Program Extraction from Proofs of Weak Head Normalization

Małgorzata Biernacka
Olivier Danvy
Kristian Støvring

BRICS Report Series RS-05-12

ISSN 0909-0878 April 2005

Program extraction from proofs of weak head normalization

Małgorzata Biernacka, Olivier Danvy, and Kristian Støvring

BRICS[1]
Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, 8200 Aarhus N, Denmark

Abstract

We formalize two proofs of weak head normalization for the simply typed lambda-calculus in first-order minimal logic: one for normal-order reduction, and one for applicative-order reduction in the object language. Subsequently we use Kreisel's modified realizability to extract evaluation algorithms from the proofs, following Berger; the proofs are based on Tait-style reducibility predicates, and hence the extracted algorithms are instances of (weak head) normalization by evaluation, as already identified by Coquand and Dybjer.

Key words: program extraction, normalization by evaluation, weak head normalization.

1 Introduction and related work

In the early nineties, Berger and Schwichtenberg introduced normalization by evaluation in a proof-theoretic setting [5]. Berger then substantiated their normalization function by extracting it from a proof of strong normalization [2], using Kreisel's modified realizability interpretation [11]. In their own study of what also turned out to be normalization by evaluation [7,8], Coquand and Dybjer constructed normalization functions interpreting source terms in so-called glueing models. They also outlined a process of "program extraction" with which their normalization algorithms can be obtained from simple instances of a normalization proof due to Martin-Löf, and noticed the connection with Berger's work. In this article, we use part of Berger's framework to formalize some of the relationship identified by Coquand and Dybjer between glueing models and Tait-style proofs as used by Martin-Löf. We consider two intuitionistic proofs of weak head normalization for the simply typed λ-calculus: A normal-order proof essentially due to Martin-Löf, and an applicative-order counterpart due to Hofmann [12, page 152].

[1] Basic Research in Computer Science (`www.brics.dk`), funded by the Danish National Research Foundation.

Our results can be described informally as follows: Applying modified realizability to the definition of the Tait-style reducibility predicate gives the definition of a glueing model. Applying modified realizability to the proof of normalization of a particular simply typed term t gives a λ-term denoting the interpretation of t in this glueing model.

The program extraction we perform can be intuitively explained as a "program optimization" [7]: Martin-Löf's normalization proof is formalized in an intuitionistic meta-language, and such a proof can informally be regarded as a function returning the normal form, together with a proof that this result actually is a normal form [7,9]. To go from such a normalization proof to a function returning *only* the normal form, one can then remove the redundant parts representing the axioms for convertibility, and simplify the types accordingly [7]. Berger's use of the modified realizability interpretation works like that (in the setting of first-order logic): the axioms for convertibility can be stated as *Harrop formulas*, and subproofs which are proofs of Harrop formulas disappear during the extraction.

Coquand and Dybjer's weak normalization function for the λ-calculus can be perceived as an optimized version of the program we extract in the applicative-order case. This is not surprising, since our focus here is on formalizing the proofs and considering two different evaluation strategies in the object language rather than optimizing the extracted programs. In doing so, we identify certain technical difficulties arising with the applicative-order case, and we adjust the extraction method to solve them. In his recent work [3], Berger has proposed a similar refinement as part of a bigger framework, the Uniform Heyting Arithmetic.

Our account has the following limitations:

- Like Berger, we only partially formalize the normalization proof. Since a part of the proof is performed at the meta-level, we cannot formally extract a normalization function, but only a λ-term denoting the glueing interpretation of t for every *particular* term t.
- For simplicity, we only consider normalization of closed terms. With this restriction, we do not need to formalize renaming of bound variables.
- We only treat the case of the simply typed λ-calculus with one uninterpreted base type, whereas Coquand and Dybjer consider a variety of more advanced examples.

In the remainder of this article, we first review the modified realizability interpretation (Section 2); we then specify the problem of weak head normalization for the λ-calculus and we extract a call-by-name λ-interpreter and then a call-by-value λ-interpreter (Section 3). ML implementations of the extracted normalization programs are presented in Appendix A.

2 Preliminaries

We begin by reviewing the techniques used by Berger to extract normalization functions from proofs [2]. The key concept is Kreisel's *modified realizability* proof interpretation [11]. Our presentation is based on Berger's article [2] and Troelstra's treatise [14].

2.1 First-order minimal logic

We formalize the normalization proofs in a first-order logic $\mathbf{M_1}$. The language of $\mathbf{M_1}$ is that of many-sorted first-order minimal logic with conjunction, defined in a standard way. Specifically, such a language is given by:

- Sorts ι, ι_1, ι_2, ...
- Constants $\mathtt{c}^\iota$, function symbols $\mathtt{f}^{\iota_1 \times \ldots \times \iota_n \to \iota}$.
- Predicate symbols $\mathbf{P}^{\iota_1 \times \ldots \times \iota_n}$.

(We will see that the sorts of $\mathbf{M_1}$ are the base types of the extracted programs.) The terms and formulas of $\mathbf{M_1}$ are:

$$\begin{array}{lrl} \textit{Terms} & t^\iota := & x^\iota \mid \mathtt{c}^\iota \mid \mathtt{f}^{\iota_1 \times \ldots \times \iota_n \to \iota}(t_1^{\iota_1}, \ldots, t_n^{\iota_n}) \\ \textit{Formulas} & \phi, \psi := & \mathbf{P}^{\iota_1 \times \ldots \times \iota_n}(t_1^{\iota_1}, \ldots, t_n^{\iota_n}) \mid \phi \wedge \psi \mid \phi \to \psi \mid \forall x^\iota.\, \phi \mid \exists x^\iota.\, \phi \end{array}$$

A natural deduction proof system of $\mathbf{M_1}$ is shown in Figure 1. Instead of presenting the proof rules graphically, we directly define a proof of a formula ϕ to be a dependently typed λ-term d^ϕ. In the definition, $\mathrm{FV}(\phi)$ denotes the set of free variables in the formula ϕ, while $\mathrm{FA}(d)$ denotes the set of free assumptions in the proof d. Only the interesting defining cases of $\mathrm{FA}(d)$ are shown.

We will also use the notation $u_1 : \psi_1, \ldots, u_n : \psi_n \vdash_{\mathbf{M_1}} d : \phi$ to mean that d^ϕ is an $\mathbf{M_1}$-proof of ϕ with free assumptions contained in the set $\{u_1^{\psi_1}, \ldots, u_n^{\psi_n}\}$.

2.2 Modified realizability

In the presentation we use one of Troelstra's variants of modified realizability [14, p. 218].

The programs extracted from proofs are terms of the simply typed λ-calculus with product and unit types, and with the sorts of $\mathbf{M_1}$ as base types:

$$\begin{array}{ll} \textit{Types} & \sigma := 1 \mid \iota_1 \mid \iota_2 \mid \ \ldots \ \mid \sigma_1 \to \sigma_2 \mid \sigma_1 \times \sigma_2 \\ \textit{Terms} & \mathrm{t} := x^\sigma \mid \mathrm{t}_0\, \mathrm{t}_1 \mid \lambda x^\sigma.\mathrm{t} \mid \mathrm{fst}\, \mathrm{t} \mid \mathrm{snd}\, \mathrm{t} \mid (\mathrm{t}_1, \mathrm{t}_2) \mid \ * \mid \mathtt{c} \mid \mathtt{f}(\mathrm{t}_1, \ldots, \mathrm{t}_n) \end{array}$$

Note that the language of λ-terms includes the constants and function symbols of $\mathbf{M_1}$. Moreover, meta-variables ranging over λ-terms are denoted with the Roman font (t), and thus differ from the notation for logical terms in $\mathbf{M_1}$ (t).

In the following, by a "program" we mean a simply typed λ-term as just defined. Only in Appendix A are actual programming language implementations considered [6].

