Programming with Classical Proofs

MSc Thesis *(Afstudeerscriptie)*

written by

Hans Bugge Grathwohl
(born January 10th 1989 in Frederiksberg, Denmark)

under the supervision of **prof. dr. Herman Geuvers** and **dr. Inge Bethke**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:
August 27th 2013

Members of the Thesis Committee:
dr. Maria Aloni
prof. dr. Herman Geuvers
dr. Inge Bethke
prof. dr. Dick de Jongh
dr. Piet Rodenburg
dr. Benno van den Berg

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

This thesis is about extracting programs from classical proofs. In the first part, we show conservativity of Peano arithmetic over Heyting arithmetic for Π^0_2-sentences, an old result of Kreisel, using Friedman's A-translation technique. Then we present some extensions by Parigot and Krebbers of the lambda-calculus with control mechanisms, that allow for some amount of classical reasoning via the Curry–Howard correspondence.

In the second part of the thesis, we present a new system by Aschieri and Berardi, $\mathsf{HA} + \mathsf{EM}_1$, a Curry–Howard system for an arithmetic with a limited amount of classical reasoning that is based on ideas from their Interactive Realizability semantics for classical arithmetic. We show Aschieri's recent proof of strong normalization of $\mathsf{HA} + \mathsf{EM}_1$ that uses a new technique based on non-deterministic choice.

Two non-trivial examples of proof terms in $\mathsf{HA} + \mathsf{EM}_1$ are then worked out, and their possible reduction paths are analyzed. On basis of this, an operational natural semantics for $\mathsf{HA} + \mathsf{EM}_1$ is developed and tested on the previous examples.

Acknowledgements

I would like to thank my supervisor Herman Geuvers for introducing me to the area of classical program extraction, and for a lot of good, fruitful meetings in Nijmegen. Furthermore, I would like to thank Inge Bethke for being willing to take up the job as my local supervisor.

I am grateful to my brother Bjørn, the computer scientist, who has carefully read my drafts and provided valuable comments and corrections.

I would also like to thank my fellow students at the ILLC, who has proved excellent company in my years in Amsterdam, and furthermore have taught me most of the logic I know. Outside logic, a special thanks goes to Roos Holleman for great times, and for invaluable support during the final stages of my writing.

Contents

1 Introduction **1**

1.1 Related work 2

1.2 Outline 3

1.3 Notation 4

2 Preliminaries **5**

2.1 Natural deduction 5

2.2 First-order logic 6

2.3 The untyped lambda calculus 9

2.4 Simply typed lambda calculus 11

2.5 Gödel's System **T** 13

2.6 Annotated first-order proofs 16

3 Friedman's A-translation **19**

3.1 The arithmetics PA and HA 19

3.2 Double-negation translation 20

3.3 A-translation 22

3.4 The proof 24

4 Control operators **27**

4.1 The system $\lambda\mu$ 28

4.2 The system $\lambda\mu^{\mathbf{T}}$ 31

5 Arithmetic with exceptions: $\mathsf{HA} + \mathsf{EM}_1$ **35**

5.1 Post rules 35

5.2 HA 37

5.3 $\mathsf{HA} + \mathsf{EM}_1$ 41

5.4 The system $\mathsf{HA} + \mathsf{EM}_1^*$ 45

5.5 Strong normalization for $\mathsf{HA} + \mathsf{EM}_1^*$ and $\mathsf{HA} + \mathsf{EM}_1$ 47

5.6 Existential witness property 54

6 Programming with terms in $\mathsf{HA} + \mathsf{EM}_1$ **55**

6.1 Searching 55

6.2 Multiplication example 61

7 Program extraction from $\mathsf{HA}+\mathsf{EM}_1$ 67

7.1 Natural semantics for $\mathsf{HA}+\mathsf{EM}_1$ 67

7.2 Searching 70

7.3 Multiplication 71

8 Conclusion 73

8.1 Further research 73

Bibliography 77

Chapter 1

Introduction

A fundamental result about the theory of computer programming is Rice's theorem, which states that there is no effective way of deciding whether an algorithm computes a partial recursive function with a given non-trivial property. A consequence of this is, that it is in general undecidable whether a given program meets its specification. One approach to solve this problem stems from a combination of two observations: Firstly, that there is a tight connection between computer programs and proofs, this is what is commonly known as the *Curry–Howard correspondence*, sometimes referred to as *proofs-as-programs* and *formulas-as-types*. Secondly, the observation that it is decidable whether a formal proof is correct. Thus, the idea is to make a mathematical proof of a specification (which, of course, might be hard), and from this extract a correct computer program. This is what is known as *program extraction*. It is well established that this method works well when we consider intuitionistic proof systems. Paulin-Mohring, e.g., in [32] presented a method to extract correct programs from proofs in the Calculus of Construction, a higher order λ-calculus with dependent types [12]. In [29], Parigot discusses the practicalities of the idea of *programming with proofs*, i.e., using formal mathematics as a programming language.

This method needs the proofs to be constructive, in the sense that from a proof of an existential statement, one can get a witness of this statement. All proofs in intuitionistic logic are constructive, and indeed, for people working in program extraction, attention was in the beginning restricted to intuitionistic logics. Classical logics are not constructive in the same sense, and thus it does not a priori seem to be possible to apply the same techniques here. However, an old result about arithmetic states that any Π^0_2-sentence is provable in Peano arithmetic if and only if it is provable in Heyting arithmetic. Thus, there is a method to transform any classical proof of a specification in arithmetic, i.e., a proof of $\forall\alpha\exists\beta.P(\alpha,\beta)$ where $P(\alpha,\beta)$ is a basic formula, into an intuitionistic proof of the same specification. This is evidence that all classical proofs of Π^0_2-sentences have some computational content. Π^0_2-sentences are indeed arguably the most important sentences in computer science, since a proof of one of these corresponds to a proof of totality of a recursive function. This leads to the area of *classical program extraction.*

There have been several approaches to extracting the computational content of these classical proofs. It was discovered by Griffin in 1989 [20] that inference by contradiction corresponds to Felleisen's control operator $\mathcal{C}$ [13], and hence the Curry–Howard correspondence was extended to include classical reasoning. This sparked a lot of research in this area. Several extensions of the λ-calculus with control operators have been proposed. To name a couple: Felleisen's $\lambda_{\mathcal{C}}$ with typing rules by Griffin; Rehof and Sørensen's λ_{Δ} [36] that extends ordinary λ-terms with a binder Δ which is typed by *reductio ad absurdum*; and Parigot's $\lambda\mu$ [30], which we will return to in Chapter 4, along with Krebbers's $\lambda\mu^{\mathbf{T}}$ which extends $\lambda\mu$ with natural numbers as a primitive datatype.

These systems correspond to classical *propositional* logic, which means that their type systems are rather simple, and that, when they are equipped with datatypes, they are more closely related to real world computer programming languages than first-order systems are. But since we are interested in proofs of statements of the form $\forall\alpha\exists\beta.\varphi(\alpha,\beta)$, we need to consider systems that correspond to first-order logic. For intuitionistic logic the standard system is IQC, and when this is extended with the Peano axioms for arithmetic, we get Heyting arithmetic, HA. In HA we do not need to add datatypes, since the natural numbers are primitive in it. In this thesis we are mainly concerned with an extension of HA with a limited amount of classical reasoning in the form of EM_1, the law of excluded middle restricted to Σ^0_1-formulas. The system $\mathsf{HA}+\mathsf{EM}_1$ that we present in Chapter 5 is a very recent system by Aschieri and Berardi, and therefore it is not yet well studied. We work out some non-trivial proofs in this system, and discuss how we can extract programs from these.

1.1 Related work

Berger, Buchholz, and Schwichtenberg [11] describe a method for extracting programs from classical proofs, by way of extracting a term in Gödel's System **T** which contains all the computationally relevant parts of the proof. This is in the style of the Gödel–Gentzen double negation translation, and indeed the target language does not contain control mechanisms.

In [28], Makarov utilizes Felleisen's $\mathcal{C}$-operator to extract a program from a classical proof of a non-trivial arithmetical proposition by adding extra inference rules and defining a structural operational semantics for the classical deduction system.

Herbelin has introduced the system $\mathrm{IQC}_{\mathrm{MP}}$ [21], which he characterizes as an intuitionistic predicate logic with just enough classical reasoning to prove Markov's principle, which is the scheme that asserts that $\neg\neg\varphi \to \varphi$ whenever φ is $\forall$-$\to$-free.

Krebbers extended Parigot's $\lambda\mu$ to contain a primitive datatype for the natural numbers, in the style of Gödel's System $\mathbf{T}$, so as to come closer to "real" programming languages, since these all have primitive datatypes. We will present this system in Chapter 4. Furthermore, he has developed $\lambda :: \texttt{catch}$, which is an extension of Herbelin's $\mathrm{IQC}_{\mathrm{MP}}$-calculus with `catch` and `throw` [21], this time with lists as a primitive datatype.

Aschieri and Berardi has developed *interactive realizability* [2,4,5,7], which is a computational semantics for classical proofs that is based on the principle of learning. Instead of following the method of Avigad [8], who characterizes his classical realizability in terms of a special double-negation translation followed by Friedman's A-translation, followed by Kreisel's modified realizability [26], Aschieri avoids the use of a double-negation translation, and instead combines modified realizability and Friedman's translation. The learning aspect is based on the idea that whenever we use an instance of excluded middle $\forall\alpha.\varphi(\alpha) \vee \exists\alpha.\neg\varphi(\alpha)$ in a proof, the realizer starts by *assuming* that $\forall\alpha.\varphi(\alpha)$ is the case, and then whenever we use an instance $\varphi(n)$ in the proof, the realizer checks to see if this is actually the case. The realizer then updates its state with this new information (it *learns*). If $\varphi(n)$ is the case, then it continues under the assumption that $\forall\alpha.\varphi(\alpha)$ holds, and if not, it has found a witness for $\exists\alpha.\neg\varphi(\alpha)$, thus this must hold, and the realizer continues in the part of the proof that work under this assumption.

It is on the basis of interactive realizability that Aschieri and Berardi have developed the classical Curry–Howard system $\mathsf{HA} + \mathsf{EM}_1$ [3,6] that we will investigate in this thesis.

1.2 Outline

In Chapter 2 we present some basic proof theory and lambda calculus, and we introduce some type systems, namely the simply typed lambda calculus $\lambda_{\rightarrow}$, Gödel's System $\mathbf{T}$, and MQC, a calculus for minimal first-order logic.

In Chapter 3 we present a proof of Kreisel's theorem that PA is a conservative extension of HA for Π^0_2-sentences, via the Gödel–Gentzen double-negation translation and Friedman's A-translation, which lays ground to most of the methods employed in the area of classical program extraction.

In Chapter 4 we discuss how to introduce control mechanisms in the λ-calculus, and specifically we present the systems $\lambda\mu$ by Parigot, and $\lambda\mu^{\mathbf{T}}$ by Krebbers. These are examples of simple programming languages with control mechanisms that correspond via Curry–Howard to classical logic.

In Chapter 5 we present a system HA, and expand this to Aschieri's system $\mathsf{HA} + \mathsf{EM}_1$. We prove strong normalization of $\mathsf{HA} + \mathsf{EM}_1$ by a new method of Aschieri [3] that uses *non-deterministic choice.*

In Chapter 6 we investigate how to use $\mathsf{HA}+\mathsf{EM}_1$ for program extraction via analysis of two concrete examples. The first example is a proof of a specification of a searching problem, and the second example is a multiplication program which uses control to increase efficiency.

In Chapter 7 we introduce a new operational semantics for $\mathsf{HA} + \mathsf{EM}_1$, and test this on some examples from Chapter 6.

1.3 Notation

We use greek letters, $\alpha, \beta, \gamma, \dots$ to refer to numeric variable, letters $x, y, z, \dots$ to refer to proof variables, and letters $a, b, c, \dots$ to refer to variables that acts as "addresses" for control mechanisms. For proof terms, we will mainly use the letters $u, v, w, \dots$, and for numeric terms we will mostly use $n, m, \dots$.

When writing λ-abstractions, we will often omit the annotated types, even if we are working in Church-style. This saves space, and the types can be deduced from the context.

For formulas φ, we will often write $\varphi(\alpha)$, which means that we can substitute α with n simply by writing $\varphi(n)$. It does not necessarily imply that α is the only free variable in φ.

Natural deduction proof trees are defined with a turnstile and an environment, Γ, and $\vdash$, but since this makes the, already bulky, trees look even more voluminous, we will often discharge variables with superscripts instead:

$$\frac{\tau \vdash \tau}{\vdash \tau \rightarrow \tau} \quad \text{versus} \quad \frac{\tau^x}{\tau \rightarrow \tau}\, x.$$

Chapter 2

Preliminaries

2.1 Natural deduction

We first define what a natural deduction system is in general.

Definition 2.1.1 (Natural deduction systems)**.** Let $\mathcal{L}$ be a language. We define a natural deduction system $\mathcal{N}$.

1. An *environment* in natural deduction is a finite set of formulas of $\mathcal{L}$, usually written Γ.

2. A natural deduction *judgment* is a pair consisting of an environment and a formula, written $\Gamma \vdash \varphi$. We do not write set-brackets when we specify the environment, thus we write $\varphi, \psi \vdash \theta$ instead of $\{\varphi, \psi\} \vdash \theta$ and $\vdash \varphi$ when the environment is empty.

3. An *n-ary rule of inference* consists of $n + 1$ judgments (n premises and one conclusion), and is written on the form

$$\frac{\Gamma_1 \vdash \varphi_1 \qquad \Gamma_2 \vdash \varphi_2 \qquad \cdots \qquad \Gamma_n \vdash \varphi_n}{\Gamma \vdash \varphi}$$

   A nullary inference rule is called an *axiom.* Different natural deduction systems are distinguished by having different inference rules.

4. A *proof* (synonym: *derivation*) of a judgment $\Gamma \vdash \varphi$ is a finite tree, where:

   - $\Gamma \vdash \varphi$ is the root label,
   - any label is obtained by its children's labels by an application of one of the natural deduction rules. If a label is obtained by an application of a nullary rule (an axiom), then it is a leaf.

In general, we will write $\Gamma \vdash \varphi$ to mean that *there is a derivation of the judgment* $\Gamma \vdash \varphi$. As we will sometimes use multiple natural deduction systems, it can be practical to annotate which system we are using, like so: $\Gamma \vdash_{\mathcal{N}} \varphi$. Mostly, this will be clear from the context.

2.2 First-order logic

In order to formalize first-order logic, we start by defining a natural deduction proof system for the so-called *minimal first-order logic* (mFOL). Minimal logic, introduced in 1936 by Ingebrigt Johansson [23], is a simplified version of intuitionistic logic where *ex falso quodlibet* does not hold. In fact, minimal logic does not contain any rules about absurdity, and therefore $\bot$ does not need to be in the language. Since negation is usually defined as $\neg A := A \to \bot$, we do not necessarily have negation in mFOL.

Firstly, we need to specify what language we work with.

Definition 2.2.1 (The language of first-order logic)**.** Given a signature $\mathcal{S}$ consisting of functional symbols and relational symbols together with their arity, we define the first-order language $\mathcal{L}_{\mathcal{S}}$:

- Let $\mathcal{V}$ be a set of distinct variable names $\alpha, \beta, \gamma, \ldots$
- We define the *terms* of $\mathcal{L}_{\mathcal{S}}$ as the least set $\mathcal{T}$ such that
  - $\mathcal{V} \subseteq \mathcal{T}$;
  - If $t_1, \ldots, t_n \in \mathcal{T}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}$ where f is an n-ary functional symbol from $\mathcal{S}$.

  A term is *closed* if it contains no variables. The closed terms are supposed to represent the objects in the domain of discourse.
- We define the *formulas* of $\mathcal{L}_{\mathcal{S}}$ as the least set $\mathcal{F}$ such that
  - $P(t_1, \ldots, t_n) \in \mathcal{F}$ where P is an n-ary relation symbol from $\mathcal{S}$, and $t_1, \ldots, t_n \in \mathcal{T}$. These are called *atomic formulas.*
  - $\varphi \wedge \psi, \varphi \vee \psi, \varphi \to \psi \in \mathcal{F}$,
  - $\forall \alpha.\varphi, \exists \alpha.\varphi \in \mathcal{F}$, where $\alpha \in \mathcal{V}$. We say that the scope of $\forall \alpha$ ($\exists \alpha$) is φ, and we say that any occurrence of α in φ is *bound.*

In the rest of this document, we will use the less cumbersome Backus-Naur notation when we specify syntax, e.g. when we define terms, formulas, types, etc. The above definition of formulas will then look like:

$$\varphi, \psi ::= P(t_1, \ldots, t_n) \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \to \psi \mid \forall \alpha.\varphi \mid \exists \alpha.\varphi.$$

Example 2.2.2. Consider the signature $\mathcal{S} = \{\mathsf{0}, \mathsf{S}, +, =\}$, where $\mathsf{0}$ is a nullary, S a unary, and $+$ a binary function symbol, and $=$ a binary relation symbol. Examples of terms of the language $\mathcal{L}_\mathcal{S}$ are

$$\mathsf{SS}\alpha,\ \mathsf{0} + \mathsf{S}0,\ \alpha + \beta,$$

and an example of a formula is

$$\forall\alpha\, \mathsf{S}\alpha = \alpha + \mathsf{S0}.$$

Definition 2.2.3 (Free variables)**.** The set of free variables of a term t, $\mathrm{FV}(t)$, is defined inductively:

- $\mathrm{FV}(\alpha) = \{\alpha\}$, where α is a variable;
- $\mathrm{FV}(f(t_1, \ldots, t_n)) = \mathrm{FV}(t_1) \cup \cdots \cup \mathrm{FV}(t_n)$.

Likewise, we inductively define the set of free variables of a formula A, $\mathrm{FV}(A)$:

- $\mathrm{FV}(P(t_1, \ldots, t_n)) = \mathrm{FV}(t_1) \cup \cdots \cup \mathrm{FV}(t_n)$;
- $\mathrm{FV}(\varphi \wedge \psi) = \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$;
- $\mathrm{FV}(\varphi \vee \psi) = \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$;
- $\mathrm{FV}(\varphi \to \psi) = \mathrm{FV}(\varphi) \cup \mathrm{FV}(\psi)$;
- $\mathrm{FV}(\forall\alpha\, \varphi) = \mathrm{FV}(\varphi) \setminus \{\alpha\}$;
- $\mathrm{FV}(\exists\alpha\, \varphi) = \mathrm{FV}(\varphi) \setminus \{\alpha\}$.

If Γ is a set of formulas, then $\mathrm{FV}(\Gamma) = \bigcup_{A \in \Gamma} \mathrm{FV}(A)$.

Definition 2.2.4 (mFOL)**.** Given a signature $\mathcal{S}$, we define *minimal first-order logic* (mFOL) over $\mathcal{S}$ as the natural deduction system with the inference rule schemata given in Figure 2.1, where all the formulas are from $\mathcal{L}_\mathcal{S}$.

Intuitionistic and classical logic

To get an intuitionistic first-order logic one needs the rule *ex falso quodlibet*:

$$\frac{\bot}{\varphi}$$

where φ is any formula and $\bot$ is a symbol for *absurdity*. Instead of adding this as a primitive rule, we will later see a method to make this rule admissible, by adding intuitionistic reasoning to the *atomic language*.

To get a classical system, one will have to add a classical rule or axiom. Typically, it is done by adding one of the following rules:

$$\Gamma, \varphi \vdash \varphi \ (\mathrm{Ax})$$

$$\frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi_0 \wedge \varphi_1}{\Gamma \vdash \varphi_i}\ (\wedge\mathrm{E}_i)\ \text{for } i = 0, 1$$

$$\frac{\Gamma \vdash \varphi_i}{\Gamma \vdash \varphi_0 \vee \varphi_1}\ (\vee\mathrm{I}_i)\ \text{for } i = 0, 1 \qquad \frac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \theta \qquad \Gamma, \psi \vdash \theta}{\Gamma \vdash \theta}\ (\vee\mathrm{E})$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ (\to\mathrm{I}) \qquad \frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ (\to\mathrm{E})$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \forall\alpha\varphi}\ (\forall\mathrm{I})\ \alpha \notin \mathrm{FV}(\Gamma) \qquad \frac{\Gamma \vdash \forall\alpha\varphi}{\Gamma \vdash \psi[\alpha := t]}\ (\forall\mathrm{E})$$

$$\frac{\Gamma \vdash \psi[\alpha := t]}{\Gamma \vdash \exists\alpha\varphi}\ (\exists\mathrm{I}) \qquad \frac{\Gamma \vdash \exists\alpha\varphi \qquad \Gamma, \varphi \vdash \psi}{\Gamma \vdash \psi}\ (\exists\mathrm{E})\ \alpha \notin \mathrm{FV}(\psi) \cup \mathrm{FV}(\Gamma)$$

Figure 2.1: Natural deduction rules for mFOL

- Peirce's law: We add

  $$\Gamma \vdash ((\varphi \to \psi) \to \varphi) \to \varphi$$

  as an axiomatic rule.

- Reductio ad absurdum: We allow reasoning of the form

  $$\frac{\begin{array}{c}[\neg\varphi]\\ \vdots\\ \bot\end{array}}{\varphi}$$

  which is equivalent to adding $\neg\neg\varphi \to \varphi$ as an axiom.

- Law of excluded middle: We add the axiom

  $$\Gamma \vdash \varphi \vee \neg\varphi.$$

All of these methods are equivalent in the sense that the systems extended with any of these rules will prove the same formulas, but intuitively and morally they are different. Later in this document we will mainly use the law of the excluded middle, which is intuitively justified by the common model theoretic intuition that something either holds or does not in a classical setting.

Reduction ad absurdum and Peirce's law have an interesting counter-part in computer programming: Continuation Passing Style programming.

We define the systems iFOL, mcFOL and cFOL, which are simple extensions of mFOL.

Definition 2.2.5 (iFOL)**.** By adding nullary relation symbol $\bot$ to the signature, and adding the inference rule *ex falso quodlibet*

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\,(\bot\mathrm{E})$$

to mFOL, we get *intuitionistic first-order logic*, iFOL. We define *negation* of a formula $\neg\varphi := \varphi \to \bot$.

Definition 2.2.6 (mcFOL)**.** By adding the *law of the excluded middle*

$$\Gamma \vdash \varphi \vee \neg\varphi \text{ (EM)}$$

as an axiom schema to mFOL, we get *minimal classical first-order logic.*

Definition 2.2.7 (cFOL)**.** By adding the law of the excluded middle to iFOL, we get *classical first-order logic.*

The systems can be ordered by deductive strength thus:

$$\begin{array}{ccc} \text{mFOL} & \subset & \text{iFOL} \\ \cap & & \cap \\ \text{mcFOL} & \subset & \text{cFOL} \end{array}$$

It is well-known that iFOL is sound and complete with respect to Heyting semantics, and that cFOL is sound and complete with respect to Tarskian semantics.

2.3 The untyped lambda calculus

We give a brief introduction to the untyped lambda calculus, mainly following [9].

Definition 2.3.1 (Untyped λ-terms)**.** We will work with an infinite set of λ-variables $x, y, z, \ldots$. The untyped λ-terms are defined as follows:

$$t, s ::= x \mid \lambda x.t \mid ts.$$

Definition 2.3.2 (Free variables)**.** We define the set of free variables of a λ-term t, $\mathrm{FV}(t)$, by induction as follows.

- $\mathrm{FV}(x) = \{x\}$, when x is a λ-variable;
- $\mathrm{FV}(ts) = \mathrm{FV}(t) \cup \mathrm{FV}(s)$;
- $\mathrm{FV}(\lambda x.t) = \mathrm{FV}(t) \setminus \{x\}$.

A term t is said to be *closed* if $\mathrm{FV}(t) = \emptyset$, and otherwise it is *open.* If a variable x occurs in a term t, but $x \notin \mathrm{FV}(t)$, then x is *bound*; in this case it must be under the scope of λx.

Definition 2.3.3 (Substitution)**.** *The substitution of* t *for* x *in* s, written $s[x := t]$, is defined as follows:

$$\begin{array}{rcl} x[x := t] & = & t; \\ y[x := t] & = & y, \text{ if } x \neq y; \\ (st)[x := t] & = & (s[x := t])(t[x := t]); \\ (\lambda x.s)[x := t] & = & \lambda x.s; \\ (\lambda y.s)[x := t] & = & \lambda y.s[x := t], \text{ if } x \neq y. \end{array}$$

It is, in other words, the result of substituting any free occurrence of x in s with t.

Definition 2.3.4 (α-equivalence)**.** Two terms t, s are said to be *α-equivalent*, $t =_\alpha s$, if they only differ on bound variables, i.e.:

- If y is neither free nor bound in t, then

$$\lambda x.t =_\alpha \lambda y.t[x := y].$$

- If $t =_\alpha s$, then

$$\begin{aligned} \lambda x.t &=_\alpha \lambda x.s, \text{ for all variables } x, \\ tr &=_\alpha sr, \text{ and} \\ rt &=_\alpha rs. \text{ for all } \lambda\text{-terms } r. \end{aligned}$$

In practice, we will not distinguish between α-equivalent terms. So we will suppress the α-subscript, and, e.g., say $\lambda x.x = \lambda y.y$.

Remark 2.3.5 (Barendregt's variable convention). If $t_1, \ldots, t_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

Because of this convention, any substitution will always be capture avoiding, which means that we will avoid problematic substitutions like

$$(\lambda x.yx)[y := x] = \lambda x.xx,$$

since x is both occurring as a bound variable (in $\lambda x.yx$) and as a free variable (in the substituendum x), hence it does not satisfy the variable convention. We will follow this variable convention in all the systems that we define in this document.

Remark 2.3.6. Sometimes it can be necessary to do a vacuous λ-abstraction, i.e., an abstraction over a non-occurring variable. Instead of writing $\lambda x.t$ *for* $x \notin \mathrm{FV}(t)$ we will use the notation $\lambda\_.t$.

Definition 2.3.7 (Compatible relations)**.** We say that a relation R on λ-terms is *compatible* if, for all terms t, s, r

- If tRs, then $\lambda x.t\ R\ \lambda x.s$, for all variables x;
- If tRs, then $trRsr$;
- If tRs, then $rtRrs$.

The *compatible closure* of a relation R is the least compatible relation R' such that $R \subseteq R'$.

Definition 2.3.8 (β-reduction)**.** The relation $\to_\beta$ is defined as the least compatible relation that satisfies

$$(\lambda x.t)s \to_\beta t[x := s].$$

A term of the shape $(\lambda x.t)s$ is called a *β-redex* (*red*ucible *ex*pression). If a term does not contain any β-redexes, then it is said to be in β-normal form.

2.4 Simply typed lambda calculus

We define a *simply typed lambda calculus*, $\lambda_\to$. This is the simplest example of a *type theory*, and all systems that we will define later will be extensions of $\lambda_\to$.

