is total, i.e.* $\mathtt{Dom}(\Phi_M)=\mathbb{T}$, *and discrete-valued for* U *in* $\mathcal{B}$. *Let* $\mathtt{J}:=\bigcup\limits_{\mu,\nu\leq\varepsilon}\mathtt{J}_{\mu,\nu}$, *each* $\mathtt{J}_{\mu,\nu}\in[\mathbb{T}_{\mu,\nu}\times\mathcal{B}\curvearrowright\mathbb{E}_{\mu,\nu}]$ *being as in the corollary. Define the adjacent canonical* ε*-covering*

$$\Omega^{(2)}:=\mathtt{Rng}(\mathtt{J}\restriction_{U\times\mathcal{B}})=\{\mathtt{J}_{\mu,\nu}(\varphi,\mathfrak{b})\mid\mu,\nu\leq\varepsilon\wedge\varphi\in U\wedge\mathfrak{b}\in\mathcal{B}\}\subseteq\bigcup_{\mu,\nu\leq\varepsilon}\mathbb{E}_{\mu,\nu}$$

and for any fixed $c\in\mathbb{N}_0^{>}$ *define the correlated "trivial" c-partition* $\Omega^{(2)}=\Omega^{(2)}_{\mathrm{S}}\cup\Omega^{(2)}_{\mathrm{L}}$ *by setting, for all* $\mu,\nu\leq\varepsilon$*:*

$$\Omega^{(2)}_{\mathrm{S}}\cap\mathbb{E}_{\mu,\nu}:=\emptyset\ \ \mathtt{if}\ \ \mu+\nu\geq c,\ \ \mathtt{else}\ \ \Omega^{(2)}_{\mathrm{L}}\cap\mathbb{E}_{\mu,\nu}:=\emptyset.$$

Definition 10. *We say* $\mathcal{A}$ *accepts* DNF *iff* $\mathcal{A}_0$ *includes (codes/gödelnumbers of) conjunction* $\wedge$ *and disjunction* $\vee$ *and two special signs* $\restriction$ *and* $\downarrow$ *which enable us to interpret positive (negative) literals as strings of* $\restriction$ *(*$\downarrow$*). To put it more precisely, we interpret a propositional variable* v_i *and its negation* $\neg v_i$, $i\in\mathbb{N}$, *by*

$\underbrace{\rceil,...,\rceil}_{i+1}$ *and* $\underbrace{\lfloor,...,\lfloor}_{i+1}$, *respectively.*[5] *Now any formula* $\digamma \in \mathrm{DNF}$ *is expressed by the (uniquely determined) isomorphic* $\mathcal{A}$*-tape* $\varphi_{\digamma} \in \mathbb{T}$ *whose skeleton has the shape of* $\digamma$*, whose leftmost sign is printed in the cell* 0*. Moreover, the body of* $\varphi_{\digamma}$ *arises from* $\digamma$ *by dropping parentheses, while expressing conjunction, disjunction and literals according to the above interpretation. For example, the translation of* $\digamma = (v_0 \wedge v_1) \vee \neg v_2$ *results in* $\varphi_{\digamma} = \boxed{\rceil \wedge \rceil^2 \vee \lfloor^3} \in \mathbb{T}_{0,4}$*. For any* $\varepsilon \in \mathbb{N}_0^{>}$*, define the adjacent canonical* DNF*-segment* $U := \{\varphi_{\digamma} \mid \digamma \in \mathrm{DNF}\} \cap \bigcup\limits_{\nu \leq \varepsilon} \mathbb{T}_{0,\nu}$.

Conclusion 11 *Let* $\|M\| = 2$, $\mathcal{B} = \{1,2\} \subseteq \mathcal{A}_0$, $\varepsilon \geq c \in \mathbb{N}_1^{>}$*. Suppose that* $\mathcal{A}$ *accepts* DNF*, let* $U \subseteq \bigcup\limits_{\nu \leq \varepsilon} \mathbb{T}_{0,\nu}$ *be the adjacent canonical* DNF*-subsegment. Suppose that* Φ_M *is total and discrete-valued for* U *in* $\mathcal{B}$*. Consider the adjacent canonical* ε*-covering* $\Omega^{(2)} \subseteq \bigcup\limits_{\nu \leq \varepsilon} \mathbb{E}_{0,\nu}$ *and the correlated* c*-partition* $\Omega^{(2)} = \Omega_{\mathrm{S}}^{(2)} \cup \Omega_{\mathrm{L}}^{(2)}$*. We have* $\Omega^{(2)} = \{\mathfrak{O}_i \neq \emptyset \mid i \in I\}$, $\Omega_{\mathrm{S}}^{(2)} = \{\mathfrak{O}_s \mid s \in S\}$, $\Omega_{\mathrm{L}}^{(2)} = \{\mathfrak{O}_l \mid l \in L\}$ *where* $S \cup L = I$ *and* $S \cap L = \emptyset$*. For a moment suppose that*

- $\Phi_M(\varphi) = \boxed{2} \Leftrightarrow \digamma \in \mathrm{TAU}_{\mathrm{DNF}}$ *holds for all* $\varphi \in U$.

Arguing as in 5.2 (see above), let $\mathcal{O}_i := \Bbbk(\mathfrak{O}_i)$*, and set*

- $X := \left\{\varphi \in U \mid \Phi_M(\varphi) = \boxed{2}\right\}$
  $Y := \left\{\varphi \in U \mid \Phi_M(\varphi) = \boxed{1}\right\}$
  We claim that
- $U = \bigcup\limits_{i \in I} \mathcal{O}_i = \bigcup\limits_{s \in S} \mathcal{O}_s \cup \bigcup\limits_{l \in L} \mathcal{O}_l$ *holds along with:*
  1. $S \cup L = I$
  2. $S \cap L = \emptyset$
  3. $(\forall i \in I)\,(\mathcal{O}_i \subseteq X \vee \mathcal{O}_i \subseteq Y)$
  4. $(\forall l \in L)\,(\mathcal{O}_l \cap Y \neq \emptyset)$
  5. $\bigcup\limits_{s \in S} \mathcal{O}_s \cup Y \neq U$

To establish this claim, it will suffice to verify the last two conditions 4, 5 (cf. 5.2 above). First note that 5 is obvious. For on the one hand $\Omega_{\mathrm{S}}^{(2)} \cap \mathbb{E}_{0,2} = \emptyset$ *(see Definition 8 above). On the other hand,* $Y \cap \mathbb{E}_{0,2} \neq U \cap \mathbb{E}_{0,2}$*, since* $\emptyset \neq X \ni \varphi_{\digamma} = \boxed{\rceil \vee \lfloor} \in \mathbb{T}_{0,2}$ *for* $\digamma = v_0 \vee \neg v_0 \in \mathrm{TAU}_{\mathrm{DNF}}$*. Consider the remaining condition 4. Take any* $l \in L$ *and* $\mathcal{O}_l = \Bbbk(\mathfrak{O}_l)$ *where* $\mathfrak{O}_l = \langle \mathfrak{S}, \mathfrak{X} \rangle \in \Phi_M^{\bigstar}(0,0,\nu,\langle\langle \mathfrak{b} \rangle, \{1\}\rangle)$ *for* $c \leq \nu \leq \varepsilon$ *and* $\mathfrak{b} \in \{1,2\}$*. Moreover, by Corollary 6, one of the following three conditions holds.*

[5] As usual in algebra, we drop superfluous parentheses, while assuming that conjunction binds stronger than disjunction. This particular interpretaion is not important for the result - one can just as well use a distinguished letter for the atomic negation $\neg$, instead of $\lfloor$.

1. $\mathfrak{X} = (\mathbb{N}_0^>)^{\nu+1}$
2. $(\exists \xi \leq \nu)\, \mathfrak{X} = (\mathbb{N}_0^>)^{\xi} \times \mathbb{N}_1^> \times (\mathbb{N}_0^>)^{\nu-\xi}$
3. $(\exists \xi : 0 < \xi \leq \nu)\, \mathfrak{X} = (\mathbb{N}_0^>)^{\xi-1} \times \{1\} \times (\mathbb{N}_0^>)^{\nu-\xi+1}$

Suppose $\varphi_F \in \mathcal{O}_l$ *and consider* F*. By the above conditions, at most one propositional variable occurrence in* F *(the one corresponding to the coordinate* ξ *in the latter condition) is* v_0*, other variable occurrences being* v_ι *for any* $\iota \in \mathbb{N}_1^>$*. That is, arbitrary variable-substitutions beyond* v_0 *and* v_1 *for all but at most one variable occurrences in* F *preserve the assumption* $\varphi_F \in \mathcal{O}_l$*, and hence the value* $\mathfrak{b}$ *of* $\Phi_M(\varphi_F)(0)$*. Assigning pairwise different indices to all variable occurrences in* F *it is readily seen, then, that at least one such* $\varphi_F \in \mathcal{O}_l$ *is not a tautology. By the main assumption* $\Phi_M(\varphi) = \boxed{2} \Leftrightarrow F \in \mathrm{TAU}_{\mathrm{DNF}}$*, the latter yields* $\mathfrak{b} = 1$*. Hence* $\mathcal{O}_l \subseteq Y$*, and in particular* $\mathcal{O}_l \cap Y \neq \emptyset$*, since* $\mathcal{O}_l \neq \emptyset$ *- which completes the proof of Condition 4. Now Proposition 1 (see Introduction) shows that the main assumption fails. Thus no Turing machine with only two initial states can decide* $\mathrm{TAU}_{\mathrm{DNF}}$*, Q.E.D.*

References

[B] S. Buss, **Bounded arithmetic**, Bibliopolis, Napoli (1986)

[H] A. Haken, *The intractability of resolution,* Theor. Comp. Sci. **39** *(1985), 297-308*

[P] E. Post, *Finite combinatory processes — formulation I*, Journ. Symb. Logic **1** (1936), 103-105

[T] G. Takeuti, *Computational complexity and proof theory*, Sugaku **39** (2), 1987, 110-123 (Transl. in **AMS** Sugaku expositions **0** (1), 1988, 1-14)

[Tu] A. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. **42** (1937), 230-265

Interpolation for Natural Deduction with Generalized Eliminations

Ralph Matthes

Institut für Informatik der Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany
matthes@informatik.uni-muenchen.de

Abstract. A modification of simply-typed λ-calculus by generalized elimination rules in the style of von Plato is presented. Its characteristic feature are permutative conversions also for function types and product types. After the addition of certain extensional reduction rules, an interpolation theorem (à la Lyndon) is proven which is also aware of the terms (a. k. a. the proofs via the Curry-Howard-isomorphism) like in Čubrić's treatment of the usual λ-calculus. Connections between interpolation and canonical liftings of positive and negative type dependencies are given which are important for the intensional treatment of inductive datatypes.

1 Introduction

In logic, interpolation is often used as a tool to reduce arguments on monotone operators to those on positive operators. Mostly, one is satisfied with a logically equivalent positive formula expressing the original monotone dependency. When it comes to functional programming with datatypes, logical equivalence of datatypes is not very helpful. (Think of all the natural numbers being proofs of Nat.) Therefore, a more intensional approach is called for. [Čub94] presented interpolation with additional information: Not only an intermediate formula/type is gained but also proofs/terms for the both implications such that their composition reduces to the original given proof/term in some strongly normalizing and confluent calculus. This idea is taken up in a modification ΛJ of λ-calculus presented in section 2 which has a different application rule than λ-calculus and therefore needs permutative conversions already for function types. Concerning provability/typability, there is no difference with λ-calculus. However, the normal forms have a more appealing structure, made very precise in the last subsection. Section 3 lists several meta-theoretic properties of ΛJ. The interpolation theorem for ΛJ is stated and proven in great detail in section 4. Many subtleties show that in ΛJ one gets an even deeper insight into interpolation than in λ-calculus. Section 5 starts with some material on η-rules and long normal forms, then introduces lifting of arbitrary type dependencies and calculates the interpolants for all of them. In the last subsection, some thoughts are given on the use of interpolation for the purpose of positivization of monotone type dependencies.

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 153–169, 2001.

Acknowledgement: Many thanks to my former colleague Thorsten Altenkirch who directed my interest towards intensional aspects of the interpolation theorem, and to Jan von Plato who introduced me to the field of generalized eliminations and inspired the present work by his visit to Munich in September 1999. Also thanks to the anonymous referee whose comments were of great help.

2 System ΛJ with Sums and Pairs

We study an extension of simply-typed λ-calculus in the spirit of von Plato [vP98]. The main idea is the use of generalized eliminations, most notably the generalized $\to$-elimination (see below) which is the closure under substitution of the left implication rule of Gentzen, written in the syntax of λ-calculus. For the implicational fragment—called ΛJ—see [JM99]. ΛJ can also be studied in an untyped fashion (hence departing completely from its logical origins): Its confluence and standardization have been established in [JM00]. On the other hand, ΛJ can be assigned intersection types, and thus a characterization of the strongly normalizing terms via typability can be given [Mat00].

2.1 Types and Terms

Types (depending on an infinite supply of type variables denoted α, β—also with decoration) are defined inductively:

$$\begin{array}{rl}
(\mathrm{V}) & \alpha \\
(\to) & \rho \to \sigma \\
(+) & \rho + \sigma \\
(\times) & \rho \times \sigma \\
(1) & 1 \\
(0) & 0
\end{array}$$

Let $\mathrm{FV}(\rho)$ be the set of type variables occurring in ρ ("free variables"). We associate $\to$ to the right and abbreviate $\rho_1 \to \ldots \to \rho_n \to \sigma$ by $\boldsymbol{\rho} \to \sigma$.

Typed terms (with types indicated as superscripts or after a colon, and also depending on an infinite supply of variables x^ρ for any type ρ) are defined inductively:

$$\begin{array}{rl}
(\mathrm{V}) & x^\rho : \rho \\
(\to\text{-I}) & \lambda x^\rho r^\sigma : \rho \to \sigma \\
(\to\text{-E}) & r^{\rho\to\sigma}(s^\rho, x^\sigma.t^\tau) : \tau \\
(+\text{-I}) & \mathrm{inj}_{1,\rho} r^\sigma : \sigma + \rho,\ \mathrm{inj}_{2,\rho} r^\sigma : \rho + \sigma \\
(+\text{-E}) & r^{\rho+\sigma}(x^\rho.s^\tau, y^\sigma.t^\tau) : \tau \\
(\times\text{-I}) & \langle r^\rho, s^\sigma \rangle : \rho \times \sigma \\
(\times\text{-E}) & r^{\rho\times\sigma}(x^\rho y^\sigma.t^\tau) : \tau \\
(1\text{-I}) & \mathrm{IN1} : 1 \\
(0\text{-E}) & r^0 \rho : \rho
\end{array}$$

Note that +-elimination always [Pra65] has been given in the "generalized style" and that ×-elimination is the same as tensor elimination in linear logic[1]. The essential new ingredient (by von Plato) is the generalized →-elimination. In λ-calculus, we would have

$$(\to\text{-E})' \quad r^{\rho\to\sigma}s^{\rho} : \sigma$$
$$(\times\text{-E})' \quad r^{\rho\times\sigma}\mathsf{L} : \rho \text{ and } r^{\rho\times\sigma}\mathsf{R} : \sigma$$

Let $\mathrm{FV}(r)$ be the set of term variables occurring free in r where x is bound in $\lambda x r$ and in $x.t$ (and also for x and y in $x.y.t$ where $x \neq y$ is understood). We identify terms which differ only in the names of their bound variables already on the syntactic level. Hence, e. g., $x(y, z.z)$ and $x(y, y.y)$ are the same term, and α-conversion is kept out of the system. With this identification, it is straightforward to define the result $r^{\rho}[x^{\sigma} := s^{\sigma}]$ of substituting s for x in r (without capture of free variables) as a term of type ρ. In the same way, one can define simultaneous substitution $r^{\rho}[x^{\sigma}, y^{\tau} := s^{\sigma}, t^{\tau}]$ of x by s and y by t (with $x \neq y$) which also gives a term of type ρ.

2.2 Reduction

Rules of β-conversion:

$$(\beta_{\to}) \quad (\lambda x^{\rho} r^{\sigma})(s^{\rho}, y^{\sigma}.t^{\tau}) \rhd t[y := r[x := s]]$$
$$(\beta_{+}) \quad (\mathrm{inj}_{i,\rho_{3-i}} r^{\rho_i})(x_1^{\rho_1}.r_1^{\tau}, x_2^{\rho_2}.r_2^{\tau}) \rhd r_i[x_i := r] \text{ for } i = 1, 2$$
$$(\beta_{\times}) \quad \langle r^{\rho}, s^{\sigma}\rangle(x^{\rho}y^{\sigma}.t^{\tau}) \rhd t[x, y := r, s]$$

Clearly, if $r \rhd s$ then s has the same type as r ("subject reduction"). Since also $\mathrm{FV}(s) \subseteq \mathrm{FV}(r)$, theses rules are a valid means of transforming proofs (via the Curry-Howard-isomorphism [How80]).

In λ-calculus, we would have

$$(\beta_{\to})' \quad (\lambda x^{\rho} r^{\sigma}) s^{\rho} \rhd r[x := s]$$
$$(\beta_{\times})' \quad \langle r^{\rho}, s^{\sigma}\rangle\mathsf{L} \rhd r \text{ and } \langle r^{\rho}, s^{\sigma}\rangle\mathsf{R} \rhd s$$

It is well-known for λ-calculus with sums that β-reduction alone does not produce terms with the subformula property (every subterm should have a type which forms a part of the type of the term or any of its free variables). In $\Lambda\mathrm{J}$, this already comes from the fragment with function types only, e. g. the term $x^{\alpha\to\alpha}(y^{\alpha}, z^{\alpha}.\lambda u^{\rho\to\rho} u)(\lambda v^{\rho} v, w^{\rho\to\rho}.y) : \alpha$ with free variables $x^{\alpha\to\alpha}$ and y^{α} is β-normal but has a subterm of uncontrolled type ρ.

Let an elimination be anything which may occur as an argument in one of the elimination rules of the term system, hence an object $(s, z.t), (x.s, y.t), (x.y.t)$ or a type (for (0-E)). They will be denoted by $R, S, \ldots$ (For λ-calculus, terms, objects of the form $(x.s, y.t)$, L, R and types would be eliminations.) $\mathrm{FV}(S)$ shall have the obvious meaning as the free variables of S.

Define for any elimination S such that $y^{\tau}S$ is a term, the following eliminations:

[1] For restrictions to linear terms this is much more useful than the two projections, expressed below by L and R.

$(\to\text{-E})$ $(s, x.t^\tau)\{S\} := (s, x.tS)$ where we assume that $x \notin \mathrm{FV}(S)$
$(+\text{-E})$ $(x_1.t_1^\tau, x_2.t_2^\tau)\{S\} := (x_1.t_1S, x_2.t_2S)$ where we assume that $x_1, x_2 \notin \mathrm{FV}(S)$
$(\times\text{-E})$ $(x_1.x_2.t^\tau)\{S\} := (x_1.x_2.tS)$ where we assume that $x_1, x_2 \notin \mathrm{FV}(S)$
(0-E) $\tau\{S\} := \sigma$ with σ the type of the term $x^\tau S$

It is immediate that whenever rRS is a term, then $R\{S\}$ is an elimination and $rR\{S\}$ is also a term.

Rule schema of permutative conversion:

(π) $rRS \rhd rR\{S\}$

Written out more explicitly, the schema comprises:

$(\pi_\to)$ $r^{\rho_1\to\rho_2}(s, x.t)S \rhd r(s, x.tS)$
(π_+) $r^{\rho_1+\rho_2}(x_1.t_1, x_2.t_2)S \rhd r(x_1.t_1S, x_2.t_2S)$
$(\pi_\times)$ $r^{\rho_1\times\rho_2}(x_1.x_2.t)S \rhd r(x_1.x_2.tS)$
(π_0) $(r^0\tau S)^\sigma \rhd r\sigma$

The rule (π_+) is the usual permutative conversion for sums in natural deduction formulation.

We also introduce the following projection rules (to be justified below):

$(\kappa_\to)$ $r(s, x.t) \rhd t$ for $x \notin \mathrm{FV}(t)$
$(\kappa_\times)$ $r(x.y.t) \rhd t$ for $x, y \notin \mathrm{FV}(t)$

Let $\twoheadrightarrow$ be the term closure of all of the above conversions and let $\twoheadrightarrow_0$ be that of all the conversions except the projection rules. Both reduction relations satisfy subject reduction since types are obviously preserved under $\rhd$.

The set NF of normal forms is defined inductively by:

(V) $x \in \mathrm{NF}$.
$(\to\text{-E})$ $s, t \in \mathrm{NF} \wedge y \in \mathrm{FV}(t) \Rightarrow x(s, y.t) \in \mathrm{NF}$.
$(+\text{-E})$ $t_1, t_2 \in \mathrm{NF} \Rightarrow x(x_1.t_1, x_2.t_2) \in \mathrm{NF}$.
$(\times\text{-E})$ $t \in \mathrm{NF} \wedge x_1$ or $x_2 \in \mathrm{FV}(t) \Rightarrow x(x_1.x_2.t) \in \mathrm{NF}$.
(0-E) $x\tau \in \mathrm{NF}$.
$(\to\text{-I})$ $r \in \mathrm{NF} \Rightarrow \lambda x r \in \mathrm{NF}$.
$(+\text{-I})$ $r \in \mathrm{NF} \Rightarrow \mathrm{inj}_i r \in \mathrm{NF}$.
$(\times\text{-I})$ $r_1, r_2 \in \mathrm{NF} \Rightarrow \langle r_1, r_2\rangle \in \mathrm{NF}$.
(1-I) $\mathrm{IN1} \in \mathrm{NF}$.

It can easily be checked that NF is the set of terms which are normal with respect to $\twoheadrightarrow$. Note that unlike λ-calculus, we do not need to consider several arguments to a variable (or even worse for sums and products) but only one elimination.

Define the set NF_0 by removing the requirements on free variables from the definition of NF. Again, it is easy to check that NF_0 characterizes the normal terms with respect to $\twoheadrightarrow_0$.

2.3 Comparison with λ-Calculus

Clearly, we can embed λ-calculus into our system by setting $rs := r(s, z.z)$ and $r\mathsf{L} := r(x.y.x)$ and $r\mathsf{R} := r(x.y.y)$ because then $(\lambda xr)s = (\lambda xr)(s, z.z) \to_{\beta_\to} r[x := s]$ and $\langle r, s\rangle\mathsf{L} = \langle r, s\rangle(x.y.x) \to_{\beta_\times} r$ and likewise for $\langle r, s\rangle\mathsf{R}$.

ΛJ can also be encoded in terms of λ-calculus: $r(s, x.t) := t[x := rs]$ and $r(x.y.t) := t[x, y := r\mathsf{L}, r\mathsf{R}]$. This time, only the equalities induced by the rewrite systems are preserved (call $=_\lambda$ that of λ-calculus):

$$
\begin{array}{rl}
(\beta_\to) & (\lambda xr)(s, z.t) = t[z := (\lambda xr)s] =_\lambda t[z := r[x := s]] \\
(\beta_\times) & \langle r, s\rangle(x.y.t) = t[x, y := \langle r, s\rangle\mathsf{L}, \langle r, s\rangle\mathsf{R}] =_\lambda t[x, y := r, s] \\
(\pi_\to) & r(s, x.t)S = t[x := rs]S = (tS)[x := rs] = r(s, x.tS) \\
(\pi_\times) & r(x.y.t)S = t[x, y := r\mathsf{L}, r\mathsf{R}]S = (tS)[x, y := r\mathsf{L}, r\mathsf{R}] = r(x.y.tS) \\
(\kappa_\to) & r(s, x.t) = t[x := rs] = t \text{ if } x \notin \mathrm{FV}(t) \\
(\kappa_\times) & r(x.y.t) = t[x, y := r\mathsf{L}, r\mathsf{R}] = t \text{ if } x, y \notin \mathrm{FV}(t)
\end{array}
$$

It is well-known that the set of normal λ-terms w. r. t. $(\beta_\to)$', (β_+), $(\beta_\times)$', (π_+) and (π_0)—henceforth the corresponding reduction will be denoted by $\twoheadrightarrow_\lambda$—is inductively characterized by the set NF', defined as follows:

$$
\begin{array}{rl}
\text{(V)} & x \in \mathrm{NF}'. \\
(\to\text{-E}) & s, t \in \mathrm{NF}' \wedge y \in \mathrm{FV}(t) \Rightarrow t[y := xs] \in \mathrm{NF}'. \\
(+\text{-E}) & t_1, t_2 \in \mathrm{NF}' \Rightarrow x(x_1.t_1, x_2.t_2) \in \mathrm{NF}'. \\
(\times\text{-E}) & t \in \mathrm{NF}' \wedge x_1 \text{ or } x_2 \in \mathrm{FV}(t) \Rightarrow t[x_1, x_2 := x\mathsf{L}, x\mathsf{R}] \in \mathrm{NF}'. \\
\text{(0-E)} & x\tau \in \mathrm{NF}'. \\
(\to\text{-I}) & r \in \mathrm{NF}' \Rightarrow \lambda xr \in \mathrm{NF}'. \\
(+\text{-I}) & r \in \mathrm{NF}' \Rightarrow \mathrm{inj}_i r \in \mathrm{NF}'. \\
(\times\text{-I}) & r_1, r_2 \in \mathrm{NF}' \Rightarrow \langle r_1, r_2\rangle \in \mathrm{NF}'. \\
\text{(1-I)} & \mathrm{IN1} \in \mathrm{NF}'.
\end{array}
$$

Apparently, NF' is nothing but the encoded version of NF. But unlike NF', NF's definition is syntax-directed and does not come as a surprise. Consequently, an analysis of normal forms in λ-calculus can be done more conveniently via a detour through ΛJ. We will see below that a more fine-grained analysis of interpolation can be achieved in this manner.

3 Strong Normalization, Standardization, and Confluence

We claim strong normalization of $\twoheadrightarrow$. This would be proved by combining the proofs of strong normalization of ΛJ (with function types only) with that of λ-calculus with sums and permutative conversions in [JM99] since we do not expect any harm from products.

It should be possible to address standardization in a similar way to [JM00] where untyped ΛJ (without case analysis and pairing) has been treated.

We would prefer to give a direct proof of confluence instead of proving local confluence and referring to strong normalization (Newman's Lemma). We believe that again the method in [JM00] should carry over to the full system. (Local confluence can easily be established by considering the critical pairs which are non-trivial, though.)

4 Interpolation

Intuitionistic logic has the interpolation property which means that from a proof of $\rho \to \sigma$ one can find a type τ with type variables among the intersection of those in ρ and σ such that $\rho \to \tau$ and $\tau \to \sigma$ are provable. This would be called Craig interpolation. From the proof, one can see that more can be said about positivity and negativity of the occurrences of the type variables in τ: Positive occurrences in τ imply positive occurrences in both ρ and σ, and similarly for negative occurrences. This is called Lyndon interpolation.

We did not yet exploit our λ-calculus structure in which we formulate intuitionistic reasoning. Since there are terms (a. k. a. proofs) of $r : \rho \to \sigma$, $s : \rho \to \tau$ and $t : \tau \to \sigma$, we may ask whether one can have s and t even such that r and $\lambda x^{\rho}.t(sx)$ are intensionally equal and not merely proofs of the same statement. [Čub94] established this fact for usual λ-calculus (and extended the result to bicartesian closed categories). In his case, intensional equality for λ-calculus meant equality w. r. t. β-reduction and permutations. Unfortunately, this does not suffice for the treatment of generalized applications. Therefore, we have to extend our notion of reduction by extended permutation rules and then prove Lyndon interpolation in a style quite similar to [Čub94]. However, we will profit a lot from the simpler structure of normal forms in our calculus.

4.1 Adding Some Extensionality

In order to formulate the interpolation theorem we need a more powerful permutation principle for function and product types. The extended permutation rules are:

$(\pi^e_{\to})$ $g[z := r(s, x.t)] \rhd r(s, x.g[z := t])$ for any term g with $z \in \mathrm{FV}(g)$ and (w. l. o. g.) $x \notin \mathrm{FV}(g)$

$(\pi^e_{\times})$ $g[z := r(x.y.t)] \rhd r(x.y.g[z := t])$ for any term g with $z \in \mathrm{FV}(g)$ and (w. l. o. g.) $x, y \notin \mathrm{FV}(g)$

Clearly, the standard permutation rules are the special case with $g := zS$. The extended rules are also justified by our encoding into λ-calculus since in that encoding, we have $g[z := r(s, x.t)] = g[z := t[x := rs]] = r(s, x.g[z := t])$ and $g[z := r(x.y.t)] = g[z := t[x, y := r\mathsf{L}, r\mathsf{R}]] = r(x.y.g[z := t])$.

Let $\twoheadrightarrow_e$ be the reduction relation which is the term closure of the β-rules, (π_+), $(\pi^e_{\to})$ and $(\pi^e_{\times})$.

The (excluded) degenerate cases with $z \notin \mathrm{FV}(g)$ would read:

$(\kappa^r_{\to})$ $g \rhd r(s, x.g)$ for $x \notin \mathrm{FV}(g)$

$(\kappa^r_{\times})$ $g \rhd r(x.y.g)$ for $x, y \notin \mathrm{FV}(g)$

They are simply the reverses of $(\kappa_{\to})$ and $(\kappa_{\times})$. Hence, with respect to equality theory, the degenerate cases are already included in $\twoheadrightarrow$.

4.2 The Interpolation Theorem

As a notation, for a set Γ of typed variables with types $\rho_1, \ldots, \rho_n$, write $\Gamma^{\boldsymbol{\rho}}$. Then, $\otimes\boldsymbol{\rho} := \rho_1 \times \ldots \times \rho_n$ (associated to the left). The operator $\otimes$ shall bind stronger than the other connectives.

Let $+(\rho)$ be the set of type variables occurring at some (not necessarily strictly) positive position in ρ and likewise let $-(\rho)$ be the set of type variables occurring negatively in ρ. E. g. $\alpha, \beta \in +((\alpha \to \alpha) \to \beta)$ and only $\alpha \in -((\alpha \to \alpha) \to \beta)$. p (for polarity) will be a variable ranging over $\{+, -\}$. Hence, $\alpha \in p(\rho)$ is a well-defined statement conditional on p's value. Also write $-p$ to mean $-$ in case $p = +$ and $+$ in case $p = -$. More formally, we may now define $+(\rho)$ and $-(\rho)$ as follows:

$$
\begin{array}{rl}
(\mathrm{V}) & +(\alpha) := \{\alpha\},\ -(\alpha) := \emptyset. \\
(\to) & \alpha \in p(\rho \to \sigma) :\Leftrightarrow \alpha \in -p(\rho) \vee \alpha \in p(\sigma) \\
(+) & \alpha \in p(\rho + \sigma) :\Leftrightarrow \alpha \in p(\rho) \vee \alpha \in p(\sigma) \\
(\times) & \alpha \in p(\rho \times \sigma) :\Leftrightarrow \alpha \in p(\rho) \vee \alpha \in p(\sigma) \\
(1) & \alpha \notin p(1) \\
(0) & \alpha \notin p(0)
\end{array}
$$

Theorem 1. Let $r^\rho \in \mathrm{NF}$ and $\mathrm{FV}(r) \subseteq \Pi^{\boldsymbol{\pi}} \uplus \Sigma^{\boldsymbol{\sigma}}$. Set $\Pi_r := \mathrm{FV}(r) \cap \Pi$ and $\Sigma_r := \mathrm{FV}(r) \cap \Sigma$. Then, there is a type $\hat{\rho}$ and terms $f^{\hat{\rho}}$ and $\lambda z^{\hat{\rho}} g^\rho$ both in NF such that

$$
\begin{array}{rl}
(\mathrm{FV}) & \Pi_r \subseteq \mathrm{FV}(f) \subseteq \Pi,\ \Sigma_r \subseteq \mathrm{FV}(\lambda z g) \subseteq \Sigma \text{ and } (z \in \mathrm{FV}(g) \Leftrightarrow \Pi_r \neq \emptyset) \\
 & \text{and } z \text{ occurs at most once in } g \\
(+-) & \alpha \in p(\hat{\rho}) \Rightarrow \alpha \in p(\otimes\boldsymbol{\pi}) \cap p(\boldsymbol{\sigma} \to \rho) \text{ (for all } \alpha \text{ and all values of } p) \\
(\twoheadrightarrow_e) & g[z := f] \twoheadrightarrow_e^* r
\end{array}
$$

Notation: $\mathrm{I}(r, \Pi, \Sigma) = (\hat{\rho}, f, \lambda z g)$. ($\uplus$ above denotes disjoint union.)

Notice that this theorem is a strengthening of interpolation: By assumption, r proves $\otimes\boldsymbol{\pi} \to \boldsymbol{\sigma} \to \rho$, and a type $\hat{\rho}$ is produced together with proofs of $\otimes\boldsymbol{\pi} \to \hat{\rho}$ and $\hat{\rho} \to \boldsymbol{\sigma} \to \rho$, and every α occurring in $\hat{\rho}$ also occurs in $\otimes\boldsymbol{\pi}$ and in $\boldsymbol{\sigma} \to \rho$. The interpolation theorem one would like to work with follows with $\boldsymbol{\pi} := \pi$ and $\boldsymbol{\sigma} := \emptyset$.

Also note that from our earlier discussion on NF versus NF′, we can infer interpolation for λ-calculus as in [Čub94][2].

Proof. Firstly, the function I is defined by recursion on $r \in \mathrm{NF}$.[3] It is always clear that the second and third component are in NF_0, have the right types and that $\mathrm{FV}(f) \subseteq \Pi$ and $\mathrm{FV}(\lambda z g) \subseteq \Sigma$, and that z occurs at most once in g .

[2] We cannot guarantee that z occurs only once in g, and we will deduce $g[z := f] \twoheadrightarrow_\lambda^* r$ from $g[z := f] =_\lambda r$ and confluence of $\twoheadrightarrow_\lambda$.

[3] We have to be careful with the disjointness assumption for Π and Σ in the recursive calls. Nevertheless, under the renaming convention, we see that in the course of the recursion we never change Π and Σ such that they intersect again.

(triv) Case $\Pi_r = \emptyset$. Set

$$\mathrm{I}(r, \Pi, \Sigma) := (1, \mathrm{IN1}, \lambda z r)$$

with a new variable z. All the other cases are under the proviso "otherwise".
(V) Case x^ρ with $x \in \Pi$. Set

$$\mathrm{I}(x, \Pi, \Sigma) := (\rho, x, \lambda x x).$$

Case x^ρ with $x \in \Sigma$. This is covered by (triv); $\mathrm{I}(x, \Pi, \Sigma) = (1, \mathrm{IN1}, \lambda z x)$.
($\to$-E) Case $x^{\rho_1 \to \rho_2}(s^{\rho_1}, y^{\rho_2} t^\tau)$ with $x \in \Pi$. We already have

$$\mathrm{I}(s, \Sigma, \Pi) = (\hat{\rho}_1, f_1, \lambda z_1 g_1)$$

(note the interchange of Π and Σ) and

$$\mathrm{I}(t, \Pi \cup \{y\}, \Sigma) = (\hat{\tau}, f_2, \lambda z_2 g_2).$$

Set

$$\mathrm{I}(x(s, y.t), \Pi, \Sigma) := (\hat{\rho}_1 \to \hat{\tau}, \lambda z_1.x(g_1, y.f_2), \lambda z.z(f_1, z_2.g_2)).$$

Case $x^{\rho_1 \to \rho_2}(s^{\rho_1}, y^{\rho_2} t^\tau)$ with $x \in \Sigma$. We already have

$$\mathrm{I}(s, \Pi, \Sigma) = (\hat{\rho}_1, f_1, \lambda z_1 g_1)$$

and

$$\mathrm{I}(t, \Pi, \Sigma \cup \{y\}) = (\hat{\tau}, f_2, \lambda z_2 g_2).$$

Set

$$\mathrm{I}(x(s, y.t), \Pi, \Sigma) := (\hat{\rho}_1 \times \hat{\tau}, \langle f_1, f_2 \rangle, \lambda z.z(z_1.z_2.x(g_1, y.g_2))).$$

(+-E) Case $x^{\rho_1 + \rho_2}(x_1^{\rho_1}.t_1^\tau, x_2^{\rho_2}.t_2^\tau)$ with $x \in \Pi$. We already have

$$\mathrm{I}(t_i, \Pi \cup \{x_i\}, \Sigma) = (\hat{\tau}_i, f_i, \lambda z_i g_i) \quad \text{for } i = 1, 2.$$

Set $\mathrm{I}(x(x_1.t_1, x_2.t_2), \Pi, \Sigma) :=$

$$(\hat{\tau}_1 + \hat{\tau}_2, x(x_1.\mathrm{inj}_{1,\hat{\tau}_2} f_1, x_2.\mathrm{inj}_{2,\hat{\tau}_1} f_2), \lambda z.z(z_1.g_1, z_2.g_2)).$$

Case $x^{\rho_1 + \rho_2}(x_1^{\rho_1}.t_1^\tau, x_2^{\rho_2}.t_2^\tau)$ with $x \in \Sigma$. We already have

$$\mathrm{I}(t_i, \Pi, \Sigma \cup \{x_i\}) = (\hat{\tau}_i, f_i, \lambda z_i g_i) \quad \text{for } i = 1, 2.$$

Set $\mathrm{I}(x(x_1.t_1, x_2.t_2), \Pi, \Sigma) :=$

$$(\hat{\tau}_1 \times \hat{\tau}_2, \langle f_1, f_2 \rangle, \lambda z.z(z_1.z_2.x(x_1.g_1, x_2.g_2))).$$

($\times$-E) Case $x^{\rho_1 \times \rho_2}(x_1^{\rho_1}.x_2^{\rho_2}.t^\tau)$ with $x \in \Pi$. We already have

$$\mathrm{I}(t, \Pi \cup \{x_1, x_2\}, \Sigma) = (\hat{\tau}, f, \lambda z g).$$

Set

$$\mathrm{I}(x(x_1.x_2.t), \Pi, \Sigma) := (\hat{\tau}, x(x_1.x_2.f), \lambda z g).$$

Case $x^{\rho_1\times\rho_2}(x_1^{\rho_1}.x_2^{\rho_2}.t^\tau)$ with $x\in\Sigma$. We already have

$$\mathrm{I}(t,\Pi,\Sigma\cup\{x_1,x_2\})=(\hat\tau,f,\lambda zg).$$

Set

$$\mathrm{I}(x(x_1.x_2.t),\Pi,\Sigma):=(\hat\tau,f,\lambda z.x(x_1.x_2.g)).$$

(0-E) Case $x^0\tau$ with $x\in\Pi$. Set

$$\mathrm{I}(x\tau,\Pi,\Sigma):=(0,x,\lambda x.x\tau).$$

Case $x^0\tau$ with $x\in\Sigma$. (triv) yields $\mathrm{I}(x\tau,\Pi,\Sigma)=(1,\mathrm{IN1},\lambda z.x\tau)$.
($\to$-I) Case $\lambda x^\rho r^\sigma$. We already have $\mathrm{I}(r,\Pi,\Sigma\cup\{x\})=(\hat\sigma,f,\lambda zg)$. Set

$$\mathrm{I}(\lambda xr,\Pi,\Sigma):=(\hat\sigma,f,\lambda z\lambda x.g).$$

($+$-I) Case $\mathrm{inj}_{i,\rho}r^\sigma$. We already have $\mathrm{I}(r,\Pi,\Sigma)=(\hat\sigma,f,\lambda zg)$. Set

$$\mathrm{I}(\mathrm{inj}_{i,\rho}r,\Pi,\Sigma):=(\hat\sigma,f,\lambda z.\mathrm{inj}_{i,\rho}g).$$

($\times$-I) Case $\langle r_1^{\rho_1},r_2^{\rho_2}\rangle$. We already have $\mathrm{I}(r_i,\Pi,\Sigma)=(\hat\rho_i,f_i,\lambda z_ig_i)$ for $i=1,2$. Set

$$\mathrm{I}(\langle r_1,r_2\rangle,\Pi,\Sigma):=(\hat\rho_1\times\hat\rho_2,\langle f_1,f_2\rangle,\lambda z.z(z_1.z_2.\langle g_1,g_2\rangle)).$$

(1-I) Case IN1. Covered by (triv), hence $\mathrm{I}(\mathrm{IN1},\Pi,\Sigma)=(1,\mathrm{IN1},\lambda z.\mathrm{IN1})$.