Definition 2.1 [Program extraction] Given an $\mathbf{M_1}$-proof d of ϕ, we define a type $\tau(d)$ and a λ-term $[\![d]\!]$ of type $\tau(d)$ as follows:

$$
\begin{array}{ll}
(\mathit{ass.}) & u^{\phi} \\
(\to^{+}) & (\lambda u^{\phi}.d^{\psi})^{\phi\to\psi} \\
(\to^{-}) & (d^{\phi\to\psi}\, e^{\phi})^{\psi} \\
(\wedge^{+}) & (d^{\phi}, e^{\psi})^{\phi\wedge\psi} \\
(\wedge_1^{-}) & (\mathrm{fst}\, d^{\phi\wedge\psi})^{\phi} \\
(\wedge_2^{-}) & (\mathrm{snd}\, d^{\phi\wedge\psi})^{\psi} \\
(\forall^{+}) & (\lambda x^{\iota}.d^{\phi})^{\forall x^{\iota}.\,\phi} \\
 & (\text{provided } x^{\iota} \notin \mathrm{FV}(\psi) \text{ for every } u^{\psi} \in \mathrm{FA}(d)) \\
(\forall^{-}) & (d^{\forall x^{\iota}.\,\phi}\, t^{\iota})^{\phi\,[t/x]} \\
(\exists^{+}) & \langle t, d^{\phi\,[t/x]}\rangle^{\exists x.\,\phi} \\
(\exists^{-}) & [e^{\exists x.\,\phi}, u^{\phi}.d^{\psi}]^{\psi} \\
 & (\text{provided } x \notin \mathrm{FV}(\psi), \\
 & \text{and } x \notin \mathrm{FV}(\chi) \text{ for every } v^{\chi} \in \mathrm{FA}(d) \setminus \{u^{\phi}\})
\end{array}
$$

$$
\begin{array}{lll}
\text{where } \mathrm{FA}(u^{\phi}) & = & \{u^{\phi}\} \\
\mathrm{FA}((\lambda u^{\phi}.d^{\psi})^{\phi\to\psi}) & = & \mathrm{FA}(d^{\psi}) \setminus \{u^{\phi}\} \\
\mathrm{FA}([e^{\exists x.\,\phi}, u^{\phi}.d^{\psi}]^{\psi}) & = & \mathrm{FA}(e^{\exists x.\,\phi}) \cup (\mathrm{FA}(d^{\psi}) \setminus \{u^{\phi}\}) \\
 & \text{etc.} &
\end{array}
$$

Fig. 1. The proof system $\mathbf{M_1}$

$$
\begin{array}{ll}
\tau(\mathbf{P}(t_1,\ldots,t_n)) & := 1 \\
\tau(\phi\wedge\psi) & := \tau(\phi)\times\tau(\psi) \\
\tau(\phi\to\psi) & := \tau(\phi)\to\tau(\psi) \\
\tau(\forall x^{\iota}.\,\phi) & := \iota\to\tau(\phi) \\
\tau(\exists x^{\iota}.\,\phi) & := \iota\times\tau(\phi)
\end{array}
\qquad
\begin{array}{ll}
[\![u^{\phi}]\!] & := x_u^{\tau(\phi)} \\
[\![\lambda u^{\phi}.d^{\psi}]\!] & := \lambda x_u^{\tau(\phi)}.[\![d]\!] \\
[\![d^{\phi\to\psi}\, e^{\phi}]\!] & := [\![d]\!]\,[\![e]\!] \\
[\![(d^{\phi}, e^{\psi})]\!] & := ([\![d]\!], [\![e]\!]) \\
[\![\mathrm{fst}\, d^{\phi\wedge\psi}]\!] & := \mathrm{fst}\,[\![d]\!] \\
[\![\mathrm{snd}\, d^{\phi\wedge\psi}]\!] & := \mathrm{snd}\,[\![d]\!] \\
[\![\lambda x^{\iota}.d^{\phi}]\!] & := \lambda x^{\iota}.[\![d]\!] \\
[\![d^{\forall x^{\iota}.\,\phi}\, t^{\iota}]\!] & := [\![d]\!]\, t \\
[\![\langle t, d^{\phi\,[t/x]}\rangle]\!] & := (t, [\![d]\!]) \\
[\![[e^{\exists x.\,\phi}, u^{\phi}.d^{\psi}]]\!] & := [\![d]\!][\mathrm{fst}\,[\![e]\!]/x, \mathrm{snd}\,[\![e]\!]/x_u]
\end{array}
$$

Subsequently, we simplify the extracted terms using the isomorphisms $A\times 1 \cong A$, $1\times B \cong B$, $A \to 1 \cong 1$, and $1 \to B \cong B$.[2] This means that the type $\tau(\phi)$ of an extracted term will either be 1 or not contain 1 at all. The first case happens exactly

[2] In the original version of modified realizability [11], as well as in newer variants [4], this "optimization" is built-in. We use the simpler version for presentational purposes.

when ϕ is a *Harrop formula*—we then informally say that ϕ "has no computational content."

2.2.1 Soundness of the extraction

We now briefly consider in what sense a λ-term extracted from a proof of ϕ "realizes" ϕ. The notion of realizability is formalized in a finite-type extension $\mathbf{M_1^-}(\lambda)$ of $\mathbf{M_1}$ [2]. The point is that every extracted term $[\![d]\!]$ is a term of $\mathbf{M_1^-}(\lambda)$.

Definition 2.2 [Modified realizability] By induction on the $\mathbf{M_1}$-formula ϕ we define an $\mathbf{M_1^-}(\lambda)$-formula $\mathrm{t}^{\tau(\phi)}\,\mathsf{mr}\,\phi$ as follows:

$$\begin{aligned}
\mathrm{t}^1\ \mathsf{mr}\ \mathbf{P}(\mathrm{t}_1,\ldots,\mathrm{t}_n) &:= \mathbf{P}(\mathrm{t}_1,\ldots,\mathrm{t}_n)\\
\mathrm{t}^{\sigma_1\times\sigma_2}\ \mathsf{mr}\ \phi\wedge\psi &:= (\mathrm{fst}\,\mathrm{t})\,\mathsf{mr}\,\phi\ \wedge\ (\mathrm{snd}\,\mathrm{t})\,\mathsf{mr}\,\psi\\
\mathrm{t}^{\sigma_1\to\sigma_2}\ \mathsf{mr}\ \phi\to\psi &:= \forall y^{\sigma_1}.\,(y\,\mathsf{mr}\,\phi\to \mathrm{t}\,y\,\mathsf{mr}\,\psi)\\
\mathrm{t}^{\iota\to\sigma}\ \mathsf{mr}\ \forall z^\iota.\,\phi(z) &:= \forall z^\iota.\,\mathrm{t}\,z\,\mathsf{mr}\,\phi(z)\\
\mathrm{t}^{\iota\times\sigma}\ \mathsf{mr}\ \exists z^\iota.\,\phi(z) &:= (\mathrm{snd}\,\mathrm{t})\,\mathsf{mr}\,\phi(\mathrm{fst}\,\mathrm{t})
\end{aligned}$$

Given an $\mathbf{M_1}$-proof d of ϕ, the goal is therefore to give an $\mathbf{M_1^-}(\lambda)$-proof of $[\![d]\!]\,\mathsf{mr}\,\phi$. It turns out that the proof d is allowed to contain free assumptions of Harrop formulas.

Theorem 2.3 (Soundness of modified realizability) *Let $\psi_1,\ldots,\psi_n$ be Harrop formulas. If $u_1:\psi_1,\ldots,u_n:\psi_n\vdash_{\mathbf{M_1}} d:\phi$, then $\psi_1,\ldots,\psi_n\vdash_{\mathbf{M_1^-}(\lambda)}[\![d]\!]\,\mathsf{mr}\,\phi$.*

Proof. Standard [2,14]. □

As an example, suppose that d is a $\mathbf{M_1}$-proof of $\forall x.\,\exists y.\,\mathbf{P}(x,y)$ containing only Harrop formulas as free assumptions. Then Theorem 2.3 gives an $\mathbf{M_1^-}(\lambda)$-proof of $\forall x.\,\mathbf{P}(x,\mathrm{fst}([\![d]\!]\,x))$ from the same free assumptions. In this way, free Harrop assumptions can be thought of as "axioms" with no effect on the extracted program.