There are two common ways of presenting simply typed lambda calculus: In *Curry style* and in *Church style*. In the Curry style, we use the untyped λ-terms, and hence the same term can be assigned multiple different types, while in Church style simply typed lambda calculus we annotate every abstractions with a type so as to ensure that every term has a unique type. We will present it in Church style.

Definition 2.4.1 (Simple types)**.** We have a non-empty set of *atomic types*, $\mathcal{A}$. The types of $\lambda_\to$ are then defined as the least set $\mathcal{T}$ such that

- $\mathcal{A} \subseteq \mathcal{T}$;
- If $\sigma, \tau \in \mathcal{T}$, then $\sigma \to \tau \in \mathcal{T}$.

Equivalently, we can express the definition of $\mathcal{T}$ with BNF-notation thus: The types of $\lambda_{\to}$ are

$$\sigma, \tau ::= a \mid \sigma \to \tau,$$

where a ranges over the atomic types.

Remark 2.4.2. When writing types, we employ association to the right, i.e., instead of writing $\sigma_1 \to (\sigma_2 \to \sigma_3)$, we will write $\sigma_1 \to \sigma_2 \to \sigma_3$.

We will use the following abbreviation

$$\begin{aligned} \sigma^0 \to \tau &:= \tau \\ \sigma^{n+1} \to \tau &:= \sigma \to \sigma^n \to \tau. \end{aligned}$$

Definition 2.4.3 (Type environments)**.** An *environment* in $\lambda_{\to}$ is a finite set of pairs of λ-variables and types, such that each variable occurs maximally once. It is typically denoted Γ, Δ, and written on the form

$$\Gamma = x_1 : \varphi_1, \cdots, x_n : \varphi_n.$$

Definition 2.4.4 ($\lambda_{\to}$-terms)**.** The difference between typed and untyped λ-terms is that all variables are annotated with a type: The terms in $\lambda_{\to}$ are defined as follows:

$$t, s := x^{\tau} \mid \lambda x^{\tau}.t \mid ts,$$

where τ is a type.

In practice we can often deduce the type of a variable from the context, in these cases we will typically omit the type annotation, but formally they are still there.

Definition 2.4.5 (Type judgments)**.** A *type judgment* is a triple consisting of an environment, a term, and a type, written $\Gamma \vdash t : \varphi$.

Definition 2.4.6 (Type derivation)**.** A *type derivation* of a judgment $\Gamma \vdash t : \varphi$ is a finite tree where:

- $\Gamma \vdash t : \varphi$ is the root label;
- Any label is obtained by its children's labels by an application of one of the typing rules from Figure 2.2.

The simply typed lambda calculus corresponds exactly to what is known as *minimal propositional logic*, which is—basically—minimal first order logic without quantifiers. This is what is originally known as the *Curry–Howard isomorphism* or *Curry–Howard correspondence*:

$$\Gamma, x:\tau \vdash x^{\tau} \qquad \frac{\Gamma, x:\sigma \vdash t:\tau}{\Gamma \vdash \lambda x^{\sigma}.t : \sigma \to \tau} \qquad \frac{\Gamma \vdash t:\sigma \to \tau \qquad \Gamma \vdash s:\sigma}{\Gamma \vdash ts:\tau}$$

Figure 2.2: Typing rules for $\lambda_{\to}$

Theorem 2.4.7 (The Curry–Howard correspondence)**.** *If* $\Gamma \vdash t : \sigma$ *in* $\lambda_{\to}$*, where* $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$*, then* $\Gamma' \vdash \sigma$ *in minimal propositional logic, where* $\Gamma' = \sigma_1, \ldots, \sigma_n$*.*

For a proof, see [37].

2.5 Gödel's System T

Gödel's System **T** ($\lambda^{\mathbf{T}}$) is an extension of $\lambda_{\to}$ that adds the natural numbers as a primitive datatype together with a recursion operator. In the following definition we also add a Boolean datatype for convenience—this is merely syntactic sugar, since we could just as well have used zero and one to correspond to true and false.

Later in this document, we will see the idea behind the transition from $\lambda_{\to}$ to $\lambda^{\mathbf{T}}$ be applied on other systems. One should see $\lambda^{\mathbf{T}}$ as a model of a simple, yet powerful, computer programming language.

Definition 2.5.1. The types of $\lambda^{\mathbf{T}}$ are

$$\sigma, \tau ::= \texttt{N} \mid \texttt{Bool} \mid \sigma \to \tau$$

Definition 2.5.2. The terms of $\lambda^{\mathbf{T}}$ are defined inductively over an infinite set of typed λ-variables $x^{\tau}, y^{\sigma}, \ldots$

$$\begin{aligned} t, u &::= c \mid x^{\tau} \mid tu \mid \lambda x^{\tau}.t \\ c &::= \mathsf{0} \mid \mathsf{S} \mid \texttt{True} \mid \texttt{False} \mid \texttt{Rec}_{\tau} \mid \texttt{if}_{\tau} \end{aligned}$$

Definition 2.5.3. The typing judgments $\Gamma \vdash t : \sigma$ in $\lambda^{\mathbf{T}}$ are given by the typing rules in Figure 2.3.

Definition 2.5.4. Reduction, $\to_{\mathbf{T}}$, on $\lambda^{\mathbf{T}}$-terms is defined as the compatible closure of the following reduction rules:

$$\begin{array}{rll} (\lambda x^{\tau}.t)u & \to_{\beta} & t[x := u] \\ \texttt{Rec}_{\tau}\ u\ v\ \mathsf{0} & \to_{\texttt{Rec}_1} & u \\ \texttt{Rec}_{\tau}\ u\ v\ (\mathsf{S}t) & \to_{\texttt{Rec}_2} & v\ t\ (\texttt{Rec}_{\tau}\ u\ v\ t) \\ \texttt{if}_{\tau}\ \texttt{True}\ u\ v & \to_{\texttt{True}} & u \\ \texttt{if}_{\tau}\ \texttt{False}\ u\ v & \to_{\texttt{False}} & v \end{array}$$

As usual, $\twoheadrightarrow_{\mathbf{T}}$ denotes the transitive and reflexive closure of $\to_{\mathbf{T}}$, while $=_{\mathbf{T}}$ denotes the transitive, reflexive and symmetric closure.

Constants:

$$\Gamma \vdash \mathbf{0} : \mathtt{N}, \quad \Gamma \vdash \mathbf{S} : \mathtt{N} \to \mathtt{N}, \quad \Gamma \vdash \mathtt{True} : \mathtt{Bool}, \quad \Gamma \vdash \mathtt{False} : \mathtt{Bool},$$

$$\Gamma \vdash \mathtt{Rec}_\tau : \tau \to (\mathtt{N} \to (\tau \to \tau)) \to \mathtt{N} \to \tau, \quad \Gamma \vdash \mathtt{if}_\tau : \mathtt{Bool} \to \tau \to \tau \to \tau$$

Variables:

$$\Gamma, x : \tau \vdash x : \tau$$

Composed terms:

$$\frac{\Gamma \vdash t : \sigma \to \tau \qquad \Gamma \vdash u : \sigma}{\Gamma \vdash tu : \tau} \qquad \frac{\Gamma, x^\sigma \vdash t : \tau}{\Gamma \vdash \lambda x^\sigma . t : \sigma \to \tau}$$

Figure 2.3: Typing rules for terms in $\lambda^{\mathbf{T}}$

Definition 2.5.5. A term t is said to be in *normal form* if $t \twoheadrightarrow t'$ if and only if $t \equiv t'$, i.e., t has no possible reductions.

The system $\lambda^{\mathbf{T}}$ satisfies the following important meta-theorems:

Theorem 2.5.6. $\lambda^{\mathbf{T}}$ *satisfies subject reduction: If* $\Gamma \vdash t : \sigma$ *and* $t \twoheadrightarrow t'$, *then* $\Gamma \vdash t' : \sigma$.

Proof. It is easy to check that all the reduction rules preserve typing. □

Theorem 2.5.7. $\lambda^{\mathbf{T}}$ *is confluent: If* $t_1 \twoheadrightarrow t_2$ *and* $t_1 \twoheadrightarrow t_3$, *then there is a term* t_4 *such that* $t_2 \twoheadrightarrow t_4$ *and* $t_3 \twoheadrightarrow t_4$.

$$\begin{tikzcd}
& t_1 \arrow[dl, two heads] \arrow[dr, two heads] & \\
t_2 \arrow[dr, dashed, two heads] & & t_3 \arrow[dl, dashed, two heads] \\
& t_4 &
\end{tikzcd}$$

Theorem 2.5.8. $\lambda^{\mathbf{T}}$ *is strongly normalizing: There are no infinite reduction chains*

$$t_1 \to t_2 \to t_3 \to \cdots$$

which means that every term has a normal form, and no matter which reductions we choose, we will eventually reach a normal form.

The proofs of Theorems 2.5.7 and 2.5.8 can be found in [37].

Example 2.5.9. We can define equality between numbers in $\lambda^{\mathbf{T}}$. A reasonable implementation of equality needs to satisfy the following:

$$\vdash \mathtt{equal} : \mathtt{N} \to \mathtt{N} \to \mathtt{Bool}$$

$$\begin{aligned}\mathtt{equal}\ \mathbf{0}\ \mathbf{0} &= \mathtt{True}\\ \mathtt{equal}\ \mathbf{0}\ (\mathbf{S}m) &= \mathtt{False}\\ \mathtt{equal}\ (\mathbf{S}n)\ \mathbf{0} &= \mathtt{False}\\ \mathtt{equal}\ (\mathbf{S}n)\ (\mathbf{S}m) &= \mathtt{equal}\ n\ m\end{aligned}$$

To begin with, we define a term that checks for zero:

$$\mathtt{isZero} := \mathtt{Rec}_{\mathtt{Bool}}\ \mathtt{True}\ (\lambda \_^{\mathtt{N}} \lambda \_^{\mathtt{Bool}}.\mathtt{False})$$

This fulfills:

$$\vdash \mathtt{isZero} : \mathtt{N} \to \mathtt{Bool}$$

$$\begin{aligned}\mathtt{isZero}\ \mathbf{0} &\twoheadrightarrow \mathtt{True}\\ \mathtt{isZero}\ (\mathbf{S}n) &\twoheadrightarrow \mathtt{False}.\end{aligned}$$

Now, the first part of `equal` can be defined thus (for some, as of yet, undefined `equal_aux`):

$$\mathtt{equal} := \mathtt{Rec}_{\mathtt{N}\to\mathtt{Bool}}\ \mathtt{isZero}\ \mathtt{equal\_aux},$$

for then

$$\mathtt{equal}\ \mathbf{0}\ \mathbf{0} \twoheadrightarrow \mathtt{isZero}\ \mathbf{0} \twoheadrightarrow \mathtt{True},$$
$$\mathtt{equal}\ \mathbf{0}\ (\mathbf{S}m) \twoheadrightarrow \mathtt{isZero}\ (\mathbf{S}m) \twoheadrightarrow \mathtt{False}.$$

We define `equal_aux` as follows:

$$\mathtt{equal\_aux} := \lambda \_^{\mathtt{N}} \lambda f^{\mathtt{N}\to\mathtt{Bool}}.\mathtt{Rec}_{\mathtt{Bool}}\ \mathtt{False}\ (\lambda m^{\mathtt{N}} \lambda \_^{\mathtt{Bool}}.fm),$$

for then

$$\begin{aligned}\mathtt{equal}\ (\mathbf{S}n)\ \mathbf{0} &\twoheadrightarrow \mathtt{equal\_aux}\ n\ (\mathtt{equal}\ n)\ \mathbf{0}\\ &\twoheadrightarrow \mathtt{Rec}_{\mathtt{Bool}}\ \mathtt{False}\ (\lambda m^{\mathtt{N}} \lambda \_^{\mathtt{Bool}}.\mathtt{equal}\ n\ m)\ \mathbf{0}\\ &\twoheadrightarrow \mathtt{False},\end{aligned}$$

and

$$\begin{aligned}\mathtt{equal}\ (\mathbf{S}n)\ (\mathbf{S}m) &\twoheadrightarrow \mathtt{equal\_aux}\ n\ (\mathtt{equal}\ n)\ (\mathbf{S}m)\\ &\twoheadrightarrow \mathtt{Rec}_{\mathtt{Bool}}\ \mathtt{False}\ (\lambda m^{\mathtt{N}} \lambda \_^{\mathtt{Bool}}.\mathtt{equal}\ n\ m)\ (\mathbf{S}m)\\ &\twoheadrightarrow \mathtt{equal}\ n\ m.\end{aligned}$$

Sometimes—whenever it does not cause confusion—we will use the notation $t_1 = t_2$ as an abbreviation of `equal` $t_1\ t_2$.

Theorem 2.5.10 (Primitive recursive functions in $\lambda^{\mathbf{T}}$)**.** *All primitive recursive functions are representable in $\lambda^{\mathbf{T}}$.*

Proof. Every primitive recursive function F, except $\mathbf{0}$ and $\mathbf{S}$, is defined by exactly one of from the following three schemes:

$$\begin{array}{rcll} F(x_1,\ldots,x_i,\ldots,x_n) & = & x_i & (\mathrm{proj}_F) \\ F(x_1,\ldots,x_n) & = & G(H_1(x_1,\ldots,x_n),\ldots,H_m(x_1,\ldots,x_n)) & (\mathrm{comp}_F) \\ F(\mathbf{0},x_1,\ldots,x_n) & = & G(x_1,\ldots,x_n) & \\ \wedge\, F(\mathbf{S}(y),x_1,\ldots,x_n) & = & H(F(y,x_1,\ldots,x_n),y,x_1,\ldots,x_n) & (\mathrm{rec}_F) \end{array}$$

where $G, H, H_1, \ldots, H_m$ are previously defined primitive recursive functions. It should be clear how to represent these in $\lambda^{\mathbf{T}}$. If, for example, G, H are represented by $\mathtt{G}, \mathtt{H}$ and F is defined by (rec_F), then F is represented by:

$$\mathtt{F} := \mathtt{Rec}_{\mathtt{N}}\ \mathtt{G}\ (\lambda n^{\mathtt{N}} \lambda f.\mathtt{H}\, f\, n). \qquad \square$$

Remark 2.5.11. The expressivity of $\lambda^{\mathbf{T}}$ is considerably larger than just the primitive recursive functions. When defining a primitive recursive function using a recursion axiom, we are only allowed to recurse over the natural numbers. In $\lambda^{\mathbf{T}}$, $\mathtt{Rec}_\tau$ can recurse over any type τ. The following is an example of a function that is definable in $\lambda^{\mathbf{T}}$ but is not primitive recursive: Let $A : \mathbb{N}^2 \to \mathbb{N}$ be a function such that

$$\begin{array}{rcl} A(0,n) & = & n+1 \\ A(m+1,0) & = & A(m,1) \\ A(m+1,n+1) & = & A(m,A(m+1,n)). \end{array}$$

In [33] this is shown not to be primitive recursive; it is a variant of the Ackermann function. But since we are allowed to recurse over functions of type $\mathtt{N} \to \mathtt{N}$, we can easily define this in $\lambda^{\mathbf{T}}$:

$$\mathtt{ack} := \mathtt{Rec}_{\mathtt{N}\to\mathtt{N}}\ \mathbf{S}\ (\lambda k^{\mathtt{N}} \lambda f^{\mathtt{N}\to\mathtt{N}}.\mathtt{Rec}_{\mathtt{N}}\ (f\,(\mathbf{S0}))(\lambda l^{\mathtt{N}} \lambda n^{\mathtt{N}}.fn)).$$

2.6 Annotated first-order proofs

Proof calculus for mFOL

We will now introduce a *proof calculus* for mFOL, which we will call MQC—minimal quantifier calculus. By proof calculus, we basically mean a type system where the type derivations correspond exactly to the proofs in mFOL.

Definition 2.6.1 (Types of MQC)**.** The *types* of MQC are the formulas of mFOL.

Definition 2.6.2 (Untyped terms of MQC)**.** The untyped *terms* of MQC are

$$\begin{array}{rcl} t, u, v & := & x \mid tu \mid tn \mid \lambda x\, u \mid \lambda \alpha\, u \\ & \mid & \langle t, u \rangle \mid \pi_0 u \mid \pi_1 u \mid \iota_0 u \mid \iota_1 u \\ & \mid & t[x.u, y.v] \mid (n, t) \mid t[(\alpha, x).u] \end{array}$$

where x, y range over an infinite set of λ-variables, α over variables of $\mathcal{L}_{\mathcal{S}}$, and n over terms of $\mathcal{L}_{\mathcal{S}}$.

Definition 2.6.3 (Typing judgments in MQC)**.** An *environment*, Γ, in MQC is a finite set of pairs of distinct λ-variables with formulas. It is typically written on the form $\Gamma = x_1 : \varphi_1, \dots, x_n : \varphi_n$.

A *typing judgment* is a triple of the form $\Gamma \vdash u : \varphi$, and we use it to mean that there exists a derivation using the typing rules from Figure 2.4 with $\Gamma \vdash u : \varphi$ at the root.

Definition 2.6.4 (Reduction rules for MQC)**.** We define the reduction relation $\rightarrow_{\mathrm{MQC}}$ as the compatible closure of the following reduction rules:

$$\begin{array}{rcl} (\lambda x.u)t & \rightarrow_{\beta_1} & u[x := t] \\ (\lambda \alpha.u)t & \rightarrow_{\beta_2} & u[\alpha := t] \\ \pi_0 \langle u_0, u_1 \rangle & \rightarrow_{\pi_0} & u_0 \\ \pi_1 \langle u_0, u_1 \rangle & \rightarrow_{\pi_1} & u_1 \\ \iota_0(u)[x_1.t_1, x_2.t_2] & \rightarrow_{\iota_0} & t_0[x_1 := u] \\ \iota_1(u)[x_1.t_1, x_2.t_2] & \rightarrow_{\iota_1} & t_1[x_2 := u] \\ (n, u)[(\alpha, x).v] & \rightarrow_{\exists} & v[\alpha := n][x := u], \text{ for each term } n \end{array}$$

Theorem 2.6.5 (Curry–Howard correspondence)**.** *$\Gamma \vdash_{mFOL} \varphi$ iff there is a t such that $\Gamma \vdash_{MQC} t : \varphi$.*

$$\Gamma, x : \varphi \vdash x : \varphi$$

$$\frac{\Gamma \vdash u : \varphi \qquad \Gamma \vdash v : \psi}{\Gamma \vdash \langle u, v \rangle : \varphi \wedge \psi} \qquad \frac{\Gamma \vdash u : \varphi_0 \wedge \varphi_1}{\Gamma \vdash \pi_i u : \varphi_i} \text{ for } i = 0, 1$$

$$\frac{\Gamma \vdash u : \varphi_i}{\Gamma \vdash \iota_i u : \varphi_0 \vee \varphi_1} \text{ for } i = 0, 1$$

$$\frac{\Gamma \vdash u : \varphi \vee \psi \qquad \Gamma, x : \varphi \vdash v_0 : \theta \qquad \Gamma, x : B \vdash v_1 : \theta}{\Gamma \vdash u[x.v_0, x.v_1] : \theta}$$

$$\frac{\Gamma, x : \varphi \vdash u : \psi}{\Gamma \vdash \lambda x\, u : A \to \psi} \qquad \frac{\Gamma \vdash u : \varphi \to \psi \qquad \Gamma \vdash v : \varphi}{\Gamma \vdash uv : \psi}$$

$$\frac{\Gamma \vdash u : \varphi}{\Gamma \vdash \lambda \alpha\, u : \forall \alpha \varphi} \ \alpha \notin \mathrm{FV}(\Gamma) \qquad \frac{\Gamma \vdash u : \forall \alpha . \varphi(\alpha)}{\Gamma \vdash ut : \varphi(t)} \ t \text{ is a term of } \mathcal{L}$$

$$\frac{\Gamma \vdash u : \varphi(t)}{\Gamma \vdash (t, u) : \exists \alpha \varphi(\alpha)} \ t \text{ is a term of } \mathcal{L}$$

$$\frac{\Gamma \vdash u : \exists \alpha \varphi \qquad \Gamma, x : \varphi \vdash v : \theta}{\Gamma \vdash u[(\alpha, x).v] : \theta} \ \alpha \notin \mathrm{FV}(C) \cup \mathrm{FV}(\Gamma)$$

Figure 2.4: Type inference rules for MQC

Chapter 3

Friedman's A-translation

In this chapter we will present a proof of the following old theorem by Kreisel [25]:

Theorem 3.0.6. *Peano Arithmetic is a conservative extension of Heyting Arithmetic over the* Π^0_2*-sentences.*

The proof will make use of two techniques that are central to area of classical program extraction, namely the *Gödel–Gentzen double negation translation* and *Friedman's A-translation.*

The theorem has the following corollary, which gives the main motivation to why we want to examine the computational content of classical proofs:

Corollary 3.0.7. *A recursive function is provably total in Peano Arithmetic if and only if it is provably total in Heyting Arithmetic.*

This tells us, that any classical proof of totality of a recursive function can be converted to an intuitionistic proof, and therefore the classical proof must be constructive, and have computational content in some sense.

3.1 The arithmetics PA and HA

We formalize arithmetic as natural deduction systems. Firstly, we have to fix the signature of the language. Notice that we assume to have the concept of primitive recursive relations defined in our meta-language.

Definition 3.1.1 (Signature of arithmetic)**.** Let

$$\mathcal{S} = \{\mathsf{0}, \mathsf{S}, =\} \cup \{P \mid P \text{ is a primitive recursive relation}\}$$

where $\mathsf{0}$ is a nullary function symbol, S is a unary function symbol, $=$ is a binary relation symbol, and P is an n-ary relation symbol, if P is an n-ary primitive recursive relation.

Then the language $\mathcal{L} = \mathcal{L}_{\mathcal{S}}$ consists of all formulas of arithmetic. We will use this language for iFOL and cFOL.

Notation 3.1.2. We will write $\Gamma \vdash_I \varphi$ if $\Gamma \vdash \varphi$ in iFOL, and $\Gamma \vdash_C \varphi$ if $\Gamma \vdash \varphi$ in cFOL.

Definition 3.1.3 (The Peano axioms)**.** Let Ω be the (countable) set of formulas consisting of the universal closures of the following formulas.

Axioms for equality:

(refl): $\alpha = \alpha$

(trans): $\alpha = \beta \wedge \beta = \gamma \to \alpha = \gamma$

(cong$_P$): $\alpha_i = \alpha'_i \to (P(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) = P(\alpha_1, \ldots, \alpha'_i, \ldots, \alpha_n))$
for every n-ary P and $1 \leq i \leq n$

(cong$_{\mathsf{S}}$) $\alpha = \beta \to \mathsf{S}\alpha = \mathsf{S}\beta$

Axioms for successor:

(succ$_1$): $\neg(\mathsf{S}\alpha = \mathsf{0})$

(succ$_2$): $\mathsf{S}\alpha = \mathsf{S}\beta \to \alpha = \beta$

Induction axiom schema:

(ind): $\varphi(\mathsf{0}) \wedge \forall\alpha.(\varphi(\alpha) \to \varphi(\mathsf{S}\alpha)) \to \forall\alpha.\varphi(\alpha)$
for every formula $\varphi(\alpha)$

Defining axioms:

(succ$_P$): $P(\alpha, \mathsf{S}\alpha)$

(const$_P$): $P(\alpha_1, \ldots, \alpha_n, \mathsf{S}^m\mathsf{0})$

(proj$_P$): $P(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_n, \alpha_i)$

(comp$_P$): $R_1(\alpha_1, \ldots, \alpha_n, \beta_1) \wedge \cdots \wedge R_m(\alpha_1, \ldots, \alpha_n, \beta_m)$
$\wedge\, Q(\beta_1, \ldots, \beta_m, \gamma) \to P(\alpha_1, \ldots, \alpha_n, \gamma)$

(rec$_P$): $(Q(\alpha_1, \ldots, \alpha_n, \beta) \to P(\mathsf{0}, \alpha_1, \ldots, \alpha_n, \beta))$
$\wedge\, (P(\gamma, \alpha_1, \ldots, \alpha_n, \delta) \wedge R(\delta, \beta, \alpha_1, \ldots, \alpha_n, \varepsilon)$
$\to P(\mathsf{S}\gamma, \alpha_1, \ldots, \alpha_n, \varepsilon))$

These are the *Peano axioms.*

Definition 3.1.4 (Peano arithmetic and Heyting arithmetic)**.** We say that a formula φ *is derivable in Peano arithmetic*, and write $\vdash_{\mathsf{PA}} \varphi$, if there is a finite subset $\Gamma \subset_\omega \Omega$ of the Peano axioms such that $\Gamma \vdash_C \varphi$. Similarly, we say that φ *is derivable in Heyting arithmetic*, $\vdash_{\mathsf{HA}}$, if $\Gamma \vdash_I \varphi$ for some $\Gamma \subset_\omega \Omega$.

3.2 Double-negation translation

We first define the *double-negation translation* of formulas. It was invented independently by Gödel and Gentzen in the early thirties [15, 18].

Definition 3.2.1 (Double-negation translation)**.** Let φ be a formula. Define the *double-negation translation* φ^- of φ as follows:

$$\begin{aligned}
\bot^- &:= \bot \\
P^- &:= \neg\neg P, \text{ where } P \neq \bot \text{ is atomic} \\
(\varphi \lor \psi)^- &:= \neg\neg(\varphi^- \lor \psi^-) \\
(\varphi \land \psi)^- &:= \varphi^- \land \psi^- \\
(\varphi \to \psi)^- &:= \varphi^- \to \psi^- \\
(\forall\alpha.\varphi)^- &:= \forall\alpha.\varphi^- \\
(\exists\alpha.\varphi)^- &:= \neg\neg\exists\alpha.\varphi^-
\end{aligned}$$

So φ^- is the result of double-negating all atomic, disjunctive and existential subformulas of φ.