Because of (triv), we trivially have that $z\in\mathrm{FV}(g)$ implies $\Pi_r\neq\emptyset$.

Concerning the polarity requirement $(+-)$ we only deal with an implication elimination $x^{\rho_1\to\rho_2}(s^{\rho_1},y^{\rho_2}.t^\tau)$. The other cases do not need any further sophistication.

Case $x\in\Pi$. Let $\alpha\in p(\hat\rho_1\to\hat\tau)$. Therefore, $\alpha\in -p(\hat\rho_1)\cup p(\hat\tau)$. Show that $\alpha\in p(\otimes\boldsymbol{\pi})\cap p(\boldsymbol{\sigma}\to\tau)$.

Case $\alpha\in -p(\hat\rho_1)$. By induction hypothesis, $\alpha\in -p(\otimes\boldsymbol{\sigma})\cap -p(\boldsymbol{\pi}\to\rho_1)$. Therefore, $\alpha\in p(\boldsymbol{\sigma}\to\tau)$. Also, $\alpha\in p(\otimes\boldsymbol{\pi})\cup -p(\rho_1)$. In case $\alpha\in -p(\rho_1)$, $\alpha\in p(\rho_1\to\rho_2)\subseteq p(\otimes\boldsymbol{\pi})$ because $x^{\rho_1\to\rho_2}\in\Pi$.

Case $\alpha\in p(\hat\tau)$. By induction hypothesis, $\alpha\in p(\otimes\boldsymbol{\pi}\times\rho_2)\cap p(\boldsymbol{\sigma}\to\tau)$. Hence, $\alpha\in p(\otimes\boldsymbol{\pi})\cup p(\rho_2)$. If $\alpha\in p(\rho_2)$, then $\alpha\in p(\rho_1\to\rho_2)\subseteq p(\otimes\boldsymbol{\pi})$ again since $x^{\rho_1\to\rho_2}\in\Pi$.

Case $x\in\Sigma$. Let $\alpha\in p(\hat\rho_1\times\hat\tau)$. Therefore, $\alpha\in p(\hat\rho_1)\cup p(\hat\tau)$. Show that $\alpha\in p(\otimes\boldsymbol{\pi})\cap p(\boldsymbol{\sigma}\to\tau)$.

Case $\alpha\in p(\hat\rho_1)$. By induction hypothesis, $\alpha\in p(\otimes\boldsymbol{\pi})\cap p(\boldsymbol{\sigma}\to\rho_1)$. Hence, $\alpha\in p(\boldsymbol{\sigma}\to\tau)\cup p(\rho_1)$. In case $\alpha\in p(\rho_1)$, $\alpha\in -p(\rho_1\to\rho_2)\subseteq p(\boldsymbol{\sigma}\to\tau)$ because $x^{\rho_1\to\rho_2}\in\Sigma$.

Case $\alpha\in p(\hat\tau)$. By induction hypothesis, $\alpha\in p(\otimes\boldsymbol{\pi})\cap p(\boldsymbol{\sigma}\to\rho_2\to\tau)$. Hence, $\alpha\in p(\boldsymbol{\sigma}\to\tau)\cup -p(\rho_2)$. If $\alpha\in -p(\rho_2)$, then $\alpha\in -p(\rho_1\to\rho_2)\subseteq p(\boldsymbol{\sigma}\to\tau)$ again since $x^{\rho_1\to\rho_2}\in\Sigma$.

The verification of the statements that f and g have certain free variables, hinges on the tacit assumption in the definition that names of the bound variables are always chosen appropriately (which can be done by the renaming convention). First show that if $z \notin \mathrm{FV}(g)$, then $\Pi_r = \emptyset$: Only the cases (triv), ($\times$-E), ($\to$-I) and ($+$-I) have to be analyzed. The most interesting case is ($\times$-E)Π: $x \in \Pi$. Therefore, we have to show that $z \in \mathrm{FV}(g)$. This holds by induction hypothesis, since due to $x(x_1.x_2.t) \in \mathrm{NF}$, x_1 or x_2 in $\mathrm{FV}(t)$. The other cases follow directly from the induction hypothesis. Therefore, we have proved

$$\Pi_r \neq \emptyset \Rightarrow z \in \mathrm{FV}(g) \quad (*)$$

Now prove $\Pi_r \subseteq \mathrm{FV}(f)$ and $\Sigma_r \subseteq \mathrm{FV}(\lambda z g)$ together with $f, \lambda z g \in \mathrm{NF}$: (Again we only deal with ($\to$-E) because no further arguments are needed for the other cases.)

Case $x \in \Pi$. $\Pi_{x(s,y.t)} = \{x\} \cup \Pi_s \cup (\Pi_t \setminus \{y\})$. $\Sigma_{x(s,y.t)} = \Sigma_s \cup (\Sigma_t \setminus \{y\})$. $y \in \mathrm{FV}(t)$ (by definition of NF), hence $(\Pi \cup \{y\})_t = \Pi_t \cup \{y\}$. By induction hypothesis, $\Sigma_s \subseteq \mathrm{FV}(f_1)$, $\Pi_s \subseteq \mathrm{FV}(\lambda z_1 g_1)$, $\Pi_t \cup \{y\} \subseteq \mathrm{FV}(f_2)$ and $\Sigma_t \subseteq \mathrm{FV}(\lambda z_2 g_2)$. Therefore, $\Pi_{x(s,y.t)} \subseteq \mathrm{FV}(\lambda z_1.x(g_1, y.f_2))$, $y \in \mathrm{FV}(f_2)$ and $\Sigma_{x(s,y.t)} \subseteq \mathrm{FV}(\lambda z.z(f_1, z_2.g_2))$. Finally, by $(*)$, $z_2 \in \mathrm{FV}(g_2)$.

Case $x \in \Sigma$. $\Pi_{x(s,y.t)} = \Pi_s \cup (\Pi_t \setminus \{y\})$. $\Sigma_{x(s,y.t)} = \{x\} \cup \Sigma_s \cup (\Sigma_t \setminus \{y\})$. $y \in \mathrm{FV}(t)$ (by definition of NF), hence $(\Sigma \cup \{y\})_t = \Sigma_t \cup \{y\}$. We (implicitly) assume that $y \notin \Pi$, hence $\Pi_t \setminus \{y\} = \Pi_t$. By induction hypothesis, $\Pi_s \subseteq \mathrm{FV}(f_1)$, $\Sigma_s \subseteq \mathrm{FV}(\lambda z_1 g_1)$, $\Pi_t \subseteq \mathrm{FV}(f_2)$ and $\Sigma_t \cup \{y\} \subseteq \mathrm{FV}(\lambda z_2 g_2)$. Therefore, $\Pi_{x(s,y.t)} \subseteq \mathrm{FV}(\langle f_1, f_2\rangle)$, $\Sigma_{x(s,y.t)} \subseteq \mathrm{FV}(\lambda z.z(z_1.z_2.x(g_1, y.g_2))$ and $y \in \mathrm{FV}(g_2)$. Because $\Pi_{x(s,y.t)} \neq \emptyset$, $\Pi_s \neq \emptyset$ or $\Pi_t \neq \emptyset$. In the first case, $z_1 \in \mathrm{FV}(g_1)$ by $(*)$, in the second case $z_2 \in \mathrm{FV}(g_2)$ by $(*)$. Hence, by induction hypothesis (and $z_2 \neq y$), $z(z_1.z_2.x(g_1, y.g_2)) \in \mathrm{NF}$.

Finally, we check ($\twoheadrightarrow_e$). Only the reduction is given. The induction hypothesis is not explicitly mentioned. The subcases are simply indicated by Π and Σ.

(triv) $r = r$.

(V) Π: $x = x$.

($\to$-E) Π: $(\lambda z_1.x(g_1, y.f_2))(f_1, z_2.g_2) \twoheadrightarrow_{\beta_\to} g_2[z_2 := x(g_1[z_1 := f_1], y.f_2)] \twoheadrightarrow_e^* g_2[z_2 := x(s, y.f_2)] \twoheadrightarrow_{\pi^e_\to} x(s, y.g_2[z_2 := f_2]) \twoheadrightarrow_e^* x(s, y.t)$. Note that due to $(\Pi \cup \{y\})_t \neq \emptyset$ (by the requirement $y \in \mathrm{FV}(t)$ in the definition of NF), we have $z_2 \in \mathrm{FV}(g_2)$ which allows to apply $(\pi^e_\to)$.

Σ: $\langle f_1, f_2\rangle(z_1.z_2.x(g_1, y.g_2)) \twoheadrightarrow_{\beta_\times} x(g_1[z_1 := f_1], y.g_2[z_2 := f_2]) \twoheadrightarrow_e^* x(s, y.t)$

($+$-E) Π: $x(x_1.\mathrm{inj}_{1,\hat{\tau}_2} f_1, x_2.\mathrm{inj}_{2,\hat{\tau}_1} f_2)(z_1.g_1, z_2.g_2) \twoheadrightarrow_{\pi_+}$
$x(x_1.(\mathrm{inj}_{1,\hat{\tau}_2} f_1)(z_1.g_1, z_2.g_2), x_2.(\mathrm{inj}_{2,\hat{\tau}_1} f_2)(z_1.g_1, z_2.g_2)) \twoheadrightarrow_{\beta_+} \twoheadrightarrow_{\beta_+}$
$x(x_1.g_1[z_1 := f_1], x_2.g_2[z_2 := f_2]) \twoheadrightarrow_e^* x(x_1.t_1, x_2.t_2)$

Σ: $\langle f_1, f_2\rangle(z_1.z_2.x(x_1.g_1, x_2.g_2)) \twoheadrightarrow_{\beta_\times} x(x_1.g_1[z_1 := f_1], g_2[z_2 := f_2]) \twoheadrightarrow_e^* x(x_1.t_1, x_2.t_2)$

($\times$-E) Π: $g[z := x(x_1.x_2.f)] \twoheadrightarrow_{\pi^e_\times} x(x_1.x_2.g[z := f]) \twoheadrightarrow_e^* x(x_1.x_2.t)$: Because of $(\Pi \cup \{x_1, x_2\})_t \neq \emptyset$ (by the requirement x_1 or $x_2 \in \mathrm{FV}(t)$ in the definition of NF), we have $z \in \mathrm{FV}(g)$ which allows to apply $(\pi^e_\times)$.

Σ: $x(x_1.x_2.g[z := f]) \twoheadrightarrow_e^* x(x_1.x_2.t)$

(0-E) $\Pi\colon x\tau = x\tau$.
(→-I) $\lambda x.g[z := f] \twoheadrightarrow_e^* \lambda x r$.
(+-I) $\mathrm{inj}_{i,\rho} g[z := f] \twoheadrightarrow_e^* \mathrm{inj}_{i,\rho} r$.
(×-I) $\langle f_1, f_2\rangle(z_1.z_2.\langle g_1, g_2\rangle) \twoheadrightarrow_{\beta\times} \langle g_1[z_1 := f_1], g_2[z_2 := f_2]\rangle \twoheadrightarrow_e^* \langle r_1, r_2\rangle$ □

Note that $\mathrm{I}(r, \Pi, \Sigma)$ does not depend on variables which are not free in r, i. e. $\mathrm{I}(r, \Pi, \Sigma) = \mathrm{I}(r, \Pi_r, \Sigma_r)$. (However, the case of (→-I) requires the more general definition.)

For the applications we slightly improve the definition of I to a function I' which only differs in the treatment of the case (→-E): In case $x \in \Pi$ and s is a variable $x_1 \in \Pi$, we now set:

$$\mathrm{I}'(x(x_1, y.t), \Pi, \Sigma) := (\hat{\tau}, x(x_1, y.f_2), \lambda z_2 g_2).$$

The justification of the properties (FV), (+−) and ($\twoheadrightarrow_e$) is obvious due to $\mathrm{I}(x_1, \Sigma, \Pi) = (1, \mathrm{IN1}, \lambda z_1 x_1)$ and hence

$$\mathrm{I}(x(x_1, y.t), \Pi, \Sigma) = (1 \to \hat{\tau}, \lambda z_1.x(x_1, y.f_2), \lambda z.z(\mathrm{IN1}, z_2.g_2)).$$

In case $x \in \Sigma$ and t is the variable y, we set:

$$\mathrm{I}'(x(s, y.y), \Pi, \Sigma) := (\hat{\rho}_1, f_1, \lambda z_1.x(g_1, y.y)).$$

Again, it is easy to verify (FV), (+−) and ($\twoheadrightarrow_e$) by exploiting

$$\mathrm{I}(y, \Pi, \Sigma \cup \{y\}) = (1, \mathrm{IN1}, \lambda z_2 y)$$

and consequently

$$\mathrm{I}(x(s, y.y), \Pi, \Sigma) = (\hat{\rho}_1 \times 1, \langle f_1, \mathrm{IN1}\rangle, \lambda z.z(z_1.z_2.x(g_1, y.y)).$$

Let us illustrate that extended permutation rules were indeed needed.

Example 1. Set $r := x^{\rho\to\sigma}(u^\rho, y^\sigma.\lambda v^\tau y^\sigma)$. By (V), $I(u, \{u\}, \{x\}) = (\rho, u, \lambda u u)$ and $I(y, \{x, y\}, \{u, v\}) = (\sigma, y, \lambda y y)$. By (→-I), $I(\lambda v y, \{x, y\}, \{u\}) = (\sigma, y, \lambda y \lambda v y)$. Hence $I(r, \{x\}, \{u\}) = (\rho \to \sigma, \lambda u.x(u, y.y), \lambda z.z(u, y.\lambda v y))$. But we only have $(z(u, y.\lambda v y))[z := \lambda u.x(u, y.y)] \twoheadrightarrow_{\beta\to} (\lambda v y)[y := x(u, y.y)] = \lambda v.x(u, y.y)$ and cannot reach r by help of $\twoheadrightarrow^*$.

5 Some Uses of the Interpolation Algorithm

In the first subsection, additional concepts are given in preparation for the statements on the interpolation of liftings.

5.1 Long Normal Forms and Eta-Reduction

Define the set LNF of long normal forms by keeping all the rules of the definition of NF (with NF replaced by LNF) except (V) which now reads

(V) If $x : \alpha$ or $x : 0$ or $x : 1$, then $x \in \mathrm{LNF}$.

Lemma 1. The interpolation functions I and I′ preserve long normal forms, i. e., for $r^\rho \in \mathrm{LNF}$, $\mathrm{I}(r, \Pi, \Sigma) = (\hat{\rho}, f, \lambda z g)$ fulfills $f, \lambda z g \in \mathrm{LNF}$, and similarly for I′.

Proof. Immediate by induction on $r \in \mathrm{LNF}$. □

Consider the following conversions:

$(\eta_\to)$ $\lambda x^\rho.r^{\rho\to\sigma}(x, y^\sigma.y) \rhd r$, if $x \notin \mathrm{FV}(r)$
(η_+) $r^{\rho+\sigma}(x^\rho.\mathrm{inj}_{1,\sigma}x, y^\sigma.\mathrm{inj}_{2,\rho}y) \rhd r$
$(\eta_\times)$ $r^{\rho\times\sigma}(x^\rho.y^\sigma.\langle x, y\rangle) \rhd r$

Let $\twoheadrightarrow_\eta$ be the term closure of those three rules. The η-rules for λ-calculus would have the following differences:

$(\eta_\to)$' $\lambda x^\rho.r^{\rho\to\sigma}x \rhd r$, if $x \notin \mathrm{FV}(r)$
$(\eta_\times)$' $\langle r\mathsf{L}, r\mathsf{R}\rangle \rhd r$

Clearly, the above η-rules of $\Lambda\mathrm{J}$ are justifiable in our encoding of $\Lambda\mathrm{J}$ into λ-calculus since, in that encoding, $\lambda x.r(x, y.y) = \lambda x.rx$ and $r(x.y.\langle x, y\rangle) = \langle r\mathsf{L}, r\mathsf{R}\rangle$.

Define η-expansion $\eta(x^\rho) \in \mathrm{LNF}$ for variables by recursion on ρ as follows:

(triv) If ρ is a type variable or $\rho \in \{0, 1\}$, then $\eta(x) := x$.
$(\to)$ $\eta(x^{\rho\to\sigma}) := \lambda y^\rho.x(\eta(y), z^\sigma.\eta(z))$.
$(+)$ $\eta(x^{\rho+\sigma}) := x(y^\rho.\mathrm{inj}_{1,\sigma}\eta(y), z^\sigma.\mathrm{inj}_{2,\rho}\eta(z))$.
$(\times)$ $\eta(x^{\rho\times\sigma}) := x(y^\rho.z^\sigma.\langle\eta(y), \eta(z)\rangle)$.

Notice that λ-calculus does not allow such an easy η-expansion of variables.

5.2 Lifting of Positive and Negative Type Dependencies

It is well-known that if a type variable α occurs only positively in some type ρ, one can "lift" terms of type $\sigma \to \tau$ to terms of type $\rho[\alpha := \sigma] \to \rho[\alpha := \tau]$ (see e. g. [Lei90]). We pursue a new approach which does not use any positivity requirement for the definition (but will give the usual definition in case it is met).

But first, we have to introduce type substitution: $\rho[\alpha := \sigma]$ shall denote the result of replacing every occurrence of α in ρ by σ. Similarly, define the result $\rho[\alpha, \beta := \sigma, \tau]$ of simultaneous substitution of α by σ and β by τ in ρ (we assume $\alpha \neq \beta$). Type substitution and simultaneous type substitution may also be carried out for terms, denoted by $r[\alpha := \sigma]$ and $r[\alpha, \beta := \sigma, \tau]$, respectively.

Associate with every type variable α a "fresh" type variable α^-. Let α^p be α for $p = +$ and let α^{-p} be α^- for $p = +$ and α for $p = -$. Set likewise for term variables h and h_-, $h_p := h$ for $p = +$ and $h_{-p} := h_-$ for $p = +$ and $h_{-p} := h$ for $p = -$.

Define for every type ρ and every type variable α a term $\mathrm{lift}^p_{\lambda\alpha\rho}$ of type ρ with free variables among the typed variables $h : \alpha^{-p} \to \alpha^p$, $h_- : \alpha^p \to \alpha^{-p}$ and $x : \rho[\alpha := \alpha^-]$ by recursion on ρ (write $r[\alpha \leftrightarrow \alpha^-]$ for $r[\alpha, \alpha^- := \alpha^-, \alpha]$):

(triv) If ρ is a type variable $\neq \alpha$ or $\rho \in \{0, 1\}$, set $\mathrm{lift}^p_{\lambda\alpha\rho} := x$ (which is type-correct!)

(V) $\mathrm{lift}^p_{\lambda\alpha\alpha} := h_p(x, y^\alpha.y)$. (Note that $h_p : \alpha^- \to \alpha$.)

($\to$) $\mathrm{lift}^p_{\lambda\alpha.\rho_1\to\rho_2} :=$

$$\lambda x^{\rho_1}.x^{(\rho_1\to\rho_2)[\alpha:=\alpha^-]}(\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}[\alpha \leftrightarrow \alpha^-], x^{\rho_2[\alpha:=\alpha^-]}.\mathrm{lift}^p_{\lambda\alpha\rho_2}).$$

Note that $\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}[\alpha \leftrightarrow \alpha^-]$ has free variables among $h : \alpha^{-p} \to \alpha^p$, $h^- : \alpha^p \to \alpha^{-p}$ and x^{ρ_1}.

(+) $\mathrm{lift}^p_{\lambda\alpha.\rho_1+\rho_2} :=$

$$x^{(\rho_1+\rho_2)[\alpha:=\alpha^-]}(x^{\rho_1[\alpha:=\alpha^-]}.\mathrm{inj}_{1,\rho_2}\mathrm{lift}^p_{\lambda\alpha\rho_1}, x^{\rho_2[\alpha:=\alpha^-]}.\mathrm{inj}_{2,\rho_1}\mathrm{lift}^p_{\lambda\alpha\rho_2}).$$

($\times$) $\mathrm{lift}^p_{\lambda\alpha.\rho_1\times\rho_2} :=$

$$x^{(\rho_1\times\rho_2)[\alpha:=\alpha^-]}(x^{\rho_1[\alpha:=\alpha^-]}.x^{\rho_2[\alpha:=\alpha^-]}.\langle\mathrm{lift}^p_{\lambda\alpha\rho_1}, \mathrm{lift}^p_{\lambda\alpha\rho_2}\rangle).$$

It should be clear that x with differently written types means different variables (even if ρ_1 and ρ_2 happen to be the same type!).

Lemma 2. (i) $\mathrm{lift}^p_{\lambda\alpha\rho} \in \mathrm{LNF}$ and $x^{\rho[\alpha:=\alpha^-]} \in \mathrm{FV}(\mathrm{lift}^p_{\lambda\alpha\rho})$.
(ii) $\alpha \in p(\rho) \Leftrightarrow h \in \mathrm{FV}(\mathrm{lift}^p_{\lambda\alpha\rho})$ and $\alpha \in -p(\rho) \Leftrightarrow h_- \in \mathrm{FV}(\mathrm{lift}^p_{\lambda\alpha\rho})$.

Proof. By induction on ρ. □

By a slight extension of our notation with a variable q ranging over $\{+, -\}$, we may rewrite (ii) to $\alpha \in qp(\rho) \Leftrightarrow h_q \in \mathrm{FV}(\mathrm{lift}^p_{\lambda\alpha\rho})$.

For applications, we normally do not want that h_- occurs in $\mathrm{lift}^p_{\lambda\alpha\rho}$. Therefore, we define the set $\mathrm{Only}_+(\rho)$ of type variables occurring only positively in ρ and the set $\mathrm{Only}_-(\rho)$ of type variables occurring only negatively in ρ by setting $\alpha \in \mathrm{Only}_p(\rho) :\Leftrightarrow \alpha \notin -p(\rho)$. It is clear that we get the following recursive (with respect to ρ) characterization of $\mathrm{Only}_+(\rho)$ and $\mathrm{Only}_-(\rho)$:

(V) $\mathrm{Only}_+(\alpha) =$ all variables, $\mathrm{Only}_-(\alpha) =$ all variables except α.
($\to$) $\mathrm{Only}_p(\rho \to \sigma) = \mathrm{Only}_{-p}(\rho) \cap \mathrm{Only}_p(\sigma)$.
(+) $\mathrm{Only}_p(\rho + \sigma) = \mathrm{Only}_p(\rho) \cap \mathrm{Only}_p(\sigma)$.
($\times$) $\mathrm{Only}_p(\rho \times \sigma) = \mathrm{Only}_p(\rho) \cap \mathrm{Only}_p(\sigma)$.
(1) $\mathrm{Only}_p(1) =$ all variables.
(0) $\mathrm{Only}_p(0) =$ all variables.

Corollary 1. If $\alpha \in \mathrm{Only}_p(\rho)$, then $h_- \notin \mathrm{FV}(\mathrm{lift}^p_{\lambda\alpha\rho})$.

Lemma 3. If $\alpha \notin \mathrm{FV}(\rho)$, then $\mathrm{lift}^p_{\lambda\alpha\rho} = \eta(x)$.

Proof. Induction on ρ. □

5.3 Interpolation and Lifting

Lemma 4. (i) $\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha\rho}, \{h, h_-, x\}, \emptyset) = (\rho, \mathrm{lift}^p_{\lambda\alpha\rho}, \lambda z.\eta(z))$.
(ii) $\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha\rho}, \{x\}, \{h, h_-\}) = (\rho[\alpha := \alpha^-], \eta(x), \lambda x.\mathrm{lift}^p_{\lambda\alpha\rho})$.

Proof. By induction on ρ. Note that we never use the case (triv) of the interpolation theorem. The case (V) needs the modifications made for I′. Only (→) (i) will be dealt with: By induction hypothesis (i) and (ii)

$$\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha\rho_2}, \{h, h_-, x^{\rho_2[\alpha:=\alpha^-]}\}, \emptyset) = (\rho_2, \mathrm{lift}^p_{\lambda\alpha\rho_2}, \lambda z_2.\eta(z_2)) \quad \text{and}$$

$$\mathrm{I}'(\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}, \{x^{\rho_1[\alpha:=\alpha^-]}\}, \{h^{\alpha^p\to\alpha^{-p}}, h_-^{\alpha^{-p}\to\alpha^p}\}) =$$

$$(\rho_1[\alpha := \alpha^-], \eta(x^{\rho_1[\alpha:=\alpha^-]}), \lambda x^{\rho_1[\alpha:=\alpha^-]}.\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}).$$

Therefore, $\mathrm{I}'(\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}[\alpha \leftrightarrow \alpha^-], \{x^{\rho_1}\}, \{h^{\alpha^{-p}\to\alpha^p}, h_-^{\alpha^p\to\alpha^{-p}}\}) =$

$$(\rho_1, \eta(x^{\rho_1}), \lambda x^{\rho_1}.\mathrm{lift}^{-p}_{\lambda\alpha\rho_1}[\alpha \leftrightarrow \alpha^-]).$$

Finally (recall that we may drop superfluous variables from Π and Σ),

$\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha.\rho_1\to\rho_2}, \{h, h_-, x^{(\rho_1\to\rho_2)[\alpha:=\alpha^-]}\}, \emptyset) =$

$$(\rho_1 \to \rho_2, \mathrm{lift}^p_{\lambda\alpha.\rho_1\to\rho_2}, \lambda z\lambda x^{\rho_1}.z(\eta(x^{\rho_1}), z_2.\eta(z_2))).$$

□

For a fixed type variable α we now define ρ^+ which is ρ with every negative occurrence of α replaced by α^- (recall that α^- shall be a "fresh" type variable) and ρ^- which is $\rho^+[\alpha \leftrightarrow \alpha^-]$. Then $\alpha \in \mathrm{Only}_p(\rho^p)$ and $\alpha^- \in \mathrm{Only}_{-p}(\rho^p)$, in short: $\alpha^q \in \mathrm{Only}_{qp}(\rho^p)$. More formally, we define ρ^p for fixed α:

(triv) If ρ is a type variable $\neq \alpha$ or $\rho \in \{0, 1\}$, then set $\rho^p := \rho$.
(V) α^p is defined as before.
(→) $(\rho \to \sigma)^p := \rho^{-p} \to \sigma^p$.
(+) $(\rho + \sigma)^p := \rho^p + \sigma^p$.
(×) $(\rho \times \sigma)^p := \rho^p \times \sigma^p$.

It is clear that $\mathrm{FV}(\rho^p) \subseteq \mathrm{FV}(\rho) \cup \{\alpha^-\}$, that $\rho^{-p} = \rho^p[\alpha \leftrightarrow \alpha^-]$, $\rho^p[\alpha^- := \alpha] = \rho$ and $\alpha^q \in \mathrm{Only}_{qp}(\rho^p)$. [Concerning the last statement, consider its equivalent $\alpha^q \notin -qp(\rho^p)$. (V): If $p = q$, then $\alpha^p \notin -(\alpha^p)$. If $p \neq q$, then $\alpha^{-p} \notin +(\alpha^p)$. (→): Show $\alpha^q \notin -qp(\rho^{-p} \to \sigma^p)$, i. e., $\alpha \notin -q(-p)(\rho^{-p})$ and $\alpha \notin -qp(\sigma^p)$ which both follow from the induction hypothesis.]

Lemma 5. Let $x : \rho[\alpha := \alpha^-]$, $h : \alpha^{-p} \to \alpha^p$, $h_- : \alpha^p \to \alpha^{-p}$ (hence $h_q : \alpha^{-qp} \to \alpha^{qp}$, and therefore $h_p : \alpha^- \to \alpha$) and let $\hat\alpha$ be a "fresh" type variable.

Then $\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha\rho}, \{x, h_q\}, \{h_{-q}\}) =$

$$(\rho^{qp}, \mathrm{lift}^p_{\lambda\alpha.\rho^{qp}[\alpha^-:=\hat\alpha]}[\hat\alpha := \alpha^-], \lambda x^{\rho^{qp}}.\mathrm{lift}^p_{\lambda\alpha.\rho^{-qp}[\alpha^-:=\hat\alpha]}[\hat\alpha := \alpha]).$$

Proof. Induction on ρ (quite tedious). □

The preceding two lemmas may be written down in a uniform notation as follows:

$$\mathrm{I}'(\mathrm{lift}^p_{\lambda\alpha\rho}, \{x\} \cup \Pi, \Sigma) =$$

$$(\rho^*, \mathrm{lift}^p_{\lambda\alpha.\rho^*[\alpha^-:=\hat{\alpha}]}[\hat{\alpha} := \alpha^-], \lambda x^{\rho^*}.\mathrm{lift}^p_{\lambda\alpha.\rho^*[\alpha:=\hat{\alpha}][\alpha^-:=\alpha]}[\hat{\alpha} := \alpha])$$

$$\text{with } \rho^* := \begin{cases} \rho & \text{, if } h, h_- \in \Pi \\ \rho[\alpha := \alpha^-] & \text{, if } h, h_- \in \Sigma \\ \rho^p & \text{, if } h \in \Pi,\ h_- \in \Sigma \\ \rho^{-p} & \text{, if } h_- \in \Pi,\ h \in \Sigma \end{cases}$$

5.4 Application: Positivization of Monotone Types

Consider a type ρ and a type variable α such that there is a term $m : \rho$ with free variables among $h : \alpha^- \to \alpha$ and $x : \rho[\alpha := \alpha^-]$. We know that if $\alpha \in \mathrm{Only}_+(\rho)$, then $\mathrm{lift}^+_{\lambda\alpha\rho}$ is such a term. By lemma 4, interpolation (the function I') gives back the type ρ and that term together with an η-expansion of the identity. What do we get for arbitrary $m \in \mathrm{NF}$? We get

$$\mathrm{I}'(m, \{h, x\}, \emptyset) = (\hat{\rho}, f^{\hat{\rho}}, \lambda z^{\hat{\rho}} g^{\rho})$$

with $f, \lambda zg \in \mathrm{NF}$, $\mathrm{FV}(f) \subseteq \{h, x\}$, $\mathrm{FV}(\lambda zg) = \emptyset$, $\mathrm{FV}(\hat{\rho}) \subseteq \mathrm{FV}(\rho)$ (especially, $\alpha^- \notin \mathrm{FV}(\hat{\rho})$), $\alpha \in \mathrm{Only}_+(\hat{\rho})$ and $g[z := f] \twoheadrightarrow^*_e m$. A proof for the last but one property: If α were in $-(\hat{\rho})$, then $\alpha \in -((\alpha^- \to \alpha) \times \rho[\alpha := \alpha^-])$ which is not true. Moreover,

$$\lambda x^{\rho}.f[\alpha^- := \alpha][h^{\alpha\to\alpha} := \lambda y^{\alpha} y]$$

is a closed term of type $\rho \to \hat{\rho}$. Therefore, we found for $\lambda\alpha\rho$ and m a positivization: A type $\hat{\rho}$ with $\alpha \in \mathrm{Only}_+(\rho)$ such that there are closed terms of types $\hat{\rho} \to \rho$ and $\rho \to \hat{\rho}$. We even have that

$$g[z := f[\alpha^- := \alpha][h^{\alpha\to\alpha} := \lambda y^{\alpha} y]] = g[z := f][\alpha^- := \alpha][h^{\alpha\to\alpha} := \lambda y^{\alpha} y] \twoheadrightarrow^*_e$$
$$m[\alpha^- := \alpha][h^{\alpha\to\alpha} := \lambda y^{\alpha} y].$$

Therefore, if $m[\alpha^- := \alpha][h^{\alpha\to\alpha} := \lambda y^{\alpha} y] \twoheadrightarrow^*_e x$ holds, we have even a retract from $\hat{\rho}$ to ρ. But this cannot be expected without adding the η-conversions: It is easy to see that

$$\mathrm{lift}^p_{\lambda\alpha\rho}[\alpha^- := \alpha][h^{\alpha\to\alpha}, h_-^{\alpha\to\alpha} := \lambda y^{\alpha} y, \lambda y^{\alpha} y] \twoheadrightarrow^*_{\eta} x.$$

Therefore, we could hope for a retract with respect to $\twoheadrightarrow_e \cup \twoheadrightarrow_\eta$. But this seems to hold only in the trivial case of $\alpha \in \mathrm{Only}_+(\rho)$ where we do not need any positivization.

I conjecture that something like

$$\mathrm{lift}^+_{\lambda\alpha\hat{\rho}}[x^{\hat{\rho}[\alpha:=\alpha^-]} := f[\alpha := \alpha^-][h^{\alpha^-\to\alpha^-} := \lambda y^{\alpha^-} y]] \twoheadrightarrow^*_e f$$

holds. In the case of $\alpha \in \mathrm{Only}_+(\rho)$, it can easily be shown by Lemma 4 (without the help of η-reduction): In fact one has to show (for arbitrary ρ and α):

Lemma 6. Two easy properties of liftings:
(i) $\mathrm{lift}^p_{\lambda\alpha\rho}[x^{\rho[\alpha:=\alpha^-]} := \mathrm{lift}^p_{\lambda\alpha\rho}[\alpha := \alpha^-][h^{\alpha^-\to\alpha^-}, h^{\alpha^-\to\alpha^-}_- := \lambda y^{\alpha^-}y, \lambda y^{\alpha^-}y] \twoheadrightarrow^*_e \mathrm{lift}^p_{\lambda\alpha\rho}$.
(ii) $\mathrm{lift}^p_{\lambda\alpha\rho}[\alpha^- := \alpha][h^{\alpha\to\alpha}, h^{\alpha\to\alpha}_- := \lambda y^\alpha y, \lambda y^\alpha y][x^\rho := \mathrm{lift}^p_{\lambda\alpha\rho}] \twoheadrightarrow^*_e \mathrm{lift}^p_{\lambda\alpha\rho}$.

Proof. Routine verification by simultaneous induction on ρ. (Note that $(\pi^e_\times)$ is not needed beyond $(\pi_\times)$.) □

Let us finally study an interesting example of positivization.

Example 2. Let $\rho := (((\alpha\to\beta)\to\alpha)\to\alpha)\to\alpha$. Set

$$m^\rho := \lambda y^{((\alpha\to\beta)\to\alpha)\to\alpha}.y\Big(\lambda z^{\alpha\to\beta}.x\Big(\lambda z_1^{(\alpha^-\to\beta)\to\alpha^-}.z_1 S, u_4^{\alpha^-}.h(u_4, u_5^\alpha.u_5)\Big), u_6^\alpha.u_6\Big)$$

with

$$S := (\lambda z_2^{\alpha^-}.h(z_2, u_1^\alpha.z(u_1, u_2^\beta.u_2)), u_3^{\alpha^-}.u_3)$$

and the only free variables $h : \alpha^- \to \alpha$ and $x : \rho[\alpha := \alpha^-]$. It is an easy exercise to calculate $I'(r, \{x, h\}, \emptyset) = ((\alpha\to\beta)\to\alpha, \ldots)$. The shown type may be seen as the optimal positivization. Thorsten Altenkirch found another one, namely $(((\alpha\to\beta)\to\alpha)\to\beta)\to\alpha$, which can also be found by using a more complicated term m. Since we are not dealing with uniform interpolation [Pit92], the interpolating types may depend on the given monotonicity proof m.

6 Conclusions

In the light of the definitional embedding of the system ΛJ with generalized eliminations into usual λ-calculus, e. g. via $r(s, z.t) \mapsto t[z := rs]$, we arrived at a more fine-grained analysis of Lyndon interpolation even when taking into account the proofs/terms and not only provability/typability. Moreover, the proof of the results on the system with generalized eliminations shows how elegant reasoning becomes by induction on the normal forms obtainable in the system in contrast to those in the standard formulation of λ-calculus resp. derivations in intuitionistic logic.