2.2.2 Eliminating computationally redundant variables

The extraction procedure can be refined in order to keep the resulting programs simple. We first present a refinement due to Berger [2] which allows computationally redundant universal variables to be eliminated from the extracted program. To this end, we add a new kind of formulas of the form $\{\forall x\}.\,\phi$ with the following introduction and elimination rules:

$$\begin{aligned}
&(\forall^+)\ (\{\lambda x^\iota\}.d^\phi)^{\{\forall x^\iota\}.\,\phi}\\
&\qquad(\text{provided } x^\iota\notin \mathrm{FV}(\psi) \text{ for every } u^\psi\in\mathrm{FA}(d)) \text{ and } x\notin\mathrm{CV}(d))\\
&(\forall^-)\ (d^{\{\forall x^\iota\}.\,\phi}\,\{t^\iota\})^{\phi\,[t/x]},
\end{aligned}$$

where the set of computationally relevant variables $\mathrm{CV}(d)$ is defined as the set of all variables occurring free in a witness for an existential quantifier, or in any term instantiating a universal quantifier in d. A universally quantified variable is called redundant if it is not computationally relevant.

The type of realizers for the new formulas simply ignores the redundant variable: $\tau(\{\forall x\}.\,\phi):=\tau(\phi)$. The corresponding clause for modified realizability is

$\mathsf{t}\,\mathsf{mr}\,\{\forall x\}.\,\phi := \forall x.\,\mathsf{t}\,\mathsf{mr}\,\phi$ (with $x \notin \mathrm{FV}(\mathsf{t})$). As desired, the extracted program does not contain the redundant variable:

$$\begin{aligned} [\![\{\lambda x\}.d]\!] &:= [\![d]\!] \\ [\![d\,\{t\}]\!] &:= [\![d]\!] \end{aligned}$$

The proof of soundness of modified realizability can be extended to handle this case [2].

The second refinement allows us to choose which of the existential quantifiers in a formula we want to have witnesses of. However, we postpone the description of this extension until Section 3.3, where it will be essential that not all of the existential quantifiers are realized.

3 Weak head normalization

We now specify the problem of weak head normalization for the λ-calculus. In the presentation, we assume that all terms are well-typed, but for clarity we omit all typing annotations. We consider only closed terms.

By normalization we understand the process of reducing a term to a normal form, where the basic reduction step is β-reduction [1]:

$$(\lambda x.t)\,s \to t\,[s/x].$$

The compatible closure of β-reduction yields the one-step reduction relation.

Weak head normalization is a restricted form of normalization producing terms in weak head normal form, which—for closed terms—stops at a λ-abstraction, without normalizing its body. Therefore any λ-abstraction is in weak head normal form.

We consider two deterministic restrictions of the one-step reduction that lead to weak head normal forms: the normal-order and applicative-order reduction strategies. Since weak head normalization is closely related to *evaluation* in the λ-calculus regarded as a programming language, where computations are not performed under λ-abstractions, we also refer to the above reduction strategies as the call-by-name and call-by-value evaluation strategies, respectively [13].

Definition 3.1 [Normal-order reduction] The normal-order reduction strategy is obtained from one-step reduction by restricting it to the following rules:

$$(\beta)\;\; (\lambda x.r)\,s \to r\,[s/x]$$

$$(\nu)\qquad \frac{r \to r'}{r\,s \to r'\,s}$$

Definition 3.2 [Applicative-order reduction] The (left-to-right) applicative-order reduction strategy is obtained from one-step reduction by restricting it to the following rules:

$$(\beta_{\mathrm{v}})\;\; (\lambda x.r)\,s \to r\,[s/x] \text{ if } s \text{ is a value}$$

$$(\nu)\qquad \frac{r \to r'}{r\,s \to r'\,s}$$

$$(\mu_{\mathrm{v}})\qquad \frac{s \to s'}{r\,s \to r\,s'} \qquad \text{if } r \text{ is a value}$$

Values are λ-abstractions.

These specifications of evaluation strategies can be axiomatized directly in the logic $\mathbf{M_1}$ using only Harrop formulas, as will be shown in the following sections.

The theorem we want to prove can be stated informally as follows:

Theorem 3.3 (Weak head normalization) *The process of reducing a closed well-typed λ-term according to either of the above strategies terminates with a (weak head) normal form.*

The proof proceeds by first defining a suitable logical relation on well-typed closed terms that implies the desired property. Next we show that every well-typed term satisfies this relation. Obviously, the exact shape of the proof relies on the chosen reduction strategy (normal-order or applicative-order), and consequently the extracted program produces the result according to the corresponding strategy in the object language (call by name or call by value).

In the rest of the section we first formalize this theorem for the two evaluation strategies, and then we use modified realizability to extract the underlying programs. For the case of call-by-name evaluation this is a straightforward exercise, whereas in the call-by-value case we need to refine the extraction procedure further. Our development in this section formalizes and extends the proof of normalization for call-by-value evaluation presented in Pierce's book [12, pp. 149-152].

3.1 The object language

We consider an explicitly typed version of the simply typed λ-calculus with variables contained in a countable set $V = x_1^{T_1}, x_2^{T_2}, \ldots$ (infinitely many of each type). This language is now encoded in a first-order minimal logic. The variables are used to index the sorts and constants of the logic, which is given by the following:

- Sorts: For every type T and finite set of variables X, we have the sort Λ_T^X of object-level λ-terms of type T containing exactly free variables X.

- Constants: The λ-term constructors are:

$$\begin{aligned}
\mathtt{VAR}_x &: \Lambda_T^{\{x\}} && \text{(for each variable } x^T\text{)}\\
\mathtt{LAM}_{x,T_1,T_2,X} &: \Lambda_{T_2}^X \to \Lambda_{T_1\to T_2}^{X\setminus\{x\}} && \text{(where } x \text{ has type } T_1\text{)}\\
\mathtt{APP}_{T_1,T_2,X,Y} &: \Lambda_{T_1\to T_2}^X \to \Lambda_{T_1}^Y \to \Lambda_{T_2}^{X\cup Y}
\end{aligned}$$

- Predicate symbols: the set of predicate symbols differs for call-by-name and call-by-value evaluation, and we specify each of them in Section 3.2 and Section 3.3, respectively.

Notation. For the sake of presentation, we use a number of notational abbreviations when constructing object terms, e.g., we omit type annotations from λ-term constructors—in most cases they can be inferred from the context; we use the "uncurried" versions of the term constructors; we also write $\mathtt{LAM}\, x_i.\, t$ instead of $\mathtt{LAM}_{x_i,T_1,T_2,X}(t)$, and $\mathtt{VAR}\, x_i$ instead of $\mathtt{VAR}_{x_i}$.

We abbreviate sorts of closed terms $\Lambda_T^{\emptyset}$ as Λ_T. In the formulas used in the rest of this article, we only quantify over sorts of closed terms.

