Proceedings

Sixth Annual IEEE Symposium on
LOGIC IN COMPUTER SCIENCE

July 15-18, 1991 Amsterdam, The Netherlands

Sponsored by
IEEE Technical Committee on
Mathematical Foundations of Computing
CWI, Amsterdam
Vrije Universiteit, Amsterdam

In cooperation with
Association for Computing Machinery
Association for Symbolic Logic
European Association for Theoretical Computer Science

1951-1991

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

Table of Contents

Session 5
Chair: P.-L. Curien

Session 6
Chair: R. Constable

Session 7
Chair: S. Ronchi Della Rocca

Session 8
Chair: S. Abiteboul

Session 9
Chair: J. Bergstra

Session 10
Chair: E. Shapiro

Session 11
Chair: K. Apt

Session 12
Chair: M. Sato

Session 13
Chair: U. Goltz

An Inverse of the Evaluation Functional for Typed λ-calculus

U. Berger

Mathematisches Institut
der LMU München
8000 München 2

H. Schwichtenberg

Mathematisches Institut
der LMU München
8000 München 2

Abstract

In any model of typed λ-calculus contaning some basic arithmetic, a functional p−e *(procedure → expression) will be defined which inverts the evaluation functional for typed λ-terms. Combined with the evaluation functional,* p−e *yields an efficient normalization algorithm. The method is extended to λ-calculi with constants and is used to normalize (the λ-representations of) natural deduction proofs of (higher order) arithmetic. A consequence of theoretical interest is a strong completeness theorem for βη-reduction, generalizing results of Friedman [1] and Statman [3]: If two λ-terms have the same value in some model containing representations of the primitive recursive functions (of level 1) then they are provably equal in the βη-calculus.*

0 Introduction

Normalization is a fundamental but expensive process in proof theory and proof implementation. In view of the Curry–Howard correspondence it is natural to try to use evaluation of typed λ–terms, which corresponds to normalization of proofs and is available in functional programming languages, to get rid of the burden of implementing a normalization procedure 'by hand'. However in trying so, one is faced with two problems: 1. Terms containing free variables are not accepted by the compiler. 2. If the term is of functional type, the normalized procedure is not shown but only a message that the result is some procedural object.

Although both problems seem to be implementation dependent, they may be formulated purely mathematically, using denotational semantics. The denotational value of a term of functional type is a functional (or procedure) which is an abstract object and therefore cannot be shown on the screen. But to solve the second problem we do not need the procedure itself but a term in normal form evaluating to it. So, it is our our task to define a functional p−e inverting the evaluation functional and returning normal forms only. The first problem asks for an environment binding every free variable x of the term to be normalized to some functional which we will call $\mathrm{mse}(x)$ (make self evaluating) for reasons (perhaps) becoming appearent when looking at the definition below.

Let us briefly discuss a first guess how p−e and mse might be defined: Because p−e should return terms, it is clear that a model M in which p−e exists must contain (representations of) λ-terms. Therefore let, for simplicity, M^o be the set of all typed λ-terms and $M^{\rho\to\sigma} = (M^\sigma)^{M^\rho}$. Define $\mathrm{p{-}e}_{\rho\to\sigma} \in M^{\rho\to\sigma}$ and $\mathrm{mse}_\rho \in M^{o\to\rho}$ simultaneously by

$$\mathrm{p{-}e}_o(r) = \mathrm{mse}_o(r) = r$$

$$\mathrm{p{-}e}_{\rho\to\sigma}(a) = \lambda z^\rho.\mathrm{p{-}e}_\sigma(a(\mathrm{mse}_\rho(z^\rho)))$$

$$\mathrm{mse}_{\rho\to\sigma}(r)(b) = \mathrm{mse}_\sigma(r(\mathrm{p{-}e}_\rho(b)))$$

In the definition of $\mathrm{p{-}e}_{\rho\to\sigma}(a)$ the bound variable z^ρ must be 'fresh', i.e. whenever a is the value of a term r then z^ρ must not occur free in r, where the evaluation of r takes place in an environment binding every free variable x to $\mathrm{mse}(x)$. Under this assumption it is not hard to show that p−e does the job.