Lemma 3.2.2 (Properties of double-negation translation)**.** *Let φ be a formula, Γ a set of formulas, and $\Gamma^- = \{\psi^- \mid \psi \in \Gamma\}$.*

1. $\vdash_C \varphi \leftrightarrow \varphi^-$*,*

2. $\neg\neg\varphi^- \vdash_I \varphi^-$*,*

3. *If $\Gamma \vdash_C \varphi$, then $\Gamma^- \vdash_I \varphi^-$ (this justifies calling it a* translation*).*

Proof. 1. We need to show that $\varphi \vdash_C \varphi^-$ and $\varphi^- \vdash_C \varphi$ for any formula φ. This is done by induction on the complexity of φ, and we only have to consider the atomic, disjunctive, and existential cases. We show the atomic case, the rest are similar. For $P \vdash_C \neg\neg P$ we have the derivation

$$\dfrac{\dfrac{\neg P^x \qquad P}{\bot}}{\neg\neg P}\,x$$

and for the case $\neg\neg P \vdash_C P$ we have

$$\dfrac{P \lor \neg P \qquad P^x \qquad \dfrac{\dfrac{\neg\neg P \qquad \neg P^x}{\bot}}{P}}{P}\,x$$

2. This is also an easy induction. We show just the atomic case, where we need $\neg\neg\neg\neg\varphi \vdash_I \neg\neg\varphi$:

$$\dfrac{\dfrac{\neg\neg\neg\neg P \qquad \dfrac{\dfrac{\neg\neg P^y \qquad \neg P^x}{\bot}}{\neg\neg\neg P}\,y}{\bot}}{\neg\neg P}\,x$$

3. We show this by induction on the depth of the derivation $\Gamma \vdash_C \varphi$. Most of the rules are trivial, those are the rules that iFOL and cFOL have in common. See for example implication elimination:

$$\frac{\Gamma, \varphi \vdash_C \psi}{\Gamma \vdash_C \varphi \to \psi} \quad \text{becomes} \quad \frac{\Gamma^-, \varphi^- \vdash_I \psi^-}{\Gamma^- \vdash_I \varphi^- \to \psi^-}$$

   So we have only the excluded middle rule left. We will only have to show that $\vdash_I \neg\neg(\varphi \vee \neg\varphi)$ for any formula φ, it will then follow that $\Gamma \vdash_I \neg\neg(\varphi^- \vee \neg\varphi^-)$. We show this with the following derivation:

$$\cfrac{\cfrac{\neg(\varphi \vee \neg\varphi)^x \qquad \cfrac{\cfrac{\cfrac{\neg(\varphi \vee \neg\varphi)^x \qquad \cfrac{\varphi^y}{\varphi \vee \neg\varphi}}{\bot}}{\neg\varphi}\; y}{\varphi \vee \neg\varphi}}{\bot}}{\neg\neg(\varphi \vee \neg\varphi)}\; x$$

□

Observation 3.2.3. *In general* not $\varphi \vdash_I \varphi^-$.

This can be shown with a counter-example. One such is $\neg\forall\alpha.P(\alpha) \nvdash_I \neg\forall\alpha.\neg\neg P(\alpha)$, which can be shown using Kripke semantics.

3.3 A-translation

The A-translation was introduced by H. Friedman in [14] to give a simple proof of Kreisel's theorem. The A in the name stems from the name Friedman used for the arbitrary formula that is inserted via the translation.

Definition 3.3.1 (A-translation)**.** Let φ and A be formulas such that no bound variable of φ is free in A. We define the *A-translation* φ^A of φ as follows:

$$\begin{aligned}
\bot^A &:= A \\
P^A &:= P \vee A, \text{ where } P \neq \bot \text{ is atomic} \\
(\varphi \wedge \psi)^A &:= \varphi^A \wedge \psi^A \\
(\varphi \vee \psi)^A &:= \varphi^A \vee \psi^A \\
(\varphi \to \psi)^A &:= \varphi^A \to \psi^A \\
(\forall\alpha.\varphi)^A &:= \forall\alpha.\varphi^A \\
(\exists\alpha.\varphi)^A &:= \exists\alpha.\varphi^A
\end{aligned}$$

So φ^A is the result of substituting all atomic subformulas P with $P \vee A$, and replacing any $\bot$ with A. Note that $(\neg P)^A = P \vee A \to A$.

Lemma 3.3.2 (Properties of the A-translation)**.** *Let φ be a formula, Γ a set of formulas and A a formula such that φ^A and Γ^A are defined, where $\Gamma^A = \{\psi^A \mid \psi \in \Gamma\}$.*

1. $\vdash_C \varphi^A \leftrightarrow \varphi \vee A$
2. $A \vdash_I \varphi^A$
3. *If* $\Gamma \vdash_I \varphi$, *then* $\Gamma^A \vdash_I \varphi^A$
4. *In general* not $\varphi \vdash_I \varphi^A$

Proof. 1. We have to show that $\varphi^A \vdash_C \varphi \vee A$ and $\varphi \vee A \vdash_C \varphi^A$. This is easily done by induction on the complexity of φ. We illustrate by showing one case, that of $(\varphi \wedge \psi) \vee A \vdash_C \varphi^A \wedge \psi^A$:

$$\begin{array}{c} (\varphi\wedge\psi)\vee A \qquad \begin{array}{c} \begin{array}{c} \varphi\wedge\psi^x \\ \hline \varphi \\ \hline \varphi\vee A \\ \hdashline \text{IH} \\ \hdashline \varphi^A \end{array} \quad \begin{array}{c} \varphi\wedge\psi^x \\ \hline \psi \\ \hline \psi\vee A \\ \hdashline \text{IH} \\ \hdashline \psi^A \end{array} \\ \hline \varphi^A\wedge\psi^A \end{array} \qquad \begin{array}{c} \begin{array}{c} A^x \\ \hline \varphi\vee A \\ \hdashline \text{IH} \\ \hdashline \varphi^A \end{array} \quad \begin{array}{c} A^x \\ \hline \psi\vee A \\ \hdashline \text{IH} \\ \hdashline \psi^A \end{array} \\ \hline \varphi^A\wedge\psi^A \end{array} \\ \hline \varphi^A\wedge\psi^A \end{array}\; x$$

2. This is a straight-forward induction on the complexity of φ.

3. This is done by induction on the depth of the derivation of $\Gamma \vdash_I \varphi$. For the ex falso quodlibet rule, the induction hypothesis is that $\Gamma^A \vdash_I A$, but from 2 we have $A \vdash_I \varphi^A$, this together gives us $\Gamma^A \vdash_I \varphi^A$. As for the rest of the rules, they are quite simple. Here is the implication introduction case:

$$\begin{array}{c} \Gamma, \varphi \vdash \psi \\ \hline \Gamma \vdash \varphi \to \psi \end{array} \quad \text{becomes} \quad \begin{array}{c} \text{IH} \\ \hdashline \Gamma^A, \varphi^A \vdash \psi^A \\ \hline \Gamma^A \vdash \varphi^A \to \psi^A \end{array}$$

The rest of the rules without quantifiers are similarly obvious. For the quantifier rules, we have to take care of variable bindings. Here existential introduction:

$$\begin{array}{c} \Gamma \vdash \varphi[\alpha := t] \\ \hline \Gamma \vdash \exists\alpha.\varphi \end{array} \quad \text{becomes} \quad \begin{array}{c} \text{IH} \\ \hdashline \Gamma^A \vdash \varphi^A[\alpha := t] \\ \hline \Gamma^A \vdash \exists\alpha.\varphi^A \end{array}\; \exists_I$$

because $(\varphi[\alpha := t])^A = \varphi^A[\alpha := t]$ and $(\exists\alpha.\varphi)^A = \exists\alpha.\varphi^A$.

□

Observation 3.3.3. *In general* not $\varphi \vdash_I \varphi^A$.

A counter-example for this is $\neg\neg A \nvdash_I (\neg\neg A)^A$.

3.4 The proof

We know from Observation 3.2.3 and Observation 3.3.3 that it does not always hold that $\varphi \vdash_I \varphi^-$ or $\varphi \vdash_I \varphi^A$. But in some cases it does hold, and these are the cases where the A-translation proof method is applicable. In our case, this is HA. We first observe some easy cases:

Observation 3.4.1. *If φ is on one of the forms*

- P,
- $P \wedge Q$,
- $P_1 \wedge \cdots \wedge P_m \to Q$, *or*
- $(P_1 \to P_2) \wedge (Q_1 \wedge Q_2 \to Q_3)$,

where $P, P_1, \ldots, P_m, Q, Q_1, Q_2, Q_3$ are atomic formulas, then $\varphi \vdash_I \varphi^-$ and $\varphi \vdash_I \varphi^A$.

This leads us to the following interesting lemma:

Lemma 3.4.2. *Let φ be a Peano axiom. Then $\vdash_{\mathsf{HA}} \varphi^-$ and $\vdash_{\mathsf{HA}} \varphi^A$.*

Proof. Every axiom, except the induction axiom, is on one of the shapes from Observation 3.4.1. So we only need to check the induction axiom: Let φ be an instance of the induction axiom:

$$\varphi = \psi(\mathbf{0}) \wedge \forall\alpha(\psi(\alpha) \to \psi(\mathbf{S}(\alpha))) \to \forall\alpha.\psi(\alpha),$$

for some formula $\psi(\alpha)$. Now:

$$\begin{aligned} \varphi^- &= \psi^-(\mathbf{0}) \wedge \forall\alpha(\psi^-(\alpha) \to \psi^-(\mathbf{S}(\alpha))) \to \forall\alpha.\psi^-(\alpha), \\ \varphi^A &= \psi^A(\mathbf{0}) \wedge \forall\alpha(\psi^A(\alpha) \to \psi^A(\mathbf{S}(\alpha))) \to \forall\alpha.\psi^A(\alpha), \end{aligned}$$

which are themselves axioms of HA. □

Corollary 3.4.3. *Let φ and A be formulas.*

1. *If $\vdash_{\mathsf{PA}} \varphi$, then $\vdash_{\mathsf{HA}} \varphi^-$;*
2. *If $\vdash_{\mathsf{HA}} \varphi$ and φ^A is defined, then $\vdash_{\mathsf{HA}} \varphi^A$.*

Proof. 1. Let Γ be the axioms used in the derivation $\vdash_{\mathsf{PA}} \varphi$.

$$\Gamma \vdash_C \varphi \implies \Gamma^- \vdash_I \varphi^- \implies \vdash_{\mathsf{HA}} \varphi^-.$$

2. Let Γ be the axioms used in the derivation $\vdash_{\mathsf{HA}} \varphi$.

$$\Gamma \vdash_I \varphi \implies \Gamma^A \vdash_I \varphi^A \implies \vdash_{\mathsf{HA}} \varphi^A.$$

□

Definition 3.4.4 (Π^0_2-, Σ^0_1-formulas)**.** A Σ^0_1-formula is of the form

$$\exists \alpha_1 \cdots \exists \alpha_n . \varphi(\alpha_1, \ldots, \alpha_n),$$

where φ is quantifier-free. If ψ is a Σ^0_1-formula, then

$$\forall \alpha_1 \cdots \forall \alpha_n . \varphi(\alpha_1, \ldots, \alpha_n),$$

is called a Π^0_2-formula.

We will use the following fact to simplify the Σ^0_1-formulas:

Lemma 3.4.5. *For any quantifier-free formula $\varphi(\alpha_1, \ldots, \alpha_n)$, there is a primitive recursive relation $P(\alpha_1, \ldots, \alpha_n)$ such that*

$$\vdash_{\mathsf{HA}} \varphi(\alpha_1, \ldots, \alpha_n) \leftrightarrow P(\alpha_1, \ldots, \alpha_n).$$

Thus, whenever we talk about a Σ^0_1-formula, we only need to consider the ones of the form $\exists \alpha . P(\alpha)$.

Lemma 3.4.6. *If φ is a Σ^0_1-formula, then $\vdash_I \varphi^A \leftrightarrow \varphi \vee A$.*

Proof. Firstly, one can check that

$$\exists \alpha . (\varphi \vee \psi) \leftrightarrow \exists \alpha . \varphi \vee \psi,$$

whenever $\alpha \notin \mathrm{FV}(\psi)$. Let now $\exists \alpha . P(\alpha)$ be a Σ^0_1-formula. Then

$$(\exists \alpha . P(\alpha))^A = \exists \alpha . (P(\alpha) \vee A),$$

and so

$$\vdash_I (\exists \alpha . P(\alpha))^A \leftrightarrow \exists \alpha P(\alpha) \vee A.$$

□

Proof of Theorem 3.0.6

We need to show that $\vdash_{\mathsf{PA}} \varphi$ if and only if $\vdash_{\mathsf{HA}} \varphi$ for any Π^0_2-sentence φ. It is sufficient to show that $\vdash_{\mathsf{PA}} \varphi$ if and only if $\vdash_{\mathsf{HA}} \varphi$ for any Σ^0_1-formula, for whenever we have a Σ^0_1-formula $\varphi(\alpha_1, \ldots, \alpha_n)$ for which $\vdash_{\mathsf{HA}} \varphi(\alpha_1, \ldots, \alpha_n)$ holds, we can apply n universal quantifier introduction rules to *close* it, in order to get a proof of the Π^0_2-sentence $\vdash_{\mathsf{HA}} \forall \alpha_1 \cdots \forall \alpha_n . \varphi(\alpha_1, \ldots, \alpha_n)$, Let $\exists\alpha.P(\alpha)$ be a given Σ^0_1-formula, and set $A := \exists\alpha.P(\alpha)$. Assume that $\vdash_{\mathsf{PA}} A$. We first do a double-negation translation, and get $\vdash_{\mathsf{HA}} \neg\neg A$. By A-translation, we get $\vdash_{\mathsf{HA}} (\neg\neg A)^A$. But

$$(\neg\neg A)^A = (A^A \to A) \to A,$$

and since $\vdash_{\mathsf{HA}} A^A \leftrightarrow A \vee A \leftrightarrow A$, and so $\vdash_{\mathsf{HA}} A^A \to A$, we get

$$\vdash_{\mathsf{HA}} (\neg\neg A)^A \leftrightarrow A.$$

Therefore we can conclude $\vdash_{\mathsf{HA}} A$, as wanted.

Chapter 4

Control operators

The λ-calculus has for a long time been seen as a natural basis for programming languages, and has thus been used as a meta-language to describe features in programming languages at least since Landin used it to study the features of Algol 60 [27]. Since the λ-calculus is purely functional it cannot be used to describe the *jumps* and *labels* of Algol 60, and therefore Landin had to extend the calculus with the non-functional operator J, an example of a *control operator*—an operator that behaves in a non-local way in order to change the control flow of the program execution. Control operators have since been introduced to functional programming languages. The Scheme dialects, e.g., have control operators equal in power to J, namely *catch* and *throw* [38] and *call-with-current-continuation* (*call/cc*) [35]. According to Talcott, the advantage of using control operators is that they "provide a way of pruning unnecessary computation and allow certain computations to be expressed by more compact and conceptually manageable programs." [40].

It was later discovered by Griffin [20] that adding control operators to typed λ-calculi corresponds, via the Curry–Howard correspondence, to adding classical reasoning to the logic. He did this by observing that Felleisen's extension of the λ-calculus with control operators [13] could be typed in such a way that the types of the control operators corresponded to *ex falso quodlibet* and *double negation elimination.*

In this chapter we will first introduce the system $\lambda\mu$ by Parigot [30] which is an extension to simply typed λ-calculus which by means of adding the μ-operator makes it possible to define *call/cc* and *catch-throw*, and with which it is possible to define terms with types that are not otherwise allowed in intuitionistic systems, e.g. Peirce's law. Secondly, in order to get closer to a "real" programming language, we will introduce the $\lambda\mu^{\mathbf{T}}$-calculus by Geuvers, Krebbers and McKinna [16] which is an extension of the $\lambda\mu$-calculus adding the natural numbers as a primitive datatype with a primitive recursor in the style of Gödel's system $\mathbf{T}$.

$$\Gamma, x : \tau; \Delta \vdash x^{\tau} \qquad \frac{\Gamma, x : \sigma; \Delta \vdash t : \tau}{\Gamma; \Delta \vdash \lambda x^{\sigma}.t : \sigma \to \tau}$$

$$\frac{\Gamma; \Delta \vdash t : \sigma \to \tau \qquad \Gamma; \Delta \vdash s : \sigma}{\Gamma; \Delta \vdash ts : \tau} \qquad \frac{\Gamma; \Delta, a : \tau \vdash k : \bot}{\Gamma; \Delta \vdash \mu a^{\tau}.k : \tau}$$

$$\frac{\Gamma; \Delta, a : \tau \vdash t : \tau}{\Gamma; \Delta, a : \tau \vdash [a]t : \bot}$$

Figure 4.1: Typing rules for $\lambda\mu$

4.1 The system $\lambda\mu$

In 1992 M. Parigot [30] introduced the $\lambda\mu$-calculus as a way of extending the Curry–Howard correspondence to classical proofs, by way of adding the control operator μ to the simply typed lambda calculus. Together with the control operator we also introduce a special kind of variables, the *μ-variables* or *addresses*. Therefore, the environments in $\lambda\mu$ will be bipartite; an environment will consist of a set Γ of λ-variables together with types as usual, and a set Δ of μ-variables together with types.

Definition 4.1.1 (Terms of $\lambda\mu$)**.** The terms of $\lambda\mu$ are defined inductively over an infinite set of λ-variables $(x, y, z, \dots)$ and an infinite set of μ-variables $(a, b, c, \dots)$ as follows

$$\begin{aligned} t, s &::= x \mid \lambda x^{\tau}.t \mid ts \mid \mu a^{\tau}.k \\ k &::= [a]t \end{aligned}$$

Here, τ ranges over simple types as defined in Definition 2.4.1.

Definition 4.1.2 (Free variables)**.** We let $\mathrm{FV}(t)$ denote the set of free λ-variables in t, while $\mathrm{FCV}(t)$ denotes the set of free μ-variables.

Definition 4.1.3 (Typing judgments in $\lambda\mu$)**.** The types of $\lambda\mu$ are the same as those in $\lambda_{\to}$ (Definition 2.4.1), with an extra atomic type $\bot$ (read *bottom*). A typing judgment $\Gamma; \Delta \vdash t : \rho$ is *derivable* in $\lambda\mu$ if there is a derivation tree that uses the rules of Figure 4.1 with $\Gamma; \Delta \vdash t : \rho$ as the conclusion.

Notice that the first three rules in Figure 4.1 are the same as the rules of $\lambda_{\to}$ (Figure 2.2). The two new rules are known as, respectively, *activate* and *passivate*.

Example 4.1.4. In $\lambda\mu$ we can inhabit the type of the non-intuitionistic *Peirce's law* $((p \to q) \to p) \to p$. We get the term

$$\texttt{peirce} := \lambda x^{(p \to q) \to p} \mu a^{p}.[a]x(\lambda z^{p} \mu b^{q}.[a]z)$$

by the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{x : (p \to q) \to p \qquad \dfrac{\dfrac{\dfrac{z : p}{[a]z : \bot}}{\mu b^q.[a]z : q}}{\lambda z^p \mu b^q.[a]z : p \to q}}{x(\lambda z^p \mu b^q.[a]z) : p}}{[a]x(\lambda z^p \mu b^q.[a]z) : \bot}}{\mu a^p.[a]x(\lambda z^p \mu b^q.[a]z) : p}}{\lambda x^{(p\to q)\to p} \mu a^p.[a]x(\lambda z^p \mu b^q.[a]z) : ((p \to q) \to p) \to p}$$

Theorem 4.1.5. *The strength of $\lambda\mu$ is exactly minimal classical propositional logic. I.e.,*

$$\Gamma \vdash \varphi \textit{ in minimal classical logic}$$

$$\Longleftrightarrow$$

$$\textit{there is some term } t \textit{ in } \lambda\mu \textit{ such that } \Gamma; \emptyset \vdash t : \varphi.$$

A proof of this can be found in [24].

Reduction in $\lambda\mu$

In order to define the reduction rules we need to introduce a new notion of substitution, namely *structural substitution.*

Definition 4.1.6 (Call-by-name contexts)**.** A *call-by-name evaluation context* is defined as

$$E ::= \Box \mid Et,$$

where t ranges over terms.

Definition 4.1.7 (Structural substitution)**.** Let t be a $\lambda\mu$-term, and let a, b be μ-variables and E a call-by-name evaluation context. We define the *structural substitution* $t[a := bE]$ of b and E for a by induction as follows:

$$\begin{aligned}
x[a := bE] &:= x \\
(\lambda x.t)[a := bE] &:= \lambda x.t[a := bE] \\
(ts)[a := bE] &:= t[a := bE]s[a := bE] \\
(\mu a.k)[a := bE] &:= \mu a.k \\
(\mu c.k)[a := bE] &:= \mu c.k[a := bE] \quad \text{if } c \neq a \\
([a]t)[a := bE] &:= [b]E[t[a := bE]] \\
([c]t)[a := bE] &:= [c]t[a := bE] \quad \text{if } c \neq a
\end{aligned}$$

Definition 4.1.8 (Reduction)**.** We define the reduction relation $\to$ on $\lambda\mu$ as the compatible closure of the following rules:

$$\begin{array}{rll} (\lambda x.t)s & \to_\beta & t[x := s] \\ (\mu a.k)t & \to_{\mu R} & \mu a.k[a := a\,(\Box t)] \\ \mu a.[a]t & \to_{\mu\eta} & t \quad \text{if } a \notin \mathrm{FCV}(t) \\ [a]\mu b.k & \to_{\mu\iota} & k[b := a\,\Box] \end{array}$$

Definition 4.1.9 (Catch and throw)**.** We define the terms $\texttt{catch}_a\ t$ and $\texttt{throw}_a\ t$ as follows:

$$\begin{aligned} &\texttt{catch}_a\ t := \mu a.[a]t \\ &\texttt{throw}_a\ t := \mu b.[a]t \quad \text{where } b \notin \mathrm{FCV}([a]t) \end{aligned}$$

Lemma 4.1.10. *The terms* `catch` *and* `throw` *behaves as follows, where E is a call-by-name context:*

1. $E[\texttt{throw}_a\ t] \twoheadrightarrow \texttt{throw}_a\ t$,

2. $\texttt{catch}_a\ (\texttt{throw}_a\ t) \twoheadrightarrow \texttt{catch}_a\ t$

3. $\texttt{catch}_a\ t \twoheadrightarrow t$ *if* $a \notin \mathrm{FCV}(t)$

4. $\texttt{throw}_b\ (\texttt{throw}_a\ t) \twoheadrightarrow \texttt{throw}_a\ t$

Proof. For the first reduction, do an induction on the structure of E. The rest follows directly from the definitions and the reduction rules. □

The $\lambda\mu$-calculus satisfies the main meta-theoretical theorems:

Theorem 4.1.11. *$\lambda\mu$ is confluent.*

Proof. A proof can be found in [24]. □

Theorem 4.1.12. *$\lambda\mu$ satisfies subject reduction.*

Proof. A proof can be found in [24]. □

Theorem 4.1.13. *$\lambda\mu$ is strongly normalizing.*

Proof. This is proven in [31]. □

$$\Gamma, x : \tau; \Delta \vdash x^{\tau} \qquad \frac{\Gamma, x : \sigma; \Delta \vdash t : \tau}{\Gamma; \Delta \vdash \lambda x^{\sigma}.t : \sigma \to \tau}$$

$$\frac{\Gamma; \Delta \vdash t : \sigma \to \tau \qquad \Gamma; \Delta \vdash s : \sigma}{\Gamma; \Delta \vdash ts : \tau} \qquad \frac{\Gamma; \Delta, a : \tau \vdash k : \bot}{\Gamma; \Delta \vdash \mu a^{\tau}.k : \tau}$$

$$\frac{\Gamma; \Delta, a : \tau \vdash t : \tau}{\Gamma; \Delta, a : \tau \vdash [a]t : \bot}$$

$$\Gamma; \Delta \vdash \mathbf{0} : \mathtt{N} \qquad \frac{\Gamma; \Delta \vdash t : \mathtt{N}}{\Gamma; \Delta \vdash \mathbf{S}t : \mathtt{N}}$$

$$\frac{\Gamma; \Delta \vdash t : \tau \qquad \Gamma; \Delta \vdash s : \mathtt{N} \to \tau \to \tau \qquad \Gamma; \Delta \vdash r : \mathtt{N}}{\Gamma; \Delta \vdash \mathtt{Rec}_{\tau}\ t\ s\ r : \tau}$$

Figure 4.2: Typing rules for $\lambda\mu^{\mathbf{T}}$

4.2 The system $\lambda\mu^{\mathbf{T}}$

The $\lambda\mu^{\mathbf{T}}$-calculus arises from the $\lambda\mu$-calculus in the same way that the $\lambda^{\mathbf{T}}$-calculus arises from the $\lambda_{\to}$-calculus, namely by "hard-coding" the natural numbers into the system by adding an atomic type $\mathtt{N}$, primitive terms $\mathbf{0} : \mathtt{N}$ and $\mathbf{S} : \mathtt{N} \to \mathtt{N}$, and a recursor $\mathtt{Rec}$.

Definition 4.2.1 (Terms of $\lambda\mu^{\mathbf{T}}$)**.** The terms of $\lambda\mu^{\mathbf{T}}$ are defined inductively over an infinite set of λ-variables $(x, y, z, \dots)$ and an infinite set of μ-variables $(a, b, c, \dots)$ as follows:

$$\begin{aligned} t, s, r &:= x \mid \lambda x^{\tau}.t \mid ts \mid \mu a^{\tau}.k \mid \mathbf{0} \mid \mathbf{S}t \mid \mathtt{Rec}_{\tau}\ t\ s\ r \\ k &:= [a]t \end{aligned}$$

Here, τ ranges over $\lambda^{\mathbf{T}}$-types, as defined in Definition 2.5.1.

Definition 4.2.2 (Free variables)**.** As in $\lambda\mu$, we let $\mathrm{FV}(t)$ and $\mathrm{FCV}(t)$ denote the sets of free λ-variables and μ-variables, respectively.

We define substitution $t[x := s]$ in the obvious way, such that it is capture avoiding for both λ- and μ-variables.

Definition 4.2.3 (Typing judgments in $\lambda\mu^{\mathbf{T}}$)**.** A typing judgment $\Gamma; \Delta \vdash t : \rho$ is *derivable in* $\lambda\mu^{\mathbf{T}}$ if there is a derivation tree that uses the rules of Figure 4.2 with $\Gamma; \Delta \vdash t : \rho$ as the conclusion, and similarly, a typing judgment $\Gamma; \Delta \vdash k : \bot$ is derivable in $\lambda\mu^{\mathbf{T}}$ in case it is the conclusion of such a derivation tree.

Lemma 4.2.4. *Typing judgments in $\lambda\mu^{\mathbf{T}}$ are closed under weakening of the environment, i.e., if $\Gamma \subseteq \Gamma'$, $\Delta \subseteq \Delta'$, and $\Gamma; \Delta \vdash t : \tau$, then $\Gamma'; \Delta' \vdash t : \tau$*

Proof. By easy induction on the depth of the derivation. □

When we work with *numerals*, we will abbreviate them as $\underline{n} := \mathsf{S}^n\mathbf{0}$.

In order to define reduction in $\lambda\mu^{\mathbf{T}}$ we will first need the concepts of *contexts* and *structural substitution*.