Finally, a new formulation of liftings has been presented whose behaviour under interpolation could have hardly been studied in standard λ-calculus due to notational burdens already in the definition of the result of interpolation.

The following questions certainly have to be addressed in the future: What is the behaviour of the extended permutation rules and of η-reduction? Is it possible to operationalize η-expansion and prove the usual theorems about it? Is it possible to prove strong normalization of our system by continuation-passing-style transformations in the spirit of [dG99]? Is there a nice categorical semantics of ΛJ which would allow to infer our theorem from the general result in [Čub94]? And, of course, the conjecture on the previous page should be settled since it would allow to reduce iteration and primitive recursion on monotone inductive types to those on positive inductive types without recourse to impredicativity (compare [Mat01]).

References

[Čub94] Djordje Čubrić. Interpolation property for bicartesian closed categories. *Archive for Mathematical Logic*, 33:291–319, 1994.

[dG99] Philippe de Groote. On the strong normalisation of natural deduction with permutation-conversions. In Paliath Narendran and Michaël Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference (RTA '99), Trento, Italy, July 2-4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 45–59. Springer Verlag, 1999.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980.

[JM99] Felix Joachimski and Ralph Matthes. Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T. *Archive for Mathematical Logic*, 1999. Accepted for publication.

[JM00] Felix Joachimski and Ralph Matthes. Standardization and confluence for a lambda calculus with generalized applications. In Leo Bachmair, editor, *Rewriting Techniques and Applications, Proceedings of the 11th International Conference RTA 2000, Norwich, UK*, volume 1833 of *Lecture Notes in Computer Science*, pages 141–155. Springer Verlag, 2000.

[Lei90] Daniel Leivant. Contracting proofs to programs. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, 1990.

[Mat00] Ralph Matthes. Characterizing strongly normalizing terms for a lambda calculus with generalized applications via intersection types. In José D. P. Rolim, Andrei Z. Broder, Andrea Corradini, Roberto Gorrieri, Reiko Heckel, Juraj Hromkovic, Ugo Vaccaro, and Joe B. Wells, editors, *ICALP Workshops 2000, Proceedings of the Satellite Workshops of the 27th International Colloquium on Automata, Languages, and Programming, Geneva, Switzerland*, volume 8 of *Proceedings in Informatics*, pages 339–353. Carleton Scientific, 2000.

[Mat01] Ralph Matthes. Tarski's fixed-point theorem and lambda calculi with monotone inductive types. To appear in Synthese, 2001.

[Pit92] Andrew Pitts. On an interpretation of second order quantification in first order intuitionistic propositional logic. *The Journal of Symbolic Logic*, 57(1):33–52, 1992.

[Pra65] Dag Prawitz. *Natural Deduction. A Proof-Theoretical Study.* Almquist and Wiksell, 1965.

[vP98] Jan von Plato. Natural deduction with general elimination rules. Submitted, 1998.

Implicit Characterizations of *Pspace*

Isabel Oitavem*

Universidade Nova de Lisboa
&
CMAF — Universidade de Lisboa
Av. Prof. Gama Pinto, 2
1649-003 Lisboa
Portugal
Tel. +351-21-7904903, Fax: +351-21-7954288
isarocha@lmc.fc.ul.pt

Abstract. In this paper we focus on the class of functions computable in deterministic polynomial space — *Pspace*. Our aim is to give a survey on implicit characterizations of *Pspace*. Special attention will be given to characterizations where no bounded recursion scheme is invoked and within them to term rewriting characterizations.

Keywords: computational complexity, implicit characterizations, term rewriting, *Pspace*.

1 Introduction

Several machine independent approaches to relevant classes of computational complexity have been developed. All them lead to characterizations of classes of computational complexity where the machine formulation and resources are implicit — *implicit characterizations*. Conceptually, implicit characterizations link computational complexity to levels of definitional and inferential abstractions of significant independent interest.

In this paper we give a survey on implicit characterizations of the class of functions computable in deterministic polynomial space — *Pspace*. Our aim is to describe and to discuss implicit characterizations of *Pspace* with different skills. Namely bounded, input-sorted, term rewriting and unsorted characterizations of *Pspace*. These characterizations, with minor variations, can be found in [15], [16] and [17].

Usually the presence of bounds in the recursion schemes is seen as a negative feature of the characterizations, mainly due to two order of reasons: The presence of bounds often cause difficulties when handling with the classes; Moreover the bounds address, more or less explicitly, to the computational constraints of the characterized classes. With the input-sorted characterizations we solve, if not both, at least the second of these problems. The reason to consider input sorts

* The author would like to thank to *CMAF* and *Fundação para a Ciência e a Tecnologia.*

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 170–190, 2001.

lays on the recursion schemes. Let us consider f defined by primitive recursion as follows: $f(\epsilon, \bar{x}) = g(\bar{x})$ and $f(z', \bar{x}) = h(z, \bar{x}, f(z, \bar{x}))$. Notice that f occurs in both sides of the last equality. Actually, the definition of $f(z', \bar{x})$ uses previous values $f(\cdot, \bar{x})$, which are inputted in the last position of h — *critical input*. By other words, one uses values $f(\cdot, \bar{x})$ before having the object (function) f formed. Therefore, one may have a full knowledge about the objects occurring in a given input, or not. This distinction leads to input-sorted characterizations. It is also possible to consider unsorted systems of function terms and to use purely syntactic measures to control the "impredicativity" of the terms. This approach leads to unsorted characterizations.

Whenever an implicit characterization is established, we are dragging a class of computational complexity to a "less computational" context. In that "less computational" context, we often have a characterization of a class of functions and not any longer a characterization of computational processes. In other words, it might happen that the function definitions of the implicit characterization yield algorithms which does not respect the constraints (time and/or space) of the characterized class of complexity. The term rewriting approach allows us to characterize implicitly classes of computational complexity without dismiss the underlined computational processes. That is achieved by using the rewriting rules to express the appropriate computation strategies. That is for instance the case of [1], where Beckmann and Weiermann describe a term rewriting system for *Ptime* — the class of functions computable in deterministic polynomial time.

The additional problem that we face when approaching *Pspace* is the presence of the (unbounded) primitive recursion scheme. It is easy to understand that the usual formulation of the safe primitive recursion scheme leads us to a term rewriting system where, in the derivations, the length of the terms is not necessarily bounded polynomially by the length of the first term. Notice that the obvious term rewriting rules corresponding to the safe primitive recursion scheme are something of the form

$$PREC[g,h](\epsilon, \bar{x}; \bar{y}) \to g(\bar{x}; \bar{y})$$

$$PREC[g,h](z', \bar{x}; \bar{y}) \to h(z, \bar{x}; \bar{y}, PREC[g,h](z, \bar{x}; \bar{y})),$$

where z' denotes the numeric successor of z. This yields the derivation

$$\begin{aligned} &PREC[g,h](z', \bar{x}; \bar{y}) \to \\ &\cdots \to \\ &h(z, \bar{x}; \bar{y}, h(\cdot, \bar{x}; \bar{y}, h(\cdot, \bar{x}; \bar{y}, h(\cdots(h(\epsilon, \bar{x}; \bar{y}, g(\bar{x}; \bar{y})))\cdots)))), \end{aligned}$$

where the symbol h occurs z' times in the last term. Thus the length of the last term grows exponentially on the length of the first term. Therefore, in this context, a suitable reformulation of the safe primitive recursion scheme is imperative.

2 *Pspace* - An Overview

Within the most studied classes of computational complexity are P and NP — the class of languages decidable in deterministic, respectively non-deterministic, polynomial time. Actually, the question whether P coincides with NP is often pointed as one of the main problems in computational complexity. These two classes, P and NP, are the first two levels of the polynomial hierarchy of time, PH. One usually defines $\Sigma_0^p = P$, $\Sigma_{i+1}^p = NP(\Sigma_i^p)$ and $PH = \cup_{i \in \mathbb{N}} \Sigma_i^p$, where $NP(\Sigma_i^p)$ denotes the class of languages decidable in non-deterministic polynomial time with oracle in Σ_i^p. It is obvious that $\Sigma_i^p \subseteq \Sigma_{i+1}^p$ for all $i \in \mathbb{N}$, but it is still open whether some of these inclusions are proper. By other words, it is unknown whether the polynomial hierarchy of time collapses to some level. The description of *complete* problems for each one of these classes may lead to a better understanding of these classes. Let us consider, for each $i \geq 1$, the $QSAT_i$ problem (quantified satisfiability with i alternations of quantifiers) described in [18] pp.227-228: Given a boolean expression ϕ, with boolean variables partitioned into i sets $X_1, \cdots, X_i$, is it true that for all partial truth assignments for variables in X_1 there is a partial truth assignment for variables in X_2 such that for all partial truth assignments for the variables in X_3, and so on up to X_i, ϕ is satisfied by the overall truth assignment? We represent an instance of $QSAT_i$ as follows (by slightly abusing our first order quantifiers):

$$\exists X_1 \, \forall X_2 \, \exists X_3 \cdots Q X_i \quad \phi$$

where the quantifier Q is $\exists$ if i is odd and $\forall$ if i is even. It is known that, for all $i \geq 1$, $QSAT_i$ is Σ_i^p-complete. Now, let us consider the problem $QSAT$ (also called QBF, for quantified boolean formula) defined like $QSAT_i$ but without any previously defined limitation of the quantifier alternations. Informally speaking $QSAT = \cup_i QSAT_i$. Since $QSAT_i$ is a Σ_i^p-complete problem and $PH = \cup_i \Sigma_i^p$, one could expect $QSAT$ to be a PH-complete problem. However, this is an open problem. In fact, what we know is that $QSAT$ is a complete problem for *Pspace* — the class of languages decidable in deterministic polynomial space. In this sense *Pspace* is the most natural class containing the polynomial hierarchy of time. It is open whether PH coincides with *Pspace* or not. Nevertheless, a positive answer would entail the collapse of the polynomial hierarchy of time to some finite level. For more details see [18].

The question of the power of non-determinism in space is much less dramatic than the same problem in the time domain. For instance, as a consequence of Savich's theorem (1970), we have the collapsing of the polynomial hierarchy of space. Thus, when we restrict ourselves to polynomial space it is irrelevant whether we consider deterministic or non-deterministic computations. Therefore *Pspace*, defined above as the class of languages decidable in deterministic polynomial space, can also be described as the class of languages decidable in non-deterministic polynomial space.

Pspace can also be characterized using parallel computations. It is know that *Pspace* coincides with the class of languages decidable in alternating polynomial time — see, for instance, [18].

In the sequel *Pspace* denotes, not only the class of languages (boolean functions) decidable in polynomial space but, the class of all functions (over $\{0,1\}^*$) computable in polynomial space.

3 Implicit Characterizations of *Pspace*

3.1 Notation

We work with functions defined in cartesian products of $\{0,1\}^*$ and assuming values in $\{0,1\}^*$, where $\{0,1\}^*$ is the set of 0-1 words. Thus we will have in mind the binary tree and all standard notation related with it: $|x|$ for the length of the sequence/word x, ϵ for the sequence of length zero, xy for the concatenation of the sequence x with the sequence y, the "product" $x \times y = x \dots x$ (similar in growth to Samuel Buss' smash function, see [7]) for the concatenation of x with itself $|y|$ times, and x' for the sequence that follows immediately after x when we consider the binary tree ordered according to length and, within the same length, lexicographically. Finally, $x_{|y} = \begin{cases} x \text{ if } |x| \leq |y| \\ z \text{ if } z \subseteq x \wedge |z| = |y| \end{cases}$ for the truncature of x to y, where $z \subseteq x$ abbreviates $\exists y\ zy = x$.

3.2 Bounded Characterization

It is known that *Ptime* (the class of functions computable in deterministic polynomial time) is the smallest class of functions containing the source, binary successor, projection and conditional functions, and that it is closed under the composition and bounded recursion on notation schemes (see [9] or [10]). It is also known that if we close *Ptime* under bounded primitive recursion we will get *Pspace*. This is the case because the number of steps that a *Pspace* machine may carry is exponential on the length of the input. Thus, fixed the binary notation, we have that *Pspace* is the smallest class of functions containing the initial functions 1-4 and that is closed under the composition, bounded recursion on notation and bounded primitive recursion:

1)	ϵ	(source)
2)	$S_i(x) = xi\ ,\ i \in \{0,1\}$	(binary successors)
3)	$\Pi^n_j(x_1, \dots, x_n) = x_j\ ,\ 1 \leq j \leq n$	(projections)
4)	$C(\epsilon, y, z_0, z_1) = y$, $C(xi, y, z_0, z_1) = z_i$, $i \in \{0,1\}$	(conditional)

Composition: $f(\bar{x}) = g(\bar{h}(\bar{x}))$

Bounded primitive recursion (exhaustive):
$f(\epsilon, \bar{x}) = g(\bar{x})$
$f(y', \bar{x}) = h(y, \bar{x}, f(y, \bar{x}))_{|t(y,\bar{x})}$

Bounded recursion on notation (over the branches):
$f(\epsilon, \bar{x}) = g(\bar{x})$
$f(yi, \bar{x}) = h_i(y, \bar{x}, f(y, \bar{x}))_{|t(y,\bar{x})}\ ,\ i \in \{0,1\}$

where t is a bounding function, i.e., is a function of the smallest class of functions containing the projection functions and the concatenation and binary product functions and which is closed under composition and assignment of values to variables.

3.3 Two-Input-Sorted Characterization

In 1995, following techniques introduced by Bellantoni and Cook in [4] we established an two-input-sorted characterization of *Pspace* where no bounded scheme is invoked, see [14] or [15]. As usually, we use a semi-colon to separate different sorts of inputs. We consider two sorts of inputs — *normal* and *safe* — and we write them by this order. The mentioned characterization, which is here denoted by *2-Pspace*, uses sorted versions of the composition and recursion schemes, but it also requires the introduction of some additional initial functions. We say that *2-Pspace* is the smallest class of functions containing the initial functions 1-8 and which is closed under the safe composition, the safe primitive recursion and the safe recursion on notation schemes:

1)	ϵ	(source)
2)	$S_i(x;) = xi$, $i \in \{0,1\}$	(binary successors)
3)	$\Pi_j^{n;m}(x_1,\ldots,x_n;x_{n+1},\ldots,x_{n+m}) = x_j$, $1 \leq j \leq n+m$	(projections)
4)	$B(x,\epsilon;z) = x$, $B(x,yi;\epsilon) = x1$, $B(x,yi;zj) = B(xj,y;z)$, $i,j \in \{0,1\}$	(B-transition)
5)	$P(;\epsilon) = \epsilon$, $P(;xi) = x$, $i \in \{0,1\}$	(binary predecessor)
6)	$p(;\epsilon) = \epsilon$, $p(;x') = x$	(numeric predecessor)
7)	$C(;\epsilon,y,z_0,z_1) = y$, $C(;xi,y,z_0,z_1) = z_i$, $i \in \{0,1\}$	(conditional)
8)	$\hat{\times}(\epsilon,x;) = \epsilon$, $\hat{\times}(yi,x;) = B(\hat{\times}(y,x;),x;x)$, $i \in \{0,1\}$	(modified (binary) product)

Safe composition: $f(\bar{x};\bar{y}) = g(\bar{r}(\bar{x};);\bar{s}(\bar{x};\bar{y}))$

Safe primitive recursion:
$$f(\epsilon,\bar{x};\bar{y}) = g(\bar{x};\bar{y})$$
$$f(z',\bar{x};\bar{y}) = h(z,\bar{x};\bar{y},f(z,\bar{x};\bar{y}))$$

Safe recursion on notation:
$$f(\epsilon,\bar{x};\bar{y}) = g(\bar{x};\bar{y})$$
$$f(zi,\bar{x};\bar{y}) = h_i(z,\bar{x};\bar{y},f(z,\bar{x};\bar{y})),\ i \in \{0,1\}$$

In the safe composition scheme the absence of some of the functions $\bar{r},\bar{s}$ is allowed.

This class *2-Pspace* characterizes *Pspace* in the sense that:

$$f(\bar{x};) \in \textit{2-Pspace} \Leftrightarrow f(\bar{x}) \in \textit{Pspace},$$

or equivalently, by dismissing the input sorts of the functions in *2-Pspace* one gets exactly the *Pspace* functions.

This characterization of *Pspace* is the analogue of the characterization of *Ptime* given by Bellantoni and Cook, see [4], [2] or [3]. In this vein the formulation of the safe composition, safe primitive recursion and safe recursion on notation schemes is quite natural. The asymmetry of safe composition scheme allows us "to move" variables from safe positions to normal positions, but not the opposite. In other words, whenever $f(\bar{x};\bar{y}) \in$ *2-Pspace*, the safe composition scheme allows us to define $F(\bar{x},\bar{y};) = f(\bar{x};\bar{y})$, but not $F(;\bar{x},\bar{y}) = f(\bar{x};\bar{y})$. Regarding the recursion schemes, we have the expected separations between the positions occupied by the recursion variable and by the values obtained recursively. The most delicate part is then to fix a set of functions to start with. In the formulation of the initial functions there are two crucial steps. Firstly, one should notice that we must have the binary successor functions, but we cannot have them over safe inputs. In other words, we must have $S_i(x;) = xi$, but we cannot allow the functions $S_i(;x) = xi$, $i \in \{0,1\}$, to be in our class. In fact, any function which increases the length of a safe input by one unit, together with the safe primitive recursion scheme, leads us out of *Pspace*. For instance, if $S_1(;x) = x1$ was in our class then the function $f(\epsilon;) = 1$ and $f(x';) = S_1(;f(x;))$ would be there as well. Such function f has exponential growth and so it is obviously not in *Pspace*. Secondly, it turns out that to have just $S_i(x;)$ is not enough — see proof of lemma 3.1 in [15]. In fact we cannot have $S_i(;x)$, but we need to have some "interesting" function involving safe inputs. This "interesting" function could, for instance, be the normal-bounded safe-binary-successor: $S_i(z;x) = \begin{cases} xi \text{ if } |x| < |z| \\ x \quad \text{otherwise} \end{cases}$, $i \in \{0,1\}$. Although, here we consider the function B described above. B enables us to carry bits from safe positions into normal positions. However, the number of bits that can be carried is dominated by the length of some normal input. Actually, $B(x,y;z) = x\hat{}(z^{-1}1)_{|y}$, where $\hat{}$ stands for concatenation, z^{-1} is the sequence z written by the reverse order, and $w_{|y}$ denotes the first $|y|$ bits of w. Finally, we would like to emphasize that we included the modified product function among the initial functions, but we could had chosen the usual binary product function instead. Notice that the modified product $\hat{\times}(y,x;)$ leads to $x^{-1}\hat{}\cdots\hat{}x^{-1}$, $|y|$-times, and the usual binary product leads to $x\hat{}\cdots\hat{}x$, $|y|$-times. What really matters is to have a function behaving like the product function, i.e. verifying $|\times(y,x;)| = |x|\cdot|y|$. The modified product is a function satisfying this property and having an easy formulation in present context.

One of the main properties of the class *2-Pspace* is expressed in the following bounding lemma:

Lemma 1. *If $f \in$ 2-Pspace then there exists a polynomial q_f such that*

$$(*) \quad \forall \bar{x},\bar{y}\ |f(\bar{x};\bar{y})| \leq \max\{q_f(|\bar{x}|)\,,\ \max_i |y_i|\}.$$

Proof. The proof is by induction on the complexity of the function definitions. The initial functions pose no problems.

If f is defined by the safe composition scheme, then we have $q_g, \bar{q_r}, \bar{q_s}$ satisfying $(*)$ and, therefore, we may take,

$$q_f(|\bar{x}|) = q_g(q_r(|\bar{x}|)) + \sum_i q_{s_i}(|\bar{x}|).$$

If f is defined by safe primitive recursion then, considering $q_f(|z|, |\bar{x}|) = q_h(|z|, |\bar{x}|) + q_g(|\bar{x}|)$, the result follows by induction on the recursion variable, since q_g, q_h verify $(*)$. We have $|f(\epsilon, \bar{x}; \bar{y})| \leq \max\{q_f(|\epsilon|, |\bar{x}|), \max_i |y_i|\}$ and, since $|f(u, \bar{x}; \bar{y})| \leq \max\{q_f(|u|, |\bar{x}|), \max_i |y_i|\}$, we get,

$$\begin{aligned}
|f(u', \bar{x}; \bar{y})| &= |h(u, \bar{x}; \bar{y}, f(u, \bar{x}; \bar{y}))| \\
&\leq \max\{q_h(|u|, |\bar{x}|), \max\{\max_i |y_i|, |f(u, \bar{x}; \bar{y})|\}\} \\
&\leq \max\{q_h(|u|, |\bar{x}|), \max\{\max_i |y_i|, q_h(|u|, |\bar{x}|) + q_g(|\bar{x}|)\}\} \\
&\leq \max\{q_h(|u|, |\bar{x}|) + q_g(|\bar{x}|), \max_i |y_i|\} \\
&\leq \max\{q_h(|u'|, |\bar{x}|) + q_g(|\bar{x}|), \max_i |y_i|\} \\
&\leq \max\{q_f(|u'|, |\bar{x}|), \max_i |y_i|\}
\end{aligned}$$

If f is defined by safe recursion on notation then, if we set $q_h \equiv q_{h_0} + q_{h_1}$ and $q_f(|z|, |\bar{x}|) = q_h(|z|, |\bar{x}|) + q_g(|\bar{x}|)$, we may carry on the result by induction on the length of the recursion variable, since q_g, q_{h_0}, q_{h_1} satisfy $(*)$. Thus,

$$\begin{aligned}
|f(\epsilon, \bar{x}; \bar{y})| &= |g(\bar{x}; \bar{y})| \\
&\leq \max\{q_g(|\bar{x}|), \max_i |y_i|\} \\
&= \max\{q_f(|\epsilon|, |\bar{x}|), \max_i |y_i|\}
\end{aligned}$$

Now, since $|f(u, \bar{x}; \bar{y})| \leq \max\{q_f(|u|, |\bar{x}|), \max_i |y_i|\}$, we get

$$\begin{aligned}
|f(ui, \bar{x}; \bar{y})| &= |h_i(u, \bar{x}; \bar{y}, f(u, \bar{x}; \bar{y}))| \\
&\leq \max\{q_h(|u|, |\bar{x}|), \max\{\max_i |y_i|, |f(u, \bar{x}; \bar{y})|\}\} \\
&\leq \max\{q_h(|u|, |\bar{x}|), \max\{\max_i |y_i|, \max\{q_f(|u|, |\bar{x}|), \max_i |y_i|\}\}\} \\
&= \max\{q_h(|u|, |\bar{x}|), \max\{q_f(|u|, |\bar{x}|), \max_i |y_i|\}\} \\
&= \max\{q_h(|u|, |\bar{x}|), \max\{q_h(|u|, |\bar{x}|) + q_g(|\bar{x}|), \max_i |y_i|\}\} \\
&\leq \max\{q_h(|u|, |\bar{x}|) + q_g(|\bar{x}|), \max_i |y_i|\} \\
&\leq \max\{q_h(|ui|, |\bar{x}|) + q_g(|\bar{x}|), \max_i |y_i|\} \\
&\leq \max\{q_f(|ui|, |\bar{x}|), \max_i |y_i|\}
\end{aligned}$$

Therefore, $\forall \bar{x}, \bar{y}\ |f(\bar{x}; \bar{y})| \leq \max\{q_f(|\bar{x}|), \max_i |y_i|\}$.

The bound that we have just established does not hold for the characterization of *Ptime* given by Bellantoni and Cook in [4] — $Ptime_{BC}$. Actually, in $Ptime_{BC}$, we just have that: If $f \in Ptime_{BC}$ then there exists a polynomial q_f such that

$$\forall \bar{x}, \bar{y}\ |f(\bar{x}; \bar{y})| \leq q_f(|\bar{x}|) + \max_i |y_i|.$$

Whenever the goal is just to dominate the length of the outputs polynomially on the length of the inputs, it is obviously irrelevant which one of these bounds holds. Nevertheless, for the term rewriting approach it is going to be crucial to have this more accurate bound in *2-Pspace* — see the proof of theorem 1, in the term rewriting section.

3.4 Three-Input-Sorted Characterization

In the previous section one described a characterization of *Pspace* without bounded recursion schemes. However, one might object to the inclusion of the modified product (or product) function among the initial functions of *2-Pspace*. One might prefer to have only initial functions of linear growth. As observed before we must be very careful about operations involving safe positions because — having safe primitive recursion involving safe positions — if they increase the safe input lengths even just one bit, we would get functions of exponential growth, which lie outside *Pspace*. Therefore, in order to remove the modified product from the initial functions we seem to have to introduce an intermediate input position, say *semi-safe*, and use it to construct the modified product. Thus, if we start with simpler initial functions we will arrive at a more elaborate characterization. Let us give a glance over this alternative characterization of *Pspace*. Here, there are three kinds of input positions in the functions: *normal*, *semi-safe* and *safe*. We write the normal, semi-safe and safe inputs by this order and separate them by semicolons. We say that *3-Pspace* is the smallest class of functions containing the following initial functions 1-7 and which is closed under the safe composition, the safe primitive recursion and the double recursion on notation schemes:

1)	ϵ	(source)
2)	$S_i(;x;) = xi$, $i \in \{0,1\}$	(binary successors)
3)	$\Pi_j^{n;m;l}(x_1,\ldots,x_n;x_{n+1},\ldots,x_{n+m};x_{n+m+1},\ldots,x_{n+m+l}) = x_j$, $1 \leq j \leq n+m+l$	(projections)
4)	$B(x,\epsilon;;z) = x$, $B(x,yi;;\epsilon) = x1$, $B(x,yi;;zj) = B(xj,y;;z)$, $i,j \in \{0,1\}$	(B-transition)
5)	$P(;;\epsilon) = \epsilon$, $P(;;xi) = x$, $i \in \{0,1\}$	(binary predecessor)
6)	$p(;;\epsilon) = \epsilon$, $p(;;x') = x$	(numeric predecessor)
7)	$C(;;\epsilon,y,z_0,z_1) = y$, $C(;;xi,y,z_0,z_1) = z_i$, $i \in \{0,1\}$	(conditional)

Safe composition: $f(\bar{x};\bar{y};\bar{z}) = g(\bar{s}(\bar{x};;);\bar{r}(\bar{x};\bar{y};);\bar{t}(\bar{x};\bar{y};\bar{z}))$

Safe primitive recursion: $f(\epsilon,\bar{x};\bar{y};\bar{z}) = g(\bar{x};\bar{y};\bar{z})$
$f(w',\bar{x};\bar{y};\bar{z}) = h(w,\bar{x};\bar{y};\bar{z},f(w,\bar{x};\bar{y};\bar{z}))$

Double recursion on notation:
$f(\epsilon,\bar{x};\bar{y};\bar{z}) = g(\bar{x};\bar{y};\bar{z})$
$f(wi,\bar{x};\bar{y};\bar{z}) = h_i(w,\bar{x};\bar{y},f(w,\bar{x};\bar{y};\bar{\epsilon});\bar{z},f(w,\bar{x};\bar{y};\bar{z}))$, $i \in \{0,1\}$

The goal of the double recursion on notation scheme is to join two schemes in one. However, it could be replaced by the two following schemes:

Semi-safe recursion on notation:
$f(\epsilon, \bar{x}; \bar{y};) = g(\bar{x}; \bar{y};)$
$f(wi, \bar{x}; \bar{y};) = h_i(w, \bar{x}; \bar{y}, f(w, \bar{x}; \bar{y};);)$, $i \in \{0, 1\}$
Safe recursion on notation:
$f(\epsilon, \bar{x}; \bar{y}; \bar{z}) = g(\bar{x}; \bar{y}; \bar{z})$
$f(wi, \bar{x}; \bar{y}; \bar{z}) = h_i(w, \bar{x}; \bar{y}; \bar{z}, f(w, \bar{x}; \bar{y}; \bar{z}))$, $i \in \{0, 1\}$

Here the bounding lemma has the following formulation:

Lemma 2. *If $f \in$ 3-Pspace then there exists a polynomial q_f such that*

$$\forall \bar{x}, \bar{y}, \bar{z} \ |f(\bar{x}; \bar{y}; \bar{z})| \leq \max\{q_f(|\bar{x}|) + \max_i |y_i| \, , \, \max_i |z_i|\}.$$

For details see [14].

The three-input-sorted characterization of *Pspace* described here will be invoked later on. Actually, it is crucial to establish the unsorted characterization of *Pspace* given in section 3.6.

3.5 Term Rewriting Characterization

The characterization of *Pspace* presented here can be found in [17] and it is a minor variation of the characterization given in [16].

Our starting point is the two-input-sorted characterization of *Pspace* described above. An appropriate variation of this characterization yields naturally a class, PS, of function symbols. Among these symbols, we distinguish three — one source and two (binary) constructors — which are used to define the numerals. We introduce a set of rewriting rules in such a way that terms with no variables are numerals or can be rewritten as such. Henceforth, our main goal is to bound polynomially the length of the terms that can occur in the rewriting chains.

The crucial steps are the reformulation of the two-input-sorted characterization of *Pspace* given above and the reasoning we carry out in order to keep the length of terms occurring in the rewriting chains polynomially bounded.

Pspace

For the present purposes it is convenient to consider another alternative characterization of *Pspace*, Ps. It results from *2-Pspace* by joining the numeric successor function, $s(x;) = x'$, and the functions $E^{k;l}(x_1, \cdots, x_k; x_{k+1}, \cdots, x_{k+l}) = \epsilon$, $k \geq 0$ and $l \geq 0$ to the initial functions. Notice that $E^{0;0}$ designates the source function ϵ. Moreover, we consider the following generalization of the recursion schemes:

- $f(\epsilon, b, \bar{m}; \bar{n}, a) = a$
  $f(mi, b, \bar{m}; \bar{n}, a) = f(m, bi, \bar{m}; \bar{n}, h_i(b, \bar{m}; \bar{n}, a))$, $i \in \{0, 1\}$

– $f(\epsilon, b, \bar{m}; \bar{n}, a) = a$
  $f(m', b, \bar{m}; \bar{n}, a) = f(m, b', \bar{m}; \bar{n}, h(b, \bar{m}; \bar{n}, a))$

Notice that by initializing these schemes with m being the recursion input (written by the reverse order in case of the generalized safe recursion on notation scheme), $b = \epsilon$ and $a = g(\bar{m}; \bar{n})$ one obtains the functions defined by safe recursion on notation and the safe recursion schemes. The goal of these schemes is to reverse the usual order of the recursion process. That enables us to avoid nested occurrences of the recursion function during the computation. For instance, if f is defined by safe recursion on notation, $f(01;)$ leads to $f(01;) \to h_1(0; f(0;)) \to h_1(0; h_0(\epsilon; f(\epsilon;))) \to h_1(0; h_0(\epsilon; g))$. If f is defined by generalized safe recursion on notation one may have $f(10, \epsilon; g) \to f(1, 0; h_0(\epsilon; g)) \to f(\epsilon, 01; h_1(0; h_0(\epsilon; g))) \to h_1(0; h_0(\epsilon; g))$, but one may also have $f(10, \epsilon; g) \to f(1, 0; h_0(\epsilon; g)) \to \cdots \to f(1, 0; a_1) \to f(\epsilon, 01; h_1(0; a_1)) \to \cdots \to f(\epsilon, 01; a_2) \to a_2$, where $h_0(\epsilon; g) = a_1$ and $h_1(0; a_1) = a_2$. This last chain results from giving to the third component of f the highest rewriting priority and it avoids nested occurrences of h_i. In conformity with this let us describe the class Ps of the number-theoretic functions computable in polynomial space.

Definition 1. *Ps is the smallest class of number-theoretic functions containing: S_i, $i \in \{0, 1\}$; B; $E^{k;l}$; $\Pi^{k;l}_j$, $1 \leq j \leq k + l$; P; p; s; C; $\hat{\times}$ and*

$$\begin{array}{ll} SUB[f, \bar{g}, \bar{h}], & \text{if } f, \bar{g}, \bar{h} \in Ps; \\ PRECN[h_0, h_1], & \text{if } h_0, h_1 \in Ps; \\ PREC[h], & \text{if } h \in Ps, \end{array}$$

where $SUB[f, \bar{g}, \bar{h}]$ denotes the unique number-theoretic function which maps $(\bar{m}, \bar{n})$ to $f(\bar{g}(\bar{m};); \bar{h}(\bar{m}; \bar{n}))$ while $PRECN[h_0, h_1]$ and $PREC[h]$ denote the unique number-theoretic functions, f and g respectively, defined as follows:

$$\begin{array}{l} f(\epsilon, b, \bar{m}; \bar{n}, a) = a \\ f(mi, b, \bar{m}; \bar{n}, a) = f(m, bi, \bar{m}; \bar{n}, h_i(b, \bar{m}; \bar{n}, a)),\ i \in \{0, 1\} \\ g(\epsilon, b, \bar{m}; \bar{n}, a) = a \\ g(m', b, \bar{m}; \bar{n}, a) = g(m, b', \bar{m}; \bar{n}, h(b, \bar{m}; \bar{n}, a)) \end{array}$$

In [16] we prove that $Ps = 2\text{-}Pspace$. This equality together with lemma 1 leads to the following bound:

Corollary 1. *If $f \in Ps$ then there exists a polynomial q_f such that*

$$\forall \bar{x}, \bar{y}\ |f(\bar{x}; \bar{y})| \leq \max\{q_f(|\bar{x}|), \max_i |y_i|\}.$$

We associate to the class of number-theoretic functions, Ps, a class of function symbols.

Definition 2. *PS is the class of function symbols defined as follows: $S_i \in PS$, for any $i \in \{0, 1\}$; $B \in PS$; $E^{k;l} \in PS$, for $k \geq 0$ and $l \geq 0$;*

$\Pi_j^{k;l} \in PS$, *for any* $1 \leq j \leq k+l$; $P \in PS$; $p \in PS$; $s \in PS$; $C \in PS$; $\hat{\times} \in PS$ *and*

$$\begin{array}{ll} SUB[f,\bar{g},\bar{h}] \in PS, & \text{if } f,\bar{g},\bar{h} \in PS; \\ PRECN[h_0,h_1] \in PS, & \text{if } h_0,h_1 \in PS; \\ PREC[h] \in PS, & \text{if } h \in PS. \end{array}$$

Notice that the standard interpretation of the function symbols S_i, B, $E^{k;l}$, $\Pi_j^{k;l}$, P, p, s, C, $\hat{\times}$, $SUB[f,\bar{g},\bar{h}]$, $PRECN[h_0,h_1]$, and $PREC[h]$ (where $f,\bar{g},\bar{h},h_0,h_1,h \in PS$ are function symbols) is the number-theoretic functions S_i, B, $E^{k;l}$, $\Pi_j^{k;l}$, P, p, s, C, $\hat{\times}$, $SUB[f,\bar{g},\bar{h}]$, $PRECN[h_0,h_1]$, and $PREC[h]$ (where $f,\bar{g},\bar{h},h_0,h_1,h \in Ps$ are the correspondent number-theoretic functions), respectively. This correspondence from PS (class of function symbols) into Ps (class of number-theoretic functions) is denoted by π. Our notation does not make any distinction between elements of Ps and elements of PS. For instance $\pi(P) = P$, where the first P belongs to PS and the second one belongs to Ps. However, it will always be clear, from the context, whether we are dealing with number-theoretic functions or with function symbols.

We define inductively the *length* of the function symbols:

Definition 3. *For* $f^* \in PS$ *we define the length of* f^*, $lh(f^*)$, *as follows:*

1. $lh(f^*) = 1$ *if* f^* *is* S_i ,B, $E^{k;l}$, $\Pi_j^{k;l}$, P, p, s, C *or* $\hat{\times}$;
2. *If* $f^* = SUB[f,\bar{g},\bar{h}]$, *for some function symbols* $f,\bar{g},\bar{h} \in PS$, *then* $lh(f^*) = lh(f) + \sum_i lh(g_i) + \sum_i lh(h_i) + 1$;
3. *If* $f^* = PRECN[h_0,h_1]$, *for some function symbols* $h_0,h_1 \in PS$, *then* $lh(f^*) = lh(h_0) + lh(h_1) + 1$;
4. *If* $f^* = PREC[h]$, *for some function symbol* $h \in PS$, *then* $lh(f^*) = lh(h)+1$.

Characterizing *Pspace*

Every sequence in $\{0,1\}^*$ can be built up from the empty sequence by successive concatenations with the bits 0 and 1. Since that $S_i(n;)$ represents the concatenation of the sequence n with the sequence i, $i \in \{0,1\}$, we define for every sequence n a numeral $\underline{n}$ as follows: $\underline{\epsilon} :\equiv E^{0;0}$ and $\underline{S_i(n;)} :\equiv S_i(\underline{n};)$, $i \in \{0,1\}$. For simplicity of notation, we usually write ϵ instead of $E^{0;0}$.

Let X be a countable infinite set of variables, disjoint from PS, we define the set of (first-order) *terms* over PS, $T(PS)$, as the smallest set containing X which satisfies the following closure property: if $f \in PS$ and $\bar{r},\bar{s} \in T(PS)$ then $f(\bar{r};\bar{s}) \in T(PS)$, where $\bar{r},\bar{s}$ are tuples of appropriate arity. We say that $t \in T(PS)$ is a *ground term* if no element of X occurs in t, and we denote the set of all ground terms by $G(PS)$.

Observe that all the numerals are ground terms and every ground term, $t \in G(PS)$, has the shape $f(\bar{r};\bar{s})$ for some $f \in PS$ and $\bar{r},\bar{s} \in G(PS)$ (for $t = \underline{\epsilon}$, f is $E^{0;0}$ and no $\bar{r},\bar{s}$ occurs). We define the *length* of a ground term $f(\bar{r};\bar{s})$ in $G(PS)$, $lh(f(\bar{r};\bar{s}))$, as being $lh(f) + \sum_i lh(r_i) + \sum_i lh(s_i)$. Extending π over $G(PS)$ according to the equation $\pi(f(\bar{r};\bar{s})) = \pi(f)(\pi(\bar{r});\pi(\bar{s}))$, where $f \in PS$

and $\bar{r}, \bar{s} \in G(PS)$, one has $\pi(\underline{n}) = n$ for any numeral $\underline{n}$. It is straightforward to prove that:

Lemma 3. *For all numeral* $\underline{n}$, $lh(\underline{n}) = |\pi(\underline{n})| + 1$.