But how can we find a fresh z?

Being faced with this problem for the first time, we were working with the LISP dialect SCHEME which is a functional language but provides also some procedural facilities. Therefore at the computer it was no problem to produce such a z. We simply used the SCHEME procedure **gensym** creating a new symbol every time it is called (such a procedure may be easily defined in any procedural language).

It is the aim of this work to solve the inversion problem purely functionally, i.e. inside the theory of typed λ–calculus, avoiding procedural elements. This will allow us to prove some interesting syntactical and semantical properties of typed λ–calculus. The most remarkable one is the following strong completeness theorem:

> If two closed typed λ-terms have the same value in some model allowing the representation of primitive recursive functions, then they are already provably equal in the βη-calculus.

This generalizes Friedmans 'extended completeness theorem' [1]. Friedman requires the models to consist

of the full (set theoretical) function space at all functional types whereas in our theorem only for types of level 1 conditions on the model are imposed.

To achieve our aim we will introduce in the Sections 2 and 3 a renaming and coding machinery for λ–terms in a large class of models. These models will be called admissible. The point is that single terms are replaced by families of α–equivalent terms for which the the 'fresh z' is very easy to compute. In Section 4 we define the inversion functional for admissible models and prove the completeness theorem. In Section 5 we construct a specific model where this functional yields a normalization algorithm which is as efficient as the one based on **gensym**. In Section 6 we extend the results of Section 4 to λ-calculi with constants. Finally in Section 7 we discuss, as an example of such an extended λ-calculus, the $\rightarrow \forall$-fragment of (higher order) logic and arithmetic and show how to normalize proofs with our method.

1 Models of typed λ–calculus

Types are built up from ground types by $\rightarrow$. It will suffice to consider only one ground type o. λ–terms are constructed from typed variables x^ρ by application $(t^{\rho\rightarrow\sigma}r^\rho)^\rho$ and abstraction $(\lambda x^\rho.s^\sigma)^{\rho\rightarrow\sigma}$. $\Lambda^{(\rho)}$ is the set of terms (of type ρ). We will frequently omit types and parentheses if they can be recovered from the context. Iterated applications are associated to the left and application binds more than abstraction. For example $\lambda x.rst$ stands for $\lambda x.((rs)t)$.

As for the notion of a model we follow Friedman [1] with some change in notation. A *pre-structure* M consists of a set M^ρ, mappings $A_{\rho,\sigma}: M^{\rho\rightarrow\sigma} \times M^\rho \rightarrow M^\sigma$ and equivalence relations $=_\rho$ on M^ρ which are congruences for the $A_{\rho,\sigma}$, i.e.

$$a =_{\rho\rightarrow\sigma} a', b =_\rho b' \Rightarrow A_{\rho,\sigma}ab =_\sigma A_{\rho,\sigma}a'b'$$

Furthermore we require extensionality

$$\forall b \in M^\rho A_{\rho,\sigma}ab =_\sigma A_{\rho,\sigma}a'b \Rightarrow a =_{\rho\rightarrow\sigma} a'$$

Hence all the $=_\sigma$ are completely determined by $=_o$ because

$$a =_{\rho\rightarrow\sigma} a' \Leftrightarrow \forall b \in M^\rho A_{\rho,\sigma}ab =_\sigma A_{\rho,\sigma}a'b$$

If $b, b' \in M^\rho$ then $b = b'$ will always mean $b =_\rho b'$. Furthermore we will write ab for $A_{\rho,\sigma}ab$ and will again associate to the left.

To define models we need environments which are type respecting mappings from the variables to M. Let ENV be the set of environments. For every environment η, variable x^ρ and $a \in M^\rho$ the environment $\eta[x^\rho \mapsto a]$ is defined by $\eta[x^\rho \mapsto a](x^\rho) = a$ and $\eta[x^\rho \mapsto a](y) = \eta(y)$ if $y \neq x^\rho$.