Definition 4.2.5 (Contexts)**.** We define the $\lambda\mu^{\mathbf{T}}$*-contexts* as follows:

$$E ::= \Box \mid Et \mid \mathsf{S}E \mid \texttt{Rec}\ t\ s\ E,$$

Such a context is *singular* if the depth of the □ is exactly one, i.e.:

$$E^s ::= \Box t \mid \mathsf{S}\Box \mid \texttt{Rec}\ t\ s\ \Box.$$

Definition 4.2.6 (Context substitution, composition)**.** Given a context E and a term t, we define $E[s]$ as follows:

$$\begin{aligned} \Box[s] &:= s \\ (Et)[s] &:= E[s]t \\ (\mathsf{S}E)[s] &:= \mathsf{S}E[s] \\ (\texttt{Rec}\ t\ s\ E)[s] &:= \texttt{Rec}\ t\ s\ E[s] \end{aligned}$$

Given two contexts E and F, we define their composition EF thus:

$$\begin{aligned} \Box F &:= F \\ (Et)F &:= (EF)t \\ (\mathsf{S}E)F &:= \mathsf{S}(EF) \\ (\texttt{Rec}\ t\ s\ E)F &:= \texttt{Rec}\ t\ s\ (EF) \end{aligned}$$

Definition 4.2.7 (Structural substitution)**.** We define the *structural substitution* $t[a := bE]$ of a μ-variable b and a context E for a μ-variable a as follows:

$$\begin{aligned} x[a := bE] &:= x \\ (\lambda x.t)[a := bE] &:= \lambda x.t[a := bE] \\ (ts)[a := bE] &:= t[a := bE]s[a := bE] \\ \mathbf{0}[a := bE] &:= \mathbf{0} \\ (\mathsf{S}t)[a := bE] &:= \mathsf{S}(t[a := bE]) \\ (\texttt{Rec}\ t\ s\ r)[a := bE] &:= \texttt{Rec}\ (t[a := bE])\ (s[a := bE])\ (r[a := bE]) \\ (\mu c.k)[a := bE] &:= \mu c.k[a := bE] \\ ([a]t)[a := bE] &:= [b]E[t[a := bE]] \\ ([c]t)[a := bE] &:= [c]t[a := bE] \quad \text{if } c \neq a \end{aligned}$$

We are now ready to define the reduction rules of $\lambda\mu^{\mathbf{T}}$.

Definition 4.2.8 (Reduction rules of $\lambda\mu^{\mathbf{T}}$)**.** We define the reduction relation $\to$ as the compatible closure of the following rules:

$$\begin{array}{rll}
(\lambda x.t)s & \to_{\beta} & t[x := s] \\
\mathsf{S}(\mu a.k) & \to_{\mu\mathsf{S}} & \mu a.k[a := a\ (\mathsf{S}\Box)] \\
(\mu a.k)t & \to_{\mu R} & \mu a.k[a := a\ (\Box t)] \\
\mu a.[a]t & \to_{\mu\eta} & t \quad \text{if } a \notin \mathrm{FCV}(t) \\
[a]\mu b.k & \to_{\mu i} & k[b := a\ \Box] \\
\mathtt{Rec}\ t\ s\ \mathbf{0} & \to_{\mathbf{0}} & t \\
\mathtt{Rec}\ t\ s\ (\mathsf{S}\underline{n}) & \to_{\mathsf{S}} & s\ \underline{n}\ (\mathtt{Rec}\ t\ s\ \underline{n}) \\
\mathtt{Rec}\ t\ s\ (\mu a.k) & \to_{\mu\mathrm{N}} & \mu a.k[a := a\ (\mathtt{Rec}\ t\ s\ \Box)]
\end{array}$$

The $\lambda\mu^{\mathbf{T}}$-calculus fulfills the following important meta-theorems, proofs for all of which can be found in [16].

Theorem 4.2.9 (Subject reduction)**.** *The $\lambda\mu^{\mathbf{T}}$-calculus satisfies subject reduction, i.e., if $\Gamma; \Delta \vdash t : \tau$ and $t \to t'$, then $\Gamma; \Delta \vdash t' : \tau$.*

Theorem 4.2.10 (Confluence)**.** *The reduction relation $\to$ is confluent, i.e., if $t_1 \twoheadrightarrow t_2$ and $t_1 \twoheadrightarrow t_3$, then there is a term t_4 such that $t_2 \twoheadrightarrow t_4$ and $t_3 \twoheadrightarrow t_4$.*

$$\begin{tikzcd}
 & t_1 \arrow[dl, two heads] \arrow[dr, two heads] & \\
t_2 \arrow[dr, dashed, two heads] & & t_3 \arrow[dl, dashed, two heads] \\
 & t_4 &
\end{tikzcd}$$

Theorem 4.2.11 (Strong normalization)**.** *$\lambda\mu^{\mathbf{T}}$ is strongly normalizing: If $\Gamma; \Delta \vdash t : \sigma$, then there is no infinite reduction chain*

$$u = u_1 \to u_2 \to u_3 \to \cdots$$

Chapter 5

Arithmetic with exceptions: $\mathsf{HA} + \mathsf{EM}_1$

In this chapter we present Aschieri and Berardi's system $\mathsf{HA} + \mathsf{EM}_1$ [6], and show its strong normalization, using a new proof method by Aschieri [3]. The system is an extension of Heyting arithmetic with a restricted version of the law of the excluded middle, EM_1, which allows us to use in our proofs all disjunctions of the form $\forall\alpha.P(\alpha) \vee \exists\alpha.\neg P(\alpha)$, where P is an atomic formula.

There are multiple reasons for choosing the restricted version EM_1. In contrast to the full EM, the truth of EM_1 can be computed in the limit, in the sense of Gold [19]. Every time an instance $P(n)$ of the hypothesis $\forall\alpha.P(n)$ is used, it can effectively be checked whether this instance is true or not. If it is not, then we are immediately provided with a witness for the truth of $\exists\alpha.\neg P(\alpha)$.

Furthermore, many important classical theorems of mathematics can be proved with only EM_1 [1,10].

5.1 Post rules

Since we will describe a mathematical theory we need an atomic language and non-logical axioms. The computations that we are interested in are not the ones that happen at the atomic level, so therefore we will not bother with actually describing it. Instead, we will use Post rules as in [34] to cover up the computations happening at the atomic level, in order to simplify the low-level reasoning.

Definition 5.1.1. A *Post rule* is an inference rule of the form

$$\frac{\mathsf{P}_1 \qquad \mathsf{P}_2 \qquad \cdots \qquad \mathsf{P}_n}{\mathsf{Q}}$$

where $\mathsf{P}_1, \mathsf{P}_2, \dots, \mathsf{P}_n, \mathsf{Q}$ are atomic formulas, such that for every substitution $\sigma = [\alpha_1 := n_1, \alpha_2 := n_2, \dots, \alpha_k := n_k]$, $\mathsf{P}_1\sigma \equiv \dots \equiv \mathsf{P}_n\sigma \equiv \texttt{True}$ implies $\mathsf{Q}\sigma \equiv \texttt{True}$.

Since we work in arithmetic, we will assume there to be Post rules for every purely universal arithmetical fact that holds in the standard model of PA, i.e. facts of the form

$$\forall \vec{x}(P_1(\vec{x}) \wedge \dots \wedge P_n(\vec{x}) \to Q(\vec{x})),$$

where P_i, Q are atomic formulas. This includes all the Peano axioms except for the induction axiom scheme. We have, for example, the axioms of equality:

$$\frac{}{\mathsf{eq}(t,t)}\ (\text{refl}) \qquad \frac{\mathsf{eq}(t_1,t_2) \qquad \mathsf{eq}(t_2,t_3)}{\mathsf{eq}(t_1,t_3)}\ (\text{trans})$$

$$\frac{\mathsf{eq}(t_1,t_2) \qquad \mathsf{P}[\alpha := t_1]}{\mathsf{P}[\alpha := t_2]}\ (\text{cong}_{\mathsf{P}})$$

And the Peano axioms for the successor:

$$\frac{\mathsf{eq}(\mathbf{S}t_1, \mathbf{S}t_2)}{\mathsf{eq}(t_1,t_2)}\ (\text{succ}_1) \qquad \frac{\mathsf{eq}(\mathbf{0}, \mathbf{S}t)}{\bot}\ (\text{succ}_2)$$

where $\bot$ is the false relation, for which we have the ex falso Post rule

$$\frac{\bot}{\mathsf{P}}$$

This rule is what makes our system intuitionistic, by making the ex falso rule admissible to the system.

Also, we have Post rules for all defining axioms of each primitive recursive relation, e.g.

$$\frac{}{\mathsf{add}(t, \mathbf{0}, t)} \qquad \frac{\mathsf{add}(t_1,t_2,t_3)}{\mathsf{add}(t_1, \mathbf{S}t_2, \mathbf{S}t_3)}$$

$$\frac{}{\mathsf{mult}(t, \mathbf{0}, \mathbf{0})} \qquad \frac{\mathsf{mult}(t_1,t_2,t_3) \qquad \mathsf{add}(t_1,t_3,t_4)}{\mathsf{mult}(t_1, \mathbf{S}t_2, t_4)}$$

A trick that we will make use of below is to weaken a Post rule. Given a rule

$$\frac{\mathsf{P}_1 \qquad \mathsf{P}_2 \qquad \cdots \qquad \mathsf{P}_n}{\mathsf{Q}}$$

it can be useful to add an irrelevant premise, such that it becomes

$$\frac{\mathsf{P}_1 \qquad \mathsf{P}_2 \qquad \cdots \qquad \mathsf{P}_n \qquad \mathsf{S}}{\mathsf{Q}}$$

The reason for using Post rules is that we then do not have to bother with low-level reasoning and computation. The idea is, that whenever a Post rule is used in a proof, it could be replaced by a computation in a simple programming language, like $\lambda^{\mathbf{T}}$.

5.2 HA

We can now definasdfe the first-order system of Heyting arithmetic, HA, which will be used as the basis on which we can add classical reasoning.

To start with, we formally fix the language.

Definition 5.2.1 (Variables)**.** We have two different types of variables:

- Numerical variables, α, β, γ, representing natural numbers.
- Proof term variables, x, y, z, which correspond to the usual lambda calculus variables.

Definition 5.2.2 (Formulas of HA)**.** We define the language $\mathcal{L}$ of HA.

1. The terms in $\mathcal{L}$:
$$t, r ::= \mathbf{0} \mid \mathbf{S}t \mid \alpha$$
where α ranges over numerical variables. A *numeral* is a closed term, i.e., a term of the form $\mathbf{S} \cdots \mathbf{S0}$.
2. There is an atomic formula $\mathsf{P}(t_1, \ldots, t_n)$ for each primitive recursive relation $P \subseteq \mathbb{N}^n$. If $\mathsf{P}(\vec{t})$ is a closed atomic formula, i.e., all t_i are numerals, then we can write either $\mathsf{P}(\vec{t}) \equiv \mathtt{True}$ or $\mathsf{P}(\vec{t}) \equiv \mathtt{False}$ if $\vec{t} \in P$ or $\vec{t} \notin P$, respectively.
3. The formulas, φ, ψ, θ, are built from atomic formulas by the connectives $\vee, \wedge, \rightarrow, \forall, \exists$ as usual, with quantifiers ranging over numeric variables $\alpha, \beta, \gamma, \ldots$.

The negation of an atomic formula $\mathsf{P}^{\perp}(\vec{t})$ is defined as the atomic formula representing the complementing primitive recursive relation $\mathbb{N}^n \setminus P$, while the negation of a non-atomic formula $\neg\varphi$ is defined in the usual way as $\varphi \rightarrow \perp$, where $\perp$ is the atom representing the empty relation. Notice that negation of atoms is an involution: $(\mathsf{P}^{\perp})^{\perp} \equiv \mathsf{P}$.

Definition 5.2.3 (Free variables)**.** Given a formula φ, the set $\mathrm{FV}(\varphi)$ is defined as the set of numerical variables occurring in φ that are not bound by any quantifiers.

Definition 5.2.4 (Capture avoiding substitution in formulas of HA)**.** Let t, r be terms of $\mathcal{L}$ and α a numerical variable. We firstly define $r[\alpha := t]$, r with t substituted for α, recursively on r as follows:

- $\mathbf{0}[\alpha := t] := \mathbf{0}$,
- $(\mathbf{S}r)[\alpha := t] := \mathbf{S}r[\alpha := t]$,
- $\alpha[\alpha := t] := t$,

- $\beta[\alpha := t] := \beta$.

Let now φ be any formula. We define φ with t substituted for α, $\varphi[\alpha := t]$, recursively on φ as follows:

- $\mathsf{P}(t_1, \dots, t_n)[\alpha := t] := \mathsf{P}(t_1[\alpha := t], \dots, t_n[\alpha := t])$,
- $(\varphi \vee \psi)[\alpha := t] := \varphi[\alpha := t] \vee \psi[\alpha := t]$,
- $(\varphi \wedge \psi)[\alpha := t] := \varphi[\alpha := t] \wedge \psi[\alpha := t]$,
- $(\varphi \to \psi)[\alpha := t] := \varphi[\alpha := t] \to \psi[\alpha := t]$,
- $(\forall\alpha.\varphi)[\alpha := t] := \forall\alpha.\varphi$,
- $(\forall\beta.\varphi)[\alpha := t] := \forall\beta.\varphi[\alpha := t]$,
- $(\exists\alpha.\varphi)[\alpha := t] := \exists\alpha.\varphi$,
- $(\exists\beta.\varphi)[\alpha := t] := \exists\beta.\varphi[\alpha := t]$.

Definition 5.2.5 (Proof terms of HA)**.** The untyped proof terms in HA are the following:

$$\begin{array}{rcl} u, v, w & := & x \mid uv \mid un \mid \lambda x\, u \mid \lambda\alpha\, u \\ & \mid & \langle u, v\rangle \mid \pi_0 u \mid \pi_1 u \mid \iota_0 u \mid \iota_1 u \\ & \mid & u[x.v, y.w] \mid (n, u) \mid u[(\alpha, x).v] \\ & \mid & \texttt{Rec}\ u\ v\ n \mid \texttt{r}\ u_1\ \cdots\ u_m \end{array}$$

where x, y range over proof term variables and n over $\mathcal{L}$-terms. The term $\texttt{r}$ will be used to represent usages of Post rules.

Definition 5.2.6 (Capture avoiding substitution in terms of HA)**.** We define two notions of capture free substitution in terms of HA: Let u, v be terms of HA, t a term of $\mathcal{L}$, α a numerical variable and x a λ-variable. We define the notions $u[x := v]$ and $u[\alpha := t]$ in the standard way.

Definition 5.2.7 (Typing judgments in HA)**.** An *environment*, Γ, in HA is a finite set of pairs of distinct λ-variables and types. It is typically written on the form $\Gamma = x_1 : \varphi_1, \dots, x_n : \varphi_n$.

A *typing judgment* is a triple of the form $\Gamma \vdash u : \varphi$, and we use it to mean that there exists a derivation using the typing rules from Figure 5.1 with $\Gamma \vdash u : \varphi$ at the root.

The following lemma tells us that we can encode any quantifier-free formula into an atom, if we wish.

Lemma 5.2.8. *Let φ be a quantifier-free formula. There is an atomic formula P such that $\vdash \varphi \leftrightarrow \mathsf{P}$.*

Axioms:

$$\Gamma, x : \varphi \vdash x : \varphi$$

Conjunction:

$$\frac{\Gamma \vdash u : \varphi \qquad \Gamma \vdash v : \psi}{\Gamma \vdash \langle u, v\rangle : \varphi \wedge \psi} \qquad \frac{\Gamma \vdash u : \varphi \wedge \psi}{\Gamma \vdash \pi_0 u : \varphi} \qquad \frac{\Gamma \vdash u : \varphi \wedge \psi}{\Gamma \vdash \pi_1 u : \psi}$$

Implication:

$$\frac{\Gamma, x : \varphi \vdash u : \psi}{\Gamma \vdash \lambda x\, u : \varphi \to \psi} \qquad \frac{\Gamma \vdash u : \varphi \to \psi \qquad \Gamma \vdash v : \varphi}{\Gamma \vdash uv : \psi}$$

Disjunction:

$$\frac{\Gamma \vdash u : \varphi}{\Gamma \vdash \iota_0 u : \varphi \vee \psi} \qquad \frac{\Gamma \vdash u : \psi}{\Gamma \vdash \iota_1 u : \varphi \vee \psi}$$

$$\frac{\Gamma \vdash u : \varphi \vee \psi \qquad \Gamma, x : \varphi \vdash v_1 : \theta \qquad \Gamma, x : \psi \vdash v_2 : \theta}{\Gamma \vdash u[x.v_1, x.v_2] : \theta}$$

Universal quantification:

$$\frac{\Gamma \vdash u : \varphi}{\Gamma \vdash \lambda \alpha\, u : \forall \alpha.\varphi} \qquad \frac{\Gamma \vdash u : \forall \alpha.\varphi(\alpha)}{\Gamma \vdash ut : \varphi(t)}$$

where t is any term of $\mathcal{L}$ and α does not occur free in any formula in Γ.

Existential quantification:

$$\frac{\Gamma \vdash u : \varphi[\alpha := t]}{\Gamma \vdash (t, u) : \exists \alpha.\varphi} \qquad \frac{\Gamma \vdash u : \exists \alpha.\varphi \qquad \Gamma, x : A \vdash v : \theta}{\Gamma \vdash u[(\alpha, x).v] : \theta}$$

where t is a term of $\mathcal{L}$ and α is not free in θ nor in any formula in Γ.

Induction:

$$\frac{\Gamma \vdash u : \varphi(\mathbf{0}) \qquad \Gamma \vdash v : \forall \alpha.\varphi(\alpha) \to \varphi(\mathsf{S}\alpha)}{\Gamma \vdash \mathtt{Rec}\, u\ v\ t : \varphi(t)}$$

where t is any term of $\mathcal{L}$.

Post rules:

$$\frac{\Gamma \vdash u_1 : \mathsf{P}_1 \qquad \Gamma \vdash u_2 : \mathsf{P}_2 \qquad \cdots \qquad \Gamma \vdash u_n : \mathsf{P}_n}{\Gamma \vdash \mathtt{r}\ u_1\ u_2\ \cdots\ u_n : \mathsf{Q}}$$

where $\mathsf{P}_1, \ldots, \mathsf{P}_n, \mathsf{Q}$ are atomic formulas and the rule is a Post rule in arithmetic. If there are no premises to the rule, we will write `True` instead of `r`.

Figure 5.1: Typing rules for HA

Proof. The proof is by induction on the complexity of φ. By definition, the atomic case is trivial. For $\varphi \wedge \psi$, there are primitive recursive relations $\mathcal{P}_1, \mathcal{P}_2$ corresponding to φ and ψ respectively. Define $\mathcal{P}$ as the primitive recursive relation that is true whenever both $\mathcal{P}_1$ and $\mathcal{P}_2$ are true. Similarly with $\vee$. For $\varphi \to \psi$, define $\mathcal{P}$ as the relation that is true when $\mathcal{P}_2$ is true, or when $\mathcal{P}_1$ is false. □

Reduction for HA

Definition 5.2.9 (Reduction rules for HA)**.** We define the reduction relation $\to_{\mathsf{HA}}$ as the compatible closure of the following reduction rules:

$$\begin{array}{rll}
(\lambda x.u)t & \to_{\beta_1} & u[x := t] \\
(\lambda \alpha.u)t & \to_{\beta_2} & u[\alpha := t] \\
\pi_0 \langle u_0, u_1 \rangle & \to_{\pi_0} & u_0 \\
\pi_1 \langle u_0, u_1 \rangle & \to_{\pi_1} & u_1 \\
\iota_0(u)[x_1.t_1, x_2.t_2] & \to_{\iota_0} & t_0[x_0 := u] \\
\iota_1(u)[x_1.t_1, x_2.t_2] & \to_{\iota_1} & t_1[x_1 := u] \\
(n, u)[(\alpha, x).v] & \to_{\exists} & v[\alpha := n][x := u], \text{ for each numeral } n \\
\mathtt{Rec}\ u\ v\ \mathbf{0} & \to_{\mathtt{Rec}_1} & u \\
\mathtt{Rec}\ u\ v\ (\mathbf{S}n) & \to_{\mathtt{Rec}_2} & v\ n\ (\mathtt{Rec}\ u\ v\ n)
\end{array}$$

The following lemma tells us that the logic is indeed intuitionistic.

Lemma 5.2.10 (Ex falso quodlibet)**.** *There exist a term* $\mathtt{efq}_\varphi$ *for any formula* φ *such that*

$$\vdash \mathtt{efq}_\varphi : \bot \to \varphi.$$

Proof. We show this by induction on the complexity of the formula φ.

- $\varphi = \mathsf{P}$ (atomic): Since we have the Post rule

$$\frac{\bot}{\mathsf{P}}$$

for any atomic formula P we have the following derivation:

$$\dfrac{\dfrac{x : \bot \vdash x : \bot}{x : \bot \vdash \mathtt{r}\, x : \mathsf{P}}}{\vdash \lambda x.\mathtt{r}\, x : \bot \to \mathsf{P}}$$

so $\mathtt{efq}_{\mathsf{P}} := \lambda x.\mathtt{r}\, x$.

- $\varphi = \psi_1 \wedge \psi_2$: Let $\mathtt{efq}_{\psi_1 \wedge \psi_2} := \lambda x \, \langle \mathtt{efq}_{\psi_1} x, \mathtt{efq}_{\psi_2} x \rangle$, for

$$\dfrac{\dfrac{x : \bot \vdash \mathtt{efq}_{\psi_1} x : \psi_1 \qquad x : \bot \vdash \mathtt{efq}_{\psi_2} x : \psi_2}{x : \bot \vdash \langle \mathtt{efq}_{\psi_1} x, \mathtt{efq}_{\psi_2} x \rangle : \psi_1 \wedge \psi_2}}{\vdash \lambda x \, \langle \mathtt{efq}_{\psi_1} x, \mathtt{efq}_{\psi_2} x \rangle : \bot \to \psi_1 \wedge \psi_2}$$

- $\varphi = \psi_1 \to \psi_2$: Let $\mathtt{efq}_{\psi_1 \to \psi_2} := \lambda x \lambda y\, \mathtt{efq}_{\psi_2}\, x$, for

$$\cfrac{\cfrac{x : \bot, y : \psi_1 \vdash \mathtt{efq}_{\psi_2}\, x : \psi_2}{x : \bot \vdash \lambda y\, \mathtt{efq}_{\psi_2}\, x : \psi_1 \to \psi_2}}{\vdash \lambda x \lambda y\, \mathtt{efq}_{\psi_2}\, x : \bot \to \psi_1 \to \psi_2}$$

- $\varphi = \forall \alpha\, \psi$: Similarly, let $\mathtt{efq}_{\forall \alpha \psi} := \lambda x \lambda \alpha\, \mathtt{efq}_{\psi}\, x$.

- $\varphi = \psi_1 \lor \psi_2$: Let $\mathtt{efq}_{\varphi} = \lambda x\, \iota_0(\mathtt{efq}_{\psi_1} x)$:

$$\cfrac{\cfrac{x : \bot \vdash \mathtt{efq}_{\psi_1} x : \psi_1}{x : \bot \vdash \iota_0(\mathtt{efq}_{\psi_1} x) : \psi_1 \lor \psi_2}}{\vdash \lambda x\, \iota_0(\mathtt{efq}_{\psi_1} x) : \bot \to \psi_1 \lor \psi_2}$$

- $\varphi = \exists \alpha\, \psi$: Let $\mathtt{efq}_{\exists \alpha\, \psi} = \lambda x\, (\mathbf{0}, \mathtt{efq}_{\psi[\alpha := \mathbf{0}]} x)$:

$$\cfrac{\cfrac{x : \bot \vdash \mathtt{efq}_{\psi[\alpha := \mathbf{0}]} x : \psi[\alpha := \mathbf{0}]}{x : \bot \vdash (\mathbf{0}, \mathtt{efq}_{\psi[\alpha := \mathbf{0}]} x) : \exists \alpha\, \psi}}{\vdash \lambda x\, (\mathbf{0}, \mathtt{efq}_{\psi[\alpha := \mathbf{0}]} x) : \bot \to \exists \alpha\, \psi}$$

□

5.3 HA + EM$_1$

The system HA + EM$_1$, introduced by Aschieri, Berardi and Birolo in [6], arises from HA by adding a limited amount of classical reasoning, namely the EM$_1$-rule, the law of excluded middle restricted to Π^0_1-formulas. Often, one sees the law of excluded middle defined as a rule of the form

$$\cfrac{}{\varphi \lor \neg \varphi}$$

But since this classical axiom does not contain any computational content by itself, we will instead combine it with the disjunction elimination rule to obtain an elimination rule of the form

$$\cfrac{\begin{matrix}[\varphi] & [\neg\varphi] \\ \vdots & \vdots \\ \psi & \psi\end{matrix}}{\psi}$$

Since we will only consider the restricted EM$_1$-rule, we can instead of φ and $\neg\varphi$ consider the formulas $\forall \alpha.\mathsf{P}(\alpha)$ and $\exists \alpha.\mathsf{P}^{\bot}(\alpha)$. Because of Lemma 5.2.8 we can restrict ourselves to formulas of the form $\forall \alpha.\mathsf{P}(\alpha)$ instead of $\forall \alpha.\varphi(\alpha)$ with quantifier free φ.

The informal computational intuition behind this proof rule is roughly the following: We start by assuming the truth of $\forall\alpha.\mathsf{P}(\alpha)$, and then each time we need the truth of an instance $\mathsf{P}(n)$ of the assumption, we check whether it is true or not; if it is true, then we continue, if it is not, we have found a witness for $\exists\alpha.\mathsf{P}^{\perp}(\alpha)$ which we can then fill in in the right-hand-side of the proof. The crucial observation is then that we will only ever need a finite number of instances of $\forall\alpha.\mathsf{P}(\alpha)$ to prove φ.

Definition 5.3.1 (Variables in HA+EM_1)**.** We will operate with three different types of variables:

- Numerical variables, α, β, γ, to represent natural numbers.
- Proof term variables, x, y, z, that act like usual lambda calculus variables.
- Hypothesis variables, a, b, c, which act as addresses to refer to uses of EM_1 hypotheses.

Definition 5.3.2 (Formulas of $\mathsf{HA} + \mathsf{EM}_1$)**.** The atomic language and the formulas of $\mathsf{HA} + \mathsf{EM}_1$ are the same as for HA, see Definition 5.2.2.

The proof terms of $\mathsf{HA} + \mathsf{EM}_1$ are similar to those of HA, except we add terms to take care of EM_1 hypotheses.

Definition 5.3.3 (Proof terms of $\mathsf{HA} + \mathsf{EM}_1$)**.** The untyped proof terms are the following:

$$\begin{aligned} u, v, w ::= {} & x \mid uv \mid um \mid \lambda x\, u \mid \lambda\alpha\, u \mid \langle u, v\rangle \mid \pi_0 u \mid \pi_1 u \mid \iota_0 u \mid \iota_1 u \\ & \mid u[x.v, y.w] \mid (m, u) \mid u[(\alpha, x).v] \mid u \parallel_a v \mid \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)} \\ & \mid \mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)} \mid \mathtt{Rec}\ u\ v\ m \mid \mathtt{r}\ u_1 \ldots u_n \end{aligned}$$

where x, y range over proof term variables, a over hypothesis variables and m over $\mathcal{L}$-terms. In terms of the form $u \parallel_a v$ we assume that a only occurs free in u in subterms of the form $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}$, and in v in subterms of the form $\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}$. If $\mathtt{r}$ occurs as a subterm without any accompanying u's, we will instead write $\mathtt{True}$.