Moreover, one has the following statement which is just a reformulation of corollary 1:

Lemma 4. *For all* $f \in PS$ *there exists a polynomial* q_f *with natural coefficients such that* $\forall \bar{r}, \bar{s} \in G(PS)\ |\pi(f(\bar{r}; \bar{s}))| \leq \max\{q_f(|\pi(\bar{r})|), \max_i |\pi(s_i)|\}$.

Considering in $\mathbb{N}$ and $\{0,1\}^*$ the usual less than or equal relations, both denoted by $\leq$, let $\beta : \{0,1\}^* \to \mathbb{N}$ be the bijection that respects these relations.

Definition 4. *Let* R *be the following set of term rewriting rules:*

1. $B(x, \epsilon; z) \to x$
2. $B(x, S_i(y;); \epsilon) \to S_1(x;),\ i \in \{0,1\}$
3. $B(x, S_i(y;); S_j(z;)) \to B(S_j(x;), y; z),\ i, j \in \{0,1\}$
4. $E^{k;l}(x_1, \cdots, x_k; x_{k+1}, \cdots, x_{k+l}) \to \epsilon$
5. $\Pi_j^{k;l}(x_1, \cdots, x_k; x_{k+1}, \cdots, x_{k+l}) \to x_j,\ 1 \leq j \leq k+l$
6. $P(; \epsilon) \to \epsilon$
7. $P(; S_i(x;)) \to x,\ i \in \{0,1\}$
8. $p(; \epsilon) \to \epsilon$
9. $p(; S_0(\epsilon;)) \to \epsilon$
10. $p(; S_0(S_i(x;);)) \to S_1(p(; S_i(x;));),\ i \in \{0,1\}$
11. $p(; S_1(x;)) \to S_0(x;)$
12. $s(\epsilon;) \to S_0(\epsilon;)$
13. $s(S_0(x;);) \to S_1(x;)$
14. $s(S_1(x;);) \to S_0(s(x;);)$
15. $C(; \epsilon, y, z, w) \to y$
16. $C(; S_0(x;), y, z, w) \to z$
17. $C(; S_1(x;), y, z, w) \to w$
18. $\hat{\times}(x, \epsilon;) \to \epsilon$
19. $\hat{\times}(x, S_i(y;);) \to B(\hat{\times}(x, y;), x; x),\ i \in \{0,1\}$
20. $SUB[f, \bar{g}, \bar{h}](\bar{x}; \bar{y}) \to f(\bar{g}(\bar{x};); \bar{h}(\bar{x}; \bar{y}))$
21. $PRECN[h_1, h_2](\epsilon, b, \bar{x}; \bar{y}, \underline{a}) \to \underline{a}$
22. $PRECN[h_1, h_2](S_i(z;), b, \bar{x}; \bar{y}, \underline{a}) \to$
    $PRECN[h_1, h_2](z, S_i(b;), \bar{x}; \bar{y}, h_i(b, \bar{x}; \bar{y}, \underline{a})),\ i \in \{0,1\}$
23. $PREC[h](\epsilon, b, \bar{x}; \bar{y}, \underline{a}) \to \underline{a}$
24. $PREC[h](S_i(\underline{m};), \underline{b}, \bar{x}; \bar{y}, \underline{a}) \to$
    $PREC[h](p(; S_i(\underline{m};)), s(\underline{b};), \bar{x}; \bar{y}, h(\underline{b}, \bar{x}; \bar{y}, \underline{a})),\ i \in \{0,1\}$

Thus, for each function symbol in $PS \setminus \{E^{0;0}, S_0, S_1\}$ one has in R very simple rewriting rules.

The term rewriting system R_{PS} is terminating and confluent on ground terms — see [17] or [16] for a proof.

Here we are going to focus on proving that for all $f \in PS$ there exists a polynomial, p_f, such that, for all numerals $\underline{\bar{m}}, \underline{\bar{n}}$, $p_f(lh(f(\underline{\bar{m}}; \underline{\bar{n}})))$ is an upper bound for the length of any term occurring in any chain $f(\underline{\bar{m}}; \underline{\bar{n}}) \to \cdots$.

Notice that underlined inputs denote numerals. Therefore, the rules 21 up to 24 can be applied only if certain components were previously rewritten as numerals. This restricts the possible reduction strategies. For instance, requiring the second component of $PREC[h](\cdot, \cdot, \cdot; \cdot, \cdot)$ to be rewritten as numerals before applying rule 24, we avoid the chain

$$PREC[h](\underline{m}, b, \cdot; \cdot, \cdot) \to \cdots \to PREC[h](\epsilon, s(s(\cdots(s(b;)\cdots;), \cdot; \cdot, \cdot).$$

Since that the length of $s(s(\cdots(s(b;)\cdots;)$ grows exponentially on the length of the first term, such a reduction strategy would make our goal impossible to achieve.

Let us start establishing some useful lemmas.

Lemma 5. *The rewriting rules 8-14 have the shape $t_1 \to t_2$ with $lh(t_2) \leq lh(t_1)$, for any assignment of ground terms to variables.*

This lemma follows directly from the observation of the mentioned rewriting rules.

Lemma 6. *Every chain $t_1 \to t_2 \to \cdots$ with $t_1 = p(; \underline{m})$, for some numeral $\underline{m}$, terminates and its last term, let us say t_*, is a numeral satisfying $lh(t_*) \leq lh(\underline{m})$.*

Proof. The proof follows by induction on $lh(\underline{m})$. $lh(\underline{m}) = 1$ implies that $\underline{m} = \epsilon$ and so the only possible chain respecting the rewriting system rules is $p(; \epsilon) \to \epsilon$. Therefore, the required condition is trivially verified.

Given $k \in \mathbb{N}$, let us assume that the result is valid for any numeral $\underline{m}$ such that $lh(\underline{m}) = k$. Any numeral with length $k+1$ has the shape $S_i(\underline{n};)$, where $i \in \{0, 1\}$ and $\underline{n}$ is a numeral with length k. If $i = 1$ or $\underline{n} = \epsilon$ the statement is immediate; Otherwise one has $p(; S_0(\underline{n};)) \to S_1(p(; \underline{n});) \to \cdots \to S_1(\underline{p(; \underline{n})};) = t_*$, where $\underline{p(; \underline{n})}$ denotes the numeral associated with the sequence represented by $\pi(p)(; n)$. By induction hypothesis, it follows that t_* is a numeral satisfying $lh(t_*) = lh(S_1) + lh(\underline{p(; \underline{n})}) \leq lh(S_1) + lh(\underline{n}) = lh(S_0(\underline{n};))$.

Lemma 7. *$\forall k \leq \beta(\pi(\underline{m}))$ $lh(\underline{s^{(k)}(\underline{b};)}) \leq lh(\underline{b}) + lh(\underline{m})$, for any numerals $\underline{b}, \underline{m}$, where $s^{(0)}(\underline{b};) = \underline{b}$, $s^{(1)}(\underline{b};) = \underline{s(\underline{b};)}$ and $s^{(k+1)}(\underline{b};) = s(s^{(k)}(\underline{b};);)$.*

Proof. This lemma is a trivial consequence of the following four facts, which can be easily proved.

1. $\pi(\underline{m}) \geq \pi(\underline{n}) \Rightarrow \pi(s^{(\beta(\pi(\underline{m})))}(\underline{b};)) \geq \pi(s^{(\beta(\pi(\underline{n})))}(\underline{b};))$;
2. $\pi(\underline{m}) \geq \pi(\underline{n}) \Rightarrow lh(\underline{m}) \geq lh(\underline{n})$;
3. $lh(\underline{s^{(\beta(\pi(\underline{m})))}(\underline{b};)}) \leq lh(\underline{B(\underline{b}, \underline{m}; \underline{m})})$;
4. $lh(\underline{B(\underline{b}, \underline{m}; \underline{m})}) \leq lh(\underline{b}) + lh(\underline{m})$.

Theorem 1. *Given $f \in PS$ there exists a polynomial p_f, with natural coefficients, such that, for any $t_1 = f(\bar{\underline{m}}; \bar{\underline{n}})$ where $\underline{m}, \underline{n}$ are numerals, all chains of ground terms $t_1 \to t_2 \to \cdots \to t_n$ ($n \in \mathbb{N}$) satisfy $lh(t_i) \leq p_f(lh(t_1))$ for all $2 \leq i \leq n$.*

Proof. The proof is by induction on the complexity of f. The interesting cases occur when f is $PRECN[h_0, h_1]$ or $PREC[h]$ for some $h_0, h_1, h \in PS$. We are going to prove only the last case. The other one follows the same reasoning.

Let us assume that $t_1 = PREC[h](\underline{t_{11}}, \underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})$, where $\underline{t_{11}} = S_i(\underline{m};)$ for some numeral $\underline{m}$, i equal to 0 or 1, and $\underline{t_{12}}, \underline{t_{13}}, \underline{t_{14}}, \underline{t_{15}}$ are numerals (the case where $\underline{t_{13}}$ and $\underline{t_{14}}$ are tuples of numerals is similar). By induction assumption, there exists a polynomial p_h such that, given any chain of ground terms

$$h(\bar{\underline{m}}; \bar{\underline{n}}) \to t_{h2} \to \cdots \to t_{hn_h}$$

where $\bar{\underline{m}}, \bar{\underline{n}}$ are numerals, one has for all $2 \leq k \leq n_h$

$$lh(t_{hk}) \leq p_h(lh(h(\bar{\underline{m}}; \bar{\underline{n}}))).$$

Since that any chain beginning with t_1 is contained in some chain of the shape

$$PREC[h](\underline{t_{11}}, \underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}}) \to \cdots \to PREC[h](\underline{t_{l1}}, \underline{t_{l2}}, \underline{t_{l3}}; \underline{t_{l4}}, \underline{t_{l5}}) \to \underline{t_{l5}}$$

for some $l \in \mathbb{N}$, let us prove that for all n, $1 \leq n \leq l$, one has:

1. $lh(t_{n1}) \leq lh(\underline{t_{11}}) + 1$;
2. $lh(t_{n2}) \leq lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + 1$;
3. $lh(t_{n5}) \leq p_h(\overline{lh}(h) + \overline{lh}(\underline{t_{11}}) + lh(\underline{t_{12}}) + 1 + lh(\underline{t_{13}}) + q_h(lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + lh(\underline{t_{13}})) + 2 \cdot lh(\underline{t_{14}}) + lh(\underline{t_{15}}))$.

Notice that in the most difficult case, which results from rule 24, one has that $PREC[h](S_i(\underline{m};), \cdots) \to_{24} PREC[h](p(; S_i(\underline{m};)), \cdots) \to \cdots \to PREC[h](p(; S_i(\underline{m};)), \cdots) \to_{24} \cdots$, $\underline{m} \neq \epsilon$ and $i \in \{0, 1\}$. In the first step the length of the term is increased by one unit. Since then and before any new appliance of rule 24, by lemma 5, the length is not any more increased. Lemma 6 enables us to conclude that $lh(\underline{p(; S_i(\underline{m};))}) \leq lh(S_i(\underline{m};))$ which put us in a good position to keep using this procedure. Therefore, $lh(t_{n1}) \leq lh(\underline{t_{11}}) + 1$. A similar reasoning, involving lemmas 5 and 7, would lead us to the second assertion. In order to prove the last assertion let us focus on the last component of the terms of the following chain:

$$\begin{aligned}
&PREC[h](\underline{t_{11}}, \underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}}) \to_{24} \\
&\quad PREC[h](t_{21}, t_{22}, \underline{t_{13}}; \underline{t_{14}}, h(\underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})) \to \\
&\quad \cdots \to \\
&\quad PREC[h](\underline{t_{21}}, \underline{t_{22}}, \underline{t_{13}}; \underline{t_{14}}, \underline{h(\underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})}) \to_{24} \\
&\quad PREC[h](t_{31}, t_{32}, \underline{t_{13}}; \underline{t_{14}}, h(\underline{t_{22}}, \underline{t_{13}}; \underline{t_{14}}, \underline{h(\underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})})) \to \\
&\quad \cdots \to \\
&\quad PREC[h](\underline{t_{31}}, \underline{t_{32}}, \underline{t_{13}}; \underline{t_{14}}, \underline{h(\underline{t_{22}}, \underline{t_{13}}; \underline{t_{14}}, \underline{h(\underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})})}) \to_{24} \\
&\quad \cdots.
\end{aligned}$$

The last component of the terms above has the shape $h(\underline{t_{j2}}, \underline{t_{13}}; \underline{t_{14}}, \underline{a})$, for some numeral $\underline{a}$, or it derives from $h(\underline{t_{j2}}, \underline{t_{13}}; \underline{t_{14}}, \underline{a})$ by the R_{PS} rewriting rules. Therefore, using the inequality 2) already established and the induction hypothesis, it is easy to argue that $lh(t_{n5}) \leq p_h(lh(h) + lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + 1 + lh(\underline{t_{13}}) + lh(\underline{t_{14}}) +$"bound for $lh(\underline{a})$"). To calculate a bound for $lh(\underline{a})$ one should notice that:
1) $\underline{a}$ is $\underline{t_{15}}$ or it has the shape

$$h\big(\underline{t_{j2}}, \underline{t_{13}}; \underline{t_{14}}, h(\underline{t_{j-1\ 2}}, \underline{t_{13}}; \underline{t_{14}}, \cdots \underbrace{h(\underline{t_{12}}, \underline{t_{13}}; \underline{t_{14}}, \underline{t_{15}})}_{\cdots} \cdots)\big);$$

2) By lemma 3, for all numeral $\underline{n}$, $lh(\underline{n}) = |\pi(\underline{n})| + 1$;
3) By lemma 4, there exists a polynomial q_h such that for all r_1, r_2, s_1 and s_2 in $G(PS)$ one has $|\pi(h(r_1, r_2; s_1, s_2))| \leq \max\{q_h(|\pi(r_1)|, |\pi(r_2)|), |\pi(s_1)|, |\pi(s_2)|\}$. Thus, for all $r_{11}, \cdots, r_{1k}, r_{21}, \cdots, r_{2k}, s_{11}, \cdots, s_{1k}, s_2$ one has

$$\begin{aligned} &|\pi(h(r_{11}, r_{21}; s_{11}, h(r_{12}, r_{22}; s_{12}, \cdots h(r_{1k}, r_{2k}; s_{1k}, s_2) \cdots)))| \\ &\leq \max\{\max_{1\leq i\leq k} q_h(|\pi(r_{1i})|, |\pi(r_{2i})|), \max_{1\leq i\leq k} |\pi(s_{1i})|, |\pi(s_2)|\}. \end{aligned}$$

From this it is simple to show that $lh(\underline{a}) \leq q_h(lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + lh(\underline{t_{13}})) + lh(\underline{t_{14}}) + lh(\underline{t_{15}})$. Therefore, we conclude that $lh(t_{n5}) \leq p_h(lh(h) + lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + 1 + lh(\underline{t_{13}}) + q_h(lh(\underline{t_{11}}) + lh(\underline{t_{12}}) + lh(\underline{t_{13}})) + 2 \cdot lh(\underline{t_{14}}) + lh(\underline{t_{15}}))$.

Based on assertions 1-3, it is trivial to construct a polynomial p_f verifying the theorem's assertion.

Now, the desired result is an almost immediate consequence of the previous theorem.

Corollary 2. *For all $f \in PS$ any R_{PS} reduction strategy yields an algorithm for $\pi(f)$ which runs in polynomial space.*

Proof. As we have already observed, every ground term in normal form is a numeral. Thus given $f \in PS$, for any R_{PS} reduction strategy we adopt and for all numerals $\underline{\bar{m}}, \underline{\bar{n}}$, we have a chain $f(\underline{\bar{m}}; \underline{\bar{n}}) \to t_2 \to \cdots \to t_n$ such that t_n is a numeral. Moreover, $t_n = \underline{\pi(f)(\bar{m}; \bar{n})}$. Now, let p_f be a polynomial given by the previous theorem. One has that: $\forall 2 \leq i \leq n\ lh(t_i) \leq p_f(lh(f(\underline{\bar{m}}; \underline{\bar{n}}))) = p_f(lh(f) + \sum_j lh(\underline{m}_j) + \sum_j lh(\underline{n}_j)) = p_f(\sum_j(|\underline{m}_j| + 1) + \sum_j(|\underline{n}_j| + 1) + lh(f))$. Therefore, setting $p_f^*(|\bar{m}|, |\bar{n}|) = p_f(\sum_j(|m_j| + 1) + \sum_j(|n_j| + 1) + lh(f))$, it is obvious that the polynomial p_f^*, in the binary lengths of $\bar{m}, \bar{n}$, dominates the length of any t_i, $2 \leq i \leq n$.

Remarks

We would like to emphasize the importance of the bound established in lemma 4. Let us recall it here: *For all $f \in PS$ there exists a polynomial, q_f, with natural coefficients such that $\forall \bar{r}, \bar{s} \in G(PS)$ $|\pi(f(\bar{r}; \bar{s}))| \leq \max\{q_f(|\pi(\bar{r})|), \max_i |\pi(s_i)|\}$.*

This lemma is used in a essential way in the proof of theorem 1. If we had only something like $|\pi(f(\bar{r};\bar{s}))| \leq q_f(|\pi(\bar{r})|) + \max_i |\pi(s_i)|$ — which is the bound for *Ptime* functions carried out by Bellantoni and Cook in [4] and used by Beckmann and Weiermann in [1] — we could not maintain the final, and also the most delicate, part of the reasoning performed in the proof of theorem 1.

The term rewriting characterizations can be used to achieve inclusion relations between classes of complexity, as claimed in [1]. Basically, they have been used to give simple and elegant proofs of known statements. An example of that is the alternative proof of the Bellantoni's result stating that *Ptime* is closed under safe recursion with parameter substitution, see [1]. For similar results involving the primitive recursive functions we refer to [8].

3.6 Unsorted Characterization

Pspace may also be implicitly characterized without using bounded schemes nor sorted function systems. That is the case of the characterization of *Pspace* given in [17]. There, an unbounded variation of the bounded characterization of *Pspace* described in section 3.2 is considered, and a purely syntactic measure of complexity, σ, is defined over the function terms. *Pspace* is identified with the terms which have rank-σ at most one. Intuitively, σ has to distinguish, for instance, when the binary successor $\mathbf{S}_1$ is taken as step-function in a primitive recursion or in a recursion on notation scheme. In the first case σ must increase drastically, because $\mathbf{S}_1$ together with primitive recursion leads us out of *Pspace*. However, σ should not increase drastically when, for example, the binary predecessor $\mathbf{P}$ is taken as step-function in a primitive recursion. Thus, σ should be accurate enough to distinguish recursion schemes and, for each recursion scheme, to identify "dangerous" and "non-dangerous" step-functions.

Some other ranks had already been considered to characterize other classes of functions — see, for instance, [5] and [13].

The Term System $\mathbf{T}^*_{Ps}$

Let $\mathbf{T}^*_{Ps}$ be the set of lambda expressions of type level at most 1 formed by lambda abstraction and application using ground-type variables V, the "initial" constant symbols $I = \{\epsilon, \mathbf{S}_0, \mathbf{S}_1, \mathbf{s}, \mathbf{P}, \mathbf{p}, \mathbf{C}, \mathbf{B}\}$, and recursion constant symbols **RN** and **PR**. Each constant symbol has the arity and type implied by the expressions used below, with the left and right sides of each conversion rule having ground type:

$$\mathbf{s}\,\epsilon \mapsto \mathbf{S}_0\,\epsilon$$
$$\mathbf{s}\,(\mathbf{S}_0\,x) \mapsto \mathbf{S}_1\,x$$
$$\mathbf{s}\,(\mathbf{S}_1\,x) \mapsto \mathbf{S}_0(\mathbf{s}\,x)$$
$$\mathbf{P}\,\epsilon \mapsto \epsilon$$
$$\mathbf{P}\,(\mathbf{S}_0\,x) \mapsto x$$
$$\mathbf{P}\,(\mathbf{S}_1\,x) \mapsto x$$

$$\begin{aligned}
&\mathbf{p}\,\epsilon \mapsto \epsilon\\
&\mathbf{p}\,(\mathbf{S}_0\,\epsilon) \mapsto \epsilon\\
&\mathbf{p}\,(\mathbf{S}_0\,(\mathbf{S}_0\,x)) \mapsto \mathbf{S}_1\,(\mathbf{p}\,(\mathbf{S}_0\,x))\\
&\mathbf{p}\,(\mathbf{S}_0\,(\mathbf{S}_1\,x)) \mapsto \mathbf{S}_1\,(\mathbf{p}\,(\mathbf{S}_1\,x))\\
&\mathbf{p}\,(\mathbf{S}_1\,x) \mapsto \mathbf{S}_0\,x\\
&\mathbf{C}\,\epsilon\,u\,v\,w \mapsto u\\
&\mathbf{C}\,(\mathbf{S}_0 x)\,u\,v\,w \mapsto v\\
&\mathbf{C}\,(\mathbf{S}_1 x)\,u\,v\,w \mapsto w\\
&\mathbf{B}\,x\,\epsilon\,z \mapsto x\\
&\mathbf{B}\,x\,(\mathbf{S}_i y)\,\epsilon \mapsto S_1 x\,,\quad i \in \{0,1\}\\
&\mathbf{B}\,x\,(\mathbf{S}_i y)\,(\mathbf{S}_j z) \mapsto \mathbf{B}\,(\mathbf{S}_j x)\,y\,z\,,\quad i,j \in \{0,1\}\\
&\mathbf{RN}\,g\,h\,\epsilon \mapsto g\,\epsilon\\
&\mathbf{RN}\,g\,h\,(\mathbf{S}_0 x) \mapsto h\,(\mathbf{S}_0 x)\,(\mathbf{RN}\,g\,h\,x)\\
&\mathbf{RN}\,g\,h\,(\mathbf{S}_1 x) \mapsto h\,(\mathbf{S}_1 x)\,(\mathbf{RN}\,g\,h\,x)\\
&\mathbf{PR}\,g\,h\,\epsilon \mapsto g\,\epsilon\\
&\mathbf{PR}\,g\,h\,(\mathbf{s}x) \mapsto h\,(\mathbf{s}x)\,(\mathbf{PR}\,g\,h\,x)
\end{aligned}$$

Of course we also have $(\lambda x\,r)s \mapsto r[s/x]$.

We restrict ourselves to type level at most 1 expressions, and by "term" in the following we mean such an expression. Every term in $\mathbf{T}^*_{Ps}$ has one of the following forms: an element of $I \cup V$; or $(\lambda x\,t)$; or $((\mathbf{RN}\,g)\,h)$; or $((\mathbf{PR}\,g)\,h)$; or (rs) with s of ground type.

Rank of Terms

For any term t, any integer $1 \leq i \leq \mathrm{arity}(t)$, and any variable $x \in FV(t)$, we define $\sigma(t;i)$ and $\sigma(t;x)$. $\sigma(t;i)$ is the rank of the ith input of t, while $\sigma(t;x)$ is the rank of variable $x \in FV(t)$. Note that σ is undefined outside this domain. In the following, for terms r, s, $1 \leq i \leq \mathrm{arity}(r)$ and $x \in FV(s)$, $K_{r,i,s,x}$ abbreviates the characteristic function of $\sigma(r;i) > \sigma(s;x) \wedge s \neq \lambda\bar{y}.x$, i.e.

$$K_{r,i,s,x} = \begin{cases} 1 & \text{if } \sigma(r;i) > \sigma(s;x) \wedge s \neq \lambda\bar{y}.x\\ 0 & \text{otherwise.}\end{cases}$$

We define:

$$\begin{aligned}
\sigma(\lambda z.r;i) &= \begin{cases}\sigma(r;z) & \text{if } i = 1\\ \sigma(r;i-1) & \text{if } i > 1\end{cases}\\
\sigma(\lambda z.r;x) &= \sigma(r;x) \text{ if } x \text{ is not } z\\
\sigma(rs;i) &= \sigma(r;i+1), \qquad \text{for } s \text{ of ground type}\\
\sigma(rs;x) &= \max\begin{cases}\sigma(r;1) + K_{r,1,s,x}\\ \sigma(s;x)\\ \sigma(r;x)\end{cases}, \text{ for } s \text{ of ground type.}
\end{aligned}$$

The clauses above are simply intended to switch back and forth between free variables and inputs reflecting the safe composition constraints. Notice that, so far, the only instance which might increase the rank is $\sigma(r;1) + K_{r,1,s,x}$, in the clause for $\sigma(rs;x)$. The rank becomes greater than one whenever x of rank $\sigma(s;x)$ in s goes, via s, into a higher rank position of r. The remaining clauses defining σ specify the way in which σ counts recursions. Let I^* be the "initial" constants except $\mathbf{S}_0, \mathbf{S}_1, \mathbf{s}$ and $\mathbf{B}$. One forces the rank of x in $\mathbf{RN}\, g\, h\, x$ to be one more than the integer approximation, with respect to $\lfloor\cdot\rfloor$, of the rank of the critical position in h. Similarly for $\mathbf{PR}\, g\, h\, x$ we use $\lceil\cdot\rceil$[1].

$$
\begin{array}{lll}
\sigma(\mathbf{S}_i;1) & = 1/2, & \text{for } i \in \{0,1\} \\
\sigma(\mathbf{s};1) & = 1 & \\
\sigma(\mathbf{B};i) & = 1, & \text{for } i \in \{1,2\} \\
\sigma(\mathbf{B};3) & = 0 & \\
\sigma(D;i) & = 0, & \text{for } D \in I^*, 1 \leq i \leq \mathrm{arity}(D) \\
\sigma(x;x) & = 0, & \text{for } x \in V
\end{array}
$$

$$
\sigma(\mathbf{RN}ghx;1) = \max \begin{cases} \sigma(h;1) \\ \sigma(g;1) \\ 1 + \lfloor \sigma(h;2) \rfloor \end{cases}
$$

$$
\sigma(\mathbf{RN}ghx;x) = \max \begin{cases} \sigma(h;x) \\ \sigma(h;2) + K_{h,2,g,x} \\ \sigma(h;2) + K_{h,2,h,x} \\ \sigma(g;x) \end{cases}
$$

$$
\sigma(\mathbf{PR}ghx;1) = \max \begin{cases} \sigma(h;1) \\ \sigma(g;1) \\ 1 + \lceil \sigma(h;2) \rceil \end{cases}
$$

$$
\sigma(\mathbf{PR}ghx;x) = \max \begin{cases} \sigma(h;x) \\ \sigma(h;2) + K_{h,2,g,x} \\ \sigma(h;2) + K_{h,2,h,x} \\ \sigma(g;x) \end{cases}
$$

Define $\sigma(t) = \max(\{\sigma(t;i) : \ 1 \leq i \leq \mathrm{arity}(t)\} \cup \{\sigma(t;x) : \ x \in FV(t)\} \cup \{0\})$.

Undefined terms are omitted from the maximums above. When all clauses are undefined, then the maximum is undefined.

Definition 5. $\mathbf{T}_{Ps} := \{t \in \mathbf{T}^*_{Ps} \ : \ \sigma(t) \leq 1\}$.

Let us open a parenthesis here to define a few $\mathbf{T}^*_{Ps}$ terms which may clarify the intended meaning of $\sigma(t) \leq 1$. In the following, when defining a term using recursion, we write it informally using equations rather than explicitly using the constants **RN** and **PR**. For $i \in \{0,1\}$, let us consider the following functions

[1] $\lfloor n \rfloor$ is the rounding-down of n, i.e. the integer part, whereas $\lceil n \rceil$ is the rounding-up.

defined by recursion on notation:

$$\epsilon \oplus x = x$$
$$(\mathbf{S}_i y) \oplus x = \mathbf{S}_i(y \oplus x)$$

$$\epsilon \otimes x = \epsilon$$
$$(\mathbf{S}_i y) \otimes x = x \oplus (y \otimes x)$$

$$e(\epsilon) = \mathbf{S}_1(\epsilon)$$
$$e(\mathbf{S}_i x) = e(x) \oplus e(x)$$

One has $\sigma(\oplus) = 1$, essentially because $\sigma(\mathbf{S}_0) = \sigma(\mathbf{S}_1) = 1/2$. The second input x of $y \oplus x$ has rank $1/2$, and this is the critical position used in the recursion defining $\otimes$. One then gets $\sigma(\otimes; 1) = 1 + \lfloor \sigma(\oplus; 2) \rfloor = 1$ and $\sigma(\otimes; 2) = \sigma(\oplus; 1)$, leading to $\sigma(\otimes) = 1$. On the other hand, $\sigma(e) = 2$ due to the fact that e is defined by a recursion through the rank 1 input of $\otimes$.

The reason why we defined $\sigma(\mathbf{S}_i; 1) = 1/2$ should now become clear. If $\mathbf{S}_i$ is used in a recursion (in a meaningful way) then one would have $\sigma(h; 2) \geq 1/2$. Consequently $\sigma(\mathbf{RN}gh; 1)$ might be less or equal than 1, but $\sigma(\mathbf{PR}gh; 1)$ will be greater than 1. This corresponds to the recursion constraints we have in the input-sorted characterizations of *Pspace*: S_i may be used in a full way together with the recursion on notation rule, but the use of S_i in a primitive recursion definition may lead us out of *Pspace*. For example $f(\epsilon;) = \epsilon$ and $f(s(x;);) = S_1(; f(x;))$ has exponential growth.

One has that:

Theorem 2. $\mathbf{T}_{Ps}$ *coincides with Pspace.*

For a proof we address to [17], pp.66-75. The proof is based on the three--input-sorted characterization of *Pspace* described in section 3.4 — *3-Pspace*. The mentioned three-input-sorted characterization results from *2-Pspace* by removing the (modified) product from the initial functions and introducing an additional input position to construct the product inside the class. Notice that in $\mathbf{T}_{Ps}$ terms may only have rank 1, 1/2 or 0. Thus on one side we have three possible ranks and on the other side we have three possible input sorts. The identification of $\mathbf{T}_{Ps}$ with *Pspace* results from building up a correspondence between input ranks and input sorts. More precisely, in [17], we prove that:

1. If $f(\bar{x}; \bar{y}; \bar{z}) \in$ *3-Pspace* then there exists a term $t_f \in \mathbf{T}_{Ps}$ such that

$$f(\bar{x}; \bar{y}; \bar{z}) \equiv t_f[\bar{x}; \bar{y}; \bar{z}].$$

   Moreover, $\sigma(t_f; x_i) = 1$, $\sigma(t_f; y_i) = 1/2$ and $\sigma(t_f; z_i) = 0$.

2. If $t \in \mathbf{T}_{Ps}$ then there exists a function $f_t \in$ *3-Pspace* such that

$$t[\bar{x}]\bar{y} \equiv f_t(\bar{y}_t^1, \bar{x}_t^1; \bar{y}_t^{1/2}, \bar{x}_t^{1/2}; \bar{y}_t^0, \bar{x}_t^0),$$

where $\bar{y} \equiv (y_1, \cdots, y_n)$, $n = \text{arity}(t)$, $\bar{x}$ is a list of all free variables of t, $\bar{x}_t^0$ and $\bar{y}_t^0$ are selected from $\bar{x}$ and $\bar{y}$ corresponding to the rank 0 or undefined rank variables and inputs of t, $\bar{x}_t^{1/2}$ and $\bar{y}_t^{1/2}$ are selected corresponding to the rank $1/2$ variables and inputs of t, and $\bar{x}_t^1$ and $\bar{y}_t^1$ are selected corresponding to the rank 1 variables and inputs of t.

A final note just to mention that the rank described here can be improved, namely by omitting the occurrences of $K_{.,.,.,.}$ in the definition of σ.

3.7 Final Comments

Implicit characterizations of *Pspace* were also given, by Leivant and Marion, in [11] and [12]. In [11] *Pspace* is characterized via a three-sorted system, which uses a scheme of recursion with parameter substitution. In [12] Leivant and Marion use higher type recursion to characterize *Pspace*. In particular, they show that using higher type recursion one can simulate the scheme of recursion with parameter substitution given in [11].

It is very common to associate schemes of recursion with parameter substitution to classes of parallel computations. We recall here that *Pspace* (boolean functions) may also be described as a class of parallel computations: alternating polynomial time. The reason to associate schemes of recursion with parameter substitution, instead of "simple" recursion schemes, to classes of parallel computations can be explained as follows: Given a deterministic computation, let T be its transition function. For each configuration c, $T(c)$ is the next configuration. Notice that $T(c)$ is uniquely determined by the computation. If $f(t, c)$ is the t'th configuration starting with c, then $f(0, c) = c$ and $f(t + 1, c) = T(f(t, c))$. Informally speaking, recursion seems to be appropriate to simulate deterministic computations. Now let us think about parallel computations. Let us say that T_1 and T_2 are the transition functions of a given parallel computation. Thus, any configuration c has two next configurations $T_1(c)$ and $T_2(c)$. If $f(t, c)$ are the t'th configurations starting with c, then $f(0, c) = c$ and $f(t + 1, c) = J(f(t, T_1(c)), f(t, T_2(c)))$, where J is a function that combines configurations. Therefore, recursion with parameter substitution seems to be appropriate to simulate parallel computations. Nevertheless, we believe that the power of parallel computations can be captured by recursion schemes over trees, without (full) parameter substitution. A characterization of NC in this vein was already achieved, see [6].

References

1. Beckmann A., Weiermann A.: *A term rewriting characterization of the polytime functions and related complexity classes*, Archive for Mathematical Logic 36, 1996, pp.11-30.
2. Bellantoni S.: *Predicative Recursion and Computational Complexity*, Ph. D. Dissertation, University of Toronto, 1993.

3. Bellantoni S.: *Predicative Recursion and the Polytime Hierarchy*, Feasible Mathematics II, ed. P.Clote and J.B.Remmel, Birkhäuser, 1995, pp.15-29.
4. Bellantoni S., Cook S.: *A New Recursion-Theoretic Characterization of Polytime Functions*, Computational Complexity, vol.2, 1992, pp.97-110.
5. Bellantoni S., Niggl K.-H.: *Ranking Primitive Recursions: The Low Grzegorczyk Classes Revisited*, SIAM Journal of Computing, vol.29, N.2, 1999, pp.401-415.
6. Bellantoni S., Oitavem I.: *Separating NC along the δ axis*, submitted.
7. Buss S.: *Bounded Arithmetic*, Bibliopolis, 1986.
8. Cichon E.A., Weiermann A.: *Term rewriting theory for the primitive recursive functions*, Annals of Pure and Applied Logic 83, 1997, pp.199-223.
9. Ferreira F.: *Polynomial Time Computable Arithmetic and Conservative Extensions*, Ph. D. Dissertation, Pennsylvania State University, 1988.
10. Ferreira F.: *Polynomial Time Computable Arithmetic*, Contemporary Mathematics, Volume 106, 1990, pp.137-156.
11. Leivant D. and Marion J.-Y.: *Ramified Recurrence and Computational Complexity II: Substitution and Poly-space*, Computer Science Logic, ed. L.Pacholski and J.Tiuryn, Springer-Verlag, 1995, pp.486-500.
12. Leivant D. and Marion J.-Y.: *Ramified Recurrence and Computational Complexity IV: Predicative functionals and poly-space* , Preprint.
13. Niggl K.-H.: *The μ-Measure as a Tool for Classifying Computacional Complexity*, Archive for Mathematical Logic, vol.39, N.7, 2000, pp.509-514.
14. Oitavem I.: *Três assuntos de Lógica e Complexidade*, Master thesis, Universidade de Lisboa, 1995.
15. Oitavem I.: *New recursive characterizations of the elementary functions and the functions computable in polynomial space*, Revista Matematica de la Universidad Complutense de Madrid, vol.10, N.1, 1997, pp.109-125.
16. Oitavem I.: *A term rewriting characterization of the functions computable in polynomial space*, Archive for Mathematical Logic, to appear.
17. Oitavem I.: *Classes of Computational Complexity: Implicit Characterizations — a Study in Mathematical Logic*, Ph.D. thesis, Universidade de Lisboa, 2001.
18. Papadimitriou C.H.: *Computational Complexity*, Addison-Wesley Publishing Company, 1994.

Iterate Logic

Peter H. Schmitt

Universität Karlsruhe, Institute for Logic,
Complexity and Deduction Systems,
D-76128 Karlsruhe, Germany.
http://i12www.ira.uka.de/

Abstract. We introduce a new logic for finite first-order structures with a linear odering. We study its expressive power. In particular we show that it is strictly stronger than first-order logic on finite structures. We close with a list of open problems.

1 Introduction

The Unified Modeling Language (UML) is a language for visualizing, specifying, constructing and documenting object-oriented software systems. It has been widely accepted as a standard for modeling software systems and is supported by a great number of CASE tools (Computer Aided Software Engineering tools). The standard document on UML is [12]. The Object Constraint Language (OCL) is part of UML [11]. The only available introduction at the time is [18]. The Object Contraint Language (OCL) is used to describe the semantics of UML with greater precision. It is extensively used in the UML meta model. The available descriptions of OCL fall short of giving a rigorous semantics, and even some syntactic details are not clear. Deficiencies have been pointed out e.g. in [14,4, 5,10].

One way to remedy this situation and provide OCL with a precise semantics is via a translation into a known, well understood specification language. This approach has been pursued e.g. in [6,3,2,8,9] . As an additional advantage of this approach the translated expressions are frequently used as input to existing tools. The disadvantage is that those not familiar with the target language will gain nothing.