A *λ-model* is a pre–structure M together with mappings $|\cdot|^\rho\cdot : \Lambda^\rho \times \mathrm{ENV} \rightarrow M^\rho$ s.t. (omitting types)

$$|x|\eta = \eta(x), |tr|\eta = |t|\eta|r|\eta, |\lambda x.s|\eta a = |s|\eta[x \mapsto a].$$

We write $M \models_\eta r = s$ if in M the equation $|r|\eta = |s|\eta$ holds. $M \models r = s :\Leftrightarrow \forall \eta: M \models_\eta r = s$. Common λ–models are the full set–theoretic models T_B where $\mathrm{T}^o_B = B$ is any set and $\mathrm{T}^{\rho\rightarrow\sigma}_B = \mathrm{T}^\rho_B{}^{\mathrm{T}^\sigma_B}$ [1], domain–theoretic models D where D^0 is a domain and $D^{\rho\rightarrow\sigma}$ is the set of continuous functions from D^ρ to D^σ or the Kleene–Kreisel functionals. A nice constructive λ–model is the structure HEO of hereditarily effective operations where HEO^ρ consist of natural numbers and $A_{\rho,\sigma}en = \{e\}n$ [4].

2 Normal forms and α–equality

β–reduction, based on β–conversion $(\lambda x.r)s \mapsto r[s/x]$, and β–normalforms are defined as usual. We will prefer *long β–normalforms* which have the form

$$\lambda x_1^{\rho_1} \ldots \lambda x_n^{\rho_n}.y^{\sigma_1\rightarrow\cdots\rightarrow\sigma_k\rightarrow o}s_1^{\sigma_1} \ldots s_k^{\sigma_k}$$

with $s_1, \ldots, s_k$ in long β–normalform ($\sigma_1 \rightarrow \ldots \rightarrow \sigma_k \rightarrow o$ is associated to the right) [3, §0]. Obviously each term in β–normalform may be transformed into long β–normalform by suitable η–expansions. Therefore each term r may be transformed into a unique long β–normalform r^* by β–conversions and η–expansions.

It is common (and was done above) to identify α–equal terms i.e. terms interconvertible by bound renaming. However, for the algorithms we will define, concrete λ–terms with specific bound variables are needed. For every finite 0–1 sequence k let $\alpha_k r$ be the result of replacing each bound variable y^σ in r with binding position $l \in \{0,1\}^*$ by x^ρ_{k*l} ($*$ denotes concatenation). We can make this precise by defining inductively

$$\alpha_k x = x$$

$$\alpha_k rs = (\alpha_{k*0}r)(\alpha_{k*1}s)$$

$$\alpha_k(\lambda y^\rho.r) = \lambda x^\rho_k.\alpha_{k*0}(r[x^\rho_k/y^\rho])$$

Lemma 1.

(a) $r =_\alpha s \Rightarrow \alpha_k r = \alpha_k s$.

(b) If r contains no free variables of the form x_{k*l} then $\alpha_k r =_\alpha r$.

Proof: Induction on the length of r. □

By this lemma, for every term $r \in \Lambda^\rho$, the term family $\mathrm{EMB}r: \{0,1\}^* \rightarrow \Lambda^\rho$, defined by

$$\mathrm{EMB}rk := \alpha_k(r)$$

may be viewed as a representation of r (by (b)) which abstracts from α–equality (by (a)). (b) also tells us how to recover an α–variant of r from $\mathrm{EMB}r$: Take $\mathrm{EMB}r\hat{k}$ where $\hat{k}$ is such that there is no index in the finite set $\{l \in \{0,1\}^* | x_l \text{ free in } r\}$ extending k. This set can be computed easily because a variable in r is free in r iff it is not changed in $\mathrm{EMB}rk$ when k

changes. Therefore we only need to determine all l s.t. x_l appears in $\mathrm{EMB}r\epsilon$ and $\mathrm{EMB}r0$ at the same position (ϵ is the empty sequence, 0 stands for the sequence containing only 0). In fact $\hat{k}$ only depends on $\mathrm{EMB}r$ and is well defined not only for $\mathrm{EMB}r$ but for any term family $F\colon\{0,1\}^* \to \Lambda$. Therefore we may define $\mathrm{EXTRACT}(F) := \mathrm{EMB}F\hat{k}$. Let us summarize some properties of $\mathrm{EMB}\colon\Lambda \to \Lambda^{\{0,1\}^*}$ and $\mathrm{EXTRACT}\colon\Lambda^{\{0,1\}^*} \to \Lambda$.