Definition 5.3.4 (Capture avoiding substitution, witness substitution)**.** We define the substitutions $u[\alpha := t]$ and $u[x := v]$ like in HA. We also define *witness substitution*: Let u be a term and n a numeral. Define $u[a := n]$ as the term obtained from replacing each subterm $\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}$ by $(n, \mathtt{True})$ if $\mathsf{P}^{\perp}(n) \equiv \mathtt{True}$ (i.e. if n is a valid witness for the existential statement), and by $(n, \mathtt{H}_a^{\forall\alpha.\mathsf{eq}(\alpha,\mathbf{0})}\ \mathbf{S0})$ otherwise.

Axioms:

$$\Gamma; \Delta, a : \forall\alpha.\mathsf{P}(\alpha) \vdash \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)} : \forall\alpha.\mathsf{P}(\alpha)$$
$$\Gamma; \Delta, a : \exists\alpha.\mathsf{P}^{\perp}(\alpha) \vdash \mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)} : \exists\alpha.\mathsf{P}^{\perp}(\alpha)$$

EM$_1$:

$$\frac{\Gamma; \Delta, a : \forall\alpha.\mathsf{P} \vdash u : \varphi \qquad \Gamma; \Delta, a : \exists\alpha.\mathsf{P}^{\perp} \vdash v : \varphi}{\Gamma; \Delta \vdash u \parallel_a v : \varphi}$$

Figure 5.2: Typing rules for EM_1

Definition 5.3.5 (Typing judgments of HA+EM_1)**.** *Environments* in HA+EM_1 are bipartite: They consist of a set Γ similar to the environments from HA consisting of λ-variables and types, and then a set Δ with pairs of hypothesis variables and formulas. We write $\Gamma; \Delta$ where $\Gamma = x_1 : \varphi_1, \ldots, x_n : \varphi_n$ and $\Delta = a_1 : \psi_1, \ldots, a_m : \psi_m$.

We write the typing judgment $\Gamma; \Delta \vdash u : \varphi$ where $\Gamma; \Delta$ is an environment, u a $\mathsf{HA} + \mathsf{EM}_1$-term and φ a formula, to mean that there is a derivation using the typing rules from Figure 5.1 (where the content of Δ is irrelevant) and from Figure 5.2.

Definition 5.3.6 (Free variables)**.** We define $\mathrm{FV}(u)$ to be the set of free λ-variables in u, $\mathrm{FNV}(u)$ to be the set of free numeric variables and $\mathrm{FCV}(u)$ to be the set of free hypothesis-variables in u, where a hypothesis variable a is said to be free if there is a subterm of the form $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}$ or $\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}$ not in the scope of $\parallel_a$.

The free numeric variables of $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}$ and $\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}$ are the free numeric variables of P minus α.

Reduction for $\mathsf{HA}+\mathsf{EM}_1$

Definition 5.3.7 (Reduction rules for $\mathsf{HA}+\mathsf{EM}_1$)**.** We define the reduction relation $\to_{\mathsf{HA}+\mathsf{EM}_1}$ as the compatible closure of the following reduction rules:

$$\begin{array}{rll}
(\lambda x.u)t & \to_{\beta_1} & u[x := t] \\
(\lambda \alpha.u)t & \to_{\beta_2} & u[\alpha := t] \\
\pi_0 \langle u_0, u_1\rangle & \to_{\pi_0} & u_0 \\
\pi_1 \langle u_0, u_1\rangle & \to_{\pi_1} & u_1 \\
(\iota_0 u)[x_0.v_0, x_1.v_1] & \to_{\iota_0} & v_0[x_0 := u] \\
(\iota_1 u)[x_0.v_0, x_1.v_1] & \to_{\iota_1} & v_1[x_1 := u] \\
(n, u)[(\alpha, x).v] & \to_{\exists} & v[\alpha := n][x := u], \text{ for each numeral } n \\
\mathtt{Rec}\ u\ v\ \mathbf{0} & \to_{\mathtt{Rec}_1} & u \\
\mathtt{Rec}\ u\ v\ (\mathsf{S}n) & \to_{\mathtt{Rec}_2} & v\, n\, (\mathtt{Rec}\ u\ v\ n) \\
(u \parallel_a v)w & \to_{\mathrm{perm}_1} & uw \parallel_a vw \\
\pi_i(u \parallel_a v) & \to_{\mathrm{perm}_2} & \pi_i u \parallel_a \pi_i v \\
(u \parallel_a v)[x.w_1, y.w_2] & \to_{\mathrm{perm}_3} & u[x.w_1, y.w_2] \parallel_a v[x.w_1, y.w_2] \\
(u \parallel_a v)[(\alpha, x).w] & \to_{\mathrm{perm}_4} & u[(\alpha, x).w] \parallel_a v[(\alpha, x).w] \\
\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n & \to_{\mathsf{EM}_{11}} & \mathtt{r},\ \text{if } \mathsf{P}(n) = \mathtt{True} \\
u \parallel_a v & \to_{\mathsf{EM}_{12}} & u,\ \text{if } a \text{ does not occur free in } u \\
u \parallel_a v & \to_{\mathsf{EM}_{13}} & v,\ \text{if } a \text{ does not occur free in } v \\
u \parallel_a v & \to_{\mathsf{EM}_{14}} & v[a := n],\ \text{if } \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n \text{ occurs in } u \\
 & & \text{and } \mathsf{P}(n) = \mathtt{False}
\end{array}$$

Definition 5.3.8 (Normal forms)**.** Define NF to be the set of all proof terms in normal form, and SN to be the set of strongly normalizing proof terms. We say that a term is in *Post normal form* if it is recursively built up with $\mathtt{r}$ and $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n$, where n is a numeral, as follows:

$$p ::= \mathtt{r}\ p \cdots p \mid \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n.$$

A term in Post normal form represents a derivation that only consists of Post rules and instances of a universal hypothesis. We use PNF to refer to the set of terms in Post normal form.

Example 5.3.9. We will see how we can perform *proofs by contradiction* in this system. Classically, we are used to be able to reason like this:

$$\frac{\Gamma, \forall\alpha\,\mathsf{P}^{\perp} \vdash \perp}{\Gamma \vdash \exists\alpha\,\mathsf{P}}$$

In the current system, we can do this with the following derivation:

$$\dfrac{\dfrac{\Gamma, a : \forall\alpha\,\mathsf{P}^{\perp} \vdash \mathtt{efq}_{\exists\alpha\,\mathsf{P}} : \perp \to \exists\alpha\,\mathsf{P} \qquad \Gamma, a : \forall\alpha\,\mathsf{P}^{\perp} \vdash u : \perp}{\Gamma, a : \forall\alpha\,\mathsf{P}^{\perp} \vdash \mathtt{efq}_{\exists\alpha\,\mathsf{P}}\, u : \exists\alpha\,\mathsf{P}} \qquad \Gamma, a : \exists\alpha\,\mathsf{P} \vdash \mathtt{W}_a^{\exists\alpha\,\mathsf{P}} : \exists\alpha\,\mathsf{P}}{\Gamma \vdash \mathtt{efq}_{\exists\alpha\,\mathsf{P}}\, u \parallel_a \mathtt{W}_a^{\exists\alpha\,\mathsf{P}} : \exists\alpha\,\mathsf{P}}$$

Axioms:

$$\Gamma;\Delta, a:\forall\alpha.\mathsf{P}(\alpha)\vdash \mathtt{H}^{\forall\alpha.\mathsf{P}(\alpha)}:\forall\alpha.\mathsf{P}(\alpha)$$
$$\Gamma;\Delta, a:\exists\alpha.\mathsf{P}^{\perp}(\alpha)\vdash \mathtt{W}^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}:\exists\alpha.\mathsf{P}^{\perp}(\alpha)$$

$\mathbf{EM}_1^*$:

$$\frac{\Gamma;\Delta, a:\forall\alpha\mathsf{P}(\alpha)\vdash u:\varphi \qquad \Gamma;\Delta, a:\exists\alpha\mathsf{P}^{\perp}(\alpha)\vdash v:\varphi}{\Gamma;\Delta\vdash u\parallel v:\varphi}$$

Figure 5.3: Typing rules for EM_1^*

5.4 The system $\mathsf{HA}+\mathsf{EM}_1^*$

In order to show strong normalization of $\mathsf{HA}+\mathsf{EM}_1$, we will introduce the system $\mathsf{HA}+\mathsf{EM}_1^*$ of [3] which is a very slight alteration of $\mathsf{HA}+\mathsf{EM}_1$, and the strong normalization of $\mathsf{HA}+\mathsf{EM}_1^*$ will imply the strong normalization of $\mathsf{HA}+\mathsf{EM}_1$. The only difference in the terms are that we discard the hypothesis variables in the terms that has to do with EM_1.

The method we use to show strong normalization of $\mathsf{HA}+\mathsf{EM}_1^*$ is Aschieri's [3] adaption of the strong normalization proofs of $\lambda_{\rightarrow}$ and System **F** in [17], that uses the idea of an abstract notion called *reducibility*, originally due to Tait [39].

Definition 5.4.1 (Proof terms of $\mathsf{HA}+\mathsf{EM}_1^*$)**.** The terms of $\mathsf{HA}+\mathsf{EM}_1^*$ are given by the following grammar

$$\begin{aligned} t,u,v ::= {} & x \mid tu \mid tm \mid \lambda x\, u \mid \lambda\alpha\, u \mid \langle t,u\rangle \mid \pi_0 u \mid \pi_1 u \mid \iota_0(u) \mid \iota_1(u) \\ & \mid t[x.u, y.v] \mid (m,t) \mid t[(\alpha,x).u] \mid u\parallel v \mid \mathtt{H}^{\forall\alpha.\mathsf{P}(\alpha)} \\ & \mid \mathtt{W}^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)} \mid \mathtt{Rec}\, uvm \mid \mathtt{r}\, t_1\ldots t_n \end{aligned}$$

where x,y range over proof term variables and m over $\mathcal{L}$-terms.

Definition 5.4.2 (Typing judgments of $\mathsf{HA}+\mathsf{EM}_1^*$)**.** *Environments* in $\mathsf{HA}+\mathsf{EM}_1^*$ are like in $\mathsf{HA}+\mathsf{EM}_1$. We write the typing judgment $\Gamma;\Delta\vdash u:\varphi$ where $\Gamma;\Delta$ is an environment, u a $\mathsf{HA}+\mathsf{EM}_1^*$-term and φ a formula, to mean that there is a derivation using the typing rules from Figure 5.1 and from Figure 5.3.

Definition 5.4.3 (Reduction rules for $\mathsf{HA}+\mathsf{EM}_1^*$)**.** The reduction rules for $\mathsf{HA}+\mathsf{EM}_1^*$ are almost the same as for $\mathsf{HA}+\mathsf{EM}_1$. We will only change the EM_1-reduction rules:

$$\begin{array}{rcl} \mathtt{H}^{\forall\alpha.\mathsf{P}(\alpha)}\ n & \rightsquigarrow_{\mathsf{EM}_{11}^*} & \mathtt{True},\ \text{if } \mathsf{P}(n)\equiv\mathtt{True} \\ u\parallel v & \rightsquigarrow_{\mathsf{EM}_{12}^*} & u \\ u\parallel v & \rightsquigarrow_{\mathsf{EM}_{13}^*} & v \\ \mathtt{W}^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)} & \rightsquigarrow_{\mathsf{EM}_{14}^*} & (n,\mathtt{True}),\ \text{for every numeral } n \end{array}$$

Let $\rightsquigarrow$ be the compatible closure of the HA reduction rules, the permutation rules, and the above four rules. We use $\rightsquigarrow^+$ to refer to the transitive closure and $\rightsquigarrow^*$ to refer to the reflexive-transitive closure.

Since $\rightsquigarrow_{\mathsf{EM}_{14}^*}$ spans every natural number, the reduction trees in $\mathsf{HA}+\mathsf{EM}_1^*$ will not necessarily be finite (because there will be ω choices for each EM_{14}^*-reduction), but they will still be *well-founded*, in the sense that they will have no infinite branches. To terms in SN we will assign an ordinal number to each node in the tree, a *height* $h(t)$, such that if $t \rightsquigarrow t'$, then $h(t) > h(t')$ [22, Theorem 2.27].

We will define a translation from $\mathsf{HA}+\mathsf{EM}_1$ terms into $\mathsf{HA}+\mathsf{EM}_1^*$ terms in the obvious way.

Definition 5.4.4 (Translation of $\mathsf{HA}+\mathsf{EM}_1$-terms into $\mathsf{HA}+\mathsf{EM}_1^*$-terms)**.** We define the translation $\cdot^*$ mapping proof terms of $\mathsf{HA}+\mathsf{EM}_1$ into proof terms of $\mathsf{HA}+\mathsf{EM}_1^*$.

$$\begin{array}{rcl}
x^* & \mapsto & x \\
(tu)^* & \mapsto & t^*u^* \\
(tm)^* & \mapsto & t^*m \\
(\lambda x.u)^* & \mapsto & \lambda x.u^* \\
(\lambda \alpha.u)^* & \mapsto & \lambda \alpha.u^* \\
\langle u,v\rangle^* & \mapsto & \langle u^*,v^*\rangle \\
(\pi_i u)^* & \mapsto & \pi_i u^* \\
(\iota_i u)^* & \mapsto & \iota_i u^* \\
(t[x.u,y.v])^* & \mapsto & t^*[x.u^*,y.v^*] \\
(\mathtt{Rec}\ u\ v\ m)^* & \mapsto & \mathtt{Rec}\ u^*\ v^*\ m \\
(\mathtt{r}\ t_1\cdots t_n)^* & \mapsto & \mathtt{r}\ t_1^*\cdots t_n^* \\
(u \parallel_a v)^* & \mapsto & u^* \parallel v^* \\
(\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)})^* & \mapsto & \mathtt{H}^{\forall\alpha.\mathsf{P}(a)} \\
(\mathtt{W}_a^{\exists\alpha.\mathsf{P}^\perp(\alpha)})^* & \mapsto & \mathtt{W}^{\exists\alpha.\mathsf{P}^\perp(\alpha)}
\end{array}$$

Basically, by applying $\cdot^*$ to a term, you erase all hypothesis variables from the term.

The following lemma is crucial, since it will allow us to transfer a termination result about $\mathsf{HA}+\mathsf{EM}_1^*$ to one about $\mathsf{HA}+\mathsf{EM}_1$.

Lemma 5.4.5 (Preservation of $\rightarrow$ by $\rightsquigarrow$)**.** *Let v be a $\mathsf{HA}+\mathsf{EM}_1$-term such that $v \rightarrow w$. Then $v^* \rightsquigarrow^+ w^*$.*

Proof. In order to show that this holds for all such v it is sufficient to show that it holds for all redexes. All of them, except EM_{14} are straight-forward, so we only show a few of them.

- $(\lambda x.u)t \rightarrow_{\beta_1} u[x := t]$. We see that

$$((\lambda x.u)t)^* = (\lambda x.u^*)t^* \rightsquigarrow u^*[x := t^*] = (u[x := t])^*.$$

- $(u \parallel_a v)[x.w_1, y.w_2] \to_{\text{perm}_3} u[x.w_1, y.w_2] \parallel_a v[x.w_1, y.w_2]$. We see that

$$\begin{aligned}&((u \parallel_a v)[x.w_1, y.w_2])^* = (u^* \parallel v^*)[x.w_1^*, y.w_2^*]\\ \rightsquigarrow &u^*[x.w_1^*, y.w_2^*] \parallel v^*[x.w_1^*, y.w_2^*] = (u[x.w_1, y.w_2] \parallel_a v[x.w_1, y.w_2])^*\end{aligned}$$

- $u \parallel_a v \to_{\mathsf{EM}_{14}} v[a := n]$. First we see that

$$(u \parallel_a v)^* = u^* \parallel v^* \rightsquigarrow v^*.$$

But this v^* may still contain subterms of the form $\mathtt{W}^{\exists\alpha.\mathsf{P}^\perp(\alpha)}$. For each of these subterms we apply $\rightsquigarrow_{\mathsf{EM}_{14}}$, replacing each $\mathtt{W}^{\exists\alpha.\mathsf{P}^\perp(\alpha)}$ with $(n, True)$:

$$v^* \rightsquigarrow^* (v[a := n])^*.$$

□

5.5 Strong normalization for $\mathsf{HA}+\mathsf{EM}_1^*$ and $\mathsf{HA}+\mathsf{EM}_1$

We aim to prove strong normalization for $\mathsf{HA}+\mathsf{EM}_1^*$. For this, we define the following abstract reducibility relation.

Definition 5.5.1 (Reducibility)**.** We define a relation red between $\mathsf{HA}+\mathsf{EM}_1^*$-terms and formulas of $\mathcal{L}$. We read $t \mathrel{\mathsf{red}} \varphi$ as *t is reducible of type φ*.

1. $t \mathrel{\mathsf{red}} \mathsf{P}$ if and only if $t \in \mathsf{SN}$;
2. $t \mathrel{\mathsf{red}} \varphi \wedge \psi$ if and only if $\pi_0 t \mathrel{\mathsf{red}} \varphi$ and $\pi_1 t \mathrel{\mathsf{red}} \psi$;
3. $t \mathrel{\mathsf{red}} \varphi \to \psi$ if and only if $u \mathrel{\mathsf{red}} \varphi$ implies $tu \mathrel{\mathsf{red}} \psi$ for all u;
4. $t \mathrel{\mathsf{red}} \varphi \vee \psi$ if and only if
   - $t \in \mathsf{SN}$,
   - if $t \rightsquigarrow^* \iota_0 u$, then $u \mathrel{\mathsf{red}} \varphi$,
   - if $t \rightsquigarrow^* \iota_1 u$, then $u \mathrel{\mathsf{red}} \psi$;
5. $t \mathrel{\mathsf{red}} \forall\alpha.\varphi(\alpha)$ if and only if $tn \mathrel{\mathsf{red}} \varphi(n)$ for all terms n of $\mathcal{L}$;
6. $t \mathrel{\mathsf{red}} \exists\alpha.\varphi(\alpha)$ if and only if
   - $t \in \mathsf{SN}$,
   - for every term n of $\mathcal{L}$, if $t \rightsquigarrow^* (n, u)$, then $u \mathrel{\mathsf{red}} \varphi(n)$.

Definition 5.5.2 (Neutrality)**.** A proof term is said to be *neutral* if it is not of one of the following forms:

$$\lambda\alpha.u, \quad \lambda x.u, \quad \langle u, v\rangle, \quad \iota_i u, \quad (t, u), \quad \mathtt{H}^{\forall\alpha.\mathsf{P}(\alpha)}, \quad u \parallel v.$$

Lemma 5.5.3 (Reducibility candidates)**.** *Let t be a $\mathsf{HA} + \mathsf{EM}_1^*$-term. Then it has the following four properties:*

(CR1) If t red φ, then $t \in \mathsf{SN}$;

(CR2) If t red φ and $t \rightsquigarrow^* t'$, then t' red φ;

(CR3) If t is neutral and if $t \rightsquigarrow t'$ implies t' red φ for every t', then t red φ;

(CR4) $u \parallel v$ red φ if and only if u red φ and v red φ.

Proof. We proceed by induction on the complexity of φ.

- $\varphi = \mathsf{P}$. If t red P, then $t \in \mathsf{SN}$.

(CR1) $t \in \mathsf{SN}$ since t red P.

(CR2) Again, $t \in \mathsf{SN}$, so if $t \rightsquigarrow^* t'$ then also $t' \in \mathsf{SN}$ and thus t' red P.

(CR3) If t' red P and $t \rightsquigarrow t'$, then also $t \in \mathsf{SN}$ and thus t red P.

(CR4) $u \parallel v \in \mathsf{SN}$ if and only if $u \in \mathsf{SN}$ and $v \in \mathsf{SN}$.

- $\varphi = \psi \to \theta$.

(CR1) Suppose that t red $\psi \to \theta$. Firstly, notice that CR3 implies that any neutral term in normal form will be a reducibility candidate for anything. Thus, by induction hypothesis for CR3, we have that x red ψ for any variable x, so tx red θ, and by induction hypothesis for CR1, $tx \in \mathsf{SN}$. Therefore $tx \in \mathsf{SN}$.

(CR2) Suppose that t red $\psi \to \theta$, $t \rightsquigarrow^* t'$ and u red ψ. We need to show that $t'u$ red θ. We know that tu red θ, so since $tu \rightsquigarrow^* t'u$, the induction hypothesis of CR2 gives us that $t'u$ red θ.

(CR3) Suppose that t is neutral, and that $t \rightsquigarrow t'$ implies t' red $\psi \to \theta$. We need to show that tu red θ for any u such that u red ψ. Suppose that such a u is given. By the induction hypothesis for CR1 we know that $u \in \mathsf{SN}$. By induction on the height $h(u)$ we will show that tu red θ.

By the induction hypothesis for CR3 it is sufficient to show that $tu \rightsquigarrow v$ implies v red θ. Since t is neutral, v can either be $t'u$ where $t \rightsquigarrow t'$, or tu' where $u \rightsquigarrow u'$. In the first case, then t' red $\psi \to \theta$ by hypothesis, so $t'u$ red θ. In the second case, u' red ψ by the induction hypothesis for CR2, and since $h(u') < h(u)$, we have that tu' red θ by the induction hypothesis for the height.

(CR4) ($\Rightarrow$) Since we have proven CR2 for $\psi \to \theta$ we can use it now: We have that $u \parallel v \rightsquigarrow u$ and $u \parallel v \rightsquigarrow v$, so u red $\psi \to \theta$ and v red $\psi \to \theta$.

($\Leftarrow$) We suppose that u red $\psi \to \theta$ and v red $\psi \to \theta$. Let w be given such that w red ψ. By CR1 $u, v, w \in \mathsf{SN}$, so we can proceed by induction on the heights $h(u), h(v), h(w)$ to show that $(u \parallel v)w$ red θ. Again, we use the induction hypothesis for CR3, so it is sufficient to show that $(u \parallel v)w \rightsquigarrow r$ implies that r red θ. There are the following possibilities for r:

1. r is uw or vw;
2. r is $(u' \parallel v)w$, $(u \parallel v')w$ or $(u \parallel v)w'$, where $u \rightsquigarrow u'$, $v \rightsquigarrow v'$ and $w \rightsquigarrow w'$;
3. r is $uw \parallel vw$.

In the first case uw, vw red θ, so we are done. For the second case, we look at $(u' \parallel v)w$, the others are analogous. By CR2 we have that u' red $\psi \to \theta$, and since $h(u') < h(u)$ we get by induction hypothesis that $(u' \parallel v)w$ red θ. In the last case, we use the induction hypothesis for CR4.

- $\varphi = \forall\alpha.\psi(\alpha)$ or $\varphi = \psi \wedge \theta$. These cases are analogous to $\varphi = \psi \to \theta$.
- $\varphi = \exists\alpha.\psi(\alpha)$.

(CR1) If t red $\exists\alpha.\psi(\alpha)$, then $t \in \mathsf{SN}$.

(CR2) Suppose t red $\exists\alpha.\psi(\alpha)$ and $t \rightsquigarrow^* t'$. Then $t \in \mathsf{SN}$, so also $t' \in \mathsf{SN}$. Let n be a numeric term. If $t' \rightsquigarrow^* (n, u)$, then also $t \rightsquigarrow^* (n, u)$ and therefore u red $\psi(n)$.

(CR3) Suppose t is neutral and that $t \rightsquigarrow t'$ implies t' red $\exists\alpha.\psi(\alpha)$. We have $t \in \mathsf{SN}$, because if $t \rightsquigarrow t'$ then $t' \in \mathsf{SN}$. Let n be a numeric term and suppose that $t \rightsquigarrow^* (n, u)$. Since t is neutral, and thus different from (n, u), there must be at least one step in the reduction: $t \rightsquigarrow t' \rightsquigarrow^* (n, u)$, and so u red $\psi(n)$.

(CR4) ($\Rightarrow$) From $u \parallel v \rightsquigarrow u$, $u \parallel v \rightsquigarrow v$ and CR2 we get that u red $\exists\alpha.\psi(\alpha)$ and v red $\exists\alpha.\psi(\alpha)$.

($\Leftarrow$) Suppose u red $\exists\alpha.\psi(\alpha)$ and v red $\exists\alpha.\psi(\alpha)$. Then $u, v \in \mathsf{SN}$ and therefore $u \parallel v \in \mathsf{SN}$. If $u \parallel v \rightsquigarrow^* (n, w)$, then we must have at least one of $u \parallel v \rightsquigarrow u \rightsquigarrow^* (n, w)$ or $u \parallel v \rightsquigarrow v \rightsquigarrow^* (n, w)$. In either case, we have w red $\psi(n)$.

- $\varphi = \psi \vee \theta$. This case is analogous to the case with $\varphi = \exists\alpha.\psi(\alpha)$.

□

The following facts are useful for the proof of the main *Adequacy Theorem.*

Lemma 5.5.4. *1. Suppose that t red $\psi_0 \vee \psi_1$, and that u_0, u_1 are terms such that, for every v such that v red ψ_1, it holds that $u_i[x := v]$ red φ. Then $t[x.u_0, x.u_1]$ red φ.*

2. *Suppose that* $u[x := v]$ red ψ *for every* v *such that* v red φ*. Then* $\lambda x.u$ red $\varphi \to \psi$*.*

3. *If, for every numeric term* n*, it holds that* $u[\alpha := n]$ red $\varphi(n)$*, then* $\lambda\alpha.u$ red $\forall\alpha.\varphi(\alpha)$*.*

Proof. For the proofs of these facts, we refer to [3] and [17]. □

The following *Adequacy Theorem* tells us—roughly— that we can pass from $\vdash$ to red.