This paper grew out of an effort to give a systematic definition of syntax and semantics of the OCL. This will be pursued elsewhere. Here we aim at a different goal, we try to formulate the logical essence behind OCL. Syntax and semantics of an OCL expression depends stongly on an associated UML class diagram. The work presented here is at a level of abstraction where this dependence is no longer directly visible. We arrive at a new kind of logic and study it in its own right.

2 Syntax and Semantics of L_{it}

In this section we introduce Iterate logic $\mathcal{L}_{it}$ by a series of definitions following the usual pattern. We will define the possible signatures Σ, the set of well-

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 191–201, 2001.

formed terms $\mathcal{T}_{it}(\Sigma)$ and formulas $\mathcal{L}_{it}(\Sigma)$ over Σ, the structures $\mathcal{M}$ and the interpretation of terms and formulas in a given $\mathcal{M}$.

Definition 1
The signature Σ for an Iterate logic consists of

1. *A non-empty set $\mathcal{S}$ of sorts. Bool is always an element of $\mathcal{S}$*
2. *A set C of sorted constant symbols.*
   We use $c : S$ to indicate that c is a constant symbol of sort S.
   The constants true and false of sort Bool are assumed to be available in any signature Σ.
3. *A set F of sorted function symbols.*
   we use $f : S_1 \times \ldots \times S_k \to S$ to indicate that f takes arguments of sort S_1 to S_k, respectively, and output values of sort S. We stipulate further that $S, S_i \in \mathcal{S}$.
4. *Function symbols $r : S_1 \times \ldots \times S_k \to Bool$ of target sort Bool will be called predicate symbols.*
5. *For every sort $S \in \mathcal{S}$ there is a predicate symbol $\doteq_S$: $S \times S \to Bool$.*
   We will usually just write $\doteq$ instead of $\doteq_S$.
6. *The will always be function symbols*
   - $\wedge : Bool \times Bool \to Bool$
   - $\vee : Bool \times Bool \to Bool$
   - $\Rightarrow$: $Bool \times Bool \to Bool$
   - $\neg : Bool \to Bool$

   in Σ
7. *There will always be a binary predicate symbol $<$ in Σ.*
   We will later insist that $<$ is always interpreted as a linear order.

There is for every sort S an infinite collection of variables Var_S of sort S.

Definition 2
Terms and their sorts over Σ are defined as follows. Terms of sort Bool will be called formulas.

1. *If c is a constant symbol of sort S then c is a term of sort S,*
2. *For every variable $x \in Var_S$ the expression x is a term of sort S,*
3. *If $f \in F$ is a function symbol with signature $f : S_1 \times \ldots \times S_k \to S$ and $t_1, \ldots, t_k$ are terms of sort $S_1, \ldots, S_k$ respectively, then $f(t_1, \ldots, t_k)$ is a term of sort S.*
4. *If*
   - *sorts $S, T \in \mathcal{S}$,*
   - *$x \in Var_S$, $y \in Var_T$, y different from x,*
   - *t is a formula not containing y,*
   - *t_0 a term of sort T not containing x or y,*
   - *u a term of sort T,*

   then
   iterate$(x : S; y : T = t_0 \mid u)t$
   is a term of sort T.

5. *If t_1, t_2 are terms of the same sort then we write $t_1 \doteq t_2$ instead of $\doteq(t_1, t_2)$. Also the propositional connectives will be written in infix notation, i.e. $t_1 \wedge t_2$ instead of $\wedge(t_1, t_2)$.*

Since the syntax of iterator terms involves five constituent entities it pays to agree on some terminology. Using the notation from clause 4 above, the variable x is called the *iterator variable*, y the *accumulator variable*, t is termed the *range formula*, u the *step term* and t_0 the *initial term*.

Definition 3
A structure $\mathcal{M}$ of Σ consists of a non-empty domain M and an interpretation function I such that

1. *$I(S)$ is a non-empty subset of M for every $S \in \mathcal{S}$. The universe M of $\mathcal{M}$ is the disjoint union of all $I(S)$, $S \in \mathcal{S}$.*
2. *$I(c) \in I(S)$ for every $(c : S) \in C$.*
3. *for every $(f : S_1 \times \ldots \times S_k \to S) \in F$ the interpretation $I(f)$ is a function from $I(S_1) \times \ldots \times I(S_k)$ into $I(S)$.*
4. *$I(\doteq)$ is the equality relation and the Boolean connectives will have their usual meaning.*
5. *$I(<)$ is a total linear order on M. Instead of $I(<)$ we will by abuse of notation write $<$ or $<_{\mathcal{M}}$ if need arises.*

Usually we will just write I for the interpretation function. If necessary we will write, more precisely, $I_{\mathcal{M}}$.

We note that Definition 3 is for the greatest part the usual definition of many-sorted structures. The only particularities are the insistence on having a linear order available, a phenomenon familiar to those working in the area of characterizing complexity classes via logical definability, and the fact that we will only consider finite structures.

Definition 4
Let $\mathcal{M} = (M, I)$ be a finite *structure.*

The interpretation function I is extended to terms and formulas. As usual we need valuations $\beta : Var_S \to I(S)$ of free variables for all $S \in \mathcal{S}$.

For a term t of sort $S \in \mathcal{S}$ $I_\beta(t)$ will be an element of $I(S)$.

1. *if $t \in Var_S$ then $I_\beta(t) = \beta(t)$,*
2. *$I_\beta(f(t_1, \ldots, t_k)) = I(f)(I_\beta(t_1), \ldots, I_\beta(t_k))$.*
3. *the interpretation $m = I_\beta(\mathbf{iterate}(x : S; y : T = t_0 \mid u)t)$ is computed as follows.*
   Let $\{a_1, \ldots, a_n\} = \{a \in I(S) \mid I_{\beta^a_x}(t) = true\}$ with $a_1 < \ldots < a_n$.
   We define m_i for $0 \leq i < n$ by:
   a) $m_0 = I_\beta(t_0)$
   b) $m_{i+1} = I_{\beta^{i+1}}(u)$
      where
      $$\beta^{i+1}(z) = \begin{cases} \beta(z) & \text{if } z \neq x, y \\ a_{i+1} & \text{if } z = x \\ m_i & \text{if } z = y \end{cases}$$
      (Observe that $m_i \in I(T)$ for all i.)

Now we set $m = m_n$.

4. $I_\beta(t_1 \doteq t_2) = true$ *iff* $I_\beta(t_1) = I_\beta(t_2)$.
5. $I_\beta(A)$ *for* A *a propositional composition of smaller formulas is defined as usual.*

Remark: If in the computation of $I_\beta(\mathbf{iterate}(x : S_1; y : T = t_0 \mid u)t)$ the set $\{a \in I(S_1) \mid I_{\beta^a_x}(t) = true\}$ is empty, then we have, according to the above definition, $I_\beta(\mathbf{iterate}(x : S_1; y : T = t_0 \mid u)t) = I_\beta(t_0)$.

Example: To get familiar with Definition 4 let us evaluate the meaning of the formula $\mathbf{iterate}(x : S; y : Bool = true \mid y \wedge u)(true)$ in structure $\mathcal{M} = (M, I)$. In addition to the restrictions on the variables imposed by the syntax definition of **interate**-terms, we require that y does not occur in u. The range formula being $true$ we obtain as the basis of iteration the set of all elements of sort S in M, say $a_1 < \ldots < a_n$. The initial value m_0 of the accumulator is the Boolean value $true$. For the first step in the iteration we compute $m_1 = I_{\beta^1}(y \wedge u)$, where $\beta^1(x) = a_1$ and $\beta^1(y) = true$. In other words $m_1 = true$ if $\mathcal{M} \models u[a_1]$ and $m_1 = false$ otherwise. We also notice that once the accumulator is assigned the value $false$ it will be stuck with this value for the rest of the iteration. Thus the final value m_n of the accumulator is $true$ if for all a_i $\mathcal{M} \models u[a_i]$ holds true and $false$ otherwise. Summing up, we may use $\mathbf{iterate}(x : S; y : Bool = true \mid y \wedge u)(true)$ as a definition of the universal quantification $\forall x_S u$.

Likewise $\mathbf{iterate}(x : S; y : Bool = false \mid y \vee u)(true)$ behaves as $\exists x_S u$.

Definition 5
Let t, s *be formulas in* $\mathcal{L}_{it}$, $\mathcal{M} = (M, I)$ *a finite structure.*

1. t *is called* valid *in* $\mathcal{M}$, *in symbols* $\mathcal{M} \models t$, *if for all* β $I_\beta(t) = true$.
2. t *is called* universally valid, *in symbols* $\models t$, *if it is valid for all finite structures* $\mathcal{M}$.
3. t *is* a logical consequence *of* s, *in symbols* $s \models t$, *if for any finite structure* $\mathcal{M}$ *such that* s *is valid in* $\mathcal{M}$ *also* t *is valid in* $\mathcal{M}$.

For formulas, i.e. terms t *of sort Boolean we will also use notations like* $(\mathcal{M}, \beta) \models t$ *instead of* $I_\beta(t) = true$, *or even* $\mathcal{M} \models t[a_1, \ldots, a_k]$, *where* $x_1, \ldots, x_k$ *are all free variables in* t *(listed in some fixed order) and* $\beta(x_i) = a_i$.

Definition 6
We obtain (a syntactic variant of) first-order logic $\mathcal{L}_{FO}$ *with a linear order if we replace item 4 in Definition 2 by*

4. *If* t *is a formula,* x *a variable of sort* S *then also* $\forall x t$ *and* $\exists x t$ *are formulas.*

and leave all other clauses unchanged.

3 Expressive Power of L_{it}

Having formulated Iterate Logic in the previous section we will now start collecting facts on its expressive power. We will use the following concept for comparing different logics with respect to their expressive power. We will only consider logics that use as interpretations structures of the kind defined in Definition 3. This is no severe restriction since it is met by most logics we are interested in, e.g. first-order logic, (monadic) second-order logic, fixed-point logics, transitive closure logics.

Definition 7
We say, logic $\mathcal{L}_2$ is as a expressive as $\mathcal{L}_1$, in symbols

$$\mathcal{L}_1 \leq \mathcal{L}_2$$

if for every formula t_1 in $\mathcal{L}_1$ there is a formula t_2 in $\mathcal{L}_2$ such that for all finite structures $\mathcal{M}$ and variable assignments β

$$(\mathcal{M}, \beta) \models_1 t_1 \text{ iff } (\mathcal{M}, \beta) \models_2 t_1$$

$\mathcal{L}_2$ is strictly more expressive than $\mathcal{L}_1$, in symbols $\mathcal{L}_1 < \mathcal{L}_2$ if $\mathcal{L}_1 \leq \mathcal{L}_2$ and $\mathcal{L}_2 \not\leq \mathcal{L}_1$.

The example following Definition 4 establishes $\mathcal{L}_{FO} \leq L_{it}$. We will eventually show that L_{it} is strictly more expressive than $\mathcal{L}_{FO}$. To prepare the ground we prove two simple definability lemmas for L_{it}.

Lemma 1. *For any formula t and terms t_1, t_2 of sort $S \in \mathcal{S}$ there is an L_{it}-term s, such that*

$$s = (\textit{if } t \textit{ then } t_1 \textit{ else } t_2)$$

Proof: Set $s = \mathbf{iterate}(x : S; y : S = t_2 \mid t_1)t$ where the variables x, y are different and do both not occur in t, t_1 and t_2. In any structure $\mathcal{M} = (M, I)$ we have for all β

$$\{a \in I(S) \mid I_{\beta^a_x}(t) = true\} = \begin{cases} M \text{ if } I_\beta(t) = true \\ \emptyset \quad \text{if } I_\beta(t) = false \end{cases}$$

From this we immediately get

$$I_\beta(s) = \begin{cases} I_\beta(t_1) \text{ if } I_\beta(t) = true \\ I_\beta(t_2) \text{ if } I_\beta(t) = false \end{cases}$$

∎

Lemma 2. *For any function symbol $f \in \Sigma$ there is a formula $s(x, y)$ in L_{it}, such that for any structure $\mathcal{M} = (M, I)$, any β, and any elements $a, b \in M$*

$$I_{\beta^{a\,b}_{x\,y}}(s) = true$$
$$\Leftrightarrow$$
$$\textit{there is } n \in \mathbb{N} \textit{ with } n \leq \textit{ card}(M) \textit{ such that } b = I_\beta(f)^n(a)$$

Proof: Set $s = \exists w((\mathbf{iterate}(z; v = x \mid u)true) \doteq y)$
where $u =$ if $(z < w)$ then $f(v)$ else v

∎

Lemma 2 already suggests that $\mathcal{L}_{it}$ is strictly stronger than first-order logic even on finite structures. But we can do better.

For the remainder of this argument, which will come to a conclusion with the proof of Lemma 3, we fix a particular family of signatures Σ_A. The parameter A is being looked at as an alphabet, as it is used in formal language theory. For the alphabet A we denote, as usual, the set of all words over A by A^*. The signature Σ_A contains the binary relation symbol $<$, a constant symbol first, a unary, postfix function $+1$ and for every $a \in A$ a unary relation symbol $Q_a()$. It does not hurt to assume in addition that Σ_A also contains a constant last and a function symbol -1.

Furthermore let Φ_A be the formula stating that

1. $<$ is a strict linear ordering with first and last element,
2. $+1$ is the successor function on non-maximal elements m and $m+1 = m$ for the greatest element
3. -1 is the predecessor function on non-minimal elements and first $-1 =$ last,
4. for every position p there is exactly one $a \in A$ such that $Q_a(p)$ is true.

Note, the asymmetric behaviour of $+1$ and -1 at the endpoints. Every finite model of Φ_A has an isomorphic copy whose elements are the natural numbers from 0 to some n. We will only consider models of Φ_A in this restricted sense. Then, the set of finite non-empty words $A^* \setminus \{\Lambda\}$ is in one-to-one correspondence with the finite models of Φ_A. For $w \in A^* \setminus \{\Lambda\}$ we denote the associated Σ_A structure by $\mathcal{M}_w$. We quickly recall the relationship between w and $\mathcal{M}_w$. Full descriptions are contained in the literature quoted in the next paragraph. Let $0, \ldots, n-1$ be all the elements of $\mathcal{M}_w$. The cardinality n of $\mathcal{M}_w$ equals the length of the word w. The elements i are thought of as the positions in the word w. $\mathcal{M}_w \models Q_a(0)$ holds true if and only if the first letter in w equals a, $\mathcal{M}_w \models Q_b(1)$ iff the second letter of w equals b and so on. We have to exclude the empty word, Λ, since in our setting the empty model $\mathcal{M}_\Lambda$ is not possible.

The following formula t_{anbn} in signature Σ_A for $A = \{a, b\}$.

$$
\begin{array}{l}
\exists x(\ \text{first} < x \wedge x < \text{last} \wedge \\
\quad (\mathbf{iterate}(z; v = (if\ Q_a(\text{first})\ \mathbf{then}\ \text{first}\ \mathbf{else}\ \text{last}) \mid u)true) \doteq \text{first}) \\
\text{where} \\
u = if(z \leq x \wedge Q_a(z))\ \text{then}\ z\ \text{else} \\
\quad (if(z \leq x \wedge \neg Q_a(z))\ \text{then last else} \\
\quad (if(x < z \wedge \neg Q_a(z))\ \text{then last else} \\
\quad (if(x < z \wedge Q_a(z))\ \text{then}\ v - 1)))
\end{array}
$$

will play the key role in the proof to follow.

Lemma 3.
For every $w \in \{a, b\}^*$, $w \neq \Lambda$ and its corresponding model $\mathcal{M}_w$ of Φ_A:

$$\mathcal{M}_w \models t_{anbn} \quad \Leftrightarrow w \in aa^n bb^n$$

Proof of the Lemma: 3
We first assume $\mathcal{M}_w \models t_{anbn}$. There is thus a natural number n, first $= 0 < n <$ last $= k$ such that

$$m = I_\beta(\textbf{iterate}(z; v = \text{first} \mid u)true) = \text{first}$$

with $\beta(x) = n$. Unravelling this according to Definition 4 we get for all $0 \leq i \leq k$:

$$m_i = \begin{cases} i & \text{if } i \leq n \wedge Q_a(i) \\ k & \text{if } i \leq n \wedge Q_b(i) \\ i-1 & \text{if } i > n \wedge Q_b(i) \\ k & \text{if } i > n \wedge Q_a(i) \end{cases}$$

We first observe that if $m_i = k$ for some i then $m = m_k$ can never be 0, as claimed. Thus we have vor all $i \leq n$ $Q_a(i)$ and for all $i > n$ $Q_b(i)$. Given that, we see that $m = n - (k - n) = 0$, i.e. $k = 2n$. Note that the stipulation first -1 = last is important here (first -1 = first would not work). So, we have seen that w is of the form $a^n b^n$ with $n > 0$.

For the reverse implication of the claim, we start with a word w of the form $a^n b^n$ with $n > 0$ and want to show $\mathcal{M}_w \models t_{anbn}$. As a witnessing element for the leading existential quantifier we use n. The rest is straight forward.

∎

The question which classes of words can be defined in first-order logic and in monadic second-order logic has been intensively investigated, surveys may be found e.g. in [16,17,13]. By a well-known result of Büchi, the sets of finite words definable by monadic second-order logic are precisely the regular sets of words. Now, $aa^n bb^n$ denotes a context-free but not regular language, thus Lemma 3 tells us that $\mathcal{L}_{it} \not\leq \mathcal{L}_{MonSO}$. We have already observed that $\mathcal{L}_{FO} \leq \mathcal{L}_{it}$ is true. Thus we also get $\mathcal{L}_{FO} < \mathcal{L}_{it}$.

We compare $\mathcal{L}_{it}$ with the transitive closure logic $\mathcal{L}_{tc}$, that has been introduced by Neil Immerman in [7, Section 3].

Definition 8
The terms of $\mathcal{L}_{tc}$ are defined as in Definition 2 exept that clause 4 is replaced by

4. *If t is a term of sort S and s is a term of sort Bool, $x_1, \ldots, x_k, y_1, \ldots, y_k$ a sequence of distinct variables, and R a relational variable not free in s, then*

$$[R(x_1, \ldots, x_k, y_1, \ldots, y_k) \equiv TC(s)]t$$

is also a term in $\mathcal{L}_{tc}$ and of sort S.

Definition 9
Let $\mathcal{M} = (M, I_\mathcal{M})$ be a finite structure, β a variable assignment.

In order to define the interpretation of $\mathcal{L}_{tc}$-terms in $\mathcal{M}$ we only need to deal with the TC-construct $[R(\boldsymbol{x}, \boldsymbol{y}) \equiv TC(s)]t$. Here $\boldsymbol{x}$ is shorthand for the tupel $x_1, \ldots, x_k$ of variables, likewise $\boldsymbol{y}$ abbreviates the tupel $y_1, \ldots, y_k$ of the same length. $x_1, \ldots, x_k$ and $y_1, \ldots, y_k$ are $2k$ different variables.

We first observe that s introduces a binary relation R_s^β on the set M^k via the definition

$$R_s^\beta(\boldsymbol{a}, \boldsymbol{b}) \Leftrightarrow (\mathcal{M}, \beta) \models s[\boldsymbol{a}, \boldsymbol{b}]$$

Again $\boldsymbol{a}$ abbreviates the tupel $a_1, \ldots, a_k$, and $\boldsymbol{b}$ the tupel $b_1, \ldots, b_k$. Furthermore we made use of $(\mathcal{M}, \beta) \models s[\boldsymbol{a}, \boldsymbol{b}]$ as a suggestive shorthand for $(\mathcal{M}, \beta') \models s$ with

$$\beta'(u) = \begin{cases} a_i & \text{if } u = x_i \\ b_i & \text{if } u = y_i \\ \beta(u) & \text{otherwise} \end{cases}$$

(Compare the remark following Definition 5.)
Then

$$I_{\mathcal{M},\beta}([R(x_1, \ldots, x_k, y_1, \ldots, y_k) \equiv TC(s)]t) = I_{\mathcal{M}^*,\beta}(t)$$

where $\mathcal{M}^*$ coincides with $\mathcal{M}$, but in addition $I_{\mathcal{M}^*}(R)$ is the transitive closure of the relation R_s^β. Notice, that the relation symbol R may occur free in t.

Remark: The superscript β in R_s^β is neccessary, since $\boldsymbol{x}, \boldsymbol{y}$ may not be all the free variables of s.

Lemma 4.

$$\mathcal{L}_{it} \leq \mathcal{L}_{tc}$$

Proof: We have to produce for every $\mathcal{L}_{it}$-formula r an $\mathcal{L}_{tc}$-formula r^{tc} such that for all structures $\mathcal{M}$ and variable assignments β

$$(\mathcal{M}, \beta) \models_{it} r \Leftrightarrow (\mathcal{M}, \beta) \models_{tc} r^{tc}$$

We will in fact construct for every $\mathcal{L}_{it}$-term r a $\mathcal{L}_{tc}$-formula $\bar{r}$, containing a new additional free variable z such that for all structures $\mathcal{M} = (M, I)$ and variable assignments β

$$I_\beta(r) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{r} = true$$

If r is a formula, i.e. a term of sort Boolean, then $r^{tc} = \bar{r}(true/z)$ yields the desired translation.

Naturally $\bar{r}$ will be constructed by structural induction. To reduce technical complications, we use always the same reserved variable z as the new variable in $\bar{r}$ for any r and assume that z does not occur among the variables in $\mathcal{L}_{it}$.

If r is simply a functional term $r = f(x_1, \ldots, x_n)$ then $\bar{r} = (z \doteq f(x_1, \ldots, x_n))$ The only challenging induction step is the case

$$r = \textbf{iterate}(x : S; y : T = t_0 \mid u)t.$$

By induction hypothesis there are $\mathcal{L}_{tc}$-formulas $\bar{t_0}$ and $\bar{u}$ such that for all $\mathcal{M} = (M, I)$ and β

$$I_\beta(t_0) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{t_0} = true$$
$$I_\beta(u) = a \Leftrightarrow (\mathcal{M}, \beta_z^a) \models_{tc} \bar{u} = true$$

Since t has to be a term of sort Boolean the induction hypothesis provides us with a $\mathcal{L}_{tc}$-formula t^{tc} such that for all $\mathcal{M}$ and β

$$(\mathcal{M},\beta) \models_{it} t \Leftrightarrow (\mathcal{M},\beta) \models_{tc} t^{tc}$$

As a first step towards the construction of the $\mathcal{L}_{tc}$-formula $\bar{r}$ consider the formula

$$\begin{aligned} e(x_1,x_2,y_1,y_2) = \exists x t^{tc} \wedge [& \\ & (first_t(x_1) \wedge next_t(x_1,y_1) \wedge \bar{t_0}(x_2/z) \wedge \bar{u}(y_1/x, x_2/y, y_2/z)) \\ & \vee \\ & (\neg first_t(x_1) \wedge next_t(x_1,y_1) \wedge \bar{u}(y_1/x, x_2/y, y_2/z)) \\ &] \end{aligned}$$

with

$$\begin{aligned} first_t(x_1) &= t^{tc}(x_1/x) \wedge \neg\exists x(t^{tc} \wedge x < x_1) \\ last_t(x_1) &= t^{tc}(x_1/x) \wedge \neg\exists x(t^{tc} \wedge x_1 < x) \\ next_t(x_1,y_1) &= t^{tc}(x_1/x) \wedge t^{tc}(y_1/x) \wedge x_1 < x_2 \wedge \neg\exists x(t^{tc} \wedge x_1 < x < x_2) \end{aligned}$$

For any terms w, s_1, s_2 the term $w(s_1/x, s_2/y)$ denotes the term obtained from w by replacing simultaneously x by s_1 and y by s_2.

The looked for translation of r can now we define as

$$\begin{aligned} \bar{r} = & (\neg\exists x t^{tc} \wedge \bar{t_0}) \vee \\ & (\neg\exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \neq z_2) \wedge \exists x \exists w(t^{tc} \wedge \bar{t_0}(w/z) \wedge \bar{u}(w/y))) \vee \\ & \exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \neq z_2) \wedge \\ & [R(\boldsymbol{x},\boldsymbol{y}) \equiv TC(e)](\exists z_1,z_2,z_3(first_t(z_1) \wedge last_t(z_2) \wedge \bar{t_0}(z_3/z) \\ & \qquad \wedge R(z_1,z_3,z_2,z))) \end{aligned}$$

Let $\mathcal{M}$ be a finite structure and β a variable assignment. The three disjunctive components of $\bar{r}$ correspond to the cases that there

1. is no element satisfying the translated range formula t^{tc},
2. is exactly one element satisfying the translated range formula t^{tc},
3. are at least two elements satisfying the translated range formula t^{tc}.

Obviously the first two cases, have been properly dealt with. We assume from now on $(\mathcal{M},\beta) \models \exists z_1,z_2(t^{tc}(z_1/x) \wedge t^{tc}(z_2/x) \wedge z_1 \neq z_2)$. We will use the notation from Definition 4, clause 3 for the evaluation of $\mathbf{iterate}(x:S; y:T = t_0 \mid u)t$ in $(\mathcal{M},\beta)$. Let further r_e denote the transitive closure of the relation induced by e on M^2. By the definition of e we have for any i, $1 \leq i < n$ and any pair (b_1,b_2) in M^2:

$$(\mathcal{M},\beta) \models e[a_i,m_i,b_1,b_2] \text{ iff } b_1 = a_{i+1}, b_2 = m_{i+1}$$

This implies for all $(b_1,b_2) \in M^2$:

$$r_e(a_1,m_0,b_1,b_2) \text{ iff } b_1 = a_i \text{ and } b_2 = m_i \text{ for some } i, 1 \leq i \leq n$$

From this it is easy to see that the formula
$[R(\boldsymbol{x},\boldsymbol{y}) \equiv TC(e)](\exists z_1,z_2,z_3(first_t(z_1) \wedge last_t(z_2) \wedge \bar{t_0}(z_3/z) \wedge R(z_1,z_3,z_2,z)))$
is true in $(\mathcal{M},\beta)$ if and only if $\beta(z) = m_n$.

∎

Open Questions

We conclude with a set of open questions, whose answers will shed more light on the status of the new logic $\mathcal{L}_{it}$. All questions concern finite structures only unless explicitely mentioned otherwise.

1. Can transitive closure be definied in $\mathcal{L}_{it}$?
2. Compare $\mathcal{L}_{it}$ to the Dynamic Logic whose programs are for-programs. This necessitates to adjust Definition 7, since the Kripke structures needed for the interpretation of formulas in Dynamic logic are different from the structures introduced in Definition 3.
3. Compare $\mathcal{L}_{it}$ to logics with a least-fix-point operator.

One wonders if in Definition 2 one could not have introduced the iterate operator with more than one accumulator, i.e. an operator of the form $\mathbf{iterate}(x; y^1 = t^1, \ldots, y^k = t^k \mid u^1, \ldots, u^k)t$. The result of this operator will be read off from the first accumulator. Formally we modify clause 3 of Definition 4 as follows:

The interpretation $m = I_\beta(\mathbf{iterate}(x; y^1 = t^1, \ldots, y^k = t^k \mid u^1, \ldots, u^k)t)$ is computed as follows.

Let $\{a_1, \ldots, a_n\} = \{a \in I(S_1) \mid I_{\beta^a_x}(t) = true\}$ with $a_1 < \ldots < a_n$.

We define m_i^j for $1 \leq j \leq k$, $0 \leq i < n$ by:

1. $m_0^j = I_\beta(t^j)$
2. $m_{i+1}^j = I_{\beta^{i+1}}(u^j)$

where

$$\beta^{i+1}(z) = \begin{cases} \beta(z) \text{ if } z \neq x, y \\ a_{i+1} \text{ if } z = x \\ m_i^j \quad \text{if } z = y^j \end{cases}$$

Now we set $m = m_n^1$.

Let us denote by L_{it-n} the logic which allows iterators with k accumulators for all $k \leq n$. Thus $L_{it-1} = L_{it}$

5. Is L_{it-n+1} more xpressive than L_{it-n}?

6. This is more a task than a question. The addition of a linear order to all structures strongly influences the expressive power of $\mathcal{L}_{it}$. Are there ways to avoid this?
7. We have restricted the semantics of $\mathcal{L}_{it}$ to finite structures, mainly for the reason that otherwise iterate-terms may not terminate. Extend the definition of $\mathcal{L}_{it}$ by allowing terms with undefined values and waive the finiteness restriction.

References

1. ACM, editor. *Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA '99*, volume 34 (10) of *ACM SIGPLAN notices*. ACM, ACM Press, 1999.

2. R. France. A problem-oriented analysis of basic uml static modeling concepts. In ACM [1].
3. M. Gogolla and M. Richters. On constraints and queries in UML. In Schader and Korthaus [15], pages 109–121.
4. A. Hamie. A formal semantics for checking and analysing UML models. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, *Proceedings of the OOPSLA'98 Workshop on Formalizing UML. Why? How?*, 1998.
5. A. Hamie, J. Howse, and S. Kent. Navigation expressions in object-oriented modelling. In *FASE'98*.
6. A. Hamie, J. Howse, and S. Kent. Interpreting the object constraint language. In *Proceedings of Asia Pacific Conference in Software Engineering*. IEEE Press, July 1998.
7. N. Immerman. Languages that capture complexity classes. *SIAM J. of Computing*, 16(4):760–778, 1987.
8. S.-K. Kim and D. Carrington. Formalizing the UML class diagram using Object-Z. volume 1723 of *LNCS*, pages 83–98. Springer, 1999.
9. P. Krishnan. Consistency Checks for UML. In *Proceedings of the Asia Pacific Software Engineering Conference (APSEC 2000)*, pages 162–169, December 2000.
10. L. Mandel and M. V. Cengarle. On the expressive power of OCL. In *FM'99 - Formal Methods. World Congress on Formal Methods in the Development of Computing Systems, Toulouse, France, September 1999. Proceedings, Volume I*, volume 1708 of *LNCS*, pages 854–874. Springer, 1999.
11. OMG. Object constraint language specification, version 1.3. chapter 7 in [12]. OMG Document, March 2000.
12. OMG. OMG unified modeling language spezification, version 1.3. OMG Document, March 2000.
13. J.-E. Pin. Logic on words. *Bulletin of the EATCS*, 54:145–165, 1994.
14. M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In T. W. Ling, S. Ram, and M. L. Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*, pages 449–464. Springer, Berlin, LNCS 1507, 1998.
15. M. Schader and A. Korthaus, editors. *The Unified Modeling Language: technical aspects and applications*. Physica-Verlag, 1998.
16. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Progress in Theoretical Computer Science. Birkhäuser, 1994.
17. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science. Vol. B : Formal Models and Semantics*, pages 135–192. Elsevier, Amsterdam, 1990.
18. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modelling with UML*. Object Technology Series. Addison-Wesley, Reading/MA, 1999.

Constructive Foundations for Featherweight Java

Thomas Studer*

Crosspoint Informatik AG
Zentrumsplatz 1
CH-3322 Schönbühl, Switzerland
thomas.studer@crosspoint.ch

Abstract. In this paper we present a recursion-theoretic denotational semantics for Featherweight Java. Our interpretation is based on a formalization of the object model of Castagna, Ghelli and Longo in a predicative theory of types and names. Although this theory is proof-theoretically weak, it allows to prove many properties of programs written in Featherweight Java. This underpins Feferman's thesis that impredicative assumptions are not needed for computational practice.

Keywords: object-oriented programming, denotational semantics, reasoning about programs, Featherweight Java, explicit mathematics

1 Introduction

The question of the mathematical meaning of a program is usually asked to gain more insight into the language the program is written in. This may be to bring out subtle issues in language design, to derive new reasoning principles or to develop an intuitive abstract model of the programming language under consideration so as to aid program development. Moreover, a precise semantics is also needed for establishing certain properties of programs (often related to some aspects concerning security) with mathematical rigor.

As far as the Java language is concerned, most of the research on its semantics is focused on the operational approach (cf. e.g. Stärk, Schmid and Börger [36], Cenciarelli, Knapp, Reus and Wirsing [11], Drossopoulou, Eisenbach and Khurshid [12], Nipkow and Oheimb [30], and Syme [40]). Notable exceptions are Oheimb [31] who introduces a Hoare-style calculus for Java as well as Alves-Foss and Lam [2] who present a denotational semantics which is, as usual, based on *domain-theoretic* notions, cf. e.g. Fiore, Jung, Moggi, O'Hearn, Riecke, Rosolini and Stark [19] for a recent survey on domains and denotational semantics. Also, the projects aiming at a verification of Java programs using modern CASE tools and theorem provers have to make use of a formalization of the Java language (cf. e.g. the KeY approach by Ahrendt, Baar, Beckert, Giese, Habermalz, Hähnle, Menzel and Schmitt [1] as well as the LOOP project by Jacobs, van den Berg, Huisman, van Berkum, Hensel and Tews [25]).

* Research supported by the Swiss National Science Foundation. This paper is a part of the author's dissertation thesis [38].

R. Kahle, P. Schroeder-Heister, and R. Stärk (Eds.): PTCS 2001, LNCS 2183, pp. 202–238, 2001.

The main purpose of the present paper is the study of a *recursion-theoretic* denotational semantics for Featherweight Java, called FJ. Igarashi, Pierce and Wadler [24,23] have proposed this system as a minimal core calculus for Java, making it easier to understand the consequences of extensions and variations. For example, they employ it to prove type safety of an extension with generic classes as well as to obtain a precise understanding of inner classes. Ancona and Zucca [4] present a module calculus where the module components are class declarations written in Featherweight Java; and the introductory text by Felleisen and Friedman [18] shows that many useful object-oriented programs can be written in a purely functional style à la Featherweight Java.

In order to give a denotational semantics for FJ, we need to formalize an object model. Often, models for statically typed object-oriented programming languages are based on a highly *impredicative* type theory. Bruce, Cardelli and Pierce [7] for example use $\mathsf{F}^{\omega}_{<:}$ as common basis to compare different object encodings. A new approach to the foundations of object-oriented programming has been proposed by Castagna, Ghelli and Longo [8,10,21] who take overloading and subtyping as basic rather than encapsulation and subtyping. Although in its full generality this approach leads to a new form of impredicativity, see Castagna [8] and Studer [39], only a *predicative* variant of their system is needed in order to model object-oriented programming. This predicative object model will be our starting point for constructing a denotational semantics for Featherweight Java in a theory of types and names.

Theories of types and names, or explicit mathematics, have originally been introduced by Feferman [13,14] to formalize Bishop style constructive mathematics. In the sequel, these systems have gained considerable importance in proof theory, particularly for the proof-theoretic analysis of subsystems of second order arithmetic and set theory. More recently, theories of types and names have been employed for the study of functional and object-oriented programming languages. In particular, they have been shown to provide a unitary axiomatic framework for representing programs, stating properties of programs and proving properties of programs. Important references for the use of explicit mathematics in this context are Feferman [15,16,17], Kahle and Studer [29], Stärk [34,35], Studer [37,39] as well as Turner [43,44]. Beeson [6] and Tatsuta [41] make use of realizability interpretations for systems of explicit mathematics to prove theorems about program extraction.

Feferman [15] claims that impredicative comprehension principles are not needed for applications in computational practice. Further evidence for this is also given by Turner [44] who presents computationally weak but highly expressive theories, which suffice for constructive functional programming. In our paper we provide constructive foundations for Featherweight Java in the sense that our denotational semantics for FJ will be formalized in a constructive theory of types and names using the predicative object model of Castagna, Ghelli and Longo [10]. This supports Feferman's thesis that impredicative assumptions are not needed. Although our theory is proof-theoretically weak we can prove soundness of our semantics with respect to subtyping, typing and reductions.

Moreover, the theory of types and names we use has a recursion-theoretic interpretation. Hence, computations in FJ will be modeled by ordinary computations. For example, a non-terminating computation is not interpreted by a function, which yields $\perp$ as result, but by a *partial* function which does not terminate, either.

The plan of the present paper is as follows. In the next section we introduce the general framework of theories of types and names and we show how to construct fixed point types in it. Further we recall a theorem about a least fixed point operator in explicit mathematics, which will be the crucial ingredient of our construction. The presentation of Featherweight Java in Section 3 is included in order to make this paper self-contained. The overloading based object model we employ for our interpretation is introduced in Section 4. Section 5 is concerned with the study of some examples written in FJ which will motivate our denotational semantics as presented in Section 6. Section 7 contains the soundness proofs of our semantics with respect to subtyping, typing and reductions. A conclusion sums up what we have achieved and suggest further work, in particular the extension to the dynamic definition of new classes.

2 Theories of Types and Names

Explicit Mathematics has been introduced by Feferman [13] for the study of constructive mathematics. In the present paper, we will not work with Feferman's original formalization of these systems; instead we treat them as theories of types and names as developed in Jäger [26]. First we will present the base theory EETJ. Then we will extend it with the principle of dependent choice and show that this implies the existence of certain fixed points. Last but not least we are going to add axioms about computability and the statement that everything is a natural number. These two additional principles make the definition of a least fixed point operator possible.

2.1 Basic Notions

The theory of types and names which we will consider in the sequel is formulated in the two sorted language $\mathcal{L}$ about individuals and types. It comprises *individual variables* $a, b, c, f, g, h, x, y, z, \ldots$ as well as *type variables* $A, B, C, X, Y, Z, \ldots$ (both possibly with subscripts).

The language $\mathcal{L}$ includes the *individual constants* k, s (combinators), $\mathsf{p}, \mathsf{p}_0, \mathsf{p}_1$ (pairing and projections), 0 (zero), s_N (successor), p_N (predecessor), d_N (definition by numerical cases) and the constant c (computation). There are additional individual constants, called *generators*, which will be used for the uniform representation of types. Namely, we have a constant c_e (elementary comprehension) for every natural number e, as well as the constants j (join) and dc (dependent choice).