Lemma 2.

(a) $r =_\alpha s \Rightarrow \mathrm{EMB}(r) = \mathrm{EMB}(s)$.

(b) $\mathrm{EXTRACT}(\mathrm{EMB}r) =_\alpha r$.

(c) $\mathrm{EMB}(\mathrm{EMB}rk)k = \mathrm{EMB}rk$.

Proof: (a) : "$\Rightarrow$" is Lemma 1(a) and "$\Leftarrow$" follows from (b) of this lemma. (b) follows from Lemma 1(b) and the definition of EXTRACT. (c) is proved by induction on r. □

3 Admissible λ–models and pr–models

Now we tackle the problem of inverting the evaluation functional $|\cdot|$. Because the final solution will be rather technical, it might be helpful to sketch the algorithms in a particular model where things are simple. Suppose M is a λ–model s.t. M^o contains syntactical material such as indices (0–1 sequences) and λ–terms and assume that all usual syntactic operations exist in M. In particular $\mathrm{EMB} \in M^{o\to o}$ and $\mathrm{EXTRACT} \in M^{(o\to o)\to o}$ (We assume that $M^{\rho\to\sigma}$ consists of set theoretic functions). The inversion functional will be based on functionals $\Phi_\rho \in M^{\rho\to(o\to o)}$ and $\Psi_\rho \in M^{(o\to o)\to\rho}$ defined by

$$\Phi_o r = \mathrm{EMB}r$$

$$\Psi_o f = \mathrm{EXTRACT}f$$

$$\Phi_{\rho\to\sigma} ak = \lambda x_k^\rho.\Phi_\rho(a(\Psi_\rho(\mathrm{EMB}x_k^\rho)))(k*0)$$

$$\Psi_{\rho\to\sigma} fa = (\text{f-APP}f(\Phi_\rho a))$$

where f-APP is application for term families i.e. $\text{f-APP}(\mathrm{EMB}r)(\mathrm{EMB}s) = \mathrm{EMB}(rs)$ should hold. The crucial property of Φ is

- If r is a closed term in long β–normalform then $\Phi|r| = \mathrm{EMB}r$.

Now we define $\mathrm{p\text{–}e}_\rho\colon M^\rho \to \Lambda^\rho$ by

$$\mathrm{p\text{–}e}_\rho(a) = \mathrm{EXTRACT}(\Phi_\rho a).$$

This works because if $a \in M_\rho$ is λ–definable then there is a term r in long β–normalform s.t. $|r| = a$ and consequently

$$|\mathrm{p\text{–}e}_\rho(a)| = |\mathrm{EXTRACT}(\mathrm{EMB}r)| = |r| = a,$$

i.e. $\mathrm{p\text{–}e}_\rho(a)$ is a term with value a. Furthermore, for closed $r \in \Lambda^\rho$,

$$\mathrm{norm}(r) := \mathrm{p\text{–}e}_\rho(|r|)$$

is the long β–normalform of r, because $|r| = |r^*|$ and therefore

$$\begin{aligned} & \mathrm{norm}(r) \\ = \ & \mathrm{EXTRACT}(\Phi_\rho|r^*|) \\ = \ & \mathrm{EXTRACT}(\mathrm{EMB}r^*) \\ =_\alpha \ & r^* \end{aligned}$$

which means that $\mathrm{norm}(r)$ is (as an α–variant of r^*) the long β–normalform of r as well. A third consequence is a completeness result for $\beta\eta$–reduction: Let $r, s \in \Lambda^\rho$ be closed with $|r| = |s|$. Then $\mathrm{norm}(r) = \mathrm{p\text{–}e}_\rho(|r|) = \mathrm{p\text{–}e}_\rho(|s|) = \mathrm{norm}(s)$ and therefore r and s are provably equal in the $\beta\eta$–calculus.