We treat substitution in λ-terms at the meta level. For a variable $x_i^{T_1}$ and logical terms $s^{\Lambda_{T_1}}$ and $t^{\Lambda_{T_2}}$, we define $t\,[s/\mathtt{VAR}\,x_i]$ as t with every subterm $\mathtt{VAR}\,x_i$ not in scope of a $\mathtt{LAM}_{x_i}$ replaced by s. As Λ_{T_1} is a sort of closed λ-terms, free object-level variables are never captured as a result of this form of substitution. For this definition of $t\,[s/\mathtt{VAR}\,x_i]$ to faithfully encode substitution, we further require that all free logical variables in t range over sorts of closed object-level terms. Thus the formal definition of substitution is as follows:

Definition 3.4 Let $x_i^{T_1}$ be a variable, and let $s^{\Lambda_{T_1}^{\emptyset}}$ and $t^{\Lambda_{T_2}^{X}}$ be logical terms such that all free logical variables in t belong to (possibly different) sorts $\Lambda_T^{\emptyset}$ of closed object-level terms. We define the term $t\,[s/\mathtt{VAR}\,x_i]$ of sort $\Lambda_{T_2}^{X\setminus\{x_i\}}$ inductively:

$$\begin{aligned}
y^{\Lambda_T^{\emptyset}}\,[s/\mathtt{VAR}\,x_i] &= y \qquad \text{(where y is a logical variable)}\\
\mathtt{VAR}\,x_i\,[s/\mathtt{VAR}\,x_i] &= s\\
\mathtt{VAR}\,x_j\,[s/\mathtt{VAR}\,x_i] &= \mathtt{VAR}\,x_j \qquad (j \neq i)\\
\mathtt{APP}(t_1,t_2)\,[s/\mathtt{VAR}\,x_i] &= \mathtt{APP}(t_1\,[s/\mathtt{VAR}\,x_i], t_2\,[s/\mathtt{VAR}\,x_i])\\
(\mathtt{LAM}_{x_i,X}\,t_1)\,[s/\mathtt{VAR}\,x_i] &= \mathtt{LAM}_{x_i,X}\,t_1\\
(\mathtt{LAM}_{x_j,X}\,t_1)\,[s/\mathtt{VAR}\,x_i] &= \mathtt{LAM}_{x_j,X\setminus\{x_i\}}\,(t_1\,[s/\mathtt{VAR}\,x_i]) \qquad (j \neq i)
\end{aligned}$$

3.2 Call-by-name evaluation

First, we give an axiomatization of call-by-name evaluation in the λ-calculus. We use two primitive predicates: $\mathbf{Ev}(t,s)$, understood as "t evaluates to s," and $\mathbf{Rd}(t,s)$, understood as "t reduces to s in one step." The process of call-by-name evaluation is defined through the following axioms:

$$\begin{aligned}
&(A_1)\ \{\forall s\}.\,\mathbf{Rd}(\mathtt{APP}(\mathtt{LAM}\,\mathbf{x_i}.\,\mathbf{t}, s), \mathbf{t}\,[s/\mathtt{VAR}\,\mathbf{x_i}])\\
&(A_2)\ \{\forall rst\}.\,\mathbf{Rd}(r,s) \to \mathbf{Rd}(\mathtt{APP}(r,t), \mathtt{APP}(s,t))\\
&(A_3)\ \{\forall rst\}.\,\mathbf{Rd}(r,s) \to \mathbf{Ev}(s,t) \to \mathbf{Ev}(r,t)\\
&(A_4)\ \mathbf{Ev}(\mathbf{t},\mathbf{t}) \text{ for all terms } \mathbf{t} = \mathtt{LAM}\,\mathbf{x_i}.\,\mathbf{s}
\end{aligned}$$

The first and the last axioms are schematic in the logical term $\mathbf{t}$ whose free logical variables must range over sorts of closed object-level terms. As explained above, this restriction is necessary for the meta-level definition of substitution to be correct.

The axioms formally capture the idea that (call-by-name) evaluation is the reflexive, transitive closure of (normal-order) one-step reduction as defined above. The notion of reduction is β-reduction (axiom (A_1)); it can be applied to left-most redexes (axiom (A_2)) yielding a one-step reduction relation. The evaluation stops when a λ-abstraction is reached (the family of axioms (A_4)); otherwise it is defined as the transitive closure of one-step reduction (axiom (A_3)).

In the proofs, we will use free assumption variables A_1, A_2, A_3, A_4 corresponding to the respective axioms above. Since all the axioms are Harrop formulas, these free variables will not occur in the extracted programs.

3.2.1 Formalizing the proof

The logical relation used in the proof is defined as follows:

$$\begin{aligned}\mathbf{R}_b(t) &:= \exists v.\mathbf{Ev}(t,v)\\ \mathbf{R}_{T_1\to T_2}(t) &:= \exists v.\mathbf{Ev}(t,v) \wedge \forall s.\,\mathbf{R}_{T_1}(s) \to \mathbf{R}_{T_2}(\mathtt{APP}(t,s))\end{aligned}$$

A term of an arrow type satisfying the relation $\mathbf{R}_T$ is not only required to evaluate to a value (or "halt", in Pierce's terms [12, p. 150]), but it should also halt when applied to another halting term. This stronger condition allows to prove the desired theorem for both call-by-value and call-by-name evaluation strategies. If we are only interested in evaluation at base types, a weaker condition is actually enough to prove the normalization theorem for call-by-name evaluation (see Section 3.4), but for the call-by-value case we still need this stronger definition.

We immediately see that every term satisfying the relation $\mathbf{R}_T$ evaluates to a value:

Lemma 3.5 $\{\forall t\}.\,\mathbf{R}_T(t) \to \exists v.\mathbf{Ev}(t,v)$.

Proof. By induction on types at the meta level. The corresponding proof terms are:

$$\mathrm{p}_1^b = \{\lambda t\}.\lambda u^{\mathbf{R}_b(t)}.u$$

$$\mathrm{p}_1^{T_1\to T_2} = \{\lambda t\}.\lambda u^{\mathbf{R}_{T_1\to T_2}(t)}.\mathrm{fst}\,u$$

□

To prove the main lemma, we need the following property.

Lemma 3.6 $\{\forall rs\}.\,\mathbf{Rd}(r,s) \to \mathbf{R}_T(s) \to \mathbf{R}_T(r)$.

Proof. By induction on types at the meta level.

Case b. Assume $\mathbf{Rd}(r,s)$ and $\mathbf{R}_b(s)$. By Lemma 3.5, we obtain $\exists v.\mathbf{Ev}(s,v)$ from $\mathbf{R}_b(s)$. Then using axiom (A_3) we deduce $\exists v.\mathbf{Ev}(r,v)$.

The proof term corresponding to this case is as follows:

$$\mathrm{p}_2^b = \{\lambda rs\}.\lambda u^{\mathbf{Rd}(r,s)} v^{\mathbf{R}_b(s)}.[\mathrm{p}_1^b\, v, w^{\mathbf{Ev}(s,v')}.\langle v', A_3\,\{rsv'\}u\,w\,\rangle]$$

Case $T_1 \to T_2$. Assume $\mathbf{Rd}(r,s)$ and $\mathbf{R}_{T_1\to T_2}(s)$. We need to prove $\exists v.\mathbf{Ev}(r,v)$ and $\forall t.\,\mathbf{R}_{T_1}(t) \to \mathbf{R}_{T_2}(\mathtt{APP}(r,t))$. The first fact is proved analogously to the base case. For the second, assume that $\mathbf{R}_{T_1}(t)$ holds for some t. By axiom (A_2) we obtain $\mathbf{Rd}(\mathtt{APP}(r,t), \mathtt{APP}(s,t))$. Next, unwinding the definition of $\mathbf{R}_{T_1\to T_2}(s)$ yields $\mathbf{R}_{T_2}(\mathtt{APP}(s,t))$. Hence, by induction hypothesis we conclude that $\mathbf{R}_{T_2}(\mathtt{APP}(r,t))$. Here is the corresponding proof term:

$$\mathrm{p}_2^{T_1\to T_2} = \{\lambda rs\}.\lambda u^{\mathbf{Rd}(r,s)} v^{\mathbf{R}_{T_1\to T_2}(s)}.(\mathrm{p}_{2,1}^{T_1\to T_2}, \mathrm{p}_{2,2}^{T_1\to T_2})$$

where

$$\mathrm{p}_{2,1}^{T_1\to T_2} = [\mathrm{p}_1^{T_1\to T_2}\, v, w^{\mathbf{Ev}(s,v')}.\langle v', A_3\,\{rsv'\}\,u\,w\rangle]$$

$$\mathrm{p}_{2,2}^{T_1\to T_2} = \lambda t^{\Lambda_{T_1}} z^{\mathbf{R}_{T_1}(t)}.\mathrm{p}_2^{T_2}\,\{\mathtt{APP}(r,t)\mathtt{APP}(s,t)\}\,(A_2\,\{rst\}\,u)\,(\mathrm{snd}\,v\,s\,z)$$

□

Lemma 3.7 *For any term t of type T, with $\mathrm{FV}(t) = \{x_1, \ldots, x_n\}$, and for any n-tuple of closed terms $\vec{r} = r_1, \ldots, r_n$ of types T_i such that $\mathbf{R}_{T_i}(r_i)$ holds for all $1 \leq i \leq n$, we have*

$$\mathbf{R}_T(t\,[\vec{r}/\vec{x}]).$$

(We use the abbreviation $t\,[\vec{r}/\vec{x}]$ for $t\,[r_1/\mathtt{VAR}\,x_1] \cdots [r_n/\mathtt{VAR}\,x_n]$.)