The *individual terms* $(r, s, t, r_1, s_1, t_1, \ldots)$ of $\mathcal{L}$ are built up from the variables and constants by means of the function symbol $\cdot$ for (partial) application. We

use (st) or st as an abbreviation for $(s \cdot t)$ and adopt the convention of association to the left, this means $s_1 s_2 \dots s_n$ stands for $(\dots(s_1 \cdot s_2)\dots s_n)$. Finally, we define general n tupling by induction on $n \geq 2$ as follows:

$$(s_1, s_2) := \mathsf{p} s_1 s_2 \quad \text{and} \quad (s_1, s_2, \dots, s_{n+1}) := (s_1, (s_2, \dots, s_{n+1})).$$

The *atomic formulas* of $\mathcal{L}$ are $s{\downarrow}$, $\mathsf{N}(s)$, $s = t$, $s \in U$ and $\Re(s, U)$. Since we work with a logic of partial terms, it is not guaranteed that all terms have values, and $s{\downarrow}$ is read as s *is defined* or s *has a value*. Moreover, $\mathsf{N}(s)$ says that s is a natural number, and the formula $\Re(s, U)$ is used to express that the individual s *represents* the type U or is a *name* of U.

The *formulas* of $\mathcal{L}$ are generated from the atomic formulas by closing against the usual propositional connectives as well as quantification in both sorts. A formula is called *elementary* if it contains neither the relation symbol $\Re$ nor bound type variables. The following table contains a list of useful abbreviations, where F is an arbitrary formula of $\mathcal{L}$:

$$\begin{array}{rcl}
s \simeq t & := & s{\downarrow} \vee t{\downarrow} \to s = t, \\
s \neq t & := & s{\downarrow} \wedge t{\downarrow} \wedge \neg(s = t), \\
s \in \mathsf{N} & := & \mathsf{N}(s), \\
(\exists x \in A) F(x) & := & \exists x (x \in A \wedge F(x)), \\
(\forall x \in A) F(x) & := & \forall x (x \in A \to F(x)), \\
f \in (A \to B) & := & (\forall x \in A) fx \in B, \\
A \subset B & := & \forall x (x \in A \to x \in B), \\
A = B & := & A \subset B \wedge A \subset B, \\
f \in (A \rightharpoonup B) & := & \forall x (x \in A \wedge fx{\downarrow} \to fx \in B), \\
x \in A \cap B & := & x \in A \wedge x \in B, \\
s \mathrel{\dot\in} t & := & \exists X (\Re(t, X) \wedge s \in X), \\
s \mathrel{\dot\subset} t & := & (\forall x \mathrel{\dot\in} s) x \mathrel{\dot\in} t, \\
s \doteq t & := & s \mathrel{\dot\subset} t \wedge t \mathrel{\dot\subset} s, \\
(\exists x \mathrel{\dot\in} s) F(x) & := & \exists x (x \mathrel{\dot\in} s \wedge F(x)), \\
(\forall x \mathrel{\dot\in} s) F(x) & := & \forall x (x \mathrel{\dot\in} s \to F(x)), \\
\Re(s) & := & \exists X \Re(s, X), \\
f \in (\Re \to \Re) & := & \forall x (\Re(x) \to \Re(fx)).
\end{array}$$

The vector notation $\vec{\mathcal{Z}}$ is sometimes used to denote finite sequences $\mathcal{Z}_1, \dots \mathcal{Z}_n$ of expressions. The length of such a sequence $\vec{\mathcal{Z}}$ is then either given by the context or irrelevant. For example, for $\vec{U} = U_1, \dots, U_n$ and $\vec{s} = s_1, \dots, s_n$ we write

$$\begin{aligned}
\Re(\vec{s}, \vec{U}) &:= \Re(s_1, U_1) \wedge \dots \wedge \Re(s_n, U_n), \\
\Re(\vec{s}) &:= \Re(s_1) \wedge \dots \wedge \Re(s_n).
\end{aligned}$$

Now we introduce the theory EETJ which provides a framework for explicit elementary types with join. Its logic is Beeson's [5] classical *logic of partial terms*

for individuals and classical logic for types. The logic of partial terms takes into account the possibility of undefined terms, i.e. terms which represent non-terminating computations. Scott [33] has given a logic similar to the logic of partial terms, but he treats existence like an ordinary predicate. Troelstra and van Dalen [42] give a discussion about the different approaches to partial terms.

Among the main features of the logic of partial terms are its *strictness axioms* stating that if a term has a value, then all its subterms must be defined, too. This corresponds to a call-by-value evaluation strategy, where all arguments of a function must first be fully evaluated before the final result will be computed. Stärk [34,35] examines variants of the logic of partial terms which also allow of call-by-name evaluation.

The nonlogical axioms of EETJ can be divided into the following three groups.

I. Applicative axioms. These axioms formalize that the individuals form a partial combinatory algebra, that we have paring and projections and the usual closure conditions on the natural numbers as well as definition by numerical cases.

(1) $\mathsf{k}ab = a$,
(2) $\mathsf{s}ab{\downarrow} \wedge \mathsf{s}abc \simeq ac(bc)$,
(3) $\mathsf{p}_0 a{\downarrow} \wedge \mathsf{p}_1 a{\downarrow}$,
(4) $\mathsf{p}_0(a,b) = a \wedge \mathsf{p}_1(a,b) = b$,
(5) $0 \in \mathsf{N} \wedge (\forall x \in \mathsf{N})(\mathsf{s}_\mathsf{N} x \in \mathsf{N})$,
(6) $(\forall x \in \mathsf{N})(\mathsf{s}_\mathsf{N} x \neq 0 \wedge \mathsf{p}_\mathsf{N}(\mathsf{s}_\mathsf{N} x) = x)$,
(7) $(\forall x \in \mathsf{N})(x \neq 0 \rightarrow \mathsf{p}_\mathsf{N} x \in \mathsf{N} \wedge \mathsf{s}_\mathsf{N}(\mathsf{p}_\mathsf{N} x) = x)$,
(8) $a \in \mathsf{N} \wedge b \in \mathsf{N} \wedge a = b \rightarrow \mathsf{d}_\mathsf{N} xyab = x$,
(9) $a \in \mathsf{N} \wedge b \in \mathsf{N} \wedge a \neq b \rightarrow \mathsf{d}_\mathsf{N} xyab = y$.

II. Explicit representation and extensionality. The following are the usual ontological axioms for systems of explicit mathematics. They state that each type has a name, that there are no homonyms and that $\Re$ respects the extensional equality of types. Note that the representation of types by their names is intensional, while the types themselves are extensional in the usual set-theoretic sense.

(10) $\exists x \Re(x, A)$,
(11) $\Re(s, A) \wedge \Re(s, B) \rightarrow A = B$,
(12) $A = B \wedge \Re(s, A) \rightarrow \Re(s, B)$.

III. Basic type existence axioms.

Elementary Comprehension. Let $F(x, \vec{y}, \vec{Z})$ be an elementary formula of $\mathcal{L}$ with at most the indicated free variables and with Gödelnumber e for any fixed Gödelnumbering, then we have the following axioms:

(13) $\Re(\vec{b}) \rightarrow \Re(\mathsf{c}_e(\vec{a}, \vec{b}))$,
(14) $\Re(\vec{b}, \vec{T}) \rightarrow \forall x(x \mathrel{\dot\in} \mathsf{c}_e(\vec{a}, \vec{b}) \leftrightarrow F(x, \vec{a}, \vec{T}))$.

With elementary comprehension we get a universal type V containing every individual. Simply let F be the elementary formula $x = x$ and apply the above axioms.

Join.

(15) $\Re(a) \wedge (\forall x \mathrel{\dot\in} a)\Re(fx) \rightarrow \Re(\mathsf{j}(a,f)) \wedge \Sigma(a,f,\mathsf{j}(a,f))$.

In this axiom the formula $\Sigma(a,f,b)$ means that b names the disjoint union of f over a, i.e.

$$\Sigma(a,f,b) := \forall x(x \mathrel{\dot\in} b \leftrightarrow \exists y \exists z(x = (y,z) \wedge y \mathrel{\dot\in} a \wedge z \mathrel{\dot\in} fy)).$$

It is a well-known result that we can introduce λ abstraction and recursion using the combinator axioms for k and s, cf. Beeson [5] or Feferman [13].

Theorem 1.

1. *For every variable x and every term t of $\mathcal{L}$, there exists a term $\lambda x.t$ of $\mathcal{L}$ whose free variables are those of t, excluding x, such that*

$$\mathsf{EETJ} \vdash \lambda x.t{\downarrow} \wedge (\lambda x.t)\, x \simeq t \text{ and } \mathsf{EETJ} \vdash s{\downarrow} \rightarrow (\lambda x.t)\, s \simeq t[s/x].$$

2. *There exists a term rec of $\mathcal{L}$ such that*

$$\mathsf{EETJ} \vdash \mathsf{rec}\, f{\downarrow} \wedge \forall x(\mathsf{rec}\, f\, x \simeq f\, (\mathsf{rec}\, f)\, x).$$

Now we introduce non-strict definition by cases (cf. e.g. Beeson [5]). Observe that if $\mathsf{d_N}abcd{\downarrow}$, then $a{\downarrow}$ and $b{\downarrow}$ hold by strictness. However, we often want to define a function by cases so that it is defined if one case holds, even if the value that would have been computed in the other case is undefined. Hence we let $\mathsf{d_s}abcd$ stand for the term $\mathsf{d_N}(\lambda z.a)(\lambda z.b)cd0$ where the variable z does not occur in the terms a and b. We will use the following notation for non-strict definition by cases

$$\mathsf{d_s}abcd \simeq \begin{cases} a & \text{if } c = d, \\ b & \text{else.} \end{cases}$$

This notation already anticipates the axiom $\forall x\mathsf{N}(x)$, otherwise we should add $\mathsf{N}(c) \wedge \mathsf{N}(d)$ as a premise; and of course, strictness still holds with respect to u and v. We have $\mathsf{d_s}rsuv{\downarrow} \rightarrow u{\downarrow} \wedge v{\downarrow}$. If u or v is undefined, then $\mathsf{d_s}rsuv$ is also undefined. However, if r is a defined term and u and v are defined natural numbers that are equal, then $\mathsf{d_s}rsuv = r$ holds even if s is not defined. In the sequel we employ type induction on the natural numbers which is given by the following axiom ($\mathsf{T\text{-}I_N}$):

$$\forall X(0 \in X \wedge (\forall x \in \mathsf{N})(x \in X \rightarrow \mathsf{s_N}x \in X) \rightarrow (\forall x \in \mathsf{N})x \in X).$$

2.2 Fixed Point Types

The classes of Java will be modeled by types in explicit mathematics. Since the Java classes may be defined by mutual recursion, i.e. class `A` may contain an attribute of class `B` and vice versa, their interpretations have to be given as fixed point types in our theory of types and names. They can be constructed by the principle of *dependent choice* (dc). These axioms have been proposed by Jäger and their proof-theoretic analysis has been carried out by Probst [32].

Dependent choice.

(dc.1) $\Re(a) \wedge f \in (\Re \to \Re) \to \mathsf{dc}(a, f) \in (\mathsf{N} \to \Re)$,
(dc.2) $\Re(a) \wedge f \in (\Re \to \Re) \to$
$\mathsf{dc}(a, f)0 \simeq a \wedge (\forall n \in \mathsf{N})\mathsf{dc}(a, f)(n + 1) \simeq f(\mathsf{dc}(a, f)n)$.

First, let us introduce some notation. By primitive recursion we can define in $\mathsf{EETJ} + (\mathsf{T\text{-}I_N})$ the usual relations $<$ and $\leq$ on the natural numbers. The i^{th} section of U is defined by $(U)_i := \{y \mid (i, y) \in U\}$. If s is a name for U, then $(s)_i$ represents the type $(U)_i$. By abuse of notation, we let the formula $(s)_i \in U$ stand for $\mathsf{p_0}s = i \wedge \mathsf{p_1}s \in U$. The context will ensure that it is clear how to read $(s)_i$. Product types are defined according to the definition of n tupling by $S_1 \times S_2 := \{(x, y) \mid x \in S_1 \wedge y \in S_2\}$ and

$$S_1 \times S_2 \times \cdots \times S_{n+1} := S_1 \times (S_2 \times \cdots \times S_{n+1}).$$

We define projection functions π_i^k for $k \geq 2$ and $1 \leq i \leq k$ so that

$$\pi_i^k(s_1, \ldots, s_k) \simeq s_i.$$

A *fixed point specification* is a system of formulas of the form

$$\begin{aligned}(X)_1 &= Y_{11} \times \cdots \times Y_{1m_1}\\ &\vdots\\ (X)_n &= Y_{n1} \times \cdots \times Y_{nm_n}\end{aligned}$$

where each Y_{ij} may be any type variable other than X or of the form

$$\{x \in X \mid \mathsf{p_0}x = k_1\} \cup \cdots \cup \{x \in X \mid \mathsf{p_0}x = k_l\}$$

for $k_i \leq n$. Those Y_{ij} which are just a type variable other than X are called *parameters* of the specification.

Our aim is to show that for every fixed point specification there exists a fixed point satisfying it and this fixed point can be named uniformly in the parameters of its specification.

Assume we are given a fixed point specification as above with parameters $\vec{Y}$. Then we find by elementary comprehension that there exists a closed individual term t of $\mathcal{L}$ so that EETJ proves for all $\vec{a}$ whose length is equal to the number of parameters of the specification:

1. $\Re(\vec{a}) \wedge \Re(b) \to \Re(t(\vec{a}, b))$,
2. $\Re(\vec{a}, \vec{Y}) \wedge \Re(b, X) \to$
$\forall x(x \mathrel{\dot\in} t(\vec{a}, b) \leftrightarrow (x)_1 \in Y_{11} \times \cdots \times Y_{1m_1} \vee \ldots \vee$
$(x)_n \in Y_{n1} \times \cdots \times Y_{nm_n})$.

In the following we assume $\Re(\vec{a})$ and let a_{ij} denote that element of $\vec{a}$ which represents Y_{ij}. The term $\lambda x.t(\vec{a}, x)$ is an operator form mapping names to names. Note that it is monotonic, i.e.

$$b \mathrel{\dot\subset} c \to t(\vec{a}, b) \mathrel{\dot\subset} t(\vec{a}, c). \tag{1}$$

Starting from the empty type, represented by $\emptyset$, this operation can be iterated in order to define the stages of the inductive definition of our fixed point. To do so, we define a function f by:

$$f(\vec{a}, n) \simeq \mathsf{dc}(\emptyset, \lambda x.t(\vec{a}, x))n.$$

As a direct consequence of (dc.1) we find $(\forall n \in \mathsf{N})\Re(f(\vec{a}, n))$. Hence, we let J be the type represented by $\mathsf{j}(\mathsf{nat}, \lambda x.f(\vec{a}, x))$. Making use of $(\mathsf{T\text{-}I_N})$ we can prove

$$(\forall n \in \mathsf{N})\forall x((n, x) \in J \to (n + 1, x) \in J)$$

and therefore

$$(\forall m \in \mathsf{N})(\forall n \in \mathsf{N})(m \leq n \ \to \ f(\vec{a}, m) \mathrel{\dot{\subset}} f(\vec{a}, n)). \tag{2}$$

We define the fixed point $\mathsf{FP} := \{x \mid (\exists n \in \mathsf{N})(n, x) \in J\}$. By the uniformity of elementary comprehension and join there exists a closed individual term fp so that $\mathsf{fp}(\vec{a})$ is a name for FP, i.e. the fixed point can be represented uniformly in its parameters. A trivial corollary of this definition is

$$(\exists n \in \mathsf{N})(x \mathrel{\dot{\in}} f(\vec{a}, n)) \leftrightarrow x \mathrel{\dot{\in}} \mathsf{fp}(\vec{a}). \tag{3}$$

The following theorem states that FP is indeed a fixed point of t. We employ $(s)_{ij} \in U$ as abbreviation for $\mathsf{p}_0 s = i \wedge \pi_j^{m_i}(\mathsf{p}_1 s) \in U$.

Theorem 2. *It is provable in* $\mathsf{EETJ} + (\mathsf{dc}) + (\mathsf{T\text{-}I_N})$ *that* FP *is a fixed point satisfying the fixed point specification, i.e.*

$$\Re(\vec{a}) \to \forall x(x \mathrel{\dot{\in}} \mathsf{fp}(\vec{a}) \leftrightarrow x \mathrel{\dot{\in}} t(\vec{a}, \mathsf{fp}(\vec{a})).$$

Proof. Assume $x \mathrel{\dot{\in}} \mathsf{fp}(\vec{a})$. By (3) there exists a natural number n so that $x \mathrel{\dot{\in}} f(\vec{a}, n)$. By (2) we find $x \in f(\vec{a}, n+1)$ and by the definition of f we get $f(\vec{a}, n+1) = t(\vec{a}, f(\vec{a}, n))$. By (3) we obtain $f(\vec{a}, n) \mathrel{\dot{\subset}} \mathsf{fp}(\vec{a})$ and with (1) we conclude $x \in t(\vec{a}, \mathsf{fp}(\vec{a}))$. Next, we show $\forall x(x \mathrel{\dot{\in}} t(\vec{a}, \mathsf{fp}(\vec{a})) \to x \mathrel{\dot{\in}} \mathsf{fp}(\vec{a}))$. Let $x \mathrel{\dot{\in}} t(\vec{a}, \mathsf{fp}(\vec{a}))$, i.e. we have for all i, j with $1 \leq i \leq n$ and $1 \leq j \leq m_i$ either $(x)_{ij} \mathrel{\dot{\in}} a_{ij}$ or by (3)

$$\begin{aligned}(x)_{ij} \in \{y \mid (\exists n \in \mathsf{N})y \mathrel{\dot{\in}} f(\vec{a}, n) \wedge \mathsf{p}_0 y = k_1\} \cup \cdots \cup \\ \{y \mid (\exists n \in \mathsf{N})y \mathrel{\dot{\in}} f(\vec{a}, n) \wedge \mathsf{p}_0 y = k_l\}\end{aligned}$$

depending on the specification. Since f is monotonic there exists a natural number n so that for all i, j with $1 \leq i \leq n$ and $1 \leq j \leq m_i$ we have either $(x)_{ij} \mathrel{\dot{\in}} a_{ij}$ or $(x)_{ij}$ is an element of

$$\{y \mid y \mathrel{\dot{\in}} f(\vec{a}, n) \wedge \mathsf{p}_0 y = k_1\} \cup \cdots \cup \{y \mid y \mathrel{\dot{\in}} f(\vec{a}, n) \wedge \mathsf{p}_0 y = k_l\}$$

depending on the specification and this implies $x \mathrel{\dot{\in}} f(\vec{a}, n+1)$. Hence we conclude by (3) that $x \mathrel{\dot{\in}} \mathsf{fp}(\vec{a})$ holds. □

2.3 Least Fixed Point Operator

As shown in Theorem 1 the combinatory axioms of EETJ provide a term rec which solves recursive equations. However, it is not provable that the solution obtained by rec is minimal. Here we are going to extend EETJ + (T-I$_{\mathsf{N}}$) with axioms about computability (Comp) and the statement that everything is a natural number. The resulting system allows to define a *least* fixed point operator and therefore it will be possible to show that recursively defined methods belong to a certain function space, cf. Kahle and Studer [29].

Computability. These axioms are intended to capture the idea that convergent computations should converge in finitely many steps. In the formal statement of the axioms the expression $\mathsf{c}(f,x,n) = 0$ can be read as "the computation fx converges in n steps." The idea of these axioms is due to Friedman (unpublished) and discussed in Beeson [5]. Note that these axioms can be satisfied in the usual recursion-theoretic model. The constant c can be interpreted by the characteristic function of Kleene's T predicate.

(Comp.1) $\forall f \forall x (\forall n \in \mathsf{N})(\mathsf{c}(f,x,n) = 0 \vee \mathsf{c}(f,x,n) = 1)$,
(Comp.2) $\forall f \forall x (fx{\downarrow} \leftrightarrow (\exists n \in \mathsf{N})\mathsf{c}(f,x,n) = 0)$.

In addition we will restrict the universe to natural numbers. This axiom is needed to construct the least fixed point operator. Of course, it is absolutely in the spirit of a recursion-theoretic interpretation.

Everything is a number. Formally, this is given by $\forall x \mathsf{N}(x)$.

For the rest of this section LFP will be the theory

$$\mathsf{EETJ} + (\mathsf{Comp}) + \forall x \mathsf{N}(x) + (\mathsf{T\text{-}I}_{\mathsf{N}}).$$

We are going to define ordering relations $\sqsubseteq_T$ for certain types T. The meaning of $f \sqsubseteq_T g$ is that f is smaller than g with respect to the usual pointwise ordering of functions, e.g. we have

$$f \sqsubseteq_{A \rightharpoonup B} g \rightarrow (\forall x \in A)(fx{\downarrow} \rightarrow fx = gx).$$

Definition 1. *Let $A_1, \ldots, A_n, B_1, \ldots, B_n$ be types Further, let $\mathcal{T}$ be the type $(A_1 \rightharpoonup B_1) \cap \cdots \cap (A_n \rightharpoonup B_n)$. We define:*

$$\begin{aligned} f \sqsubseteq g &:= f{\downarrow} \rightarrow f = g, \\ f \sqsubseteq_{\mathcal{T}} g &:= \bigwedge_{1 \le i \le n} (\forall x \in A_i) fx \sqsubseteq gx), \\ f \cong_{\mathcal{T}} g &:= f \sqsubseteq_{\mathcal{T}} g \wedge g \sqsubseteq_{\mathcal{T}} f. \end{aligned}$$

Definition 2. *Let $\mathcal{T}$ be as given in Definition 1. A function $f \in (\mathcal{T} \rightarrow \mathcal{T})$ is called* $\mathcal{T}$ monotonic, if

$$(\forall g \in \mathcal{T})(\forall h \in \mathcal{T})(g \sqsubseteq_{\mathcal{T}} h \rightarrow fg \sqsubseteq_{\mathcal{T}} fh).$$

Using the rec term we will find a fixed point for every operation g. But as we have mentioned above we cannot prove in EETJ that this is a least fixed point, and of course, there are terms g that do not have a least fixed point. However, there exists a closed individual term l of $\mathcal{L}$ so that LFP proves that $\mathsf{l}g$ is the least fixed point of a monotonic functional $g \in (\mathcal{T} \to \mathcal{T})$. A proof of the following theorem can be found in Kahle and Studer [29].

Theorem 3. *There exists a closed individual term* l *of* $\mathcal{L}$ *such that we can prove in* LFP *that if* $g \in (\mathcal{T} \to \mathcal{T})$ *is* $\mathcal{T}$ *monotonic for* $\mathcal{T}$ *given as in Definition 1, then*

1. $\mathsf{l}g \in \mathcal{T}$,
2. $\mathsf{l}g \cong_{\mathcal{T}} g(\mathsf{l}g)$,
3. $f \in \mathcal{T} \wedge gf \cong_{\mathcal{T}} f \to \mathsf{l}g \sqsubseteq_{\mathcal{T}} f$.

Now we define the theory PTN about programming with types and names as the union of all these axioms:

$$\mathsf{PTN} := \mathsf{EETJ} + (\mathsf{dc}) + (\mathsf{Comp}) + \forall x \mathsf{N}(x) + (\mathsf{T\text{-}I_N}).$$

The axioms about computability can be interpreted in the usual recursion-theoretic model, see Beeson [5] or Kahle [28]. This means that applications $a \cdot b$ in $\mathcal{L}$ are translated into $\{a\}(b)$, where $\{n\}$ for $n = 0, 1, 2, 3, \ldots$ is a standard enumeration of the partial recursive functions. In fact, the computability axioms are motivated by Kleene's T predicate which is a ternary primitive recursive relation on the natural numbers so that $\{a\}(\vec{m}) \simeq n$ holds if and only if there exists a computation sequence u with $\mathsf{T}(a, \langle \vec{m} \rangle, u)$ and $(u)_0 = n$. Hence, it can be used to verify the axioms in a recursion-theoretic interpretation; and of course, $\forall x \mathsf{N}(x)$ will also be satisfied in such a model.

Probst [32] presents a recursion-theoretic model for the system $\mathsf{EETJ} + (\mathsf{dc}) + (\mathsf{T\text{-}I_N})$ and shows that the proof-theoretic ordinal of this theory is $\varphi\omega 0$, i.e. it is slightly stronger than Peano arithmetic but weaker than Martin-Löf type theory with one universe ML_1 or the system $\mathsf{EETJ} + (\mathcal{L}\text{-}\mathsf{I_N})$ of explicit mathematics with elementary comprehension, join and full induction on the natural numbers.

These two constructions can be combined in order to get a recursion-theoretic model for PTN so that computations in PTN are modeled by ordinary recursion-theoretic functions. We obtain that PTN is proof-theoretically still equivalent to $\mathsf{EETJ} + (\mathsf{dc}) + (\mathsf{T\text{-}I_N})$. Hence, PTN is a predicative theory which is proof-theoretically much weaker than the systems that are usually used to talk about object-oriented programming, for most of these calculi are extensions of system F that already contains full analysis, cf. e.g. Bruce, Cardelli and Pierce [7]. Nevertheless, PTN is sufficiently strong to model Featherweight Java and to prove many properties of the represented programs. In PTN we can also prove soundness of our interpretation with respect to subtyping, typing and reductions.

3 Featherweight Java

Featherweight Java is a minimal core calculus for Java proposed by Igarashi, Pierce and Wadler [24] for the formal study of an extension of Java with parameterized classes. Igarashi and Pierce [23] employed Featherweight Java also

to obtain a precise understanding of inner classes. FJ is a minimal core calculus in the sense that as many features of Java as possible are omitted, while maintaining the essential flavor of the language and its type system. Nonetheless, this fragment is large enough to include many useful programs. In particular, most of the examples in Felleisen and Friedman's text [18] are written in the purely functional style of Featherweight Java. In this section we will present the formulation of Featherweight Java given in [23].

Syntax. The abstract syntax of FJ class declarations, constructor declarations, method declarations and expressions is given by:

$$
\begin{aligned}
\texttt{CL} &::= \texttt{class C extends C } \{\overline{\texttt{C f}};\ \texttt{K M}\}\\
\texttt{K} &::= \texttt{C } (\overline{\texttt{C f}})\ \{\texttt{super}(\overline{\texttt{f}});\ \texttt{this.}\overline{\texttt{f}}\ =\ \overline{\texttt{f}};\}\\
\texttt{M} &::= \texttt{C m}(\overline{\texttt{C x}})\ \{\texttt{return e};\}\\
\texttt{e} &::= \texttt{x}\\
&\quad \mid \texttt{e.f}\\
&\quad \mid \texttt{e.m}(\overline{\texttt{e}})\\
&\quad \mid \texttt{new C}(\overline{\texttt{e}})\\
&\quad \mid \texttt{(C)e}
\end{aligned}
$$

The meta-variables `A,B,C,D,E` range over class names, `f` and `g` range over field names, `m` ranges over method names, `x` ranges over variable names and `d,e` range over expressions (all possibly with subscripts). `CL` ranges over class declarations, `K` ranges over constructor declarations and `M` ranges over method declarations. We assume that the set of variables includes the special variable `this`, but that `this` is never used as the name of an argument to a method.

We write $\overline{\texttt{f}}$ as shorthand for $\texttt{f}_1, \ldots, \texttt{f}_n$ (and similarly for $\overline{\texttt{C}}, \overline{\texttt{x}}, \overline{\texttt{e}}$, etc.) and we use $\overline{\texttt{M}}$ for $\texttt{M}_1 \ldots \texttt{M}_n$ (without commas). The empty sequence is written as $\bullet$ and $\sharp(\overline{\texttt{x}})$ denotes the length of the sequence $\overline{\texttt{x}}$. Operations on pairs of sequences are abbreviated in the obvious way, e.g. "$\overline{\texttt{C}}\ \overline{\texttt{f}}$" stands for "$\texttt{C}_1\ \texttt{f}_1, \ldots, \texttt{C}_n\ \texttt{f}_n$" and "$\overline{\texttt{C}}\ \overline{\texttt{f}};$" is a shorthand for "$\texttt{C}_1\ \texttt{f}_1; \ldots; \texttt{C}_n\ \texttt{f}_n;$" and similarly "$\texttt{this.}\overline{\texttt{f}} = \overline{\texttt{f}};$" abbreviates "$\texttt{this.f}_1 = \texttt{f}_1; \ldots; \texttt{this.f}_n = \texttt{f}_n;$". We assume that sequences of field declarations, parameter names and method declarations contain no duplicate names.

A *class table* CT is a mapping from class names `C` to class declarations `CL`. A program is a pair $(CT, \texttt{e})$ of a class table and an expression. In the following we always assume that we have a *fixed* class table CT which satisfies the following sanity conditions:

1. $CT(\texttt{C}) = \texttt{class C}\ldots$ for every `C` in the domain of CT, i.e. the class name `C` is mapped to the declaration of the class `C`,
2. `Object` is not an element of the domain of CT,
3. every class `C` (except `Object`) appearing anywhere in CT belongs to the domain of CT,

4. there are no cycles in the subtype relation induced by CT, i.e. the $<:$ relation is antisymmetric.

Subtyping. The following rules define the subtyping relation $<:$ which is induced by the class table. Note that every class defined in the class table has a super class, declared with `extends`.

$$\mathtt{C} <: \mathtt{C}$$

$$\frac{\mathtt{C} <: \mathtt{D} \qquad \mathtt{D} <: \mathtt{E}}{\mathtt{C} <: \mathtt{E}}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\ldots\}}{\mathtt{C} <: \mathtt{D}}$$

Computation. These rules define the reduction relation $\longrightarrow$ which models field accesses, method calls and casts. In order to look up fields and method declarations in the class table we use some auxiliary functions that will be defined later on. We write $\mathtt{e}_0[\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{e}/\mathtt{this}]$ for the result of simultaneously replacing $\mathtt{x}_1$ by $\mathtt{d}_1, \ldots, \mathtt{x}_n$ by $\mathtt{d}_n$ and `this` by $\mathtt{e}$ in the expression $\mathtt{e}_0$.

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\mathtt{new\ C}(\overline{\mathtt{e}}).\mathtt{f_i} \longrightarrow \mathtt{e_i}}$$

$$\frac{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e}_0)}{\mathtt{new\ C}(\overline{\mathtt{e}}).\mathtt{m}(\overline{\mathtt{d}}) \longrightarrow \mathtt{e}_o[\overline{\mathtt{d}}/\overline{\mathtt{x}}, \mathtt{new\ C}(\overline{\mathtt{e}})/\mathtt{this}]}$$

$$\frac{\mathtt{C} <: \mathtt{D}}{(\mathtt{D})\mathtt{new\ C}(\overline{\mathtt{e}}) \longrightarrow \mathtt{new\ C}(\overline{\mathtt{e}})}$$

We say that an expression $\mathtt{e}$ is in *normal form* if there is no expression $\mathtt{d}$ so that $\mathtt{e} \longrightarrow \mathtt{d}$.

Now we present the typing rules for expressions, method declarations and class declarations. An environment Γ is a finite mapping from variables to class names, written $\overline{\mathtt{x}} : \overline{\mathtt{C}}$. Again, we employ some auxiliary functions which will be given later. Stupid casts (the last of the expression typing rules) are included only for technical reasons, cf. Igarashi, Pierce and Wadler [24]. The Java compiler will reject expressions containing stupid casts as ill typed. This is expressed by the hypothesis *stupid warning* in the typing rule for stupid casts.

Expression typing.

$$\Gamma \vdash \mathtt{x} \in \Gamma(\mathtt{x})$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{e}_0.\mathtt{f_i} \in \mathtt{C_i}}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{e_o} \in \mathtt{C}_0 \\ \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}}) \in \mathtt{C}}$$

$$\frac{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}} \qquad \Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}}{\Gamma \vdash \mathtt{new\ C}(\overline{\mathtt{e}}) \in \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{D} <: \mathtt{C}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{C} <: \mathtt{D} \qquad \mathtt{C} \neq \mathtt{D}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}}$$

$$\frac{\Gamma \vdash \mathtt{e}_0 \in \mathtt{D} \qquad \mathtt{C} \not<: \mathtt{D} \qquad \mathtt{D} \not<: \mathtt{C} \qquad \text{stupid warning}}{\Gamma \vdash (\mathtt{C})\mathtt{e}_0 \in \mathtt{C}}$$

Method typing.

$$\frac{\begin{array}{c}\overline{\mathtt{x}} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{e}_0 \in \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}_0 \\ CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\ldots\} \\ \text{if } \mathit{mtype}(\mathtt{m},\mathtt{D}) = \overline{\mathtt{D}} \to \mathtt{D}_0, \text{ then } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C}_0 = \mathtt{D}_0\end{array}}{\mathtt{C}_0\ \mathtt{m}\ (\overline{\mathtt{C}}\ \overline{\mathtt{x}})\ \{\mathtt{return\ e}_0;\}\ \text{OK in C}}$$

Class typing.

$$\frac{\begin{array}{c}\mathtt{K = C}(\overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}})\ \{\mathtt{super}(\overline{\mathtt{g}});\ \mathtt{this.}\overline{\mathtt{f}} = \overline{\mathtt{f}};\} \\ \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}} \qquad \overline{\mathtt{M}}\ \text{OK in C}\end{array}}{\mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\}\ \text{OK}}$$

We define the auxiliary function which are used in the rules for computation and typing.

Field lookup.

$$\mathit{fields}(\mathtt{Object}) = \bullet$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathit{fields}(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}}}{\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}},\ \overline{\mathtt{C}}\ \overline{\mathtt{f}}}$$

Method type lookup.

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{B\ m}\ (\overline{\mathtt{B}}\ \overline{\mathtt{x}})\ \{\mathtt{return\ e};\}\ \text{belongs to } \overline{\mathtt{M}}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{B}} \to \mathtt{B}}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{m}\ \text{is not defined in } \overline{\mathtt{M}}}{\mathit{mtype}(\mathtt{m}, \mathtt{C}) = \mathit{mtype}(\mathtt{m}, \mathtt{D})}$$

Method body lookup.

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{B\ m}\ (\overline{\mathtt{B}}\ \overline{\mathtt{x}})\ \{\mathtt{return\ e};\}\ \text{belongs to } \overline{\mathtt{M}}}{\mathit{mbody}(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e})}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\overline{\mathtt{C}}\ \overline{\mathtt{f}};\ \mathtt{K}\ \overline{\mathtt{M}}\} \qquad \mathtt{m} \text{ is not defined in } \overline{\mathtt{M}}}{mbody(\mathtt{m}, \mathtt{C}) = mbody(\mathtt{m}, \mathtt{D})}$$

We call a Featherweight Java expression $\mathtt{e}$ *well-typed* if $\Gamma \vdash \mathtt{g} \in \mathtt{C}$ can be derived for some environment Γ and some class $\mathtt{C}$.

Igarashi, Pierce and Wadler [24] prove that if an FJ program is well-typed, then the only way it can get stuck is if it reaches a point where it cannot perform a downcast. This is stated in the following theorem about progress.

Theorem 4. *Suppose* $\mathtt{e}$ *is a well-typed expression.*

1. *If the expression* $\mathtt{e}$ *is of the form* $\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}).\mathtt{f}$ *or contains such a subexpression, then* $fields(\mathtt{C}_0) = \overline{\mathtt{D}}\ \overline{\mathtt{f}}$ *and* $\mathtt{f} \in \overline{\mathtt{f}}$.
2. *If* $\mathtt{e}$ *is of the form* $\mathtt{new}\ \mathtt{C}_0(\overline{\mathtt{e}}).\mathtt{m}(\overline{\mathtt{d}})$ *or contains such a subexpression, then* $mbody(\mathtt{m}, \mathtt{C}_0) = (\overline{\mathtt{x}}, \mathtt{e}_0)$ *and* $\sharp(\overline{\mathtt{x}}) = \sharp(\overline{\mathtt{d}})$.

4 The Object Model

In this section we present the object model of Castagna, Ghelli and Longo [8,10] which will be employed to interpret Featherweight Java in explicit mathematics. In his book [8], Castagna introduces a kernel object-oriented language KOOL and defines its semantics via an interpretation in a meta-language λ_object for which an operational semantics is given. Our construction will be very similar in spirit to Castagna's interpretation, although we will have to solve many new problems since we start with an already existing language and will end up with a recursion-theoretic model for it.

In the object model we use, the state of an object is separated from its methods. Only the fields of an object are bundled together as one unit, whereas the methods of an object are *not* encapsulated inside it. Indeed, methods are implemented as branches of global *overloaded* functions. If a message is sent to an object, then this message determines a function and this function will be applied to the receiving object. However, messages are not ordinary functions. For if the same message is sent to objects of different classes, then different methods may be retrieved, i.e. different code may be executed. Hence, messages represent overloaded functions: depending on the type of their argument (the object the message is passed to), a different method is chosen. Since this selection of the method is based on the dynamic type of the object, i.e. its type at run-time, we also have to deal with *late-binding*.

In our semantics of FJ, objects are modeled as pairs (type, value). This encoding of objects was proposed by Castagna [8] in order to interpret late-binding. The value component will be a record consisting of all the fields of the object and the type component codes the run-time type of the object. All the methods with the same name, although maybe defined in different classes, are combined to one overloaded function which takes the receiving object as an additional argument. Hence, the type information contained in the interpretation of the receiving object can be used to resolve this overloaded application, i.e. to select

the code to be executed. Since methods may be defined recursively, the interpreting function has to be given by recursion, too. As we will see later, in order to obtain a sound model with respect to typing, we even need a *least* fixed point operator for the definition of the semantics of methods. Since several methods may call each other by mutual recursion, we have to give their interpretation using one recursive definition treating all methods in one go.

As Castagna [8] notices, even λ calculi with overloading and late-binding do not posses enough structure to reason about object-oriented programs. Hence he introduces the meta-language λ_object which is still based on overloading and late-binding but which also is enriched with new features (e.g. commands to define new types or to handle the subtyping hierarchy) that are necessary to reproduce the constructs of a programming language. Therefore λ_object is an appropriate tool for representing object-oriented programs. However, in λ_object it is not possible to state (or even prove) properties of these programs.