To prove the stated property of Φ for closed terms, one has to show a more general statement involving also Ψ and open terms. For any substitution θ define $\eta_\theta \in \mathrm{ENV}$ by $\eta_\theta(y^\rho) = \Psi_\rho(\mathrm{EMB}(\theta y^\rho))$. Now it is an easy but very instructive exercise to show (by induction on r distinguishing the cases $\rho = o$ and $\rho \neq o$) that the equation

$$\Phi(|r|\eta_\theta) = \mathrm{EMB}(r\theta)$$

holds for every term r in long β–normalform. A detailed proof of this equation in a more general framework will be carried out in Lemma 5 in the next section.

Now we will precisely describe the requirements on a λ–model making these constructions possible. In fact we will describe a more general situation which will strengthen our completeness result and allows for the definition of an efficient normalization procedure.

Call a λ–model *admissible* if there are types f (type of term families) and ι (type of indices), coding functions $\lceil\cdot\rceil^o\colon\Lambda \to M^o$, $\lceil\cdot\rceil^\iota\colon\{0,1\}^* \to M^\iota$ and objects $\mathrm{append} \in M^{\iota\to\iota\to\iota}$, $\mathrm{mvar}_\rho \in M^{\iota\to o}$, $\mathrm{app} \in M^{o\to o\to o}$, $\mathrm{abst}_\rho \in M^{\iota\to o\to o}$ (for every type ρ), $\mathrm{emb} \in M^{o\to\mathrm{f}}$, $\mathrm{fun} \in M^{\mathrm{f}\to(\iota\to o)}$ and $\mathrm{extract} \in M^{(\iota\to o)\to o}$ s.t. the following holds (omitting types)

(append) $\mathrm{append}\lceil k\rceil\lceil l\rceil = \lceil k*l\rceil$

(mvar) $\mathrm{mvar}_\rho\lceil k\rceil = \lceil x_k^\rho\rceil$

(app) $\mathrm{app}\lceil r\rceil\lceil s\rceil = \lceil rs\rceil$

(abst) $\mathrm{abst}_\rho\lceil k\rceil\lceil r\rceil = \lceil\lambda x_k^\rho.r\rceil$

($\lceil\cdot\rceil$) $\lceil r\rceil = \lceil s\rceil \Rightarrow \mathrm{EMB}r = \mathrm{EMB}s$

(emb) $\mathrm{EMB}r = \mathrm{EMB}s \Rightarrow \mathrm{emb}\lceil r\rceil = \mathrm{emb}\lceil s\rceil$

(fun) $\mathrm{fun}(\mathrm{emb}\lceil r\rceil)\lceil k\rceil = \lceil\mathrm{EMB}rk\rceil$

(extract) $\forall k\colon a\lceil k\rceil = \lceil\mathrm{EMB}rk\rceil$
$\Rightarrow \mathrm{extract}(a) = \lceil\mathrm{EXTRACT}(\mathrm{EMB}r)\rceil$

All infinite λ–models mentioned in Section 1 are easily seen to be admissible by letting $\iota = o$ and $\mathrm{f} = o \to o$.

Because the definition given above is not very comprehensible, we will define a more natural, smaller but still very large class of λ–models, showing that admissibility is in fact a very weak requirement.

Call a λ–model M a *pr–model* if all binary primitive recursive functions are represented in it. This shall mean that there is an injection $\nu : \omega \to M^o$ and for every binary primitive recursive function $h : \omega^2 \to \omega$ an object $\overline{h} \in M^{o \to o \to o}$ s.t. $\overline{h}(\nu n)(\nu m) = \nu(hnm)$ for all $n, m \in \omega$. In fact in a pr–model all primitive recursive functions are represented because every primitive recursive function is explicitely definable from binary ones. However we will only need that in pr–models all unary and binary primitive recursive functions are represented.