Theorem 5.5.5 (Adequacy Theorem)**.** *Let* $\varphi(\alpha_1, \ldots, \alpha_k)$ *be a formula, let* u *be a* $\mathsf{HA} + \mathsf{EM}_1^*$*-term, and let*

$$\Gamma = x_1 : \varphi_1(\alpha_1, \ldots, \alpha_k), \ldots, x_n : \varphi_n(\alpha_1, \ldots, \alpha_k)$$

such that no formula in Γ *nor* φ *has more free variables than* $\alpha_1, \ldots, \alpha_k$*. Let also* Δ *be given, and assume that* $\Gamma; \Delta \vdash u : \varphi$*. Now suppose that there for all numeric terms* $m_1, \ldots, m_k$ *are terms* $t_1, \ldots, t_n$ *such that*

$$t_i \ \mathsf{red}\ \varphi_i(m_1, \ldots, m_k) \quad \textit{for } i = 1, \ldots, n.$$

Then

$$u[x_1 := t_1, \ldots, x_n := t_n][\alpha_1 := m_1, \ldots, \alpha_k := m_k] \ \mathsf{red}\ \varphi(m_1, \ldots, m_k).$$

Proof. For the sake of readability, we will introduce the following notation:

$$\begin{aligned}\bar{t} &:= t[x_1 := t_1, \ldots, x_n := t_n][\alpha_1 := m_1, \ldots, \alpha_k := m_k]\\ \bar{\theta} &:= \theta(m_1, \ldots, m_k)\end{aligned}$$

The proof proceeds by induction on u. We look at the last applied rule in the derivation of $\Gamma; \Delta \vdash u : \varphi$.

Axioms:

- Last rule is $\Gamma; \Delta \vdash x_i : \varphi_i$. Then $\bar{x}_i = t_i$ and $\bar{\varphi}_i = \varphi_i(m_1, \ldots, m_k)$, and so $\bar{x}_i$ red $\bar{\varphi}_i$ by hypothesis.

- Last rule is $\Gamma; \Delta \vdash \mathtt{H}^{\forall\alpha.\mathsf{P}(\alpha)} : \forall\alpha.\mathsf{P}(\alpha)$. Then $\bar{u} = \mathtt{H}^{\forall\alpha.\bar{\mathsf{P}}(\alpha)}$ and $\bar{\varphi} = \forall\alpha.\bar{\mathsf{P}}(\alpha)$. For any numeric term n, $\bar{u}n \in \mathsf{SN}$, so $\bar{u}n$ red $\bar{\mathsf{P}}(n)$. Thus $\bar{u}$ red $\bar{\varphi}$.

- Last rule is $\Gamma; \Delta \vdash \mathtt{W}^{\exists\alpha.\mathsf{P}^\perp(\alpha)} : \exists\alpha.\mathsf{P}^\perp(\alpha)$. Then $\bar{u} = \mathtt{W}^{\exists\alpha.\bar{\mathsf{P}}^\perp(\alpha)}$ and $\bar{\varphi} = \exists\alpha.\bar{\mathsf{P}}^\perp(\alpha)$. We have $\bar{u} \in \mathsf{SN}$, and for every numeral n we have $\bar{u} \rightsquigarrow (n, \mathtt{True})$, and we also have $\mathtt{True}$ red $\bar{\mathsf{P}}^\perp(n)$. Thus, we get $\bar{u}$ red $\bar{\varphi}$.

Conjunction:

- If the last rule is a conjunction introduction rule, then $u = \langle v, w\rangle$ and $\varphi = \psi \wedge \theta$, with $\Gamma;\Delta \vdash v : \psi$ and $\Gamma;\Delta \vdash w : \theta$; thus $\bar{u} = \langle \bar{v}, \bar{w}\rangle$ and $\bar{\varphi} = \bar{\psi} \wedge \bar{\theta}$. By the induction hypothesis, $\bar{v}\ \mathsf{red}\ \bar{\psi}$ and $\bar{w}\ \mathsf{red}\ \bar{\theta}$, and we have to show that $\pi_0\langle \bar{v}, \bar{w}\rangle\ \mathsf{red}\ \bar{\psi}$ and $\pi_1\langle \bar{v}, \bar{w}\rangle\ \mathsf{red}\ \bar{\theta}$. Notice that $\pi_0\langle \bar{v}, \bar{w}\rangle$ is a neutral term, and that there are the following possible reductions:

  $$\pi_0\langle \bar{v}, \bar{w}\rangle \rightsquigarrow \bar{v}, \quad \pi_0\langle \bar{v}, \bar{w}\rangle \rightsquigarrow \pi_0\langle \bar{v}', \bar{w}\rangle, \quad \pi_0\langle \bar{v}, \bar{w}\rangle \rightsquigarrow \pi_0\langle \bar{v}, \bar{w}'\rangle,$$

  where $\bar{v} \rightsquigarrow \bar{v}'$ and $\bar{w} \rightsquigarrow \bar{w}'$. Using that the reduction trees are always well-founded, we can argue by double induction on the heights $h(\bar{v}), h(\bar{w})$ that if $\pi_0\langle \bar{v}, \bar{w}\rangle \rightsquigarrow t$, then $t\ \mathsf{red}\ \bar{\psi}$. Therefore, using CR3, we get $\pi_0\langle \bar{v}, \bar{w}\rangle\ \mathsf{red}\ \bar{\psi}$. Similarly for $\pi_1\langle \bar{v}, \bar{w}\rangle\ \mathsf{red}\ \bar{\theta}$. Hence, $\langle \bar{v}, \bar{w}\rangle\ \mathsf{red}\ \bar{\psi} \wedge \bar{\theta}$.

- If the last rule is a conjunction elimination rule, like so:

  $$\frac{\Gamma;\Delta \vdash v : \varphi \wedge \psi}{\Gamma;\Delta \vdash \pi_0 v : \varphi}$$

  Then by induction hypothesis, $\bar{v}\ \mathsf{red}\ \bar{\varphi} \wedge \bar{\psi}$, and therefore, by definition, $\pi_0\bar{v}\ \mathsf{red}\ \bar{\varphi}$, as wanted. Similarly for the π_1-case.

Implication:

- The last rule is implication introduction:

  $$\frac{\Gamma, x : \psi; \Delta \vdash v : \theta}{\Gamma;\Delta \vdash \lambda x.v : \psi \to \theta}$$

  For showing $\lambda x.\bar{v}\ \mathsf{red}\ \bar{\psi} \to \bar{\theta}$, it is by Lemma 5.5.4 sufficient to show that $\bar{v}[x := w]\ \mathsf{red}\ \bar{\theta}$ for every w such that $w\ \mathsf{red}\ \bar{\psi}$. Let such a w be given. We have

  $$t_i\ \mathsf{red}\ \bar{\varphi}_i \text{ for } i = 1, \dots, k, \quad \text{and} \quad w\ \mathsf{red}\ \bar{\psi},$$

  so by the induction hypothesis we get that $\bar{v}[x := w]\ \mathsf{red}\ \bar{\theta}$.

- The last rule is implication elimination:

  $$\frac{\Gamma;\Delta \vdash v : \psi \to \varphi \qquad \Gamma;\Delta \vdash w : \psi}{\Gamma;\Delta \vdash vw : \varphi}$$

  By the induction hypothesis, we have $\bar{v}\ \mathsf{red}\ \bar{\psi} \to \bar{\varphi}$ and $\bar{w}\ \mathsf{red}\ \bar{\psi}$, so by definition of red we also have $\bar{v}\bar{w}\ \mathsf{red}\ \bar{\varphi}$.

Disjunction:

- If the last rule is a disjunction introduction rule, say, without loss of generality, the left rule

$$\frac{\Gamma; \Delta \vdash v : \psi}{\Gamma; \Delta \vdash \iota_0 v : \psi \vee \theta}$$

  then by induction hypothesis, $\bar{v}$ red $\bar{\psi}$, and therefore, by CR1, $\bar{v} \in \mathsf{SN}$. Since, trivially, $\iota_0 \bar{v} \rightsquigarrow^* \iota_0 \bar{v}$, we get $\iota_0 \bar{v}$ red $\bar{\psi} \vee \bar{\theta}$, by the definition of red.

- If the last rule is a disjunction elimination rule:

$$\frac{\Gamma; \Delta \vdash v : \psi \vee \theta \qquad \Gamma, x : \psi; \Delta \vdash w_1 : \varphi \qquad \Gamma, x : \theta; \Delta \vdash}{\Gamma; \Delta \vdash v[x.w_1, x.w_2] : \varphi}$$

  Then the induction hypothesis tells us the following: $\bar{v}$ red $\bar{\psi} \vee \bar{\theta}$; for every t such that t red $\bar{\psi}$ we have $\bar{w}_1[x := t]$ red φ; and for every t such that t red $\bar{\theta}$ we have $\bar{w}_2[x := t]$ red $\bar{\varphi}$. By Lemma 5.5.4, we have that $\bar{v}[x.\bar{w}_1, x.\bar{w}_2]$ red $\bar{\varphi}$.

Existential quantification: These cases are analogous to the disjunction cases.

Universal quantification:

- The last rule is universal introduction:

$$\frac{\Gamma; \Delta \vdash v : \psi(\alpha)}{\Gamma; \Delta \vdash \lambda\alpha.v : \forall\alpha.\psi(\alpha)}\ \alpha \notin \mathrm{FV}(\Gamma; \Delta)$$

  We need to show that $\lambda\alpha.\bar{v}$ red $\forall\alpha.\bar{\psi}(\alpha)$, and by Lemma 5.5.4 it is sufficient to show that $\bar{v}[\alpha := t]$ red $\bar{\psi}(t)$, for t a numeric term. But we can assume that $\alpha \neq \alpha_1, \ldots, \alpha_k$, so $\bar{\psi}_i(\alpha) = \bar{\psi}_i(t)$ since α is not free in ψ_i, and thus

$$t_i \ \mathsf{red}\ \bar{\varphi}_i(t), \quad \text{for } i = 1, \ldots, k.$$

  Therefore we can apply the induction hypothesis on v and get $\bar{v}[\alpha := t]$ red $\bar{\psi}(t)$.

- The last rule is universal elimination:

$$\frac{\Gamma; \Delta \vdash v\forall\alpha.\varphi(\alpha)}{\Gamma; \Delta \vdash vt : \varphi(t)}$$

  By the induction hypothesis, $\bar{v}$ red $\forall\alpha.\bar{\varphi}(\alpha)$, and thus, by definition of red, $\bar{v}\bar{t}$ red $\bar{\varphi}(t)$, as needed.

Induction: The last rule is the induction rule:

$$\frac{\Gamma;\Delta\vdash v:\psi(\mathbf{0})\qquad \Gamma;\Delta\vdash w:\forall\alpha.\psi(\alpha)\to\psi(\mathbf{S}\alpha)}{\Gamma;\Delta\vdash \mathtt{Rec}\ v\ w\ t:\psi(t)}$$

We first notice that $\bar{t}=n$ for some numeral n, so it will suffice to show $\mathtt{Rec}\ \bar{v}\ \bar{w}\ n\ \mathsf{red}\ \bar{\psi}(n)$ for all numerals n. $\mathtt{Rec}\ \bar{v}\ \bar{w}\ n$ is neutral, so by CR3 it is enough to show

$$\mathtt{Rec}\ \bar{v}\ \bar{w}\ n\rightsquigarrow v' \quad\text{implies}\quad v'\ \mathsf{red}\ \bar{\psi}(n).$$

We will do this by a triple induction on n and the heights $h(\bar{v})$ and $h(\bar{w})$. If $n=\mathbf{0}$ and $\mathtt{Rec}\ \bar{v}\ \bar{w}\ \mathbf{0}\rightsquigarrow\bar{v}$, then $\bar{v}\rightsquigarrow\bar{\psi}(\mathbf{0})$ by the main induction hypothesis. Suppose $n=\mathbf{S}m$ and

$$\mathtt{Rec}\ \bar{v}\ \bar{w}\ (\mathbf{S}m)\rightsquigarrow\bar{w}m(\mathtt{Rec}\ \bar{v}\ \bar{w}\ m).$$

By the main induction hypothesis we have $\bar{w}\ \mathsf{red}\ \forall\alpha.\psi(\alpha)\to\psi(\mathbf{S}\alpha)$, and by the induction hypotheses for n we have that $\mathtt{Rec}\ \bar{v}\ \bar{w}\ m\ \mathsf{red}\ \bar{\psi}(m)$, so therefore

$$\bar{w}m(\mathtt{Rec}\ \bar{v}\ \bar{w}\ m\ \mathsf{red}\ \bar{\psi}(m))\ \mathsf{red}\ \psi(\mathbf{S}m).$$

In the cases where

$$\mathtt{Rec}\ \bar{v}\ \bar{w}\ n\rightsquigarrow\mathtt{Rec}\ \bar{v}'\ \bar{w}\ n\quad\text{or}\quad\mathtt{Rec}\ \bar{v}\ \bar{w}\ n\rightsquigarrow\mathtt{Rec}\ \bar{v}\ \bar{w}'\ n,$$

where $\bar{v}\rightsquigarrow\bar{v}'$ and $\bar{w}\rightsquigarrow\bar{w}'$, we can apply the induction hypotheses for the heights.

Post rules: If the last rule is a Post rule, then $u=\mathtt{r}\ u_1\cdots u_l$ and $\Gamma;\Delta\vdash u:\mathsf{Q}$, and since $\bar{u}_1,\dots,\bar{u}_l\in\mathsf{SN}$ by the induction hypothesis, then also $\bar{u}\in\mathsf{SN}$, and so $\bar{u}\ \mathsf{red}\ \bar{\mathsf{Q}}$.

$\mathbf{EM}_1^*$: The last rule is

$$\frac{\Gamma;\Delta,a:\forall\alpha.\mathsf{P}(\alpha)\vdash v:\varphi\qquad\Gamma;\Delta,a:\exists\alpha.\mathsf{P}^{\perp}(\alpha)\vdash w:\varphi}{\Gamma;\Delta\vdash v\parallel w:\varphi}$$

By induction hypothesis, $\bar{v}\ \mathsf{red}\ \bar{\varphi}$ and $\bar{w}\ \mathsf{red}\ \bar{\varphi}$, so CR4 gives us that $\bar{v}\parallel\bar{w}\ \mathsf{red}\ \bar{\varphi}$. □

Corollary 5.5.6 (Strong normalization for $\mathsf{HA}+\mathsf{EM}_1^*$)**.** *If* $\Gamma;\Delta\vdash u:\varphi$ *in* $\mathsf{HA}+\mathsf{EM}_1^*$*, then* $u\in\mathsf{SN}$.

Proof. Suppose $\Gamma;\Delta\vdash u:\varphi$, with $\Gamma=x_1:\varphi_1,\dots,x_n:\varphi_n$. Since x_i are neutral terms in normal form, CR3 gives us that $x_i\ \mathsf{red}\ \varphi_i$. Hence, by Adequacy Theorem 5.5.5, $u\ \mathsf{red}\ \varphi$, and therefore, by CR1, $u\in\mathsf{SN}$. □

Corollary 5.5.7 (Strong normalization for $\mathsf{HA} + \mathsf{EM}_1$)**.** *If* $\Gamma; \Delta \vdash u : \varphi$ *in* $\mathsf{HA} + \mathsf{EM}_1$*, then* $u \in \mathsf{SN}$*.*

Proof. Suppose $\Gamma; \Delta \vdash u : \varphi$ in $\mathsf{HA} + \mathsf{EM}_1$. Then we also have $\Gamma; \Delta \vdash u^* : \varphi$ in $\mathsf{HA} + \mathsf{EM}_1^*$. Now, suppose for contradiction that we have an infinite reduction

$$u = u_1 \to u_2 \to u_3 \to \cdots .$$

By Lemma 5.4.5, this gives rise to an infinite reduction

$$u^* = u_1^* \rightsquigarrow^+ u_2^* \rightsquigarrow^+ u_3^* \rightsquigarrow \cdots ,$$

which, by Corollary 5.5.6, is impossible. Therefore, $u \in \mathsf{SN}$. □

5.6 Existential witness property

An important property of the system $\mathsf{HA} + \mathsf{EM}_1$ is the following:

Theorem 5.6.1 (Existential Witness Property)**.** *Suppose that*

$$\vdash u : \exists \alpha . \mathsf{P}(\alpha).$$

Then there is a term (n, u') *in normal form such that* $u \twoheadrightarrow (n, u')$*, and* $\mathsf{P}(n) \equiv$ `True`*.*

This is proved in [6] using Interactive Realizability.

Chapter 6

Programming with terms in $\mathsf{HA} + \mathsf{EM}_1$

Since we study the system $\mathsf{HA} + \mathsf{EM}_1$ for the purpose of examining the computational content of classical proofs, the most natural thing to use the system for is programming. In this chapter we will study some cases of different specifications for which we find terms, and then examine how these behave computationally.

The purposes with the following two examples are different: In the first example we start with a proof and then we turn this into a program and analyze this. In the second example it is the other way around. We start with an idea of how we want the program to behave, and then we seek out a proof that will accommodate this idea.

6.1 Searching

In this situation, we investigate a problem of searching. We imagine that some decidable unary predicate P and a number n is given, whereof we know that $\neg P(0)$ and $P(n)$ hold. The problem then consists of finding a number k between 0 and n such that $\neg P(k)$ and $P(k+1)$ holds. We will investigate the computational differences between a term based solely on intuitionistic reasoning (i.e. a term from HA), and a term that makes use of the EM_1-rule.

Intuitionistic proof

We first demonstrate how we would solve this problem without the use of any classical reasoning. Firstly, we fix the atomic formulas that we will use:

$$\begin{aligned} \mathsf{P}(\alpha) &: \text{ holds if } P(\alpha) \text{ is true;} \\ \mathsf{Q}(\alpha) &: \text{ holds if } \neg P(\alpha) \wedge P(\mathbf{S}\alpha). \end{aligned}$$

Then there are some Post rules we can make use of. Obviously, we have

$$\frac{\mathsf{P}^{\perp}(\alpha) \qquad \mathsf{P}(\mathbf{S}\alpha)}{\mathsf{Q}(\alpha)}$$

and since $\mathsf{P}(\alpha)$ is decidable, we also have

$$\frac{}{\mathsf{P}(\alpha) \vee \mathsf{P}^{\perp}(\alpha)}$$

Next, we formulate our premises:

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0})$$
$$h_2 : \mathsf{P}(n).$$

Proposition 6.1.1. *There is a proof term* `search_in` *in* HA *such that*

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}), h_2 : \mathsf{P}(n) \vdash \texttt{search\_in} : \exists\alpha.\mathsf{Q}(\alpha).$$

We will find the proof term in 3 steps: Firstly, we describe an informal proof, which we then turn into a formal proof, and lastly we annotate this proof with proof terms.

Informal proof The proof will be an induction on β, showing that $\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)$. The base case is then trivial, since we get a contradiction from $\mathsf{P}^{\perp}(\mathbf{0})$ and $\mathsf{P}(\mathbf{0})$ right away. In the induction step we assume $\mathsf{P}(\mathbf{S}\beta)$ and consider two possibilities: If $\mathsf{P}(\beta)$ holds, then $\exists\alpha.\mathsf{Q}(\alpha)$ follows from the IH, and if it does not then $\mathsf{Q}(\beta)$ holds.

Formal proof We want a proof of $\mathsf{P}(n) \to \exists\alpha.\mathsf{Q}(\alpha)$, and we will do this by induction. From Lemma 5.2.10 we know that we can do ex falso reasoning, so the base case is just

$$\frac{\dfrac{\dfrac{\dfrac{\mathsf{P}^{\perp}(\mathbf{0}) \qquad \mathsf{P}(\mathbf{0})}{\perp}}{\exists\alpha.\mathsf{Q}(\alpha)}}{\mathsf{P}(\mathbf{0}) \to \exists\alpha.\mathsf{Q}(\alpha)} \qquad \begin{matrix}\text{(induction step)}\\ \vdots \\ \forall\beta.((\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)) \to \mathsf{P}(\mathbf{S}\beta) \to \exists\alpha.\mathsf{Q}(\alpha))\end{matrix}}{\mathsf{P}(n) \to \exists\alpha.\mathsf{Q}(\alpha)}$$

In the induction step we will make use of the Post rules:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{P}(\beta) \vee \mathsf{P}^{\perp}(\beta)} \qquad \dfrac{\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)^{x} \qquad \mathsf{P}(\beta)^{z}}{\exists\alpha.\mathsf{Q}(\alpha)} \qquad \dfrac{\dfrac{\mathsf{P}^{\perp}(\beta)^{z} \qquad \mathsf{P}(\mathbf{S}\beta)^{y}}{\mathsf{Q}(\beta)}}{\exists\alpha.\mathsf{Q}(\alpha)}}{\exists\alpha.\mathsf{Q}(\alpha)}\,z}{\mathsf{P}(\mathbf{S}\beta) \to \exists\alpha.\mathsf{Q}(\alpha)}\,y}{(\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)) \to \mathsf{P}(\mathbf{S}\beta) \to \exists\alpha.\mathsf{Q}(\alpha)}\,x}{\forall\beta.((\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)) \to \mathsf{P}(\mathbf{S}\beta) \to \exists\alpha.\mathsf{Q}(\alpha))}$$

Proof term By simply annotating the above proof trees, we acquire the corresponding proof term. The induction step will have the proof term

$$\texttt{search\_in\_step} := \lambda\beta\lambda x\lambda y.\texttt{True}[z.(xz), z.(\beta, (\texttt{r}\ z\ y))]$$
$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}) \vdash \texttt{search\_in\_step} : \forall\beta.((\mathsf{P}(\beta) \to \exists\alpha.\mathsf{Q}(\alpha)) \to \mathsf{P}(\mathbf{S}\beta) \to \exists\alpha.\mathsf{Q}(\alpha))$$

as can easily be checked (remember that the annotation for a Post rule is `True` or $\texttt{r}\ u_1\ \cdots\ u_m$). Similarly we find the term for the base case:

$$\texttt{search\_in\_base} := \lambda x.\texttt{efq}_{\exists\alpha.\mathsf{Q}(\alpha)}\ (\texttt{r}\ h_1\ x)$$
$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}) \vdash \texttt{search\_in\_base} : \mathsf{P}(\mathbf{0}) \to \exists\alpha.\mathsf{Q}(\alpha)$$

We can now put the pieces together to find the sought-after term. The term $\texttt{Rec search\_in\_base search\_in\_step}\ n$ will have the type $\mathsf{P}(n) \to \exists\alpha.\mathsf{Q}(\alpha)$. Therefore the final term will be

$$\texttt{search\_in} := \texttt{Rec search\_in\_base search\_in\_step}\ n\ h_2$$

for then

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}), h_2 : \mathsf{P}(n) \vdash \texttt{search\_in} : \exists\alpha.\mathsf{Q}(\alpha).$$

Classical proof

Now we will try to find another solution to the problem, this time using classical reasoning, and thus ending up with a program that uses control operators. We still have the same primitive recursive predicate P, and we also reuse the atomic formulas:

$$\begin{aligned} \mathsf{P}(\alpha) &: \text{ holds if } P(\alpha) \text{ is true;} \\ \mathsf{Q}(\alpha) &: \text{ holds if } \neg P(\alpha) \wedge P(\mathbf{S}\alpha). \end{aligned}$$

Again, this gives rise to some Post rules, and this time we will make use of the following two:

$$\frac{\mathsf{P}^{\perp}(\alpha) \qquad \mathsf{P}(\mathbf{S}\alpha)}{\mathsf{Q}(\alpha)} \qquad \frac{\mathsf{P}^{\perp}(\alpha) \qquad \mathsf{Q}^{\perp}(\alpha)}{\mathsf{P}^{\perp}(\mathbf{S}\alpha)}$$

Proposition 6.1.2. *There is a proof term* `search_cl` *in* $\mathsf{HA} + \mathsf{EM}_1$ *but not in* HA *such that*

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}), h_2 : \mathsf{P}(n) \vdash \texttt{search\_cl} : \exists\alpha.\mathsf{Q}(\alpha).$$

Again, we will divide the process in three: Firstly, we describe the proof strategy informally, then we make a formal derivation, and finally we extract the proof term from this proof.

Informal proof The proof will take the shape of a contradiction argument. We want to show that there is an α such that $\neg P(\alpha) \wedge P(\mathbf{S}\alpha)$, so we assume the opposite: For all α, $\neg(\neg P(\alpha) \wedge P(\mathbf{S}\alpha))$. We use this to show $\forall\beta\neg P(\beta)$ by induction: $\neg P(0)$ is a premise, so assume $\neg P(\beta)$; if $P(\mathbf{S}\beta)$, then we have $\neg P(\beta) \wedge P(\mathbf{S}\beta)$ which is in contradiction with our first assumption, so therefore $\neg P(\mathbf{S}\beta)$ must be the case, so we have $\forall\beta\neg P(\beta)$. This gives us a contradiction with the premise that $P(n)$ must hold. Therefore we can conclude $\exists\alpha\neg P(\alpha) \wedge P(\mathbf{S}\alpha)$.

Formal proof Since the proof is by contradiction, it will have the following form

$$\dfrac{\dfrac{\begin{matrix}[\forall\alpha.\mathsf{Q}^{\perp}(\alpha)]\\ \vdots\\ \perp\end{matrix}}{\exists\alpha.\mathsf{Q}(\alpha)} \qquad \exists\alpha.\mathsf{Q}(\alpha)}{\exists\alpha.\mathsf{Q}(\alpha)}$$

where the last rule application is an instance of EM_1. In the missing part we fill in the following:

$$\dfrac{\mathsf{P}(n) \qquad \dfrac{\mathsf{P}^{\perp}(\mathbf{0}) \qquad \begin{matrix}\text{(induction step)}\\ \vdots\\ \forall\beta.\mathsf{P}^{\perp}(\beta)\rightarrow\mathsf{P}^{\perp}(\mathbf{S}\beta)\end{matrix}}{\mathsf{P}^{\perp}(n)}}{\perp}$$

In the induction step we make use of a Post rule:

$$\dfrac{\dfrac{\dfrac{\mathsf{P}^{\perp}(\beta) \qquad \dfrac{\forall\alpha.\mathsf{Q}^{\perp}(\alpha)}{\mathsf{Q}^{\perp}(\beta)}}{\mathsf{P}^{\perp}(\mathbf{S}\beta)}}{\mathsf{P}^{\perp}(\beta)\rightarrow\mathsf{P}^{\perp}(\mathbf{S}\beta)}}{\forall\beta.\mathsf{P}^{\perp}(\beta)\rightarrow\mathsf{P}^{\perp}(\mathbf{S}\beta)}$$

Proof term Since we are using the EM_1-rule, we will use the terms $\mathtt{H}_a^{\forall\alpha.\mathsf{Q}^{\perp}(\alpha)}$ and $\mathtt{W}_a^{\exists\alpha.\mathsf{Q}(\alpha)}$ to refer to the left and right hand side of the EM_1-disjunction. The induction step proof tree will correspond to the following term:

$$\mathtt{search\_cl\_step} := \lambda\beta\lambda x.\mathtt{r}\ x\ (\mathtt{H}_a^{\forall\alpha.\mathsf{Q}^{\perp}(\alpha)}\ \beta),$$

for then

$$a : \forall\alpha.\mathsf{Q}^{\perp}(\alpha) \vdash \mathtt{search\_cl\_step} : \forall\beta.\mathsf{P}^{\perp}(\beta) \rightarrow \mathsf{P}^{\perp}(\mathbf{S}\beta).$$

We can now use this to annotate the contradiction part of the derivation. Notice that, since we regard $\mathsf{P}^{\perp}$ as an atomic formula, we will need an $\mathtt{r}$ to get $\perp$ from $\mathsf{P}^{\perp}(n)$ and $\mathsf{P}(n)$ because we use a Post rule from the following scheme of rules, where S can be any atomic formula:

$$\frac{\mathsf{S} \qquad \mathsf{S}^{\perp}}{\perp}$$

We get

$$\mathtt{search\_cl\_contr} := \mathtt{r}\ h_2\ (\mathtt{Rec}\ h_1\ \mathtt{search\_cl\_step}\ n),$$

for then

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}), h_2 : \mathsf{P}(n), a : \forall\alpha.\mathsf{Q}^{\perp}(\alpha) \vdash \mathtt{search\_cl\_contr} : \perp$$

and then we reach

$$\mathtt{search\_cl} := \mathtt{efq}_{\exists\alpha.\mathsf{Q}(\alpha)}\ \mathtt{search\_cl\_contr} \parallel_a \mathtt{W}_a^{\exists\alpha.\mathsf{Q}(\alpha)}$$

with

$$h_1 : \mathsf{P}^{\perp}(\mathbf{0}), h_2 : \mathsf{P}(n) \vdash \mathtt{search\_cl} : \exists\alpha.\mathsf{Q}(\alpha)$$

as desired.