Our theories of types and names have much more expressive power. They provide not only an axiomatic framework for representing programs, but also for stating and proving properties of programs. Nevertheless, from a proof-theoretic point of view they are quite weak (only a bit stronger than Peano Arithmetic) as demanded by Feferman [15,16,17] or Turner [44]. In its general form overloading and late-binding seem to be proof-theoretically very strong, cf. Studer [37]. However, our work shows that in order to model real object-oriented programming languages we do not need the full power of these principles; as already mentioned by Castagna, Ghelli and Longo [8,10] a predicative variant suffices for our practical purposes.

Last but not least, the theory of types and names we use in this paper has a standard recursion-theoretic model. Hence, our interpretation of FJ in explicit mathematics shows how computations in Featherweight Java can be seen as ordinary mathematical functions and the interpretation of the classes will be given by sets in the usual mathematical sense.

5 Evaluation Strategy and Typing

Now we study some examples written in Featherweight Java which will motivate our semantics for FJ as presented in the next section. We will focus on Java's evaluation strategy and on typing issues. In the last example the interplay of free variables, static types and late-binding is investigated.

Java features a call-by-value evaluation strategy, cf. the Java language specification by Gosling, Joy and Steele [22]. This corresponds to the strictness axioms of the logic of partial terms upon which explicit mathematics is built. They imply that an application only has a value, i.e. is terminating, if all its arguments have a value.

In Java we not only have non-terminating programs we also have run-time exceptions, for example when an illegal down cast should be performed. With respect to these features, Featherweight Java is much more coarse grained. There is no possibility to state that a program terminates and exceptions are completely

ignored. For good reasons, as we should say, since it is intended as a minimal core calculus for modeling Java's type system. However, this lack of expressiveness has some important consequences which will be studied in the sequel. Let us first look at the following example.

Example 1.

```
class A extends Object {
  A () { super(); }
  C m() {
    return this.m();
  }
}

class C extends Object {
    int x;
    A y;
    C (int a,A b) {
        super();
        this.x = a;
        this.y = b;
    }
}
```

Of course, `new A().m()` is a non-terminating loop. Although, if it is evaluated on an actual Java implementation, then we get after a short while a `java.lang.StackOverflowError` because of too many recursive method calls. In Featherweight Java `new A().m()` has no normal form which reflects the fact that it loops forever.

Now let `e` be the expression `new C(5,new A().m()).x`. Due to Java's call-by-value evaluation strategy, the computation of this expression will not terminate either, for `new A().m()` is a subexpression of `e` and has therefore to be evaluated first.

The operational semantics of Featherweight Java uses a non-deterministic small-step reduction relation which does not enforce a call-by-value strategy. Hence we have two different possibilities for reduction paths starting from `e`. If we adopt a call-by-value strategy, then we have to evaluate `new A().m()` first and we obtain an infinite reduction path starting from `e`. Since FJ's reduction relation is non-deterministic we also have the possibility to apply the computation rule for field access. If we decide to do so, then `e` reduces to 5 which is in normal form.

In theories of types and names we have the possibility to state that a computation terminates. The formula $t{\downarrow}$ expresses that t has a value, meaning the computation represented by t is terminating. Let $[\![\mathtt{e}]\!]$ be the interpretation of the expression `e`. In our mathematical model Java's call-by-value strategy is implemented by the strictness axioms, hence $\neg[\![\mathtt{e}]\!]{\downarrow}$ will be provable. Since 5 surely has a value we obtain $[\![\mathtt{e}]\!] \not\simeq 5$ although 5 is the normal form of `e`. This means that

in our interpretation we cannot model the non-deterministic reduction relation of Featherweight Java but we will implement a call-by-value strategy.

Non-terminating programs are not the only problem in modeling computations of Java. A second problem is the lack of a notion of run-time exception in Featherweight Java. For example, if a term is in normal form, then we cannot tell, whether this is the case because the computation finished properly, or because an illegal down-cast should be performed. It may even be the case that the final expression does not contain any down-casts at all, but earlier during the computation an exception should have been thrown. Let us illustrate this fact with the following example, where the class `C` is as in Example 1.

Example 2.

```
class main extends Object{
  public static void main (String arg[]) {
      System.out.println(new C(5,(A)(new Object())).x);
  }
}
```

If we run this `main` method, then Java throws the following exception:

```
java.lang.ClassCastException: java.lang.Object
        at main.main(main.java:4)
```

Whereas in Featherweight Java the expression

```
new C(5,(A)(new Object())).x
```

reduces to 5. This is due to the fact that the term (`C`)(`new Object`()), which causes the exception in Java, is treated as final value in Featherweight Java and therefore it can be used as argument in further method calls.

In our model we will introduce a special value `ex` to denote the result of a computation which throws an exception. An illegal down cast produces $(\mathsf{ex}, 0)$ as result and we can check every time an expression is used as argument in a method invocation or in a constructor call whether its value is $(\mathsf{ex}, 0)$ or not. If it is not, then the computation can continue; but if an argument value represents an exception, then the result of the computation is this exception value. Therefore in our model we can distinguish whether an exception occurred or not. For example, the above expression evaluates to $(\mathsf{ex}, 0)$. We will have

$$\begin{aligned}[\![\texttt{(A)(new Object())}]\!] &= \mathsf{cast}\,\texttt{A}^*\,[\![\texttt{new Object()}]\!] \\ &= \mathsf{cast}\,\texttt{A}^*\,(\texttt{Object}^*, 0) \\ &= (\mathsf{ex}, 0)\end{aligned}$$

since $\mathsf{sub}(\texttt{Object}^*, \texttt{A}^*) \neq 1$, i.e. `Object` is not a subclass of `A`. Later, we will define the operation $*$ so that if `A` is a name for a class, then $\texttt{A}^*$ is a numeral in $\mathcal{L}$. Then we define a term `sub` which decides the subclass relation. That is for two class names `A` and `B`, we have $\mathsf{sub}(\texttt{A}^*, \texttt{B}^*) = 1$ if and only if `A` is a subclass of `B`. The term `cast` will be used to model casts. If o is the interpretation of an

object and A is a class to which this object should be casted, then $\mathsf{cast}\,\mathtt{A}^*\,o$ is the result of this cast. The term cast uses sub to decide whether it is a legal cast. If it is not, as in the above example, the cast will return $(\mathsf{ex}, 0)$.

From these considerations it follows that we cannot prove soundness of our model construction with respect to reductions as formalized in Featherweight Java. However, we are going to equip FJ with a restricted reduction relation $\longrightarrow'$ which enforces a call-by-value evaluation strategy as it is used in the Java language and which also respects illegal down casts. With respect to this new notion of reduction we will be able to prove that our semantics adequately models FJ computations.

In Featherweight Java we cannot talk about the termination of programs. As usual in type systems for programming languages the statement "expression e has type T" has to be read as "if the computation of e terminates, then its result is of type T". Let A be the class of Example 1. Then in FJ `new A().m()` $\in$ C is derivable, although the expression `new A().m()` denotes a non-terminating loop. Hence in our model we will have to interpret $\mathtt{e} \in \mathtt{C}$ as $[\![\mathtt{e}]\!]{\downarrow} \to [\![\mathtt{e}]\!] \in [\![\mathtt{C}]\!]$.

As we have seen before, the computation of e may result in an exception. In this case we have $[\![\mathtt{e}]\!] = (\mathsf{ex}, 0)$ which is a defined value. Hence, by our interpretation of the typing relation, we have to include $(\mathsf{ex}, 0)$ to the interpretation of every type.

In the following we consider a class B which is the same as A except that the result type of the method m is changed to D.

```
class A extends Object {
  A () { super(); }
  C m() {
    return this.m();
  }
}

class B extends Object {
  B () { super(); }
  D m() {
    return this.m();
  }
}
```

Since the method bodies for m are the same in both classes A and B we can assume that the interpretations of `(new A()).m()` and `(new B()).m()` will be the same, that is

$$[\![\texttt{new A().m()}]\!] \simeq [\![\texttt{new B().m()}]\!].$$

In this example the classes C and D may be chosen arbitrarily. In particular, they may be disjoint, meaning that maybe, there is no object belonging to both of them. Hence, if our modeling of the typing relation is sound, it follows that we are in the position to prove that the computation of `new B().m()` is non-terminating, that is $\neg[\![\texttt{(new B()).m()}]\!]{\downarrow}$.

Usually, in lambda calculi such recursive functions are modeled using a fixed point combinator. In continuous λ-models, such as $\mathsf{P}\omega$ or D_∞, these fixed point combinators are interpreted by *least* fixed point operators and hence one can show that certain functions do not terminate. In applicative theories on the other hand, recursive equations are solved with the rec term provided by the recursion theorem. Unfortunately, one cannot prove that this operator yields a least fixed point; and hence, it is not provable that certain recursive functions do not terminate. Therefore we have to employ the special term I to define the semantics of FJ expressions. Since this term provides a least solution to certain fixed point equations, it will be possible to show $\neg[\![\texttt{(new B()).m()}]\!]{\downarrow}$ which is necessary for proving soundness of our interpretation with respect to typing.

Now we are going to examine the role of free variables in the context of static types and late-binding. In the following example let `C` be an arbitrary class with no fields.

Example 3.

```
class A extends Object{
  A () { super(); }
  C m() {
    return this.m();
  }
}

class B extends A{
  B () { super(); }
  C m() {
    return new C();
  }
}
```

As in Example 1 class `A` defines the method `m` which does not terminate. Class `B` extends `A`, hence it is a subclass of `A`, and it overrides method `m`. Here `m` creates a new object of type `C` and returns it to the calling object.

Let `x` be a free variable with (static) type `A`. The rules for method typing guarantee that the return type of `m` cannot be changed by the overriding method; and by the typing rules of FJ we can derive $\texttt{x:A} \vdash \texttt{x.m()} \in \texttt{C}$. As we have seen before this means "if `x.m()` yields a result, then it belongs to `C`." Indeed, as a consequence of Java's late-binding evaluation strategy, knowing only the static type `A` of `x` we cannot tell whether in `x.m()` the method `m` defined in class `A` or the one of class `B` will be executed. Hence we do not know whether this computation terminates or not. Only if we know the object which is referenced by `x` we can look at its dynamic type and then say, by the rules of method body lookup, which method actually gets called.

This behavior has the consequence that there are FJ expression in normal form whose interpretation will not have a value. For example, `x.m()` is in normal form, but maybe `x` references an object of type `A` and in this case the

interpretation of `x.m()` will not have a value. Therefore we conclude that only the interpretation of a *closed* term in normal form will always be defined.

6 Interpreting Featherweight Java

Assume we are given a program written in Featherweight Java. This consist of a fixed class table CT and an expression `e`. In this section we will show how to translate such a program into the language of types and names. This allows us to state and prove properties of FJ programs. In the sequel we will work with the theory PTN of explicit mathematics.

We generally assume that all classes and methods occurring in our fixed class table CT are well-typed. This means for every class `C` of CT we can derive `class C...OK` by the rules for class typing and for every method `m` defined in this class we can derive `...m...OK IN C` by the rules for method typing.

The *basic types* of Java such as `boolean`, `int`, ... are not included in FJ. However, EETJ provides a rich type structure which is well-suited to model these basic data types, cf. e.g. Feferman [15] or Jäger [27]. Hence we will include them in our modeling of Featherweight Java.

Let $*$ be an injective mapping from all the names for classes, basic types, fields and methods occurring in the class table CT into the numerals of $\mathcal{L}$. This mapping will be employed to handle the run-time type information of FJ terms as well as to model field access and method selection.

First we show how objects will be encoded as sequences in our theory of types and names. Let `C` be a class of our class table CT with $fields(\mathtt{C}) = \mathtt{D_1}\ \mathtt{g_1}, \ldots, \mathtt{D_n}\ \mathtt{g_n}$ and let $m_{\mathtt{C}}$ be the least natural number such that for all field names $\mathtt{g_j}$ occurring in $fields(\mathtt{C})$ we have ${\mathtt{g_j}}^* < m_{\mathtt{C}}$. An object of type `C` will be interpreted by a sequence $(\mathtt{C}^*, (s_1, \ldots, s_{m_{\mathtt{C}}}))$, where s_i is the interpretation of the field $\mathtt{g_j}$ if $i = {\mathtt{g_j}}^*$ and $s_i = 0$ if there is no corresponding field. In particular, we always have $s_{m_{\mathtt{C}}} = 0$. Note that in this model the type of an object is encoded in the interpretation of the object.

We have to find a way for dealing with invalid down casts. What should be the value of `(A) new Object()` in our model, when `A` is a class different from `Object`? In FJ the computation simply gets stuck, no more reductions will be performed. In our model we choose a natural number `ex` which is not in the range of * and set the interpretation of illegal down casts to $(\mathtt{ex}, 0)$. This allows us to distinguish them from other expressions using definition by cases on the natural numbers. Hence, every time when an expression gets evaluated we can check whether one of its arguments is the result of an illegal cast.

This is the reason why we will have to add run-time type information to elements of basic types, too. Let us look for example at the constant `17` of Java which surely is of type `int`. If it is simply modeled by the $\mathcal{L}$ term 17, then it might happen that $17 = (\mathtt{ex}, 0)$ and we could not decide whether this $\mathcal{L}$ term indicates that an illegal down cast occurred or whether it denotes the constant `17` of Java. On the other hand, if the Java constant `17` is modeled by $(\mathtt{int}^*, 17)$,

i.e. with run-time type information, then it is provably different from $(\mathsf{ex}, 0)$. The next example illustrates the coding of objects.

Example 4. Assume we have the following class modeling points.

```
class Point extends Object{
  int x;
  int y;

  Point (int a, int b) {
    super();
    this.x = a;
    this.y = b;
  }
}
```

Assume $*$ is such that $\mathtt{x}^* = 1$ and $\mathtt{y}^* = 3$. Hence, we have $m_{\mathtt{Point}} = 4$ An object of the class `Point` where `x=5` and `y=6` is now modeled by by the sequence $(\mathtt{Point}^*, ((\mathtt{int}^*, 5), 0, (\mathtt{int}^*, 6), 0))$.

In order to deal with the subtype hierarchy of FJ, we define a term sub modeling the subtype relation.

Definition 3 (of the term sub). *Let the term* sub *be so that for all* $a, b \in \mathsf{N}$ *we have:*

1. *If* a *or* b *codes a basic type, e.g.* $a = \mathtt{A}^*$ *for a basic type* $\mathtt{A}$*, and* $a = b$*, then* $\mathsf{sub}(a, b) = 1$.
2. *If* $\mathtt{C} <: \mathtt{D}$ *can be derived for two classes* $\mathtt{C}$ *and* $\mathtt{D}$ *and* $\mathtt{C}^* = a$ *as well as* $\mathtt{D}^* = b$ *hold, then* $\mathsf{sub}(a, b) = 1$.
3. *Otherwise we set* $\mathsf{sub}(a, b) = 0$.

Since CT is finite, sub can be defined using definition by cases on the natural numbers; recursion is not needed.

In the following, we will define a semantics for expression of Featherweight Java. In a first step, this semantics will be given only relative to a term invk which is used to model method calls. As we have shown in our discussion of the object model, we can define invk in a second step as the least fixed point of a recursive equation involving all methods occurring in our fixed class table.

Of course, if at some stage of a computation an invalid down cast occurs and we obtain $(\mathsf{ex}, 0)$ as intermediate result, then we have to propagate it to the end of the computation. Therefore all of the following terms are defined by distinguishing two cases: if none of the arguments equals $(\mathsf{ex}, 0)$, then the application will be evaluated; if one of the arguments is $(\mathsf{ex}, 0)$, then the result is also $(\mathsf{ex}, 0)$.

We we define a term proj in order to model field access.

Definition 4 (of the term proj). *Let the term* proj *be so that*

$$\mathsf{proj}\, i\, x \simeq \begin{cases} x & \mathsf{p}_0 x = \mathsf{ex}, \\ \mathsf{p}_0(\mathsf{tail}\, i\, (\mathsf{p}_1 x)) & \textit{otherwise}, \end{cases}$$

where tail *is defined by primitive recursion such that*

$$\mathsf{tail}\,1\,s \simeq s \qquad \mathsf{tail}\,(n+1)\,s \simeq \mathsf{p}_1(\mathsf{tail}\,n\,s)$$

for all natural numbers $n \geq 1$.

Hence for $i \leq n$ and $t \neq \mathsf{ex}$ we have

$$\mathsf{proj}\,i\,(t,(s_1,\ldots,s_n,s_{n+1})) \simeq s_i.$$

Next, we show how to define the interpretation of the keyword `new`.

Definition 5 (of the term new). *For every class* `C` *of our class table CT with*

$$fields(\mathtt{C}) = \mathtt{D_1}\ \mathtt{g_1},\ldots,\mathtt{D_n}\ \mathtt{g_n}$$

we find a closed $\mathcal{L}$ *term* $t_{\mathtt{C}}$ *such that:*

1. *If* $a_i{\downarrow}$ *holds for all* a_i $(i \leq n)$ *and if there is a natural number* j *such that* $\mathsf{p}_0 a_j = \mathsf{ex}$, *then we get* $t_{\mathtt{C}}(a_1,\ldots,a_n) = a_j$ *where* j *is the least number satisfying* $\mathsf{p}_0 a_j = \mathsf{ex}$.
2. *Else we find* $t_C(a_1,\ldots,a_n) \simeq (\mathtt{C}^*,(b_1,\ldots,b_{m_{\mathtt{C}}}))$, *where* $b_i \simeq a_j$ *if there exists* $j \leq n$ *with* $i = \mathtt{g_j}^*$ *and* $b_i = 0$ *otherwise.*

Using definition by cases on the natural numbers we can build a term new *so that* $\mathsf{new}\,\mathtt{C}^*\,(\vec{s}) \simeq t_{\mathtt{C}}(\vec{s})$ *for every class* `C` *in CT.*

The next example shows how the terms $t_{\mathtt{C}}$ work.

Example 5. Consider the class `Point` of Example 4. We get

$$t_{\mathtt{Point}}((\mathtt{int}^*,5),(\mathtt{int}^*,6)) \simeq (\mathtt{Point}^*,((\mathtt{int}^*,5),0,(\mathtt{int}^*,6),0))$$

and

$$t_{\mathtt{Point}}((\mathtt{int}^*,5),(\mathsf{ex},0)) \simeq (\mathsf{ex},0).$$

If b is a term so that $\neg b{\downarrow}$ holds, then we get $\neg t_{\mathtt{Point}}(a,b){\downarrow}$ by strictness even if $a = (\mathsf{ex},0)$. The strictness principle of explicit mathematics implies Java's call-by-value strategy.

Again, using definition by cases we build a term cast.

Definition 6 (of the term cast). *Let the term* cast *be so that*

$$\mathsf{cast}\,a\,b \simeq \begin{cases} (\mathsf{ex},0) & \mathsf{sub}(\mathsf{p}_0 b, a) = 0, \\ b & \textit{otherwise.} \end{cases}$$

Now, we give the translation $[\![\mathsf{e}]\!]_{\mathsf{invk}}$ of a Featherweight Java expression `e` into an $\mathcal{L}$ term relative to a term invk.

Definition 7 (of the interpretation $[\![\mathsf{e}]\!]_{\mathsf{invk}}$ relative to invk). *For a sequence $\overline{\mathsf{e}} = \mathsf{e}_1, \ldots, \mathsf{e}_n$ we write $[\![\overline{\mathsf{e}}]\!]_{\mathsf{invk}}$ for $[\![\mathsf{e}_1]\!]_{\mathsf{invk}}, \ldots, [\![\mathsf{e}_n]\!]_{\mathsf{invk}}$. We assume that for every variable $\boldsymbol{x}$ of Featherweight Java there exists a corresponding variable x of $\mathcal{L}$ such that two different variables of FJ are mapped to different variables of $\mathcal{L}$. In particular, we suppose that our language $\mathcal{L}$ of types and names includes a variable this so that $[\![\boldsymbol{this}]\!]_{\mathsf{invk}} = this$.*

$$
\begin{aligned}
[\![\mathtt{x}]\!]_{\mathsf{invk}} &:= x \\
[\![\mathtt{e.f}]\!]_{\mathsf{invk}} &:= \mathsf{proj}\,\mathtt{f}^*\,[\![\mathtt{e}]\!]_{\mathsf{invk}} \\
[\![\mathtt{e.m}(\overline{\mathtt{f}})]\!]_{\mathsf{invk}} &:= \mathsf{invk}(\mathtt{m}^*, [\![\mathtt{e}]\!]_{\mathsf{invk}}, [\![\overline{\mathtt{f}}]\!]_{\mathsf{invk}}) \\
[\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!]_{\mathsf{invk}} &:= \mathsf{new}\,\mathtt{C}^*([\![\overline{\mathtt{e}}]\!]_{\mathsf{invk}}) \\
[\![(\mathtt{C})\mathtt{e}]\!]_{\mathsf{invk}} &:= \mathsf{cast}\,\mathtt{C}^*\,[\![\mathtt{e}]\!]_{\mathsf{invk}}
\end{aligned}
$$

In the following we are going to define the term invk which models method calls. To this aim, we have to deal with overloading and late-binding in explicit mathematics, cf. Studer [37,39].

Definition 8 (of overloaded functions). *Assume we are given n natural numbers $s_1, \ldots, s_n$. Using sub we build for each $j \leq n$ a term $\mathsf{min}^j_{s_1,\ldots,s_n}$ such that for all natural numbers s we have $\mathsf{min}^j_{s_1,\ldots,s_n}(s) = 0 \vee \mathsf{min}^j_{s_1,\ldots,s_n}(s) = 1$ and $\mathsf{min}^j_{s_1,\ldots,s_n}(s) = 1$ if and only if*

$$
\mathsf{sub}(s, s_j) = 1 \wedge \bigwedge_{\substack{1 \leq l \leq n \\ l \neq j}} (\mathsf{sub}(s, s_l) = 1 \rightarrow \mathsf{sub}(s_l, s_j) = 0).
$$

Hence, $\mathsf{min}^j_{s_1,\ldots,s_n}(s) = 1$ holds if s_j is a minimal element (with respect to sub) of the set $\{s_i \,|\, \mathsf{sub}(s, s_i) = 1 \wedge 1 \leq i \leq n\}$; and otherwise we have $\mathsf{min}^j_{s_1,\ldots,s_n}(s) = 0$.

We can define a term $\mathsf{over}_{s_1,\ldots,s_n}$ which combines several functions $f_1, \ldots, f_n$ to one overloaded function $\mathsf{over}_{s_1,\ldots,s_n}(f_1, \ldots, f_n)$ such that

$$
\mathsf{over}_{s_1,\ldots,s_n}(f_1, \ldots, f_n)\,(x, \vec{y}) \simeq \begin{cases} f_1(x, \vec{y}) & \mathsf{min}^1_{s_1,\ldots,s_n}(\mathsf{p}_0 x) = 1, \\ \vdots & \\ f_n(x, \vec{y}) & \mathsf{min}^n_{s_1,\ldots,s_n}(\mathsf{p}_0 x) = 1 \\ & \bigwedge\limits_{i<n} \mathsf{min}^i_{s_1,\ldots,s_n}(\mathsf{p}_0 x) \neq 1, \\ (\mathsf{ex}, 0) & \mathsf{p}_0 x = \mathsf{ex}. \end{cases}
$$

Next, we define the term r which gives the recursive equation which will be solved by invk.

Definition 9 (of the term r). *Assume the method $\mathtt{m}$ is defined exactly in the classes $\mathtt{C}_1, \ldots, \mathtt{C}_n$ and $mbody(\mathtt{m}, \mathtt{C}_i) = (\overline{\mathtt{x}}_i, \mathtt{e}_i)$ for all $i \leq n$. Assume further that $\overline{\mathtt{x}}_i$ is $\mathtt{x}_1, \ldots, \mathtt{x}_z$, then we can define an $\mathcal{L}$ term $g^{\mathsf{invk}}_{\mathsf{e}_i}$ so that we have for $\vec{b} = b_1, \ldots, b_z$:*

1. *If* $a{\downarrow}$ *and* $\vec{b}{\downarrow}$ *hold and if there is a natural number* j *such that* $\mathsf{p}_0 b_j = \mathsf{ex}$, *then we get* $g^{\mathsf{invk}}_{\mathsf{e}_i}(a, \vec{b}) = b_j$ *where* j *is the least number satisfying* $\mathsf{p}_0 b_j = \mathsf{ex}$.
2. *Else we find* $g^{\mathsf{invk}}_{\mathsf{e}_i}(a, \vec{b}) \simeq (\lambda this.\lambda[\![\overline{\mathtt{x}}_i]\!]_{\mathsf{invk}}.[\![\mathtt{e}_i]\!]_{\mathsf{invk}})a\vec{b}$.

We see that the terms $g^{\mathsf{invk}}_{\mathsf{e}_i}$ *depend on* invk. *Now we let the* $\mathcal{L}$ *term* r *be such that for every method* m *in our class table CT we have*

$$\mathsf{r}\,\mathsf{invk}(\mathtt{m}^*, x, \vec{y}) \simeq \mathsf{over}_{\mathtt{C}_1^*,\ldots,\mathtt{C}_n^*}(g^{\mathsf{invk}}_{\mathsf{e}_1}, \ldots, g^{\mathsf{invk}}_{\mathsf{e}_n})(x, \vec{y}). \tag{4}$$

We define the term invk to be the least fixed point of r.

Definition 10 (of the term invk). *We set* $\mathsf{invk} := \mathsf{I}\,\mathsf{r}$.

In the following we will write only $[\![\mathtt{e}]\!]$ for the translation of an expression e relative to the term r invk defined as above.

It remains to define the interpretation of Featherweight Java classes. Let us begin with the basic types. The example of the type `boolean` will show how we can use the type structure of explicit mathematics to model the basic types of Java. If we let 0 and 1 denote "false" and "true", respectively, then the interpretation $[\![\mathtt{boolean}]\!]$ of the basic type `boolean` is given by

$$\{(\mathtt{boolean}^*, b) \mid b = 0 \vee b = 1\}.$$

Here we see that an element $x \in [\![\mathtt{boolean}]\!]$ is pair whose first component carries the run-time type information of x, namely $\mathtt{boolean}^*$, and whose second component is the actual truth value. For the Java expressions `false` and `true` we can set

$$[\![\mathtt{false}]\!] = (\mathtt{boolean}^*, 0) \qquad [\![\mathtt{true}]\!] = (\mathtt{boolean}^*, 1).$$

We will interpret the classes of FJ as fixed point types in explicit mathematics satisfying the following fixed point specification.

Definition 11 (of the fixed point FP). *If the class table CT contains a class named* C *with* $\mathtt{C}^* = i$, *then the following formula is included in our specification:*

$$(X)_i = Y_{i1} \times \cdots \times Y_{im_{\mathtt{C}}},$$

where $m_{\mathtt{C}}$ *is again the least natural number such that for all field names* f *occurring in fields*(C) *we have* $\mathtt{f}^* < m_{\mathtt{C}}$. Y_{ij} *is defined according to the following three clauses:*

1. *If there is a basic type* D *and a field name* f *such that* D f *belongs to fields*(C), *then* Y_{if^*} *is equal to the interpretation of* D.
2. *If there is a class* D *and a field name* f *such that* D f *belongs to fields*(C) *and if* $E_1, \ldots, E_n$ *is the list of all classes* E_j *in CT for which* $E_j <: D$ *is derivable, then*

$$Y_{if^*} = \{(E_1^*, x) \mid x \in (X)_{E_1^*}\} \cup \cdots \cup \{(E_n^*, x) \mid x \in (X)_{E_n^*}\} \cup \{(\mathsf{ex}, 0)\}.$$

3. *If there is no field name* $\mathtt{f}$ *occurring in fields*$(\mathtt{C})$ *so that* $\mathtt{f}^* = j$*, then* Y_{ij} *is the universal type* V*, in particular we find* $Y_{im_\mathtt{C}} = \mathsf{V}$.

As we have shown before, in PTN *there provably exists a fixed point* FP *satisfying the above specification.*

Since our fixed class table CT contains only finitely many classes we can set up the following definition for the interpretation $[\![\mathtt{C}]\!]$ of a class $\mathtt{C}$.

Definition 12 (of the interpretation $[\![\mathtt{C}]\!]$ of a class $\mathtt{C}$). *If* $E_1, \ldots, E_n$ *is the list of all classes* E_i *in FJ for which* $E_i <: \mathtt{C}$ *is derivable, then*

$$[\![\mathtt{C}]\!] = \{(E_1^*, x) \mid x \in (\mathsf{FP})_{E_1^*}\} \cup \cdots \cup \{(E_n^*, x) \mid x \in (\mathsf{FP})_{E_n^*}\} \cup \{(\mathsf{ex}, 0)\}.$$

We include the value $(\mathsf{ex}, 0)$ to the interpretation of all classes because this simplifies the presentation of the proofs about soundness with respect to typing. Of course we could exclude $(\mathsf{ex}, 0)$ from the above types, which would be more natural, but then we had to treat it as special case in all the proofs.

Now we will present to examples for the definition of classes.

Example 6. We take the class `Point` given in Example 4 and extend it to a class `ColorPoint`.

```
class ColorPoint extends Point{
  String color;

  ColorPoint (int a, int b, String c) {
    super(a,b);
    this.color = c;
  }
}
```

Assume $*$ is such that $\mathtt{color}^* = 4$. Hence, we have $m_{\mathtt{ColorPoint}} = 5$ Consider an object of the class `ColorPoint` with `x=5`, `y=6` and `color="black"` where the string `"black"` is modeled by say 256. This object is now interpreted by the sequence

$$(\mathtt{Point}^*, ((\mathtt{int}^*, 5), 0, (\mathtt{int}^*, 6), (\mathtt{String}, 256), 0)).$$

Since the `Point` belongs to our class table, the fixed point specification contains a line

$$(X)_{\mathtt{Point}^*} = [\![\mathtt{int}]\!] \times (\mathsf{V} \times ([\![\mathtt{int}]\!] \times \mathsf{V})).$$

We obtain that the value of our *colored* point object

$$((\mathtt{int}^*, 5), (0, ((\mathtt{int}^*, 6), ((\mathtt{String}, 256), 0))))$$

belongs to the interpretation of `Point` since $((\mathtt{String}, 256), 0) \in \mathsf{V}$. This example illustrates why we take the product with V in the interpretation of objects. It guarantees that the model is sound with respect to subtyping, see Theorem 6 below. This encoding of objects in records is due to Cardelli.

The next example shows why we have to employ fixed points to model the classes.

Example 7. Consider a class `Node` pointing to a next `Node`.

```
class Node extends Object{
   Node next;

   Node(Node x){
     super();
     this.next=x;
   }
}
```

This class will be interpreted as fixed point of

$$X = (\{(\mathtt{Node}^*, x) \mid x \in X\} \cup \{(\mathsf{ex}, 0)\}) \times \mathsf{V}.$$

Iterating this operator form will yield a fixed point only after ω many steps. Therefore, one has to employ dependent choice in the construction of the fixed points modeling classes.

7 Soundness Results

In this section we will prove that our model for Featherweight Java is sound with respect to subtyping, typing and reductions. We start with a theorem about soundness with respect to subtyping which is a trivial consequence of the interpretation of classes.

Theorem 5. *For all classes* `C` *and* `D` *of the class table with* $\mathtt{C} <: \mathtt{D}$ *it is provable in* PTN *that* $[\![\mathtt{C}]\!] \subset [\![\mathtt{D}]\!]$.

The soundness of the semantics of the subtype relation does not depend on the fact that a type is interpreted as the union of all its subtypes. As the next theorem states, our model is sound with respect to subtyping if we allow to coerce the run-time type of an object into a super type. Coercions are operations which change the type of an object. In models of object-oriented programming these constructs can be used to give a semantics for features which are based on early-binding, cf. Castagna, Ghelli and Longo [8,9]. This is achieved by coercing the type of an object into its static type before selecting the best matching branch. In Java for example, the resolution of overloaded methods is based on static types, i.e. the choice of the function to be applied is based on early-binding.

Theorem 6. *For all classes* `C` *and* `D` *with* $\mathtt{C} <: \mathtt{D}$ *the following is provable in* PTN*:*

$$x \in [\![\mathtt{C}]\!] \wedge \mathsf{p}_0 x \neq \mathsf{ex} \rightarrow (\mathtt{D}^*, \mathsf{p}_1 x) \in [\![\mathtt{D}]\!].$$

Proof. By induction on the length of the derivation of $\mathtt{C} <: \mathtt{D}$ we show that $(\mathsf{FP})_{\mathtt{C}^*} \subset (\mathsf{FP})_{\mathtt{D}^*}$. The only non-trivial case is, when the following rule has been applied

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\ldots\}}{\mathtt{C} <: \mathtt{D}}.$$

So we assume $CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\ \{\ldots\}$. By our definition of FP we obtain

$$(\mathsf{FP})_{\mathtt{C}^*} = Y_{\mathtt{C}^*1} \times \cdots \times Y_{\mathtt{C}^* m_{\mathtt{C}}}$$

and $(\mathsf{FP})_{\mathtt{D}^*}$ is of the form $Y_{\mathtt{D}^*1} \times \cdots \times Y_{\mathtt{D}^* m_{\mathtt{D}}}$. By the rules for field lookup we know that if *fields*(D) contains $\mathtt{E\ g}$, then $\mathtt{E\ g}$ also belongs to *fields*(C). Therefore we have $m_{\mathtt{D}} \leq m_{\mathtt{C}}$ and for all $i < m_{\mathtt{D}}$ we get $Y_{\mathtt{C}^* i} \subset Y_{\mathtt{D}^* i}$ by the fixed point specification for FP and our general assumption that class typing is ok. Moreover, we obviously have

$$Y_{\mathtt{C}^* m_{\mathtt{D}}} \times \cdots \times Y_{\mathtt{C}^* m_{\mathtt{C}}} \subset Y_{\mathtt{D}^* m_{\mathtt{D}}} = \mathsf{V}.$$

Therefore, we conclude that the claim holds. □

Before proving soundness with respect to typing we have to show some preparatory lemmas.

Definition 13. *If* $\overline{\mathtt{D}}$ *is the list* $\mathtt{D}_1, \ldots, \mathtt{D}_n$*, then* $[\![\overline{\mathtt{D}}]\!]$ *stands for* $[\![\mathtt{D}_1]\!] \times \cdots \times [\![\mathtt{D}_n]\!]$*; and if* $\overline{\mathtt{e}} = \mathtt{e}_1, \ldots \mathtt{e}_n$*, then* $[\![\overline{\mathtt{e}}]\!]_{\mathsf{invk}} \in [\![\overline{\mathtt{D}}]\!]$ *means*

$$([\![\mathtt{e}_1]\!]_{\mathsf{invk}}, \ldots, [\![\mathtt{e}_n]\!]_{\mathsf{invk}}) \in [\![\mathtt{D}_1]\!] \times \cdots \times [\![\mathtt{D}_n]\!].$$

For $\Gamma = \mathtt{x}_1 : \mathtt{D}_1, \ldots, \mathtt{x}_n : \mathtt{D}_n$ *we set*

$$[\![\Gamma]\!]_{\mathsf{invk}} := [\![\mathtt{x}_1]\!]_{\mathsf{invk}} \in [\![\mathtt{D}_1]\!] \wedge \cdots \wedge [\![\mathtt{x}_n]\!]_{\mathsf{invk}} \in [\![\mathtt{D}_n]\!].$$

Definition 14. *We define the type* $\mathcal{T}$ *to be the intersection of all the types*

$$(\{\mathtt{m}^*\} \times [\![\mathtt{C}]\!] \times [\![\overline{\mathtt{D}}]\!]) \curvearrowright [\![\mathtt{B}]\!]$$

for all methods m *and all classes* C *occurring in CT with* $mtype(\mathtt{m}, \mathtt{C}) = \overline{\mathtt{D}} \to \mathtt{B}$.

The next lemma states that if we have an interpretation relative to a function h belonging to $\mathcal{T}$, then this interpretation is sound with respect to typing.

Lemma 1. *If* $\Gamma \vdash \mathtt{e} \in \mathtt{C}$ *is derivable in FJ, then we can prove in* PTN *that* $h \in \mathcal{T}$ *implies*

$$[\![\Gamma]\!]_h \wedge [\![\mathtt{e}]\!]_h\!\downarrow \to [\![\mathtt{e}]\!]_h \in [\![\mathtt{C}]\!].$$

Proof. Proof by induction on the derivation length of $\Gamma \vdash \mathtt{e} \in \mathtt{C}$. We assume $[\![\Gamma]\!]_h \wedge [\![\mathtt{e}]\!]_h\!\downarrow$ and distinguish the different cases for the last rule in the derivation of $\Gamma \vdash \mathtt{e} \in \mathtt{C}$:

1. $\Gamma \vdash \mathtt{x} \in \Gamma(\mathtt{x})$: trivial.
2. $\Gamma \vdash \mathtt{e.f}_i \in \mathtt{C}_i$: $[\![\mathtt{e.f}_i]\!]_h{\downarrow}$ implies $[\![\mathtt{e}]\!]_h{\downarrow}$ by strictness. Hence, we get by the induction hypothesis $[\![\mathtt{e}]\!]_h \in [\![\mathtt{C}_0]\!]$ and $\mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}$. By the definition of $[\![\mathtt{C}_0]\!]$ this yields $\mathsf{proj}\,\mathtt{f}_i^*\,[\![\mathtt{e}]\!]_h \in [\![\mathtt{C}_i]\!]$. Finally we conclude by $\mathsf{proj}\,\mathtt{f}_i^*\,[\![\mathtt{e}]\!]_h \simeq [\![\mathtt{e.f}_i]\!]_h$ that the claim holds.
3. $\Gamma \vdash \mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}}) \in \mathtt{C}$: by the induction hypothesis and Theorem 5 we obtain $[\![\overline{\mathtt{e}}]\!]_h \in [\![\overline{\mathtt{C}}]\!] \subset [\![\overline{\mathtt{D}}]\!]$ and $[\![\mathtt{e}_0]\!]_h \in [\![\mathtt{C}_0]\!]$. Moreover, we have

$$\mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C}.$$

   Hence we conclude by $h \in \mathcal{T}$ and $[\![\mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}})]\!]_h \simeq h(\mathtt{m}^*, [\![\mathtt{e}_0]\!]_h, [\![\overline{\mathtt{e}}]\!]_h)$ that the claim holds.
4. $\Gamma \vdash \mathtt{new\ C}(\overline{\mathtt{e}}) \in \mathtt{C}$: by the induction hypothesis and Theorem 5 we have $[\![\overline{\mathtt{e}}]\!]_h \in [\![\overline{\mathtt{C}}]\!] \subset [\![\overline{\mathtt{D}}]\!]$. Further we know $\mathit{fields}(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{f}}$. Therefore the claim holds by the definition of new.
5. If the last rule was an upcast, then the claim follows immediately from the induction hypothesis, the definition of the term cast and Theorem 5.
6. Assume the last rule was a downcast or a stupid cast. By the induction hypothesis we get $[\![\mathtt{e}]\!]_h \in [\![\mathtt{D}]\!]$. Then $\mathtt{D} \not<: \mathtt{C}$ implies $\mathsf{sub}(\mathtt{p}_0[\![\mathtt{e}]\!]_h, \mathtt{C}^*) = 0$. We get $[\![\mathtt{(C)e}]\!]_h \simeq (\mathsf{ex}, 0)$ by the definition of cast. Hence the claim holds. □

The following lemma says that our interpretation of FJ expressions is in accordance with the definedness ordering $\sqsubseteq_{\mathcal{T}}$.