Lemma 3. *Every pr–model is admissible.*

Proof: Let $r_1, r_2, \ldots$ resp. $k_1, k_2, \ldots$ be effective repitition free numberings of Λ resp. $\{0,1\}^*$. 'Effective' shall mean that there are primitive recursive functions APPEND, MVAR_ρ, APP, ABST, EXEMB and ALPHA s.t.

$$k_{\mathrm{APPEND}(n,m)} = k_n * k_m,$$

$$r_{\mathrm{MVAR}_\rho(n)} = x^\rho_{k_n}$$

$$r_{\mathrm{APP}(n,m)} = r_n r_m$$

$$k_{\mathrm{ABST}_\rho(n,m)} = \lambda x^\rho_{k_n} r_m$$

$$r_{\mathrm{EXEMB}(n)} = \mathrm{EXTRACT}(\mathrm{EMB}r_n)$$

$$r_{\mathrm{ALPHA}(n,m)} = \mathrm{EMB}r_n k_m$$

In the previous section we defined $\mathrm{EXTRACT}(\mathrm{EMB}r) = \mathrm{EMB}r\hat{k}$ where $\hat{k}$ can be computed from $\mathrm{EMB}r\epsilon$ and $\mathrm{EMB}r0$. Hence we may assume furthermore that there exists a primitive recursive function INDEXT s.t. if $\mathrm{EMB}r\epsilon = r_n$ and $\mathrm{EMB}r0 = r_m$ then

$$\mathrm{EXTRACT}(\mathrm{EMB}r) = \mathrm{EMB}r k_{\mathrm{INDEXT}(n,m)}$$

Now we turn M into an admissible model by letting $\iota = o$, $\mathrm{f} = o$, $\ulcorner r_n \urcorner^o = \nu n$, $\ulcorner k_n \urcorner^\iota = \nu n$, $\mathrm{append} = \overline{\mathrm{APPEND}}$, $\mathrm{mvar}_\rho = \overline{\mathrm{MVAR}_\rho}$, $\mathrm{app} = \overline{\mathrm{APP}}$, $\mathrm{abst}_\rho = \overline{\mathrm{ABST}_\rho}$, $\mathrm{emb} = \overline{\mathrm{EXEMB}}$, $\mathrm{fun} = \overline{\mathrm{ALPHA}}$ and define $\mathrm{extract} \in M^{(\iota \to o) \to o}$ by

$$\mathrm{extract}(a) = a(\overline{\mathrm{INDEXT}}(a\ulcorner\epsilon\urcorner)(a\ulcorner 0\urcorner))$$

extract exists in M because it is explicitely defined from elements of M and M is a λ–model. We have to verify the laws for admissibility. (append), (mvar), (app), (abst), ($\ulcorner\cdot\urcorner$) and (emb) clearly hold.
(fun):

$$\begin{aligned}
&\mathrm{fun}(\mathrm{emb}\ulcorner r_n\urcorner)\ulcorner k_m\urcorner \\
=\ & \overline{\mathrm{ALPHA}}(\overline{\mathrm{EXEMB}}\nu n)(\nu m) \\
=\ & \nu(\mathrm{ALPHA}(\mathrm{EXEMB}n)m) \\
=\ & \ulcorner r_{\mathrm{ALPHA}(\mathrm{EXEMB}n)m}\urcorner \\
=\ & \ulcorner \mathrm{EMB}r_{\mathrm{EXEMB}n} k_m\urcorner \\
=\ & \ulcorner \mathrm{EMB}(\mathrm{EXTRACT}(\mathrm{EMB}r_n))k_m\urcorner \\
=\ & \ulcorner \mathrm{EMB}r_n k_m\urcorner
\end{aligned}$$

by Lemma 2 (a) and (b).
(extract): Let $\mathrm{EMB}r\epsilon = r_n$ and $\mathrm{EMB}r0 = r_m$ and assume $a\ulcorner k\urcorner = \ulcorner \mathrm{EMB}rk\urcorner$ for all $k \in \{0,1\}^*$. Then in particular $a\ulcorner\epsilon\urcorner = \ulcorner \mathrm{EMB}r\epsilon\urcorner = \ulcorner r_n\urcorner = \nu n$ and likewise $\ulcorner a0\urcorner = \nu m$. Hence