Proof. By induction on the typing derivation (or, on the structure of t, parameterized by the set of free variables). The formula to prove is

$$\forall \vec{r}.\,(\mathbf{R}_{T_1}(r_1) \wedge \ldots \wedge \mathbf{R}_{T_n}(r_n)) \rightarrow \mathbf{R}_T(t\,[\vec{r}/\vec{x}]).$$

Case $t = \mathtt{VAR}\,x_i^T$. Obvious. $\mathrm{p}_3^{\mathtt{VAR}\,x_i,\vec{x}} = \lambda \vec{r}\vec{u}.u_i$.

Case $t = \mathtt{APP}(s_1^{T_1 \rightarrow T}, s_2^{T_1})$. We apply the induction hypothesis to both subterms to obtain $\mathbf{R}_{T_1 \rightarrow T}(s_1\,[\vec{r}/\vec{x}])$ and $\mathbf{R}_{T_1}(s_2\,[\vec{r}/\vec{x}])$. Unwinding the definition of $\mathbf{R}_{T_1 \rightarrow T}(s_1\,[\vec{r}/\vec{x}])$ then yields $\mathbf{R}_T(\mathtt{APP}(s_1, s_2)\,[\vec{r}/\vec{x}])$ (using $\mathtt{APP}(s_1\,[\vec{r}/\vec{x}], s_2\,[\vec{r}/\vec{x}]) = \mathtt{APP}(s_1, s_2)\,[\vec{r}/\vec{x}]$).

$$\mathrm{p}_3^{\mathtt{APP}(s_1,s_2),\vec{x}} = \lambda \vec{r}\vec{u}.\mathrm{snd}(\mathrm{p}_3^{s_1,\vec{x}}\,\vec{r}\vec{u})\,(s_2\,[\vec{r}/\vec{x}])\,(\mathrm{p}_3^{s_2,\vec{x}}\,\vec{r}\vec{u}).$$

Case $t = \mathtt{LAM}\,x_{n+1}^{T_1}.\,r^{T_2} (T = T_1 \rightarrow T_2)$. We need to show that $\exists v.\mathbf{Ev}(t\,[\vec{r}/\vec{x}], v)$ and $\forall s.\,\mathbf{R}_{T_1}(s) \rightarrow \mathbf{R}_{T_2}(\mathtt{APP}(t\,[\vec{r}/\vec{x}], s))$. The first fact follows from (an instance of) the axiom (A_4), since $(\mathtt{LAM}\,x_{n+1}.\,r)\,[\vec{r}/\vec{x}]$ is a λ-abstraction. For the second, assume that $\mathbf{R}_{T_1}(s)$ holds for some s. By induction hypothesis, $\mathbf{R}_{T_2}(r\,[\vec{r}/\vec{x}]\,[s/x_{n+1}])$ holds. We now obtain $\mathbf{R}_{T_2}(\mathtt{APP}(\mathtt{LAM}\,x_{n+1}.\,r\,[\vec{r}/\vec{x}], s))$ using axiom (A_1) and Lemma 3.6, which concludes the proof. The corresponding proof term reads as follows:

$$\mathrm{p}_3^{\mathtt{LAM}\,x_{n+1}.\,r,\vec{x}} = \lambda \vec{r}\vec{u}.(\mathrm{p}_{3,1}, \mathrm{p}_{3,2})$$

where

$$\mathrm{p}_{3,1} = \langle (\mathtt{LAM}\,x_{n+1}.\,r)\,[\vec{r}/\vec{x}], A_4\rangle$$
$$\mathrm{p}_{3,2} = \lambda s^{\Lambda_{T_1}} v^{\mathbf{R}_{T_1}(s)}.\mathrm{p}_2^{T_2}\,\{t_1 t_2\}\,(A_1\,\{s\})\,\mathrm{p}_3^{r,\vec{x}x_{n+1}}\,(\vec{r}s)\,(\vec{u}v)$$

with

$$t_1 = \mathtt{APP}(\mathtt{LAM}\,x_{n+1}.\,r\,[\vec{r}/\vec{x}], s)$$
$$t_2 = r\,[\vec{r}/\vec{x}]\,[s/\mathtt{VAR}\,x_{n+1}]$$

□

The normalization theorem can now be stated formally as follows.

Theorem 3.8 *For any closed term t of type T, $\exists v.\mathbf{Ev}(t, v)$ holds.*

Proof. By Lemma 3.7, $\mathbf{R}_T(t)$ holds. Hence, by Lemma 3.5, $\exists v.\mathbf{Ev}(t, v)$ holds.

$$\mathrm{p} = \mathrm{p}_1^T\,(\mathrm{p}_3^t\,\varepsilon\,\varepsilon),$$

where ε denotes the empty tuple. □

3.2.2 Extracted program

Since the induction on the structure of terms in the proof of Lemma 3.7 is done at the meta level, from the proof of Theorem 3.8 we do not obtain one extracted program of type $\Lambda_T \to \Lambda_T$ realizing the formula $\forall t^{\Lambda_T}.\, \exists v^{\Lambda_T}.\, \mathbf{Ev}(t, v)$, but rather—for each term t^{Λ_T}—we extract a program 'computing' a term v such that $\mathbf{Ev}(t, v)$ is provable in $\mathbf{M}_{\mathbf{1}}^{-}(\lambda)$ [2].

We first consider the types $\tau(\mathbf{R}_T(t))$ of programs extracted from Lemma 3.7 (for specific terms t^{Λ_T}.) We see that the types $\tau(\mathbf{R}_T(t))$ are independent of t, and that they can be characterized inductively like this:

$$\begin{aligned} \tau(\mathbf{R}_b) &:= \Lambda_b \\ \tau(\mathbf{R}_{T_1 \to T_2}) &:= \Lambda_{T_1 \to T_2} \times (\Lambda_{T_1} \to \tau(\mathbf{R}_{T_1}) \to \tau(\mathbf{R}_{T_2})) \end{aligned}$$

This defines the semantic domains of a glueing model similar to the ones considered by Coquand and Dybjer (relative to any particular model of $\mathbf{M}_{\mathbf{1}}^{-}(\lambda)$).