Reduction of search_cl

In order to visualize how this search term can operate, we will draw a reduction graph for a simple example. We will need to fix an n and some truth-values for P. The size of the graph grows very quickly when we increase n, so in order to keep it simple we will consider the trivial situation where $n = 1$, and $P(0), \neg P(1)$ holds. This means we have the atomic formulas $\mathsf{P}(0), \mathsf{P}(1)$, and $\mathsf{Q}(0)$ (where the numbers represent the terms $\mathbf{0}$ respectively $\mathbf{S0}$).

The term search_cl contains β_1-redexes because of the definition of efq, and since these in the end will be uninteresting for the computation, we start by getting rid of these, and instead consider the following term:

$$(\mathbf{0}, \mathtt{r}\ (\mathtt{r}\ h_2\ (\mathtt{Rec}\ h_1\ \mathtt{search\_cl\_step}\ 1))) \parallel_a \mathtt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)}.$$

All the redexes, except for the ones of the form $u \parallel_a v$, will occur inside the subterm Rec h_1 search_cl_step 1, and so we can restrict out attention to this. We will abbreviate some subterms: search_cl_step with step, and $\mathtt{H}_a^{\forall\alpha\mathsf{Q}^{\perp}(\alpha)}$

with H.

$$\begin{tikzcd}
 & \underline{\mathtt{Rec}\ h_1\ \mathtt{step}\ 1} \arrow[d] & \\
 & \underline{\mathtt{step}\ 0}\ \underline{(\mathtt{Rec}\ h_1\ \mathtt{step}\ 0)} \arrow[dl] \arrow[dr] & \\
\underline{(\lambda x.\mathtt{r}\ x\ (\utilde{\mathtt{H}\ 0}))(\underline{\mathtt{Rec}\ h_1\ \mathtt{step}\ 0})} \arrow[d] \arrow[drr] & & \underline{\mathtt{step}\ 0\ h_1} \arrow[d] \\
\mathtt{r}\ (\underline{\mathtt{Rec}\ h_1\ \mathtt{step}\ 0})\ (\utilde{\mathtt{H}\ 0}) \arrow[dr] & & \underline{(\lambda x.\mathtt{r}\ x\ (\utilde{\mathtt{H}\ 0}))\ h_1} \arrow[dl] \\
 & \mathtt{r}\ h_1\ (\utilde{\mathtt{H}\ 0}) &
\end{tikzcd}$$

The underlined subterms are redexes, and the ones that are underlined with a wavy line are subterms that makes a EM_{14}-reduction possible at the root level. This is where the *exceptions* can occur, where we can escape the reduction on the left-hand side, as for example:

$$(\mathbf{0}, \mathtt{r}\ (\mathtt{r}\ h_2\ ((\lambda x.\mathtt{r}\ x\ (\mathtt{H}\ 0))(\mathtt{Rec}\ h_1\ \mathtt{step}\ 0)))) \parallel_a \mathtt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)} \to (\mathbf{0}, \mathtt{True}).$$

We will now try and look at a slightly more complicated example. Let now $n=3$, and suppose that $\neg P(0), P(1), \neg P(2), P(3)$ holds, thus we have the atomic formulas: $\mathsf{Q}(0), \mathsf{Q}^{\perp}(1), \mathsf{Q}(2)$. If we prioritize the $\mathtt{Rec}_2$-reduction, then we can do the following reduction:

$$\begin{aligned}
\mathtt{Rec}\ h_1\ \mathtt{step}\ 3 &\twoheadrightarrow \mathtt{step}\ 2\ (\mathtt{step}\ 1\ (\mathtt{step}\ 0\ h_1)) \\
&\twoheadrightarrow \mathtt{r}\ (\mathtt{r}\ (\mathtt{r}\ h_1\ (\utilde{\mathtt{H}\ 0}))\ (\mathtt{H}\ 1))\ (\utilde{\mathtt{H}\ 2}) \\
&\twoheadrightarrow \mathtt{r}\ (\mathtt{r}\ (\mathtt{r}\ h_1\ (\utilde{\mathtt{H}\ 0}))\ \mathtt{True})\ (\utilde{\mathtt{H}\ 2}).
\end{aligned}$$

This reduction leaves us with the choice of two exceptional exits, namely

$$(\mathbf{0}, \mathtt{r}\ (\mathtt{r}\ h_2\ (\mathtt{r}\ (\mathtt{r}\ (\mathtt{r}\ h_1\ (\mathtt{H}\ 0))\ \mathtt{True})\ (\mathtt{H}\ 2)))) \parallel_a \mathtt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)} \to (0, \mathtt{True}),$$

and

$$(\mathbf{0}, \mathtt{r}\ (\mathtt{r}\ h_2\ (\mathtt{r}\ (\mathtt{r}\ (\mathtt{r}\ h_1\ (\mathtt{H}\ 0))\ \mathtt{True})\ (\mathtt{H}\ 2)))) \parallel_a \mathtt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)} \to (2, \mathtt{True}).$$

It is clear that we can expand this for any situation: If there are m valid candidates, then the search term has m different normal forms. Therefore, this term cannot be said to contain one search algorithm per se, since the outcome is entirely decided by the reduction strategy. But it does seem that there is a natural choice to make: The most efficient way of finding a normal form, will be to choose the first exceptional exit. In the example, instead of reducing to

$$\mathtt{r}\ (\mathtt{r}\ (\mathtt{r}\ h_1\ (\utilde{\mathtt{H}\ 0}))\ \mathtt{True})\ (\utilde{\mathtt{H}\ 2}),$$

one could instead just stop the reduction much earlier at

$$(\lambda x.\mathtt{r}\ x\ (\mathtt{H}\ 2))(\mathtt{Rec}\ h_1\ \mathtt{step}\ 2),$$

and then do the exceptional exit:

$$(\mathbf{0}, \mathtt{r}\ (\mathtt{r}\ h_2\ ((\lambda x.\mathtt{r}\ x\ (\mathtt{H}\ 2))(\mathtt{Rec}\ h_1\ \mathtt{step}\ 2)))) \parallel_a \mathtt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)} \to (2, \mathtt{True}).$$

Thus, if we can fix a reduction strategy that behaves in this “natural” way, then the search algorithm we get is *top-down search*.

6.2 Multiplication example

An example of a computer program that can be made more efficient with the use of exception operators is *list multiplication*. The following is a traditional implementation of a program that multiplies the elements in a list (here in Haskell notation):

```haskell
listmult :: [Integer] -> Integer
listmult [] = 1
listmult (x:xs) = x * (listmult xs)
```

However, since we know that the product of any list containing a zero will be zero, we would like the program to stop the calculation as soon as a zero is encountered. In the traditional implementation this does not happen. The naïve solution is to add the following pattern matching case to the code:

```haskell
listmult (0:xs) = 0
```

But this is not satisfactory, since this does not break the recursion, as the following calculation example shows:

```
listmult [3,5,0,3]  →  3 * listmult [5,0,3]
                    →  3 * (5 * listmult [0,3])
                    →  3 * (5 * 0)
                    →  3 * 0
                    →  0
```

Notice, that even though we should already know that the result will be 0 after the third reduction, we continue calculating. One solution to this problem is to use an *exception operator*, which will let us abort the recursion once a zero is encountered.

We want to find a solution to this problem using $\mathsf{HA}+\mathsf{EM}_1$, but since this system does not have any data structure for lists, we will have to reformulate the problem to a problem of pure arithmetic. Let instead some primitive recursive function $f : \mathtt{N} \to \mathtt{N}$ be given (notice that this corresponds to an infinite list). The product of the list $[f(0), f(1), \ldots, f(n-1)]$ will then be $\prod_{i=1}^{n} f(i-1)$. It is clearly a primitive recursive task to decide whether $\prod_{i=1}^{n} f(i-1) = m$ for given n and m, so we can push this task to the atomic level and introduce it as an atomic formula. We will also need to be able to say whether f will evaluate to zero on a certain input.

$$\mathsf{M}(\alpha,\beta): \text{ holds if } \prod_{i=1}^{\alpha} f(i-1) = \beta$$

$$\mathsf{N}(\alpha,\beta): \text{ holds if } f(\beta) = 0 \text{ and } \beta < \alpha$$

For M we have the following Post rules:

$$\frac{}{\mathsf{M}(0,1)} \qquad \frac{\mathsf{M}(\alpha,\beta)}{\mathsf{M}(\mathbf{S}\alpha, f(\alpha)*\beta)}$$

Notice that we have introduced the symbol $*$ which stands for multiplication. How this actually reduces is not really relevant; an answer of the form $f(2)*f(1)*f(0)$ is sufficiently informative for our purposes.

Intuitionistic proof

Firstly, we will solve the problem without using classical reasoning, and as we will see this is quite straightforward.

Proposition 6.2.1. *There is a term* `mult_in` *in* HA *such that it fulfills the specification:*

$$\vdash \mathtt{mult\_in} : \forall\alpha\exists\beta.\mathsf{M}(\alpha,\beta).$$

The proof is a straightforward induction proof, viz.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{M}(0,1)}}{\exists\beta.\mathsf{M}(0,\beta)} \qquad \dfrac{\dfrac{\dfrac{\exists\beta.\mathsf{M}(\gamma,\beta)^x \qquad \dfrac{\dfrac{\mathsf{M}(\gamma,\beta)^y}{\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)}}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}\,y}{\exists\beta.\mathsf{M}(\gamma,\beta) \to \exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}\,x}{\forall\gamma(\exists\beta.\mathsf{M}(\gamma,\beta) \to \exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta))}}{\exists\beta.\mathsf{M}(\alpha,\beta)}}{\forall\alpha\exists\beta.\mathsf{M}(\alpha,\beta)}$$

By annotating this proof tree we get

$$\mathtt{mult\_in} := \lambda\alpha.\mathtt{Rec}\ (1,\mathtt{True})\ (\lambda\gamma\lambda x.x[(\beta,y).(f(\gamma)*\beta, \mathtt{r}\ y)])\ \alpha.$$

Classical proof

Now we want to introduce classical reasoning in such a way that we can break the recursion once a zero is encountered. We will do this by applying the EM rule to $\forall\gamma.\mathsf{N}^\perp(\alpha,\gamma) \vee \exists\gamma.\mathsf{N}(\alpha,\gamma)$. The intention is then that the program will first assume $\mathsf{N}^\perp$, i.e. that all the list values are non-zero, and then execute the intuitionistic program whilst testing the EM-hypothesis. When this hypothesis is tested to be false, the program will exit the calculation, learn that N is the case and with this knowledge return a zero.

Proposition 6.2.2. *There is a term* `mult_cl` *which is in* $\mathsf{HA} + \mathsf{EM}_1$*, but not in* HA*, such that*

$$\vdash \texttt{mult\_cl} : \forall\alpha\exists\beta.\mathsf{M}(\alpha,\beta).$$

The proof will take the following form:

$$\dfrac{\dfrac{\begin{matrix}[\forall\gamma.\mathsf{N}^\perp(\alpha,\gamma)] & \qquad & [\exists\gamma.\mathsf{N}(\alpha,\gamma)]\\ \vdots & & \vdots \\ \exists\beta.\mathsf{M}(\alpha,\beta) & & \exists\beta.\mathsf{M}(\alpha,\beta)\end{matrix}}{\exists\beta.\mathsf{M}(\alpha,\beta)}}{\forall\alpha\exists\beta.\mathsf{M}(\alpha,\beta)}$$

In order to fill in the rest of the proof we will need some Post rules for N. Given m, n such that $m < n$ and $f(m) = 0$, it is necessarily true that $\prod_{i=1}^{n} f(i-1) = 0$, so therefore we can introduce the following Post rule:

$$\dfrac{\mathsf{N}(\alpha,\gamma)}{\mathsf{M}(\alpha,0)}$$

This rule is needed for the right hand side of the proof, which can now be finished:

$$\dfrac{\dfrac{\exists\gamma.\mathsf{N}(\alpha,\gamma) \qquad \dfrac{\mathsf{N}(\alpha,\gamma)}{\mathsf{M}(\alpha,0)}}{\mathsf{M}(\alpha,0)}}{\exists\beta.\mathsf{M}(\alpha,\beta)}$$

The proof term for this is

$$\texttt{mult\_cl\_rhs} := (0, \mathtt{W}_a^{\exists\gamma.\mathsf{N}(\alpha,\gamma)}[(\gamma,x).\mathtt{r}\ x]).$$

The left hand side is more tricky. Since we do not *need* the assumption of $\forall\gamma.\mathsf{N}^\perp(\alpha,\gamma)$ for anything, one could initially be tempted to just copy and paste the original intuitionistic version of the proof here. This, however, is not the solution we are looking for, since this would give no opportunity to throw an exception. So we must somehow include the assumption in the proof. One naïve way of doing this could be to insert a *detour* in the proof. Apply the following transformation to the intuitionistic proof:

$$\dfrac{\dfrac{\vdots}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\dfrac{\exists\beta.\mathsf{M}(\gamma,\beta)\to\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}{\vdots}} \quad\longmapsto\quad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\mathsf{N}^{\perp}(\alpha,\gamma)\to\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)} \qquad \dfrac{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}{\mathsf{N}^{\perp}(\alpha,\gamma)}}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\dfrac{\exists\beta.\mathsf{M}(\gamma,\beta)\to\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}{\vdots}}$$

This will yield a proof term like the following:

$$\texttt{mult\_cl\_step\_attempt} := (\lambda\_.x[(\beta,y).(f(\gamma)*\beta,\texttt{r}\ y)])(\texttt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}\ \gamma)$$

This approach does make it possible to exit the recursion once a zero is encountered. But this would require that the reduction path would not β-reduce the vacuous λ-abstraction, which is something that no common evaluation strategy would respect. Therefore we have to come up with another trick in order to get a term that will behave as wanted under a reasonable evaluation strategy. The trick lies in describing a new Post rule: Since $\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)$ holds whenever $\mathsf{M}(\gamma,\beta)$ holds, then it is certainly also the case that $\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)$ holds whenever $\mathsf{M}(\gamma,\beta)$ *and* $\mathsf{N}^{\perp}(\alpha,\gamma)$ holds. This becomes the Post rule

$$\dfrac{\mathsf{M}(\gamma,\beta) \qquad \mathsf{N}^{\perp}(\alpha,\gamma)}{\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)}$$

Thereby the aim is to encode the use of the hypothesis into the atomic level, such that the reduction rules of $\mathsf{HA}+\mathsf{EM}_1$ cannot remove it. To obtain the sought-after proof we simply make the following transformation on the intuitionistic proof:

$$\dfrac{\dfrac{\dfrac{\mathsf{M}(\gamma,\beta)}{\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)}}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\vdots} \quad\longmapsto\quad \dfrac{\dfrac{\dfrac{\mathsf{M}(\gamma,\beta) \qquad \dfrac{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}{\mathsf{N}^{\perp}(\alpha,\gamma)}}{\mathsf{M}(\mathbf{S}\gamma, f(\gamma)*\beta)}}{\exists\beta.\mathsf{M}(\mathbf{S}\gamma,\beta)}}{\vdots}$$

In the proof term, this transformation amounts to replacing the occurrence of $\texttt{r}\ y$ with $\texttt{r}\ y\ (\texttt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}\ \gamma)$. Thus, the term for the left hand side will be

$$\texttt{mult\_cl\_lhs} := \texttt{Rec}\ (1,\texttt{True})\ (\lambda\gamma\lambda x.x[(\beta,y).(f(\gamma)*\beta,\texttt{r}\ y\ (\texttt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}\ \gamma))])\ \alpha,$$

and the complete term is

$$\texttt{mult\_cl} := \lambda\alpha.(\texttt{mult\_cl\_lhs} \parallel_a \texttt{mult\_cl\_rhs}).$$

Reduction of `mult_cl`

We will examine how `mult_cl` reduces by example. The term is in normal form, so we need to apply it to a number. To keep the reduction graph on a reasonable size, we will choose the number 1. Since the right hand side does not have any occurrences of α, then all the redexes, except for the exceptional exits, will occur on the left hand side. We use the abbreviations `step` for $\lambda\gamma\lambda x.x[(\beta, y).(f(\gamma) * \beta, \mathtt{r}\ y\ (\mathtt{H}_a^{\forall\gamma.\mathsf{N}^\perp(1,\gamma)}\ \gamma))]$, and `H` for $\mathtt{H}_a^{\forall\gamma.\mathsf{N}^\perp(1,\gamma)}$. Thus, we will examine the reduction graph of the term $\mathtt{Rec}\ (1, \mathtt{True})\ \mathtt{step}\ 1$:

$$\begin{tikzcd}
 & \underline{\mathtt{Rec}\ (1,\mathtt{True})\ \mathtt{step}\ 1} \arrow[d] & \\
 & \underline{\mathtt{step}\ 0}\ (\underline{\mathtt{Rec}\ (1,\mathtt{True})\ \mathtt{step}\ 0}) \arrow[dr] \arrow[ddl] & \\
 & & \underline{\mathtt{step}\ 0}\ (1,\mathtt{True}) \arrow[dd, bend left] \\
\underline{(\lambda x.x[(\beta,y).(f(0)*\beta,\mathtt{r}\ y\ (\utilde{\mathtt{H}\ 0}))])(\underline{\mathtt{Rec}\ (1,\mathtt{True})\ \mathtt{step}\ 0})} \arrow[drr] \arrow[dd] & & \\
 & & \underline{(\lambda x.x[(\beta,y).(f(0)*\beta,\mathtt{r}\ y\ (\utilde{\mathtt{H}\ 0}))])\ (1,\mathtt{True})} \arrow[ddl, bend left] \\
(\underline{\mathtt{Rec}\ (1,\mathtt{True})\ \mathtt{step}\ 0})[(\beta,y).(f(0)*\beta,\mathtt{r}\ y\ (\utilde{\mathtt{H}\ 0}))] \arrow[dr] & & \\
 & \underline{(1,\mathtt{True})[(\beta,y).(f(0)*\beta,\mathtt{r}\ y\ (\utilde{\mathtt{H}\ 0}))]} \arrow[d] & \\
 & (f(0)*1,\mathtt{r}\ \mathtt{True}\ (\utilde{\mathtt{H}\ 0})) &
\end{tikzcd}$$

The wavy underline signifies that one of two things can happen: Either $f(0) \neq 0$, so $\mathsf{N}^\perp(1,0)$ holds, and then we can do an EM_{11}-reduction to replace the subterm with `True`, or $f(0) = 0$, and thus we can do an exceptional exit with an EM_{14}-reduction, for example:

$$\begin{aligned}
&(1,\texttt{True})[(\beta,y).(f(0)*\beta,\texttt{r}\ y\ (\texttt{H}\ 0))] \parallel_a (0,\texttt{W}_a^{\exists\gamma.\mathsf{N}(\alpha,\gamma)}[(\gamma,x).\texttt{r}\ x])\\
\to{}& (0,(0,\texttt{True})[(\gamma,x).\texttt{r}\ x])\\
\to{}& (0,\texttt{r}\ \texttt{True}).
\end{aligned}$$

Therefore, if we use a reduction strategy that prioritizes EM_{14}-reductions, it is clear that we can evaluate in a more efficient manner with this operator, especially if there is an exception early: Assume for instance that $f(999)=0$, and that want to evaluate `mult_cl` 1000. Then we can find the quite short reduction path

$$\begin{aligned}
&\texttt{mult\_cl}\ 1000\\
\twoheadrightarrow{}& (\lambda x.x[(\beta,y).(f(999)*\beta,\texttt{r}\ y\ (\texttt{H}\ 999))])(\texttt{Rec}\ (1,\texttt{True})\ \texttt{step}\ 999) \parallel_a \texttt{rhs}\\
\to{}& (0,(999,\texttt{True})[(\gamma,x).\texttt{r}\ x])\\
\to{}& (0,\texttt{r}\ \texttt{True}),
\end{aligned}$$

where $\texttt{rhs} = (0,\texttt{W}_a^{\exists\gamma.\mathsf{N}(\alpha,\gamma)}[(\gamma,x).\texttt{r}\ x])$.

Chapter 7

Program extraction from $\mathsf{HA} + \mathsf{EM}_1$

In this chapter, we introduce an operational semantics for system $\mathsf{HA} + \mathsf{EM}_1$ that is based on a call-by-name evaluation. We will test this on some situations with the examples from Chapter 6.

7.1 Natural semantics for $\mathsf{HA} + \mathsf{EM}_1$

We provide $\mathsf{HA} + \mathsf{EM}_1$ with a *natural semantics* (also known under the names *big-step semantics* or *evaluation semantics*).

Definition 7.1.1. We define the judgments $u \Downarrow (u'; \Delta)$ by the inference rules in the Figures 7.1 and 7.2, where Δ is a sequence of terms of the form $\mathtt{H}_a^{\forall \alpha . \mathsf{P}(\alpha)}\ n$, where n is a numeric term. In all of the rules, n represents numeric terms, and the other letters represent $\mathsf{HA} + \mathsf{EM}_1$-terms.

By $a \mathrel{\varepsilon} \Delta$, we mean that there is a $\mathtt{H}_a^{\forall \alpha . \mathsf{P}(\alpha)}$ such that $\mathtt{H}_a^{\forall \alpha . \mathsf{P}(\alpha)} \in \Delta$.

The intended meaning of $u \Downarrow (u'; \Delta)$ is, that u will *evaluate* to the normal form u', and so if $\vdash u : \exists \alpha . \mathsf{P}(\alpha)$, then by Theorem 5.6.1, u' will be of the form (n, u''), where n is a numeral such that $\mathsf{P}(n)$.

We can see from the following rule, that this semantics is based on a call-by-name strategy:

$$\frac{u \Downarrow (\lambda x . u'; \Delta) \qquad u'[x := v] \Downarrow (v'; \Delta')}{uv \Downarrow (v'; \Delta')}$$

The purpose of the Δ is to keep track of the EM_1-hypotheses such that the rules in Figure 7.2 can handle exceptions. It is indeed the intention that Δ will be empty whenever $u \Downarrow (u'; \Delta)$ and $\mathrm{FCV}(u) = \emptyset$.

$$\frac{}{x \Downarrow (x;\emptyset)} \qquad \frac{u \Downarrow (\lambda x.u';\Delta) \qquad u'[x := v] \Downarrow (v';\Delta')}{uv \Downarrow (v';\Delta')}$$

$$\frac{u_1 \Downarrow (u_1';\Delta_1) \qquad u_2 \Downarrow (u_2';\Delta_2) \qquad \cdots \qquad u_n \Downarrow (u_n';\Delta_n)}{\mathtt{r}\ u_1\ u_2\ \cdots\ u_n \Downarrow (\mathtt{r}\ u_1'\ u_2'\ \cdots\ u_n';\Delta_1,\Delta_2,\ldots,\Delta_n)}$$

$$\frac{u \Downarrow (u';\Delta)}{\lambda x.u \Downarrow (\lambda x.u';\Delta)} \qquad \frac{u \Downarrow (u';\Delta)}{\lambda \alpha.u \Downarrow (\lambda \alpha.u';\Delta)}$$

$$\frac{u \Downarrow (\lambda \alpha.u';\Delta) \qquad u'[\alpha := n] \Downarrow (v;\Delta')}{un \Downarrow (v;\Delta')} \qquad \frac{u_0 \Downarrow (v_0;\Delta_0) \qquad u_1 \Downarrow (v_1;\Delta_1)}{\langle u_0, u_1\rangle \Downarrow (\langle v_0, v_1\rangle;\Delta_0,\Delta_1)}$$

$$\frac{u \Downarrow (\langle v_0, v_1\rangle;\Delta)}{\pi_i u \Downarrow (v_i;\Delta)} \qquad \frac{u \Downarrow (u',\Delta) \qquad v \Downarrow (v',\Delta') \qquad w \Downarrow (w',\Delta'')}{u[x.v,y.w] \Downarrow (u'[x.v',y.w'];\Delta,\Delta',\Delta'')}$$

$$\frac{u \Downarrow (\iota_i u';\Delta) \qquad v_i[x_i := u'] \Downarrow (v';\Delta')}{u[x_0.v_0, x_1.v_1] \Downarrow (v';\Delta,\Delta')} \qquad \frac{u \Downarrow (u';\Delta)}{\iota_i u \Downarrow (\iota_i u';\Delta)}$$

$$\frac{u \Downarrow (u';\Delta)}{(n,u) \Downarrow ((n,u');\Delta)} \qquad \frac{u \Downarrow ((n,u');\Delta) \qquad v[\alpha := n][x := u'] \Downarrow (v';\Delta')}{u[(\alpha,x).v] \Downarrow (v';\Delta,\Delta')}$$

$$\frac{u \Downarrow (u';\Delta)}{\mathtt{Rec}\ u\ v\ \mathbf{0} \Downarrow (u';\Delta)} \qquad \frac{\mathtt{Rec}\ u\ v\ n \Downarrow (w;\Delta) \qquad v\ n\ w \Downarrow (w';\Delta')}{\mathtt{Rec}\ u\ v\ (\mathsf{S}n) \Downarrow (w';\Delta')}$$

$$\frac{uw \parallel_a vw \Downarrow (u';\Delta)}{(u \parallel_a v)w \Downarrow (u';\Delta)} \qquad \frac{\pi_i u \parallel_a \pi_i v \Downarrow (u';\Delta)}{\pi_i(u \parallel_a v) \Downarrow (u';\Delta)}$$

$$\frac{u[x_0.w_0, x_1.w_1] \parallel_a v[x_0.w_0, x_1.w_1] \Downarrow (u';\Delta)}{(u \parallel_a v)[x_0.w_0, x_1.w_1] \Downarrow (u';\Delta)}$$

$$\frac{u[(\alpha,x).w] \parallel_a v[(\alpha,x).w] \Downarrow (u';\Delta)}{(u \parallel_a v)[(\alpha,x).w] \Downarrow (u';\Delta)}$$

Figure 7.1: Natural semantics for $\mathsf{HA} + \mathsf{EM}_1$, part 1 of 2

$$\frac{n \text{ is not closed, or } \mathsf{P}(n) \equiv \mathtt{False}}{\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n \Downarrow (\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n, \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n)} \qquad \frac{}{\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}\ n \Downarrow (\mathtt{W}_a^{\exists\alpha.\mathsf{P}^{\perp}(\alpha)}\ n; \emptyset)}$$

$$\frac{n \text{ is closed, and } \mathsf{P}(n) \equiv \mathtt{True}}{\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n \Downarrow (\mathtt{True}, \emptyset)}$$

$$\frac{u \Downarrow (u'; \Delta) \qquad a \not\in \Delta}{u \parallel_a v \Downarrow (u'; \Delta)}$$

$$\frac{u \Downarrow (u'; \Delta) \qquad v[a := n] \Downarrow (v'; \Delta') \qquad \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n \in \Delta,\ \mathsf{P}(n) \equiv \mathtt{False}}{u \parallel_a v \Downarrow (v'; \Delta')}$$

where $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n$ is the first occurrence in Δ with a.

$$\frac{u \Downarrow (u', \Delta) \qquad v \Downarrow (v'; \Delta') \qquad a \,\varepsilon\, \Delta, \text{ but if } \mathtt{H}_a^{\forall\alpha\mathsf{P}(\alpha)}\ n \in \Delta, \text{ then } \mathsf{P}(n) \equiv \mathtt{False}}{u \parallel_a v \Downarrow (u' \parallel_a v'; \Delta, \Delta')}$$

Figure 7.2: Natural semantics for $\mathsf{HA}+\mathsf{EM}_1$, part 2 of 2

Most of the rules are obvious choices for a call-by-name semantics. The more interesting rules are the ones in Figure 7.2. Especially the rule

$$\frac{u \Downarrow (u'; \Delta) \qquad v[a := n] \Downarrow (v'; \Delta') \qquad \mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n \in \Delta,\ \mathsf{P}(n) \equiv \mathtt{False}}{u \parallel_a v \Downarrow (v'; \Delta')}$$

where $\mathtt{H}_a^{\forall\alpha.\mathsf{P}(\alpha)}\ n$ is the first occurrence in Δ with a, is important. It is here that an exception is thrown. By choosing the first occurrence in Δ that gives rise to the exception, we make sure that it is the inner-most occurrence of $\mathtt{H}_a$ that is checked first, and therefore the problem with confluence is solved. This will ensure, for example, that the searching example from Chapter 6 provides a top-down search algorithm.