$$\begin{aligned}
&\mathrm{extract}(a) \\
=\ & a(\overline{\mathrm{INDEXT}}(a\ulcorner\epsilon\urcorner)(a\ulcorner o\urcorner)) \\
=\ & a(\overline{\mathrm{INDEXT}}(\nu n)(\nu m)) \\
=\ & a(\nu(\mathrm{INDEXT}nm)) \\
=\ & a\ulcorner k_{\mathrm{INDEXT}(n,m)}\urcorner \\
=\ & \ulcorner \mathrm{EMB}r k_{\mathrm{INDEXT}(n,m)}\urcorner \\
=\ & \ulcorner \mathrm{EXTRACT}(\mathrm{EMB}r)\urcorner
\end{aligned}$$

using the assumption again. □

To improve the readability of the calculations in the next section we introduce some auxiliary objects. Let M be an admissible model and define $\text{f-extract} \in M^{\mathrm{f} \to o}$, $\text{f-app} \in M^{\mathrm{f} \to \mathrm{f} \to \mathrm{f}}$ and $\mathrm{fam} \in M^{(\iota \to o) \to \mathrm{f}}$ by

$$\text{f-extract}f = \mathrm{extract}(\mathrm{fun}f)$$

$$\text{f-app}fg = \mathrm{emb}(\mathrm{app}(\text{f-extract}f)(\text{f-extract}g))$$

$$\mathrm{fam}(a) = \mathrm{emb}(\mathrm{extract}(a))$$

Lemma 4. *In every admissible λ–model M the following equations hold:*

(a) $\text{f-extract}(\mathrm{emb}\ulcorner r\urcorner) = \ulcorner \mathrm{EXTRACT}(\mathrm{EMB}r)\urcorner$

(b) $\mathrm{emb}(\text{f-extract}(\mathrm{emb}\ulcorner r\urcorner)) = \mathrm{emb}\ulcorner r\urcorner$

(c) $\text{f-app}(\mathrm{emb}\ulcorner r\urcorner)(\mathrm{emb}\ulcorner s\urcorner) = \mathrm{emb}\ulcorner rs\urcorner$

(d) $\forall k : a\ulcorner k\urcorner = \ulcorner \mathrm{EMB}rk\urcorner \Rightarrow \mathrm{fam}(a) = \mathrm{emb}\ulcorner r\urcorner$

Proof:
(a): $\text{f-extract}(\mathrm{emb}\ulcorner r\urcorner) = \mathrm{extract}(\mathrm{fun}(\mathrm{emb}\ulcorner r\urcorner))$. By (fun), $\mathrm{fun}(\mathrm{emb}\ulcorner r\urcorner)\ulcorner k\urcorner = \ulcorner \mathrm{EMB}rk\urcorner$ for all k and hence, by (extract),

$$\mathrm{extract}(\mathrm{fun}(\mathrm{emb}\ulcorner r\urcorner)) = \ulcorner \mathrm{EXTRACT}(\mathrm{EMB}r)\urcorner$$

(b): $\mathrm{EMB}(\mathrm{EXTRACT}(\mathrm{EMB}r)) = \mathrm{EMB}r$, by Lemma 2. Hence, by (emb), $\mathrm{emb}\ulcorner \mathrm{EXTRACT}(\mathrm{EMB}r)\urcorner = \mathrm{emb}\ulcorner r\urcorner$. Therefore, by (a), $\mathrm{emb}(\text{f-extract}(\mathrm{emb}\ulcorner r\urcorner)) =$

$\mathrm{emb}\lceil r\rceil$.
(c): By (a), (app) and (emb), we have

$$\begin{aligned} & \text{f-app}(\mathrm{emb}\lceil r\rceil)(\mathrm{emb}\lceil s\rceil) \\ = \ & \mathrm{emb}(\mathrm{app}(\text{f-extract}(\mathrm{emb}\lceil r\rceil))(\text{f-extract}(\mathrm{emb}\lceil s\rceil))) \\ = \ & \mathrm{emb}\lceil(\mathrm{EXTRACT}(\mathrm{EMB}r))(\mathrm{EXTRACT}(\mathrm{EMB}s))\rceil \\ = \ & \mathrm{emb}\lceil rs\rceil \end{aligned}$$