The terms extracted from Lemma 3.7 can be inductively described as follows (they are parameterized by a tuple of free variables $\vec{x}$):

$$\begin{aligned} \mathrm{eval}_{\,\mathtt{VAR}\, x_i, \vec{x}} &= \lambda \vec{t}\vec{u}. u_i \\ \mathrm{eval}_{\,\mathtt{APP}(r,s), \vec{x}} &= \lambda \vec{t}\vec{u}. \mathrm{snd}(\mathrm{eval}_{\,r,\vec{x}}\, \vec{t}\vec{u})\, (s\, [\vec{t}/\vec{x}])\, (\mathrm{eval}_{\,s,\vec{x}}\, \vec{t}\vec{u}) \\ \mathrm{eval}_{\,\mathtt{LAM}\, x_{n+1}.\, t, \vec{x}} &= \lambda \vec{t}\vec{u}. (\mathtt{LAM}\, x_{n+1}.\, t\, [\vec{t}/\vec{x}], \lambda s v. [\mathrm{p}_2^{T}]\, (\mathrm{eval}_{\,t, \vec{x}x_{n+1}}\, (\vec{t}s)(\vec{u}v))) \end{aligned}$$

with

$$\begin{aligned} [\mathrm{p}_2^{b}] &= \lambda u. u \\ [\mathrm{p}_2^{T_1 \to T_2}] &= \lambda x. (\mathrm{fst}\, x, \lambda s v. [\mathrm{p}_2^{T_2}]\, ((\mathrm{snd}\, x)\, s\, v)) \end{aligned}$$

(Note that $[\mathrm{p}_2^{T}]$ is $\beta\eta\times$-equivalent to the identity function.) For every closed term t^{Λ_T}, $\mathrm{eval}_{\,t,\varepsilon}$ denotes the glueing model interpretation of the object-level term denoted by t^{Λ_T}.

From Lemma 3.5 we obtain the 'reification' function mapping semantic values back to syntax (parameterized with the type of a given term):

$$\begin{aligned} \downarrow_b &= \lambda u^{\Lambda_b}. u \\ \downarrow_{T_1 \to T_2} &= \lambda u^{\Lambda_{T_1 \to T_2} \times (\Lambda_{T_1} \to \tau(\mathbf{R}_{T_1}) \to \tau(\mathbf{R}_{T_2}))}. \mathrm{fst}\, u \end{aligned}$$

The complete program is the composition of the two functions and it is therefore an instance of (weak head) normalization by evaluation:

$$[\mathrm{p}_{t^T}] = \downarrow_T (\mathrm{eval}_{\,t,\varepsilon}\, \varepsilon\varepsilon)$$

In this presentation of the evaluation function there are two environments, represented by the vectors $\vec{t}$ and $\vec{u}$, whose elements can be substituted for the respective variables in the vector $\vec{x}$ (by construction, the length of all the vectors is the same). The program produces weak head normal forms, according to the call-by-name strategy given by the axioms, and it is correct in the sense that the formula $\mathbf{Ev}(t, [\mathrm{p}_{t^T}])$ is provable in $\mathbf{M}_{\mathbf{1}}^{-}(\lambda)$ for every closed simply typed term t of type T.

3.3 Call-by-value evaluation

The process of call-by-value evaluation of closed terms is defined through the following axioms:

$$
\begin{array}{ll}
(A_1) & \{\forall s\}.\, \mathbf{V}(s) \rightarrow \mathbf{Rd}(\mathtt{APP}(\mathtt{LAM}\,\mathbf{x_i}.\,\mathbf{t}, s), \mathbf{t}\,[s/\mathtt{VAR}\,\mathbf{x_i}]) \\
(A_2) & \{\forall rst\}.\, \mathbf{Rd}(r,s) \rightarrow \mathbf{Rd}(\mathtt{APP}(r,t), \mathtt{APP}(s,t)) \\
(A_2') & \{\forall rst\}.\, \mathbf{V}(r) \rightarrow \mathbf{Rd}(s,t) \rightarrow \mathbf{Rd}(\mathtt{APP}(r,s), \mathtt{APP}(r,t)) \\
(A_4) & \mathbf{V}(\mathbf{t}) \text{ for all terms } \mathbf{t} = \mathtt{LAM}\,\mathbf{x_i}.\,\mathbf{s}
\end{array}
$$

Similarly to the call-by-name case, these axioms directly encode the definition of one-step call-by-value evaluation strategy. In this case, however, the predicate **Ev** cannot be taken as primitive anymore, because we need to know more about the evaluation process. Informally, this is due to the fact that under call by value—in order for the proof to go through—we have to verify that whenever r reduces to s in zero or more steps, and $\mathbf{R}_T(r)$ holds, then also $\mathbf{R}_T(s)$ holds. Thus we need to proceed by induction on the length of the reduction sequence $r \rightarrow \ldots \rightarrow s$.

To this end we define an auxiliary relation $\mathbf{Rd}^*_n$:

$$
\begin{array}{ll}
\mathbf{Rd}^*_{0}(t,s) & := t = s \\
\mathbf{Rd}^*_{n+1}(t,s) & := \exists r.\, \mathbf{Rd}(t,r) \wedge \mathbf{Rd}^*_n(r,s)
\end{array}
$$

A formula $\mathbf{Rd}^*_n(t,s)$ is to be understood as "t reduces to s in n steps." Just as for the simple types, we do not formalize the induction on natural numbers used in proofs of properties of $\mathbf{Rd}^*_n$—it is done at the meta level.

Then we can define the evaluation predicate as follows:

$$
\mathbf{Ev}_n(t,v) := \mathbf{Rd}^*_n(t,v) \wedge \mathbf{V}(v)
$$

This definition requires extending the logic $\mathbf{M_1}$ with the usual axioms for equality (the soundness of modified realizability is preserved with this extension [14]).

3.3.1 Computationally irrelevant existential variables

The problem with the above specification of the predicate $\mathbf{Ev}_n$ is that via modified realizability a witness to the formula $\exists v.\, \mathbf{Ev}_n(t,v)$ will be a sequence of terms, representing the whole reduction sequence $t \rightarrow \ldots \rightarrow v$, while we are only interested in the final result, i.e., v. To rectify this, we introduce a further refinement of the program extraction procedure that allows to choose which of the existential quantifiers in a formula we want to realize. We use the same notation $\{\}$ for "uninteresting" existential variables as that for computationally redundant universal variables. In his work on Uniform Heyting Arithmetic [3], Berger independently proposed a similar refinement.

Definition 3.9 [Computationally irrelevant existential variables] Let us define an extension of the logic $\mathbf{M_1}$ by adding formulas of the form $\{\exists x\}.\, \phi$, and the corresponding introduction and elimination rules:

$(\exists^+)$ $\langle \{t\}, d^{\phi\,[t/x]} \rangle^{\{\exists x\}.\,\phi}$

$(\exists^-)$ $[e^{\{\exists x\}.\,\phi}, u^{\phi}.d^{\psi}]^{\psi}$

(provided $x \notin \mathrm{FV}(\psi)$, $x \notin \mathrm{FV}(\chi)$ for every $v^{\chi} \in \mathrm{FA}(d) \setminus \{u^{\phi}\}$ and $x \notin \mathrm{CV}(d)$)

Here the set of computationally relevant variables extends the previous definition in the following way:

$$\begin{aligned} \mathrm{CV}([e^{\{\exists x\}.\,\phi}, u^{\phi}.d^{\psi}]) &:= \mathrm{CV}(d) \cup \mathrm{CV}(e) \\ \mathrm{CV}(\langle \{t\}, d^{\phi\,[t/x]} \rangle) &:= \mathrm{CV}(d) \end{aligned}$$

The type of realizers for these new formulas is defined as $\tau(\{\exists x\}.\,\phi) := \tau(\phi)$. Furthermore, $r \,\mathsf{mr}\, \{\exists x\}.\,\phi := \exists x.\, r \,\mathsf{mr}\, \phi$ (with $x \notin \mathrm{FV}(r)$), and

$$\begin{aligned} [\![\langle \{t\}, d \rangle]\!] &:= [\![d]\!] \\ [\![[e^{\{\exists x\}.\,\phi}, u.d]]\!] &:= [\![d]\!]\,[[\![e]\!]/x_u] \end{aligned}$$

Example 3.10 The formula $(\{\exists x\}.\,P(x)) \to \exists x.\,P(x)$ is not provable: intuitively, the witness for the succedent is exactly the one provided by the proof of the antecedent of the implication, and since we do not want to know what that witness is, we also cannot produce a witness for the succedent (in other words, the witness for $\{\exists x\}.\,P(x)$ is local to the proof of this formula).