Lemma 7.1.2. *If $u \Downarrow (u', \Delta)$, then u' is in normal form.*

Proof. From strong normalization of $\mathsf{HA}+\mathsf{EM}_1$ we get that all such derivations must be finite, and by induction on the derivations we get that u' must be in normal form. □

Lemma 7.1.3. *If $u \Downarrow (u', \Delta)$ and $u \Downarrow (u'', \Delta')$, then $u' = u''$.*

Proof. By induction on the derivation. There is always only one possible reduction rule. Especially, in the $u \parallel_a v$-situation, only one of the three rules may apply. □

7.2 Searching

We return to the search example from Chapter 6. Consider again the situation where $n = 1$, and $P(0), \neg P(1)$ holds. This means we have the atomic formulas $\mathsf{P}(0), \mathsf{P}(1)$, and $\mathsf{Q}(0)$. Since we in this example are working with hypothesis variables, we have to add the following rules:

$$\overline{h_1 \Downarrow (h_1;\emptyset)} \qquad \overline{h_2 \Downarrow (h_2;\emptyset)}$$

Now, we wish to find a term u and a Δ such that

$$(\mathbf{0}, \texttt{r}\ (\texttt{r}\ h_2\ (\texttt{Rec}\ h_1\ \texttt{search\_cl\_step}\ 1))) \parallel_a \texttt{W}_a^{\exists\alpha\mathsf{Q}(\alpha)} \Downarrow (u;\Delta).$$

It is easy to check that

$$(\lambda\beta\lambda x.\texttt{r}\ x\ (\texttt{H}_a\ \beta))\ 0\ h_1 \Downarrow (\texttt{r}\ h_1\ (\texttt{H}_a\ 0), \texttt{H}_a\ 0),$$

and using this we can derive

$$\frac{\dfrac{\overline{h_1 \Downarrow (h_1;\emptyset)}}{\texttt{Rec}\ h_1\ \texttt{step}\ 0 \Downarrow (h_1,\emptyset)} \qquad (\lambda\beta\lambda x.\texttt{r}\ x\ (\texttt{H}_a\ \beta))\ 0\ h_1 \Downarrow (\texttt{r}\ h_1\ (\texttt{H}_a\ 0), \texttt{H}_a\ 0)}{\texttt{Rec}\ h_1\ \texttt{step}\ 1 \Downarrow (\texttt{r}\ h_1\ (\texttt{H}_a\ 0), \texttt{H}_a\ 0)}$$

which furthermore brings us to the conclusion that

$$(\mathbf{0}, \texttt{r}\ (\texttt{r}\ h_2\ (\texttt{Rec}\ h_1\ \texttt{search\_cl\_step}\ 1))) \Downarrow ((\mathbf{0}, \texttt{r}\ (\texttt{r}\ h_1\ (\texttt{H}_a\ 0))); \texttt{H}_a\ 0).$$

Let us abbreviate this so: $\texttt{lhs} \Downarrow (\texttt{lhs}'; \texttt{H}_a\ 0)$. Thus, knowing what the left-hand side reduction is, we can finalize the derivation:

$$\frac{\texttt{lhs} \Downarrow (\texttt{lhs}'; \texttt{H}_a^{\forall\alpha.\mathsf{Q}^{\perp}(\alpha)}\ 0) \qquad \dfrac{\overline{\texttt{True} \Downarrow (\texttt{True};\emptyset)}}{(0,\texttt{True}) \Downarrow ((0,\texttt{True});\emptyset)} \qquad \mathsf{Q}(0) \equiv \texttt{False}}{\texttt{lhs} \parallel_a \texttt{W}_a^{\exists\alpha.\mathsf{Q}(\alpha)} \Downarrow ((0,\texttt{True}),\emptyset)}$$

Thus, we can conclude that, in our semantics, the searching term in this situation will evaluate to $(0, \texttt{True})$, as expected.

7.3 Multiplication

We will now return to the multiplication example of Chapter 6, and test our natural semantics on the term $\texttt{mult\_cl}\ 1$, where we assume that $f(0) = 0$ (which means that $\mathsf{N}(1,0) \equiv \texttt{True}$). Recall that $\texttt{mult\_cl}$ is defined as

$$\texttt{mult\_cl} := \lambda\alpha.(\texttt{mult\_cl\_lhs} \parallel_a \texttt{mult\_cl\_rhs}),$$

where

$$\texttt{mult\_cl\_lhs} := \texttt{Rec}\ (1, \texttt{True})\ (\lambda\gamma\lambda x.x[(\beta, y).(f(\gamma) * \beta, \texttt{r}\ y\ (\texttt{H}_a^{\mathsf{N}^\perp}\ \gamma))])\ \alpha,$$
$$\texttt{mult\_cl\_rhs} := (0, \texttt{W}_a^{\exists\gamma.\mathsf{N}(\alpha,\gamma)}[(\gamma, x).\texttt{r}\ x]).$$

The last rule in the derivation will be

$$\dfrac{\texttt{mult\_cl} \Downarrow (\texttt{mult\_cl}, \texttt{H}_a\ \gamma) \qquad \texttt{lhs} \parallel_a \texttt{rhs} \Downarrow (u, \Delta)}{\texttt{mult\_cl}\ 1 \Downarrow (u, \Delta)}$$

where

$$\texttt{lhs} := \texttt{Rec}\ (1, \texttt{True})\ (\lambda\gamma\lambda x.x[(\beta, y).(f(\gamma) * \beta, \texttt{r}\ y\ (\texttt{H}_a^{\mathsf{N}^\perp}\ \gamma))])\ 1$$
$$\texttt{rhs} := \texttt{mult\_cl\_rhs}.$$

In order to find (u, Δ), we will first need to find (v, Δ') such that $\texttt{lhs} \Downarrow (v, \Delta')$. For this we first observe that

$$\texttt{Rec}\ (1, \texttt{True})\ \texttt{step}\ 0 \Downarrow (1, \texttt{True}, \emptyset),$$

and then:

$$\dfrac{\texttt{step}\ 0 \Downarrow (\lambda x.x[(\beta, y).w']; \texttt{H}_a\ 0) \qquad \dfrac{(1, \texttt{True}) \Downarrow ((1, \texttt{True}), \emptyset) \qquad w \Downarrow (w; \texttt{H}_a\ 0)}{(1, \texttt{True})[(\beta, y).w'] \Downarrow (w; \texttt{H}_a\ 0)}}{\texttt{step}\ 0\ (1, \texttt{True}) \Downarrow (w; \texttt{H}_a\ 0)}$$

where

$$w := (f(0) * 1, \texttt{r}\ \texttt{True}\ (\texttt{H}_a\ 0)),$$
$$w' := (f(0) * \beta, \texttt{r}\ \texttt{True}\ (\texttt{H}_a\ 0)).$$

Hence we have

$$\dfrac{\texttt{Rec}\ (1, \texttt{True})\ \texttt{step}\ 0 \Downarrow (1, \texttt{True}, \emptyset) \qquad \texttt{step}\ 0\ (1, \texttt{True}) \Downarrow (w; \texttt{H}_a\ 0)}{\texttt{lhs} \Downarrow (w; \texttt{H}_a\ 0)}$$

The right hand side will be evaluated thus:

$$\dfrac{\dfrac{(0, \texttt{True}) \Downarrow ((0, \texttt{True}); \emptyset) \qquad \texttt{r}\ \texttt{True} \Downarrow (\texttt{r}\ \texttt{True}; \emptyset)}{(0, \texttt{True})[(\gamma, x).\texttt{r}\ x] \Downarrow (\texttt{r}\ \texttt{True}; \emptyset)}}{(0, (0, \texttt{True})[(\gamma, x).\texttt{r}\ x]) \Downarrow ((0, \texttt{r}\ \texttt{True}); \emptyset)}$$

So we can finish the derivation by tying these together:

$$\frac{\texttt{lhs} \Downarrow (w; \texttt{H}_a\ 0) \qquad (0, (0, \texttt{True})[(\gamma, x).\texttt{r}\ x]) \Downarrow ((0, \texttt{r}\ \texttt{True}); \emptyset) \qquad \mathsf{N}(1,0) \equiv \texttt{True}}{\texttt{lhs} \parallel_a \texttt{rhs} \Downarrow ((0, \texttt{r}\ \texttt{True}); \emptyset)}$$

Therefore,

$$\texttt{mult\_cl}\ 1 \Downarrow ((0, \texttt{r}\ \texttt{True}); \emptyset),$$

as expected.

Chapter 8

Conclusion

In this thesis, the basics of classical program extraction have been discussed. We have introduced the systems $\lambda\mu$ and $\lambda\mu^{\mathbf{T}}$ as examples of confluent λ-calculi with control, that correspond to classical logic via the Curry–Howard correspondence. Furthermore, we have discussed the system $\mathsf{HA}+\mathsf{EM}_1$ which is a non-confluent Curry–Howard system for an arithmetic with limited classical reasoning, and we have presented Aschieri's new proof of strong normalization of $\mathsf{HA}+\mathsf{EM}_1$. Then, we have worked out some examples of proofs in $\mathsf{HA}+\mathsf{EM}_1$ and analyzed their reduction possibilities. Lastly, we have developed an operational semantics for $\mathsf{HA}+\mathsf{EM}_1$ which gives a deterministic way of extracting witnesses from proofs of Σ^0_1-sentences and tested it on our examples.

8.1 Further research

The semantics introduced in Chapter 7 does not very well describe *how* the witnesses are extracted, for this it would be better with a *structural operational semantics* (also known as *small-step semantics*), which would describe each individual step in the computation, and not just the overall result, as is the case with the natural semantics (or *big-step semantics*). I did not succeed in doing this in a satisfactory manner, but it would be interesting to see such a semantics.

Extraction to $\lambda\mu^{\mathbf{T}}$

One of my hopes was to define a translation $[\![\cdot]\!]$ of $\mathsf{HA}+\mathsf{EM}_1$-terms into $\lambda\mu^{\mathbf{T}}$-terms, such that if $\vdash_{\mathsf{HA}+\mathsf{EM}_1} u : \forall\alpha\exists\beta.\mathsf{P}(\alpha,\beta)$, then $\vdash_{\lambda\mu^{\mathbf{T}}} [\![u]\!] : \mathtt{N} \to \mathtt{N}$ in such a way that if $[\![u]\!]\, n \twoheadrightarrow m$, then $\mathsf{P}(n,m) \equiv \mathtt{True}$. My approach was to define $[\![u]\!]^{\Delta}$, where the set Δ is used to store the hypothesis variables along with their relevant information (when in the left hand side of $\|_a$, this relevant information is the term on the right hand side, and when in the right hand side, the relevant information is the witness provided by the left hand side).

The following is my proposed definition:

Definition 8.1.1 (Term extraction)**.** Let u be a $\mathsf{HA}+\mathsf{EM}_1$-term in normal form. We define $\llbracket u \rrbracket$ as $\llbracket u \rrbracket^{\emptyset}$, where this is given recursively by:

$$\begin{array}{lcl}
\llbracket m \rrbracket^{\Delta} & = & m, \quad \text{if } m \text{ is a numeric term} \\
\llbracket \lambda\alpha.u \rrbracket^{\Delta} & = & \lambda\alpha^{\mathtt{N}}.\, \llbracket u \rrbracket^{\Delta} \\
\llbracket \lambda x^{\tau}.u \rrbracket^{\Delta} & = & \lambda x^{\llbracket \tau \rrbracket}.\, \llbracket u \rrbracket^{\Delta} \\
\llbracket uv \rrbracket^{\Delta} & = & \llbracket u \rrbracket^{\Delta}\, \llbracket v \rrbracket^{\Delta} \\
\llbracket \mathtt{Rec}\ u\ v\ n \rrbracket^{\Delta} & = & \mathtt{Rec}\ \llbracket u \rrbracket^{\Delta}\ \llbracket v \rrbracket^{\Delta}\ \llbracket n \rrbracket^{\Delta} \\
\llbracket u \parallel_a v \rrbracket^{\Delta} & = & \mathtt{catch}_a\, \llbracket u \rrbracket^{\Delta,(a,v)} \\
\left\llbracket \mathtt{W}_a^{\exists\gamma \mathsf{Q}^{\perp}(\gamma)} \right\rrbracket^{\Delta,(a,n)} & = & n \\
\llbracket u[(\alpha,x).v] \rrbracket^{\Delta} & = & \llbracket v \rrbracket^{\Delta}\, [\alpha := \llbracket u \rrbracket^{\Delta}] \\
\llbracket (m,u) \rrbracket^{\Delta} & = & m, \quad \text{if } u \text{ contains no } \mathtt{H}_a \text{ with } a \in \Delta \\
\llbracket (m,u) \rrbracket^{\Delta,(a,v)} & = & \mathtt{if}\ \mathsf{Q}(\varepsilon)\ \mathtt{then}\ \llbracket (m,u) \rrbracket^{\Delta}\ \mathtt{else}\ \mathtt{throw}_a\ \llbracket v \rrbracket^{\Delta,(a,\varepsilon)}, \\
 & & \text{if } u \text{ has } \mathtt{H}_a^{\forall\gamma.\mathsf{Q}(\gamma)}\ \varepsilon \text{ as subterm,} \\
 & & \text{and } \varepsilon \text{ is not bound in } u
\end{array}$$

When we apply this transformation on the term `mult_cl` from Chapter 6 we get the following:

$$\llbracket \mathtt{mult\_cl} \rrbracket = \lambda\alpha^{\mathtt{N}}.\mathtt{catch}_a\ \llbracket \mathtt{mult\_cl\_lhs} \rrbracket^{(a,\mathtt{mult\_cl\_rhs})}.$$

Let $\Delta = (a, \mathtt{mult\_cl\_rhs})$, and say that

$$\mathtt{step} := \lambda\gamma\lambda x.x[(\beta,y).(f(\gamma)*\beta, \mathtt{r}\ y\ (\mathtt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}\ \gamma))],$$

for then

$$\begin{aligned}
\llbracket \mathtt{mult\_cl\_lhs} \rrbracket^{\Delta} &= \llbracket \mathtt{Rec}\ (1, \mathtt{True})\ \mathtt{step}\ \alpha \rrbracket^{\Delta} \\
&= \mathtt{Rec}\ 1\ \ \llbracket \mathtt{step} \rrbracket^{\Delta}\ \ \alpha
\end{aligned}$$

where

$$\begin{aligned}
\llbracket \mathtt{step} \rrbracket^{\Delta} &= \left\llbracket \lambda\gamma\lambda x.x[(\beta,y).(f(\gamma)*\beta, \mathtt{r}\ y\ (\mathtt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)}\ \gamma))] \right\rrbracket^{\Delta} \\
&= \lambda\gamma^{\mathtt{N}}\lambda x^{\mathtt{N}}.\ \left\llbracket f(\gamma*\beta, \mathtt{r}\ y\ (\mathtt{H}_a^{\forall\gamma.\mathsf{N}^{\perp}(\alpha,\gamma)})) \right\rrbracket^{\Delta} [\beta := \llbracket x \rrbracket^{\Delta}] \\
&= \lambda\gamma^{\mathtt{N}}\lambda x^{\mathtt{N}}.\mathtt{if}\ \mathsf{N}^{\perp}(\alpha,\gamma)\ \mathtt{then}\ f(\gamma)*x \\
&\quad \mathtt{else}\ \mathtt{throw}_a\ \ \llbracket \mathtt{mult\_cl\_rhs} \rrbracket^{(a,\gamma)},
\end{aligned}$$

and

$$\begin{aligned}
\llbracket \mathtt{mult\_cl\_rhs} \rrbracket^{(a,\gamma)} &= \left\llbracket (0, \mathtt{W}_a^{\exists\gamma.\mathsf{N}(\alpha,\gamma)}[(\gamma,x).\mathtt{r}\ x]) \right\rrbracket^{(a,\gamma)} \\
&= 0.
\end{aligned}$$

Therefore the complete extracted term will be

$$\lambda\alpha^{\mathtt{N}}.\mathtt{catch}_a\ \mathtt{Rec}\ 1\ (\lambda\gamma^{\mathtt{N}}\lambda x^{\mathtt{N}}.\mathtt{if}\ \mathsf{N}^{\perp}(\alpha,\gamma)\ \mathtt{then}\ f(\gamma)*x\ \mathtt{else}\ \mathtt{throw}_a\ 0)\ \alpha.$$

It can be checked that this term fulfills the specification.

My hope is that the following question can be answered positively, but I was not able to prove it.

Question 8.1.2. *Suppose that* $\vdash_{\mathsf{HA}+\mathsf{EM}_1} t : \forall\alpha\exists\beta.\mathsf{P}(\alpha,\beta)$*, and that* t *is a closed* $\mathsf{HA}+\mathsf{EM}_1$*-term in normal form. Does it hold that:*

- $\vdash_{\lambda\mu\mathtt{T}} [\![t]\!] : \mathtt{N}\to\mathtt{N}$*, and*
- *for any* $n \in \mathbb{N}$*,* $\mathsf{P}(n, [\![t]\!]\,(n))$ *holds?*

Bibliography

[1] Yohji Akama, Stefano Berardi, Susumu Hayashi, and Ulrich Kohlenbach, *An Arithmetical Hierarchy of the Law of Excluded Middle and Related Principles*, Logic in Computer Science, 2004. Proceedings of the 19th Annual IEEE Symposium on, IEEE, 2004, pp. 192–201.

[2] Federico Aschieri, *Learning, Realizability and Games in Classical Arithmetic*, Ph.D. thesis, Università degli Studi di Torino, Dipartimento di Informatica and Queen Mary, University of London, School of Electronic Engineering and Computer Science, 2011.

[3] ———, *Strong Normalization for HA+EM1 by Non-Deterministic Choice*, EPTCS (2013), to appear.

[4] Federico Aschieri and Stefano Berardi, *Interactive Learning-Based Realizability for Heyting Arithmetic with EM$_1$*, Logical Methods in Computer Science **6** (2010), no. 3.

[5] Federico Aschieri and Stefano Berardi, *A New Use of Friedman's Translation: Interactive Realizability*, Logic, Construction, Computation, Ontos-Verlag Series in Mathematical Logic, Berger et al. editors (2012).

[6] Federico Aschieri, Stefano Berardi, and Giovanni Birolo, *Realizability and Strong Normalization for a Curry-Howard interpretation of HA + EM1*, LIPIcs, to appear.

[7] Federico Aschieri and Margherita Zorzi, *Interactive Realizability and the elimination of Skolem functions in Peano Arithmetic*, arXiv preprint arXiv:1210.3114 (2012).

[8] Jeremy Avigad, *A Realizability Interpretation for Classical Arithmetic*, Logic Colloquim '98, Lecture Notes in Logic 13 (Pudlák Buss, Hájek, ed.), 2000.

[9] Henk P. Barendregt, *The Lambda Calculus: Its Syntax and Semantics*, 2nd ed., Studies in Logic, vol. 103, Elsevier, 1984.

[10] Stefano Berardi, *Some intuitionistic equivalents of classical principles for degree 2 formulas*, Annals of Pure and Applied Logic **139** (2006), no. 1, 185–200.

[11] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg, *Refined Program Extraction from Classical Proofs*, Annals of Pure and Applied Logic **114** (2002), no. 1, 3–25.

[12] Thierry Coquand and Gerard Huet, *The Calculus of Constructions*, Information and Control **76** (1986).

[13] Matthias Felleisen and Daniel P. Friedman, *Control operators, the SECD-machine, and the λ-calculus.*, 3rd Working Conference on the Formal Description of Programming Concepts (Martin Wirsing, ed.), North-Holland Publishing, 1986, pp. pp. 193–219.

[14] Harvey Friedman, *Classically and Intuitionistically Provably Recursive Functions*, Müller and Scott (eds.): Higher Set Theory, Springer, 1978, pp. 21–27.

[15] Gerhard Gentzen, *Über das Verhältnis zwischen intuitionistischer und klassischer Arithmetik*, Archive for Mathematical Logic. **16** (1974), no. 3, 119–132 (Originally to appear in Mathematische Annalen 1933, but was withdrawn).

[16] Herman Geuvers, Robbert Krebbers, and James McKinna, *The $\lambda\mu T$-calculus*, Annals of Pure and Applied Logic **164** (2013), no. 6, 676–701.

[17] Jean-Yves Girard, Paul Taylor, and Yves Lafont, *Proofs and Types*, Cambridge University Press, 1989.

[18] Kurt Gödel, *Zur intuitionistischen Arithmetik und Zahlentheorie*, Ergebnisse eines mathematischen Kolloquiums **4** (1933), 34–38.

[19] E Mark Gold, *Limiting Recursion*, The Journal of Symbolic Logic **30** (1965), no. 1, 28–48.

[20] Timothy G Griffin, *A formulae-as-type notion of control*, Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, 1989, pp. 47–58.

[21] Hugo Herbelin, *An intuitionistic logic that proves Markov's principle*, Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on, IEEE, 2010, pp. 50–56.

[22] Thomas Jech, *Set Theory*, 3rd millenium ed., Springer monographs in mathematics, Springer, 2002.

[23] Ingebrigt Johansson, *Der Minimalkalkül, ein reduzierter intuitionistischer Formalismus*, Compositio mathematica **4** (1937), 119–136.

[24] Robbert Krebbers, *Classical logic, control calculi and data types*, Master's thesis, Radboud Universiteit Nijmegen, 2010.

[25] Georg Kreisel, *Mathematical Significance of Consistency Proofs*, The Journal of Symbolic Logic **23** (1958), no. 2, 155–182.

[26] ______, *On Weak Completeness of Intuitionistic Predicate Logic*, The Journal of Symbolic Logic **27** (1962), no. 2, 139–158.

[27] P. J. Landin, *Correspondence between ALGOL 60 and Church's Lambda-notation: part I*, Commun. ACM **8** (1965), no. 2, 89–101.

[28] Yevgeniy Makarov, *Practical Program Extraction from Classical Proofs*, Electronic Notes in Theoretical Computer Science **155** (2006), 521–542.

[29] Michel Parigot, *Programming with proofs: A second order type theory*, ESOP '88 (H. Ganzinger, ed.), Lecture Notes in Computer Science, vol. 300, Springer Berlin Heidelberg, 1988, pp. 145–159.

[30] ______, *$\lambda\mu$-calculus: An Algorithmic Interpretation of Classical Natural Deduction*, Logic programming and automated reasoning, Springer, 1992, pp. 190–201.

[31] ______, *Proofs of Strong Normalisation for Second Order Classical Natural Deduction*, Journal of Symbolic Logic (1997), 1461–1479.

[32] Christine Paulin-Mohring, *Extracting F_ω's programs from proofs in the calculus of constructions*, Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (New York, NY, USA), POPL '89, ACM, 1989, pp. 89–104.

[33] Rózsa Péter, *Recursive Functions*, Academic Press, New York and London, 1967.

[34] Dag Prawitz, *Ideas and Results in Proof Theory*, Proceedings of the second Scandinavian logic symposium, 1971, pp. 235–307.

[35] J. Rees and W. Clinger, *Revised Report on the Algorithmic Language Scheme*, SIGPLAN Not. **21** (1986), no. 12, 37–79.

[36] Niels Jakob Rehof and Morten Heine Sørensen, *The $\lambda\Delta$-calculus*, Theoretical Aspects of Computer Software, Springer, 1994, pp. 516–542.

[37] Morten Heine Sørensen and Paweł Urzyczyn, *Lectures on the Curry-Howard Isomorphism*, 1st ed., Studies in Logic, vol. 149, Elsevier, Amsterdam, 2006.

[38] Gerald Jay Sussman and Guy L. Steele, Jr., *Scheme: A Interpreter for Extended Lambda Calculus*, Higher-Order and Symbolic Computation **11** (1998), no. 4, 405–439 (English).

[39] W. W. Tait, *Intensional Interpretations of Functionals of Finite Type I*, The Journal of Symbolic Logic **32** (1967), no. 2, 198–212 (English).

[40] C.L. Talcott, *The essence of Rum: A theory of the intensional and extensional aspects of Lisp-type computation*, Ph.D. thesis, Stanford University, 1985.