Lemma 2. *If $\Gamma \vdash \mathtt{e} \in \mathtt{C}$ is derivable in FJ, then we can prove in* PTN *that $g, h \in \mathcal{T}$ and $g \sqsubseteq_{\mathcal{T}} h$ imply $[\![\Gamma]\!]_g \wedge [\![\mathtt{e}]\!]_g{\downarrow} \to [\![\mathtt{e}]\!]_g = [\![\mathtt{e}]\!]_h$.*

Proof. Proof by induction on the derivation length of $\Gamma \vdash \mathtt{e} \in \mathtt{C}$. Assume $[\![\Gamma]\!]_g$ and $[\![\mathtt{e}]\!]_g{\downarrow}$ hold. We distinguish the following cases:

1. $\Gamma \vdash \mathtt{x} \in \Gamma(\mathtt{x})$: trivial.
2. $\Gamma \vdash \mathtt{e}_0\mathtt{.f}_i \in \mathtt{C}_i$: we know $\Gamma \vdash \mathtt{e}_0 \in \mathtt{C}_0$. Hence we get by the induction hypothesis $[\![\mathtt{e}_0]\!]_g = [\![\mathtt{e}_0]\!]_h$ and therefore the claim holds.
3. $\Gamma \vdash \mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}}) \in \mathtt{C}$: we get $\Gamma \vdash \mathtt{e}_0 \in \mathtt{C}_0$, $\Gamma \vdash \overline{\mathtt{e}} \in \overline{\mathtt{C}}$ and

$$\mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \quad \text{as well as} \quad \overline{\mathtt{C}} <: \overline{\mathtt{D}}. \tag{5}$$

   Because of $[\![\mathtt{e}]\!]_g{\downarrow}$ we obtain $[\![\mathtt{e}_0]\!]_g{\downarrow}$ and $[\![\overline{\mathtt{e}}]\!]_g{\downarrow}$. Hence the induction hypothesis yields $[\![\mathtt{e}_0]\!]_g = [\![\mathtt{e}_0]\!]_h$ as well as $[\![\overline{\mathtt{e}}]\!]_g = [\![\overline{\mathtt{e}}]\!]_h$. Using Lemma 1, we get $[\![\Gamma]\!]_g \vdash [\![\mathtt{e}_0]\!]_g \in [\![\mathtt{C}_0]\!]$, $[\![\Gamma]\!]_g \vdash [\![\overline{\mathtt{e}}]\!]_g \in [\![\overline{\mathtt{C}}]\!]$ as well as $[\![\Gamma]\!]_g \vdash [\![\mathtt{e}]\!]_g \in [\![\mathtt{C}]\!]$. With (5), $g \in \mathcal{T}$, $h \in \mathcal{T}$ and $g \sqsubseteq_{\mathcal{T}} h$ we conclude

$$\begin{aligned} [\![\mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}})]\!]_g &= g(\mathtt{m}^*, [\![\mathtt{e}_0]\!]_g, [\![\overline{\mathtt{e}}]\!]_g) \\ &= h(\mathtt{m}^*, [\![\mathtt{e}_0]\!]_h, [\![\overline{\mathtt{e}}]\!]_h) \\ &= [\![\mathtt{e}_0\mathtt{.m}(\overline{\mathtt{e}})]\!]_h \end{aligned}$$

4. $\Gamma \vdash \mathtt{new\ C}(\overline{\mathtt{e}}) \in \mathtt{C}$: as in the second case, the claim follows immediately from the induction hypothesis.

5. If the last rule was a cast, then again the claim is a direct consequence of the induction hypothesis. □

Remark 1. In the logic of partial terms it is provable that

$$\forall x F \wedge t{\downarrow} \rightarrow F[t/x]$$

for all formulas F of $\mathcal{L}$, cf. e.g. Beeson [5].

Now, we want to show that $\mathsf{invk} \in \mathcal{T}$. First we prove $\mathsf{r} \in (\mathcal{T} \rightarrow \mathcal{T})$.

Lemma 3. *In* PTN *it is provable that* $\mathsf{r} \in (\mathcal{T} \rightarrow \mathcal{T})$.

Proof. Assume $h \in \mathcal{T}$ and let $(\mathtt{m}^*, c, \vec{d}) \in (\{\mathtt{m}^*\} \times [\![\mathtt{C}_0]\!] \times [\![\overline{\mathtt{D}}]\!])$ for a method $\mathtt{m}$ and classes $\mathtt{C}_0, \overline{\mathtt{D}}, \mathtt{C}$ with $mtype(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \rightarrow \mathtt{C}$. We have to show

$$\mathsf{r}h(\mathtt{m}^*, c, \vec{d}){\downarrow} \rightarrow \mathsf{r}h(\mathtt{m}^*, c, \vec{d}) \in [\![\mathtt{C}]\!].$$

So assume $\mathsf{r}h(\mathtt{m}^*, c, \vec{d}){\downarrow}$. By (4) we find

$$\mathsf{r}h(\mathtt{m}^*, c, \vec{d}) = \mathsf{over}_{\mathtt{C}_1^*, \ldots, \mathtt{C}_n^*}(g^h_{\mathtt{e}_1}, \ldots, g^h_{\mathtt{e}_n})(c, \vec{d}).$$

Hence, if $c = (\mathsf{ex}, 0)$ then we obtain $\mathsf{r}h(\mathtt{m}^*, c, \vec{d}) = (\mathsf{ex}, 0)$ and the claim holds. If $c \neq (\mathsf{ex}, 0)$ then we get $\mathsf{sub}(\mathsf{p}_0 c, \mathtt{C}_0) = 1$. By our interpretation of classes, there exists a class $\mathtt{B}$ such that $\mathtt{B} <: \mathtt{C}_0$, $\mathsf{p}_0 c = \mathtt{B}^*$ as well as $c \in [\![\mathtt{B}]\!]$. Let $mbody(\mathtt{m}, \mathtt{B}) = (\overline{\mathtt{x}}_i, \mathtt{e}_i)$. Hence we have

$$\mathsf{r}\, h\, (\mathtt{m}^*, c, \vec{d}) = g^h_{\mathtt{e}_i}(c, \vec{d}) = [\![\mathtt{e}_i]\!]_h[c/this, \vec{d}/\vec{x}_i]. \tag{6}$$

By the rules for method body lookup there exists a class $\mathtt{A}$ such that $\mathtt{B} <: \mathtt{A} <: \mathtt{C}_0$ and the method $\mathtt{m}$ is defined in $\mathtt{A}$ by the expression $\mathtt{e}_i$. By our general assumption that method typing is ok we obtain

$$\overline{\mathtt{x}} : \overline{\mathtt{D}}, \mathtt{this} : \mathtt{A} \vdash \mathtt{e}_i \in \mathtt{E}_0 \qquad \mathtt{E}_0 <: \mathtt{C}. \tag{7}$$

By Theorem 5 we get $c \in [\![A]\!]$ and therefore we conclude by $h \in \mathcal{T}$, (6), Lemma 1 and Remark 1 that $\mathsf{r}h(\mathtt{m}^*, c, \vec{d}) \in [\![\mathtt{C}]\!]$ holds. □

Now, we prove that r is $\mathcal{T}$ monotonic which implies $\mathsf{invk} \in \mathcal{T}$.

Lemma 4. *In* PTN *it is provable that* $\mathsf{invk} \in \mathcal{T}$.

Proof. We have defined invk as lr. Lemma 3 states $\mathsf{r} \in (\mathcal{T} \rightarrow \mathcal{T})$. Therefore it remains to show that r is $\mathcal{T}$ monotonic. Let $g, h \in \mathcal{T}$ such that $g \sqsubseteq_{\mathcal{T}} h$. We have to show $\mathsf{r}\, g \sqsubseteq_{\mathcal{T}} \mathsf{r}\, h$. So assume $\mathsf{r}g \in \mathcal{T}$. Now we have to show

$$\mathsf{r}h \in \mathcal{T}. \tag{8}$$

Moreover, we have to show for all methods $\mathtt{m}$ and classes $\mathtt{C}_0$, $\overline{\mathtt{D}}$ and $\mathtt{C}$ with $mtype(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C}$ that

$$(\forall x \in \{\mathtt{m}^*\} \times [\![\mathtt{C}_0]\!] \times [\![\overline{\mathtt{D}}]\!])\mathsf{r}\, g\, x \sqsubseteq_{\mathtt{C}} \mathsf{r}\, h\, x. \tag{9}$$

(8) is a direct consequence of Lemma 3. In order to show (9) we let $(\mathtt{m}^*, c, \vec{d}) \in \{\mathtt{m}^*\} \times [\![\mathtt{C}_0]\!] \times [\![\overline{\mathtt{D}}]\!]$ and $\mathsf{r}\, g\, (\mathtt{m}^*, c, \vec{d}) \in [\![\mathtt{C}]\!]$. It remains to show

$$\mathsf{r}\, g(\mathtt{m}^*, c, \vec{d}) = \mathsf{r}\, h\, (\mathtt{m}^*, c, \vec{d}). \tag{10}$$

As in Lemma 3 we find $\mathsf{r}\, g\, (\mathtt{m}^*, c, \vec{d}) = [\![\mathtt{e}_i]\!]_g[c/this, \vec{d}/\vec{x}_i]$ for some i and we have to show that this is equal to $[\![\mathtt{e}_i]\!]_h[c/this, \vec{d}/\vec{x}_i]$. By (7), which was a consequence of our general assumption that method typing is OK, and Lemma 2 we find

$$x \in [\![\overline{\mathtt{D}}]\!] \wedge this \in [\![\mathtt{C}_0]\!] \to [\![\mathtt{e}_i]\!]_g = [\![\mathtt{e}_i]\!]_h.$$

Hence by Remark 1 we obtain

$$[\![\mathtt{e}_i]\!]_g[c/this, \vec{d}/\vec{x}_i] = [\![\mathtt{e}_i]\!]_h[c/this, \vec{d}/\vec{x}_i]$$

and we finally conclude that (10) holds. □

The next theorem states that our model is sound with respect to typing.

Theorem 7. *If* $\Gamma \vdash \mathtt{e} \in \mathtt{C}$ *is derivable in FJ, then in* PTN *it is provable that*

$$[\![\Gamma]\!] \wedge [\![\mathtt{e}]\!]{\downarrow} \to [\![\mathtt{e}]\!] \in [\![\mathtt{C}]\!].$$

Proof. By the previous lemma we obtain $\mathsf{invk} \in \mathcal{T}$ and therefore $\mathsf{r}\,\mathsf{invk} \in \mathcal{T}$ by Lemma 3. Then we apply Lemma 1 in order to verify our claim. □

As we have seen in Example 1 we cannot prove soundness with respect to reductions of our model construction for the original formulation of reductions in Featherweight Java. The reason is that FJ does not enforce a call-by-value evaluation strategy whereas theories of types and names adopt call-by-value evaluation via their strictness axioms. Moreover, Examples 2 and 3 show that we also have to take care of exceptions and the role of late-binding. Let $\longrightarrow'$ be the variant of the reduction relation $\longrightarrow$ with a call-by-value evaluation strategy which respects exceptions.

Definition 15. *Let* $\mathtt{a}$ *and* $\mathtt{b}$ *be two FJ expressions. We define the reduction relation* $\longrightarrow$ *by induction on the structure of* $\mathtt{a}$*:* $\mathtt{a} \longrightarrow' \mathtt{b}$ *if and only if* $\mathtt{a} \longrightarrow \mathtt{b}$*, where all subexpressions of* $\mathtt{a}$ *are in closed normal form with respect to* $\longrightarrow'$ *and* $\mathtt{a}$ *does not contain subexpressions like* $(\mathtt{D})\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})$ *with* $\mathtt{C} \not<: \mathtt{D}$*.*

As shown in Example 3 the following lemma can only be established for *closed* expressions in normal form.

Lemma 5. *Let* $\mathtt{e}$ *be a well-typed Featherweight Java expression in closed normal form with respect to* $\longrightarrow'$*. Then in* PTN *it is provable that* $[\![\mathtt{e}]\!]{\downarrow}$*. Moreover, if* $\mathtt{e}$ *is not of the form* $(\mathtt{D})\mathtt{new}\ \mathtt{C}(\overline{\mathtt{e}})$ *with* $C \not<: D$ *and does not contain subexpressions of this form, then it is provable in* PTN *that* $\mathsf{p}_0[\![\mathtt{e}]\!] \neq \mathsf{ex}$*.*

Proof. Let $\mathtt{e}$ be a well-typed closed FJ expression in normal form. First, we prove by induction on the structure of $\mathtt{e}$ that one of the following holds: $\mathtt{e}$ itself is of the form $(\mathtt{D})\mathtt{new\ C}(\overline{\mathtt{e}})$ with $\mathtt{C} \not<: \mathtt{D}$ or it contains a subexpression of this form or $\mathtt{e}$ does not contain such subexpressions and $\mathtt{e}$ is $\mathtt{new\ C}(\overline{\mathtt{e}})$ for a class $\mathtt{C}$ and expressions $\overline{\mathtt{e}}$. We distinguish the five cases for the built up of $\mathtt{e}$ as given by the syntax for expressions.

1. $\mathtt{x}$: is not possible since $\mathtt{e}$ is a closed term.
2. $\mathtt{e}_0.\mathtt{f}$: the induction hypothesis applies to $\mathtt{e}_0$. In the first two cases we obtain that $\mathtt{e}$ contains a subexpression of the form $(\mathtt{D})\mathtt{new\ C}(\overline{\mathtt{e}})$ with $\mathtt{C} \not<: \mathtt{D}$. In the last case we get by Theorem 4 about progress that $\mathtt{e}$ cannot be in normal form.
3. $\mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}})$: similar to the previous case.
4. $\mathtt{new\ C}(\overline{\mathtt{e}})$: the induction hypothesis applies to $\overline{\mathtt{e}}$. Again, we obtain in the first two cases that $\mathtt{e}$ contains a subexpression of the form $(\mathtt{D})\mathtt{new\ C}(\overline{\mathtt{e}})$ with $\mathtt{C} \not<: \mathtt{D}$. In the last case we see that $\mathtt{e}$ also fulfills the conditions of the last case.
5. $(\mathtt{C})\mathtt{e}_0$: we apply the induction hypothesis to infer that $\mathtt{e}$ must satisfy condition one or two since it is in normal form.

Now, we know that $\mathtt{e}$ satisfies one of the three conditions above. In the first two cases we obtain by induction on the structure of $\mathtt{e}$ that $[\![\mathtt{e}]\!] = (\mathsf{ex}, 0)$. If the first two cases do not apply, then $\mathtt{e}$ is built up of $\mathtt{new}$ expressions only and we can prove by induction on the structure of $\mathtt{e}$ that $[\![\mathtt{e}]\!]\downarrow$ and $\mathsf{p}_0[\![\mathtt{e}]\!] \neq \mathsf{ex}$. □

Our interpretation of FJ expressions respects substitutions.

Lemma 6. *For all FJ expressions* $\mathtt{e}, \overline{\mathtt{d}}$ *and variables* $\overline{\mathtt{x}}$ *it is provable in* PTN *that* $[\![\mathtt{e}]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]] \simeq [\![\mathtt{e}[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!]$.

Proof. We proceed by induction on the term structure of $\mathtt{e}$. The following cases have to be distinguished.

1. $\mathtt{e}$ is a variable, then the claim obviously holds.
2. $\mathtt{e}$ is of the form $\mathtt{e}_0.\mathtt{g}$. We have

$$[\![\mathtt{e}_0.\mathtt{g}]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]] \simeq (\mathsf{proj}\,\mathtt{g}^*\,[\![\mathtt{e}_0]\!])[[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]].$$

   Since none of the variables of $[\![\overline{\mathtt{x}}]\!]$ occurs freely in proj or $\mathtt{g}^*$, this is equal to $\mathsf{proj}\,\mathtt{g}^*\,([\![\mathtt{e}_0]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]])$, which equals $\mathsf{proj}\,\mathtt{g}^*\,([\![\mathtt{e}_0[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!])$ by the induction hypothesis. Finally, we obtain $[\![\mathtt{e}_0.\mathtt{g}[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!]$.
3. $\mathtt{e}$ is of the form $\mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}})$. We have

$$[\![\mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}})]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]] \simeq (\mathsf{r\,invk}\,(\mathtt{m}^*, [\![\mathtt{e}_0]\!], [\![\overline{\mathtt{e}}]\!]))[[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]].$$

   Again, since none of the variables of $[\![\overline{\mathtt{x}}]\!]$ occurs freely in $\mathsf{r\,invk}$ or in $\mathtt{m}^*$ this is equal to $\mathsf{r\,invk}\,(\mathtt{m}^*, [\![\mathtt{e}_0]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]], [\![\overline{\mathtt{e}}]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]])$. By the induction hypothesis we obtain $\mathsf{r\,invk}\,(\mathtt{m}^*, [\![\mathtt{e}_0[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!], [\![\overline{\mathtt{e}}[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!])$, and finally we get $[\![\mathtt{e}_0.\mathtt{m}(\overline{\mathtt{e}})[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!]$.

4. $\mathtt{e}$ is of the form $\mathtt{new\ C}(\overline{\mathtt{e}})$. We have

$$[\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]] \simeq (\mathsf{new}\,\mathtt{C}^*([\![\overline{\mathtt{e}}]\!]))[[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]].$$

   Again this is equal to $\mathsf{new}\,\mathtt{C}^*([\![\overline{\mathtt{e}}]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]])$ which is by the induction hypothesis $\mathsf{new}\,\mathtt{C}^*([\![\overline{\mathtt{e}}[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!])$. This is $[\![\mathtt{new\ C}(\overline{\mathtt{e}})[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!]$ by the interpretation of $\mathtt{new}$.
5. $\mathtt{e}$ is of the form $(\mathtt{C})\mathtt{e}_0$. We obtain

$$[\![(\mathtt{C})\mathtt{e}_0]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]] \simeq \mathsf{cast}\,\mathtt{C}^*([\![\mathtt{e}_0]\!][[\![\overline{\mathtt{d}}]\!]/[\![\overline{\mathtt{x}}]\!]]).$$

   By the induction hypothesis this is equal to

$$\mathsf{cast}\,\mathtt{C}^*\,[\![\mathtt{e}_0[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!] \simeq [\![(\mathtt{C})\mathtt{e}_0[\overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!].$$

□

Now we prove soundness with respect to call-by-value reductions.

Theorem 8. *Let* $\mathtt{g}, \mathtt{h}$ *be two FJ expressions so that* $\mathtt{g}$ *is well-typed and* $\mathtt{g} \longrightarrow' \mathtt{h}$ *is derivable in FJ, then in* PTN *it is provable that* $[\![\mathtt{g}]\!] \simeq [\![\mathtt{h}]\!]$.

Proof. We distinguish the three different rules for computations.

1. $\mathtt{g}$ is of the form $(\mathtt{new\ C}(\overline{\mathtt{e}}))\mathtt{.f_i}$ and $\mathit{fields}(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}$. We obtain

$$[\![(\mathtt{new\ C}(\overline{\mathtt{e}}))\mathtt{.f_i}]\!] \simeq \mathsf{proj}\,\mathtt{f}_i^*\,[\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!].$$

   By the definition of new this is equal to $\mathsf{proj}\,\mathtt{f}_i^*\,(t_{\mathtt{C}}[\![\overline{\mathtt{e}}]\!])$. For $\mathtt{g} \longrightarrow' \mathtt{h}$, we know that all subexpressions of $\mathtt{g}$ are closed, fully evaluated and not of the form $(\mathtt{D})\mathtt{new\ C}(\overline{\mathtt{e}})$ with $\mathtt{C} \not<: \mathtt{D}$. Hence we obtain by the Lemma 5 that $\mathtt{p}_0[\![\overline{\mathtt{e}}]\!] \neq \mathsf{ex}$ holds and therefore $\mathsf{proj}\,\mathtt{f}_i^*\,(t_{\mathtt{C}}[\![\overline{\mathtt{e}}]\!]) \simeq [\![\mathtt{e}_i]\!]$.
2. $\mathtt{g}$ is of the form $(\mathtt{new\ C}(\overline{\mathtt{e}}))\mathtt{.m}(\overline{\mathtt{d}})$ and $\mathit{mbody}(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e}_0)$. Assume $\mathtt{m}$ is defined exactly in the classes $\mathtt{C_1}, \ldots, \mathtt{C_n}$. Now we show by induction on the length of the derivation of $\mathit{mbody}(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{e}_0)$ that there exists k so that $1 \leq k \leq n$,

$$\mathsf{min}^k_{\mathtt{C}_1^*,\ldots,\mathtt{C}_n^*}(\mathtt{C}^*) = 1 \bigwedge_{l<k} \mathsf{min}^l_{\mathtt{C}_1^*,\ldots,\mathtt{C}_n^*}(\mathtt{C}^*) \neq 1 \tag{11}$$

   and

$$\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \mathit{mbody}(\mathtt{m}, \mathtt{C_k}). \tag{12}$$

   If $\mathtt{m}$ is defined in $\mathtt{C}$, then there exists a k in $1, \ldots, n$ so that $\mathtt{C} = \mathtt{C_k}$. Hence (11) and (12) trivially hold. If $\mathtt{m}$ is not defined in $\mathtt{C}$, then $\mathtt{C}$ extends a class $\mathtt{B}$ with $\mathit{mbody}(\mathtt{m}, \mathtt{B}) = (\overline{\mathtt{x}}, \mathtt{e}_0)$. In this case we have $\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \mathit{mbody}(\mathtt{m}, \mathtt{B})$. Therefore (11) and (12) follow by the induction hypothesis.

   We have assumed that $(\mathtt{new\ C}(\overline{\mathtt{e}}))\mathtt{.m}(\overline{\mathtt{d}})$ is well-typed. Therefore we get that $\mathtt{new\ C}(\overline{\mathtt{e}})$ and $\overline{\mathtt{d}}$ are well-typed. Let $\mathtt{B}$ be the type satisfying $\mathit{mbody}(\mathtt{m}, \mathtt{C}) = \mathit{mbody}(\mathtt{m}, \mathtt{B})$ so that $\mathtt{m}$ is defined in $\mathtt{B}$. We find $\mathtt{C} <: \mathtt{B}$. Furthermore, let $\overline{\mathtt{D}}$ be

the types with $\overline{\mathtt{d}} \in \overline{\mathtt{D}}$. The expressions $\overline{\mathtt{e}}, \overline{\mathtt{d}}$ are in closed normal form and hence we obtain by Lemma 5, Theorem 7 and Theorem 5

$$[\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!]_{\mathsf{r\,invk}} \in [\![\mathtt{B}]\!] \qquad [\![\overline{\mathtt{d}}]\!]_{\mathsf{r\,invk}} \in [\![\overline{\mathtt{D}}]\!]. \tag{13}$$

By our general assumption that method typing is ok we obtain

$$\overline{\mathtt{x}} : \overline{\mathtt{D}}, \mathtt{this} : \mathtt{B} \vdash \mathtt{e}_0 \in \mathtt{E}_0.$$

Hence applying Lemma 2 with $\mathsf{r\,invk} \cong_{\mathcal{T}} \mathsf{invk}$ yields

$$[\![\overline{\mathtt{x}}]\!]_{\mathsf{invk}} \in [\![\overline{\mathtt{D}}]\!] \wedge [\![\mathtt{this}]\!]_{\mathsf{invk}} \in [\![\mathtt{B}]\!] \to [\![\mathtt{e}_0]\!]_{\mathsf{invk}} \simeq [\![\mathtt{e}_0]\!]_{\mathsf{r\,invk}}. \tag{14}$$

Summing up, we get by (13), (14) and Remark 1

$$\begin{aligned}[\![(\mathtt{new\ C}(\overline{\mathtt{e}})).\mathtt{m}(\overline{\mathtt{d}})]\!]_{\mathsf{r\,invk}} &\simeq \mathsf{r\,invk}\,(\mathtt{m}^*, [\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!]_{\mathsf{r\,invk}}, [\![\overline{\mathtt{d}}]\!]_{\mathsf{r\,invk}}) \\ &\simeq [\![\mathtt{e}_0]\!]_{\mathsf{invk}}([\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!]_{\mathsf{r\,invk}}/this, [\![\overline{\mathtt{d}}]\!]_{\mathsf{r\,invk}}/[\![\overline{\mathtt{x}}]\!]) \\ &\simeq [\![\mathtt{e}_0]\!]_{\mathsf{r\,invk}}([\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!]_{\mathsf{r\,invk}}/this, [\![\overline{\mathtt{d}}]\!]_{\mathsf{r\,invk}}/[\![\overline{\mathtt{x}}]\!]).\end{aligned}$$

In view of Lemma 6 this is partially equal to

$$[\![\mathtt{e}_0[\mathtt{new\ C}(\overline{\mathtt{e}})/\mathtt{this}, \overline{\mathtt{d}}/\overline{\mathtt{x}}]]\!]_{\mathsf{r\,invk}}.$$

3. $\mathtt{g}$ is of the form $(\mathtt{D})(\mathtt{new\ C}(\overline{\mathtt{e}}))$ and $\mathtt{C} <: \mathtt{D}$. We have $\mathsf{sub}(\mathtt{C}^*, \mathtt{D}^*) = 1$ and therefore

$$[\![(\mathtt{D})(\mathtt{new\ C}(\overline{\mathtt{e}}))]\!] \simeq [\![\mathtt{new\ C}(\overline{\mathtt{e}})]\!].$$

□

8 Conclusion

Usually, the research on Java's semantics takes an *operational* approach. If a denotational semantics for object-oriented principles is presented, then it is often given in *domain-theoretic* notions. In contrast to this work, we investigate a *denotational* semantics for Featherweight Java which is based on *recursion-theoretic* concepts.

Our interpretation of Featherweight Java is based upon a formalization of the object model of Castagna, Ghelli and Longo [10]. Its underlying type theory can be given using predicative notions only, whereas most other object encodings are impredicative, cf. e.g. Bruce, Cardelli and Pierce [7]. We have formalized the object model in a predicative theory of types and names which shows that this model is really simple from a proof-theoretic perspective. Hence, our formalization provides constructive foundations for object-oriented programming. Moreover, this gives further evidence for Feferman's claim that impredicative assumptions are not needed for computational practice. A claim which has, up to now, only been verified for polymorphic functional programs. Our work yields

first positive results about its status in the context of object-oriented programming.

We have a proof-theoretically weak but highly expressive theory for representing object-oriented programs and for stating and proving many properties of them similar to the systems provided by Feferman [15,16,17] and Turner [43, 44] for functional programming. Due to the fact that a least fixed point operator is definable in our theory, we also can prove that certain programs will not terminate. This is not possible in the systems of Feferman and Turner.

Since Featherweight Java is the functional core of the Java language and since the object model we employ provides a unified foundation for both Simula's and CLOS's style of programming, our work also contributes to the study of the relationship between object-oriented and functional programming. It shows that these two paradigms of programming fit well together and that their combination has a sound mathematical model.

Usually, denotational semantics are given in domain-theoretic notions. In such a semantics one has to include to each type an element $\perp$ which denotes the result of a non-terminating computation of this type, cf. e.g. Alves-Foss and Lam [2]; whereas our recursion-theoretic model has the advantage that computations are interpreted as ordinary computations. This means we work with *partial* functions which possibly do not yield a result for certain arguments, i.e. computations may really not terminate. In our opinion this model is very natural and captures well our intuition about non-termination.

As already pointed out by Castagna, Ghelli and Longo [10] the dynamic definition of new classes is one of the main problems when overloaded functions are used to define methods. Indeed, in our semantics we assumed a fixed class table, i.e. the classes are given from the beginning and they will not change. This fact makes our semantics non-compositional. If we add new classes to our class table, then we get a new interpretation for our objects. An important goal would be to investigate an overloading based semantics for object-oriented programs with dynamic class definitions. Our work has also shown that theories of types and names are a powerful tool for analyzing concepts of object-oriented programming languages. Therefore, we think it would be worthwhile to employ such theories for exploring further principles, i.e. the combination of overloading and parametric polymorphism, cf. Castagna [8], or the addition of mixins to class-based object-oriented languages, cf. e.g. Ancona, Lagorio and Zucca [3] or Flatt, Krishnamurthi and Felleisen [20]. Last but not least it would be very interesting to have a semantics for concurrent and distributed computations in explicit mathematics.

References

1. Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Wolfram Menzel, and Peter H. Schmitt. The KeY approach: integrating object oriented design and formal verification. In Gerhard Brewka and Luís Moniz Pereira, editors, *Proc. 8th European Workshop on Logics in AI (JELIA)*, Lecture Notes in Artificial Intelligence. Springer, 2000.

2. Jim Alves-Foss and Fong Shing Lam. Dynamic denotational semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*, pages 201–240. Springer, 1999.
3. Davide Ancona, Giovanni Lagorio, and Elena Zucca. Jam: A smooth extension of Java with mixins. In E. Bertino, editor, *ECOOP 2000 - 14th European Conference on Object-Oriented Programming*, volume 1850 of *Lecture Notes in Computer Science*. Springer, 2000.
4. Davide Ancona and Elena Zucca. A module calculus for Featherweight Java. Submitted.
5. Michael J. Beeson. *Foundations of Constructive Mathematics: Metamathematical Studies*. Springer, 1985.
6. Michael J. Beeson. Proving programs and programming proofs. In R. Barcan Marcus, G.J.W. Dorn, and P. Weingartner, editors, *Logic, Methodology and Philosophy of Science VII*, pages 51–82. North-Holland, 1986.
7. Kim B. Bruce, Luca Cardelli, and Benjamin C. Pierce. Comparing object encodings. *Information and Computation*, 155:108–133, 1999.
8. Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation*. Birkhäuser, 1997.
9. Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A semantics for λ&-early: a calculus with overloading and early binding. In M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 1993.
10. Giuseppe Castagna, Giorgio Ghelli, and Giuseppe Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, 1995.
11. Pietro Cenciarelli, Alexander Knapp, Bernhard Reus, and Martin Wirsing. An event-based structural operational semantics of multi-threaded Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*, pages 157–200. Springer, 1999.
12. Sophia Drossopoulou, Susan Eisenbach, and Sarfraz Khurshid. Is the Java type system sound? *Theory and practice of object systems*, 5(1):3–24, 1999.
13. Solomon Feferman. A language and axioms for explicit mathematics. In J.N. Crossley, editor, *Algebra and Logic*, volume 450 of *Lecture Notes in Mathematics*, pages 87–139. Springer, 1975.
14. Solomon Feferman. Constructive theories of functions and classes. In M. Boffa, D. van Dalen, and K. McAloon, editors, *Logic Colloquium '78*, pages 159–224. North Holland, 1979.
15. Solomon Feferman. Polymorphic typed lambda-calculi in a type-free axiomatic framework. In W. Sieg, editor, *Logic and Computation*, volume 106 of *Contemporary Mathematics*, pages 101–136. American Mathematical Society, 1990.
16. Solomon Feferman. Logics for termination and correctness of functional programs. In Y. N. Moschovakis, editor, *Logic from Computer Science*, volume 21 of *MSRI Publications*, pages 95–127. Springer, 1991.
17. Solomon Feferman. Logics for termination and correctness of functional programs II: Logics of strength PRA. In P. Aczel, H. Simmons, and S. S. Wainer, editors, *Proof Theory*, pages 195–225. Cambridge University Press, 1992.
18. Matthias Felleisen and Daniel P. Friedman. *A Little Java, A Few Patterns*. MIT Press, 1998.
19. Marcello Fiore, Achim Jung, Eugenio Moggi, Peter O'Hearn, Jon Riecke, Giuseppe Rosolini, and Ian Stark. Domains and denotational semantics: history, accomplishments and open problems. *Bulletin of the European Association for Theoretical Computer Science*, 59:227–256, 1996.

20. Matthew Flatt, Shriram Krishnamurthi, and Matthias Felleisen. A programmer's reduction semantics for classes and mixins. Technical report, Rice University, 1999. Corrected Version, original in J. Alvess-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*, pages 241–269, Springer, 1999.
21. Giorgio Ghelli. A static type system for late binding overloading. In A. Paepcke, editor, *Proc. of the Sixth International ACM Conference on Object-Oriented Programming Systems and Applications*, pages 129–145. Addison-Wesley, 1991.
22. James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison Wesley, 1996. Also available via `http://java.sun.com/docs/`.
23. Atsushi Igarashi and Benjamin Pierce. On inner classes. In *Informal Proceedings of the Seventh Internatianal Workshop on Foundations of Object-Oriented Languages (FOOL)*, 2000.
24. Atsushi Igarashi, Benjamin Pierce, and Philip Wadler. Featherweight Java: A minimal core calculus for Java and GJ. In *Object Oriented Programming: Systems, Languages and Applications (OOPSLA '99)*, volume 34 of *ACM SIGPLAN Notices*, pages 132–146, 1999.
25. Bart Jacobs, Joachim van den Berg, Marieke Huisman, Martijn van Berkum, Ulrich Hensel, and Hendrik Tews. Reasoning about Java classes (preliminary report). In *Object Oriented Programming: Systems, Languages and Applications (OOPSLA '98)*, volume 33 of *ACM SIGPLAN Notices*, pages 329–340, 1998.
26. Gerhard Jäger. Induction in the elementary theory of types and names. In E. Börger, H. Kleine Büning, and M.M. Richter, editors, *Computer Science Logic '87*, volume 329 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 1988.
27. Gerhard Jäger. Type theory and explicit mathematics. In H.-D. Ebbinghaus, J. Fernandez-Prida, M. Garrido, M. Lascar, and M. Rodriguez Artalejo, editors, *Logic Colloquium '87*, pages 117–135. North-Holland, 1989.
28. Reinhard Kahle. Einbettung des Beweissystems Lambda in eine Theorie von Operationen und Zahlen. Diploma thesis, Mathematisches Institut der Universität München, 1992.
29. Reinhard Kahle and Thomas Studer. Formalizing non-termination of recursive programs. To appear in *Journal of Logic and Algebraic Programming*.
30. Tobias Nipkow and David von Oheimb. Java$_{light}$ is type-safe — definitely. In *Proc. 25th ACM Symp. Principles of Programming Languages*. ACM Press, New York, 1998.
31. David von Oheimb. Axiomatic semantics for Java$_{light}$. In S. Drossopoulou, S. Eisenbach, B. Jacobs, G. T. Leavens, P. Müller, and A. Poetzsch-Heffter, editors, *Formal Techniques for Java Programs*. Fernuniversität Hagen, 2000.
32. Dieter Probst. Dependent choice in explicit mathematics. Diploma thesis, Institut für Informatik und angewandte Mathematik, Universität Bern, 1999.
33. Dana S. Scott. Identity and existence in intuitionistic logic. In M. Fourman, C. Mulvey, and D. Scott, editors, *Applications of Sheaves*, volume 753 of *Lecture Notes in Mathematics*, pages 660–696. Springer, 1979.
34. Robert Stärk. Call-by-value, call-by-name and the logic of values. In D. van Dalen and M. Bezem, editors, *Computer Science Logic '96*, volume 1258 of *Lecture Notes in Computer Science*, pages 431–445. Springer, 1997.
35. Robert Stärk. Why the constant 'undefined'? Logics of partial terms for strict and non-strict functional programming languages. *Journal of Functional Programming*, 8(2):97–129, 1998.

36. Robert Stärk, Joachim Schmid, and Egon Börger. *Java and the Java Virtual Machine*. Springer, 2001.
37. Thomas Studer. A semantics for $\lambda_{str}^{\{\}}$: a calculus with overloading and late-binding. To appear in *Journal of Logic and Computation*.
38. Thomas Studer. *Object-Oriented Programming in Explicit Mathematics: Towards the Mathematics of Objects*. PhD thesis, Institut für Informatik und angewandte Mathematik, Universität Bern, 2001.
39. Thomas Studer. Impredicative overloading in explicit mathematics. Submitted.
40. Don Syme. Proving Java type soundness. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*, pages 83–118. Springer, 1999.
41. Makoto Tatsuta. Realizability for constructive theory of functions and classes and its application to program synthesis. In *Proceedings of Thirteenth Annual IEEE Symposium on Logic in Computer Science, LICS '98*, pages 358–367, 1998.
42. Anne Sjerp Troelstra and Dirk van Dalen. *Constructivism in Mathematics, vol II*. North Holland, 1988.
43. Raymond Turner. *Constructive Foundations for Functional Languages*. McGraw Hill, 1991.
44. Raymond Turner. Weak theories of operations and types. *Journal of Logic and Computation*, 6(1):5–31, 1996.

Author Index