On the other hand, the formula $(\exists x.\,P(x)) \to \{\exists x\}.\,P(x)$ is provable, but it does not have any computational content if P is a Harrop formula; otherwise the extracted program is a function that only "forgets" the existential witness.

The proof of soundness for modified realizability (Theorem 2.3) can be extended to handle the additional cases.

3.3.2 Formalizing the proof

Having introduced the necessary refinement, we are now in a position to redefine the reducibility predicate in the following way (making the existential variables computationally irrelevant):

$$\begin{aligned} \mathbf{Rd}^*_0(t,s) &:= t = s \\ \mathbf{Rd}^*_{n+1}(t,s) &:= \{\exists r\}.\,\mathbf{Rd}(t,r) \wedge \mathbf{Rd}^*_n(r,s) \end{aligned}$$

We can see that the type of realizers for $\mathbf{Rd}^*_n(t,s)$ is now the unit type.

As remarked before, the logical relation used in the proof is defined as in the call-by-name case, except that now it can be refined—the universal variable becomes computationally redundant under call by value (we announce it in advance, but this observation can only be made after we actually write down the proof):

$$\begin{aligned} \mathbf{R}_b(t) &:= \exists v.\mathbf{Ev}_n(t,v) \\ \mathbf{R}_{T_1 \to T_2}(t) &:= \exists v.\mathbf{Ev}_n(t,v) \wedge \{\forall s\}.\,\mathbf{R}_{T_1}(s) \to \mathbf{R}_{T_2}(\mathtt{APP}(t,s)) \end{aligned}$$

For simplicity, we omit the parameterization of $\mathbf{R}_T$ by natural numbers, induced by the definition of the predicate $\mathbf{Ev}_n$.

The call-by-value analog of Lemma 3.5 is stated and proved in the same way:

Lemma 3.11 $\{\forall t\}.\, \mathbf{R}_T(t) \to \exists v.\, \mathbf{Ev}_n(t, v)$.

In order to prove the call-by-value version of Theorem 3.8 we need a few more properties of evaluation, stated in Lemmas 3.12-3.15.

Lemma 3.12 $\{\forall st\}.\, \mathbf{Rd}(s,t) \to \mathbf{R}_T(t) \to \mathbf{R}_T(s)$.

Proof. Induction on types, using the following property: $\{\forall stv\}.\, \mathbf{Rd}(s,t) \to \mathbf{Rd}^*_n(t,v) \to \mathbf{Rd}^*_{n+1}(s,v)$, which itself is proved by induction on n. □

Lemma 3.13 $\{\forall st\}.\, \mathbf{Rd}^*_n(s,t) \to \mathbf{R}_T(t) \to \mathbf{R}_T(s)$.

Proof. Induction on n, using Lemma 3.12. □

Lemma 3.14 $\{\forall stv\}.\, \mathbf{Rd}^*_n(s,v) \to \mathbf{V}(t) \to \mathbf{Rd}^*_n(\mathtt{APP}(t,s), \mathtt{APP}(t,v))$.

Proof. Induction on n. □

Lemma 3.15 $\{\forall st\}.\, \mathbf{Rd}^*_n(s,t) \to \mathbf{R}_T(s) \to \mathbf{R}_T(t)$, *where* $m \geq n$.

Proof. Induction on n, using the following properties:

(i) $\{\forall rst\}.\, \mathbf{Rd}^*_{n+1}(t,s) \to \mathbf{Rd}(t,r) \to \mathbf{Rd}^*_n(r,s)$

(ii) $\{\forall st\}.\, \mathbf{Rd}(t,s) \to \mathbf{R}_T(t) \to \mathbf{R}_T(s)$

The proof of Property i requires an additional axiom expressing determinism of the reduction relation:

$$(Det)\ \{\forall rst\}.\, \mathbf{Rd}(t,r) \to \mathbf{Rd}(t,s) \to r = s.$$

□

The call-by-value analog of Lemma 3.7 is stated just as before, and its proof—which we omit for lack of space—relies on Lemmas 3.12-3.15:

Lemma 3.16 *For any term t of type T, with* $\mathrm{FV}(t) = \{x_1, \ldots, x_n\}$, *and for any n-tuple of closed terms $\vec{r} = r_1, \ldots, r_n$ of types T_i such that $\mathbf{R}_{T_i}(r_i)$ holds for all $1 \leq i \leq n$, we have*

$$\mathbf{R}_T(t\,[\vec{r}/\vec{x}]).$$

The main theorem is also stated and proved as before.

3.3.3 Extracted program

Again we see that the types $\tau(\mathbf{R}_T(t))$ are independent of t. They describe the domains of a glueing model as follows:

$$\begin{aligned}\tau(\mathbf{R}_b) &:= \Lambda_b \\ \tau(\mathbf{R}_{T_1\to T_2}) &:= \Lambda_{T_1\to T_2} \times (\tau(\mathbf{R}_{T_1}) \to \tau(\mathbf{R}_{T_2}))\end{aligned}$$

Similarly to the previous case, the program we obtain for call by value is the composition of the term extracted from Lemma 3.11 (the same as for call by name), and the one extracted from Lemma 3.16, which looks as follows:

$$\begin{aligned}
\mathrm{eval}_{\,\mathtt{VAR}\,x_i,\vec{x}} &= \lambda\vec{t}\vec{u}.u_i\\
\mathrm{eval}_{\,\mathtt{APP}(r,s),\vec{x}} &= \lambda\vec{t}\vec{u}.\mathrm{snd}(\mathrm{eval}_{\,r,\vec{x}}\,\vec{t}\vec{u})\,(\mathrm{eval}_{\,s,\vec{x}}\,\vec{t}\vec{u})\\
\mathrm{eval}_{\,\mathtt{LAM}\,x_{n+1}^{T_1}.\,t^{T_2},\vec{x}} &= \lambda\vec{t}\vec{u}.((\mathtt{LAM}\,x_{n+1}.\,t)\,[\vec{t}/\vec{x}],\lambda v.\mathrm{eval}_{\,t,\vec{x}x_{n+1}}\,(\vec{t}(\downarrow_{T_1} v))(\vec{u}v))
\end{aligned}$$

This program also threads two environments, but the first of them (represented by the vector $\vec{t}$) contains already evaluated terms. As before, for every closed term t^{Λ_T}, $\mathrm{eval}_{\,t,\varepsilon}$ denotes the glueing model interpretation of the object-level term denoted by t^{Λ_T}.

Remark 3.17 In the original formulation of Lemma 3.16 in Pierce's book [12, p. 151], the terms to be substituted for free variables in a given term were required to be values. This restriction, however, is not necessary for the proof to go through, and the resulting program is exactly the same as the one obtained here.

3.4 Weak head normalization for closed terms of base type

We now show a variant of the proof of weak head normalization where we are only interested in evaluating terms of base type. In order to be able to observe the behavior of programs, we extend the object language with integers, formed with the zero constant 0 and the successor constant S in the usual way. The set of base types now includes the type ι for integers. As mentioned before, for call-by-name evaluation we can simplify the definition of the relation $\mathbf{R}_T$, which consequently leads to a simpler extracted program that we will show next.

We add the following two axioms specifying the evaluation strategy for the new terms:

$$\begin{aligned}
&(A_5)\ \mathbf{Ev}(\mathtt{0},\mathtt{0})\\
&(A_6)\ \forall tv.\,\mathbf{Ev}(t,v)\to\mathbf{Ev}(\mathtt{S}\,t,\mathtt{S}\,v)
\end{aligned}$$

The definition of the logical relation is now less restrictive for higher types:

$$\begin{aligned}
\mathbf{R}_b(t) &:= \exists v.\mathbf{Ev}(t,v)\\
\mathbf{R}_{T_1\to T_2}(t) &:= \{\forall s\}.\,\mathbf{R}_{T_1}(s)\to\mathbf{R}_{T_2}(\mathtt{APP}(t,s))
\end{aligned}$$

Theorem 3.18 *For any closed term t of type ι, $\exists v.\mathbf{Ev}(t,v)$ holds.*

The proof is carried out almost as before, and it relies on the base-type version of Lemma 3.5, on Lemma 3.6 as before, and on the base-type counterpart of Lemma 3.7 which now reads as follows (note that the vector of terms $\vec{r}$ is now computationally redundant):

Lemma 3.19 *For any term t of type T, with $\mathrm{FV}(t)=\{x_1,\ldots,x_n\}$,*

$$\{\forall\vec{r}\}.\,(\mathbf{R}_{T_1}(r_1)\wedge\ldots\wedge\mathbf{R}_{T_n}(r_n))\to\mathbf{R}_T(t\,[\vec{r}/\vec{x}]).$$

For the proof of Lemma 3.19 we need to show that $\mathbf{R}_\iota(\mathtt{0})$ holds, and that for any term t of type ι, $\mathbf{R}_\iota(t)\to\mathbf{R}_\iota(\mathtt{S}\,t)$ holds.

Remark 3.20 The proof does not go through if we use the call-by-value axiomatization instead of call by name; this is due to the fact that in the proof of the main lemma, in the case for abstraction, we must know that an arbitrary term of an arbitrary type evaluates to a value. However, with the weakened definition of the relation $\mathbf{R}_T$ we cannot prove this fact any more.

The program extracted from the proof looks as follows:

$$\begin{aligned}
\mathrm{eval}_{\,\mathtt{0},\vec{x}} &= \lambda\vec{u}.\mathtt{0}\\
\mathrm{eval}_{\,\mathtt{S}\,t,\vec{x}} &= \lambda\vec{u}.\mathtt{S}\,(\mathrm{eval}_{\,t,\vec{x}}\,\vec{u})\\
\mathrm{eval}_{\,\mathtt{VAR}\,x_i,\vec{x}} &= \lambda\vec{u}.u_i\\
\mathrm{eval}_{\,\mathtt{APP}(r,s),\vec{x}} &= \lambda\vec{u}.(\mathrm{eval}_{\,r,\vec{x}}\,\vec{u})\,(\mathrm{eval}_{\,s,\vec{x}}\,\vec{u})\\
\mathrm{eval}_{\,\mathtt{LAM}\,x_{n+1}.\,t,\vec{x}} &= \lambda\vec{u}.\lambda v.\mathrm{eval}_{\,t,\vec{x}x_{n+1}}\,\vec{u}v
\end{aligned}$$

Acknowledgments:

We are grateful to Dariusz Biernacki, Philipp Gerhardy, and the MFPS reviewers for their comprehensive feedback. Special thanks to Andrzej Filinski for two rounds of insightful comments.

This work is partially supported by the ESPRIT Working Group APPSEM II (http://www.appsem.org) and by the Danish Natural Science Research Council, Grant no. 21-03-0545.

A Implementation

This appendix contains an ML implementation of the normalization programs from Sections 3.2.2 and 3.3.3. The implementation ignores the dependencies in the definition of the object-level terms and the semantic domains:

```
type ide = string

datatype term = VAR of ide
              | APP of term * term
              | LAM of ide * term
```

The ML programs work by optimistically trying to interpret an untyped object-level term (defined in the data type `term` just above) into a semantic domain defined by a reflexive type (see the data type `R` below for call by name and call by value). However, as stressed by Filinski [9,10], it is a non-trivial task to prove that such implementations are correct.

We use the following auxiliary functions, whose definitions are omitted:

```
subst_all : term * (ide * term) list -> term
lookup : ''a * (''a * 'b) list -> 'b
```

The function `subst_all` implements simultaneous substitution of terms for variables. The function `lookup` implements a standard association-list lookup.

A.1 Call by name

```
datatype R = BASE of term
           | ARROW of term * (term -> R -> R)

fun reify (BASE t)
    = t
  | reify (ARROW (t, f))
    = t

fun eval (VAR x, ts, us)
    = lookup (x, us)
  | eval (APP (t1, t2), ts, us)
    = let val ARROW (_, f) = eval (t1, ts, us)
      in f (subst_all (t2, ts)) (eval (t2, ts, us))
      end
  | eval (LAM (y, t1), ts, us)
    = let val t = subst_all (LAM (y, t1), ts)
          val f = fn s => fn u => eval (t1, (y, s) :: ts, (y, u) :: us)
      in ARROW (t, f)
      end

fun normalize t
    = reify (eval (t, [], []))
```

A.2 Call by value

```
datatype R = BASE of term
           | ARROW of term * (R -> R)

fun reify (BASE t)
    = t
  | reify (ARROW (t, f))
    = t

fun eval (VAR x, ts, us)
    = lookup (x, us)
  | eval (APP (t1, t2), ts, us)
    = let val ARROW (_, f) = eval (t1, ts, us)
      in f (eval (t2, ts, us))
      end
  | eval (LAM (y, t1), ts, us)
    = let val t = subst_all (LAM (y, t1), ts)
          val f = fn u => eval (t1, (y, reify u) :: ts, (y, u) :: us)
      in ARROW (t, f)
      end

fun normalize t
    = reify (eval (t, [], []))
```

References

[1] Barendregt, H., "The Lambda Calculus: Its Syntax and Semantics," Studies in Logic and the Foundation of Mathematics **103**, North-Holland, 1984, revised edition.

[2] Berger, U., *Program extraction from normalization proofs*, in: M. Bezem and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science (1993), pp. 91–106.

[3] Berger, U., *Uniform Heyting arithmetic*, Annals of Pure and Applied Logic **133** (2005), pp. 125–148.

[4] Berger, U., W. Buchholz and H. Schwichtenberg, *Refined program extraction from classical proofs*, Annals of Pure and Applied Logic **114** (2002), pp. 3–25.

[5] Berger, U. and H. Schwichtenberg, *An inverse of the evaluation functional for typed λ-calculus*, in: G. Kahn, editor, *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science* (1991), pp. 203–211.

[6] Biernacka, M., O. Danvy and K. Støvring, *Extracting evaluators from proofs of weak head normalization*, Research Report BRICS RS-05-12, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark (2005), extended version of an article to appear in the preliminary proceedings of MFPS XXI, Birmingham, UK, May 2005.

[7] Coquand, T. and P. Dybjer, *Intuitionistic model constructions and normalization proofs*, in: *International Workshop TYPES'93*, Nijmegen, The Netherlands, 1993, available at http://www.cs.chalmers.se/~peterd/papers/nbe.html.

[8] Coquand, T. and P. Dybjer, *Intuitionistic model constructions and normalization proofs*, Mathematical Structures in Computer Science **7** (1997), pp. 75–94.

[9] Dybjer, P. and A. Filinski, *Normalization and partial evaluation*, in: G. Barthe, P. Dybjer, L. Pinto and J. Saraiva, editors, *Applied Semantics – Advanced Lectures*, number 2395 in Lecture Notes in Computer Science (2000), pp. 137–192.

[10] Filinski, A. and H. K. Rohde, *A denotational account of untyped normalization by evaluation*, in: I. Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International Conference, FOSSACS 2004*, number 2987 in Lecture Notes in Computer Science (2002), pp. 167–181.

[11] Kreisel, G., *Interpretation of analysis by means of functionals of finite type*, in: *Constructivity in Mathematics, Proc. Colloq. Amsterdam (1957)*, Studies in Logic and the Foundation of Mathematics (1959), pp. 101–128.

[12] Pierce, B. C., "Types and Programming Languages," The MIT Press, 2002.

[13] Plotkin, G. D., *Call-by-name, call-by-value and the λ-calculus*, Theoretical Computer Science **1** (1975), pp. 125–159.

[14] Troelstra, A. S., editor, "Metamathematical Investigation of Intuitionistic Arithmetic and Analysis," Lecture Notes in Mathematics **344**, Springer-Verlag, 1973.

Recent BRICS Report Series Publications