(d): If $a\lceil k\rceil = \lceil \mathrm{EMB}rk\rceil$ for all k then, by (extract), $\mathrm{extract}(a) = \lceil \mathrm{EXTRACT}(\mathrm{EMB}r)\rceil$. Hence

$$\mathrm{fam}(a) = \mathrm{emb}(\mathrm{extract}(a)) = \mathrm{emb}\lceil r\rceil$$

by (emb). □

4 Inversion, normalization and completeness

Let M be an admissible λ-model. By recursion on ρ we define terms $\Phi_\rho \in \Lambda^{\rho\to\mathrm{f}}$ and $\Psi_\rho \in \Lambda^{\mathrm{f}\to\rho}$:

$\Phi_o = \mathrm{emb}$,

$\Psi_o = \text{f-extract}$,

$\Phi_{\rho\to\sigma} = \lambda a^{\rho\to\sigma}.\mathrm{fam}(\lambda p^\iota.\mathrm{abst}p(\mathrm{fun}(\Phi_\sigma(a\psi[p]))(p0)))$

where $\psi[p]$ stands for $\Psi_\rho(\mathrm{emb}(\mathrm{mvar}_\rho p))$ and $p0$ is short for $\mathrm{append}p\lceil 0\rceil$.

$\Psi_{\rho\to\sigma} = \lambda f^{\mathrm{f}}.\lambda a^\rho.\Psi_\sigma(\text{f-app}f(\Phi_\rho a))$.

Here of course $\mathrm{emb},\ldots,\text{f-app}$ denote constants, i.e. variables with the valuations $\mathrm{emb},\ldots,\text{f-app} \in M$ respectively. As with $\mathrm{emb},\ldots,\text{f-app}$ the values $|\Phi_\rho| \in M^{\rho\to\mathrm{f}}$ resp. $|\Psi_\rho| \in M^{\mathrm{f}\to\sigma}$ are denoted by Φ_ρ resp. Ψ_ρ again. Furthermore for $f, g \in M^{\mathrm{f}}$ we will write fg for $\text{f-app}fg$ and $f_1 f_2 \ldots f_n$ will be associated to the left. Clearly for $\rho = \rho_1 \to \ldots \to \rho_n \to o$ and $a_1 \in M^{\rho_1}, \ldots, a_n \in M^{\rho_n}$

$$\Psi_\rho f a_1 \ldots a_n = \text{f-extract}(f(\Phi_{\rho_1} a_1)(\Phi_{\rho_n} a_n))$$

For any substitution θ define $\eta_\theta \in \mathrm{ENV}$ by $\eta_\theta(y^\rho) = \Psi(\mathrm{emb}\lceil\theta y^\rho\rceil)$.

Lemma 5 (Main Lemma). *For every λ-term $r \in \Lambda^\rho$ in long β-normalform and every substitution θ*

$$\Phi_\rho(|r|\eta_\theta) = \mathrm{emb}(\lceil r\theta\rceil)$$

Moreover if $\rho = o$ then $|r|\eta_\theta = \lceil r\theta\rceil$.

Proof: Induction on r. Recall that fg stands for $\text{f-app}fg$.

1. $\rho = o$, $r = x^{\rho_1\to\ldots\to\rho_n\to o} s_1^{\rho_1} \ldots s_n^{\rho_n}$:

$$\begin{aligned} & |xs_1\ldots s_n|\eta_\theta \\ = \ & \Psi(\mathrm{emb}\lceil\theta x\rceil)|s_1|\eta\ldots|s_n|\eta \\ = \ & \text{f-extract}((\mathrm{emb}\lceil x\theta\rceil)(\Phi|s_1|\eta_\theta)\ldots(\Phi|s_n|\eta_\theta)) \\ = \ & \text{f-extract}((\mathrm{emb}\lceil x\theta\rceil)(\mathrm{emb}\lceil s_1\theta\rceil)\ldots(\mathrm{emb}\lceil s_n\theta\rceil)) \\ = \ & \text{f-extract}(\mathrm{emb}\lceil(xs_1\ldots s_n)\theta\rceil) \\ = \ & \lceil\mathrm{EXTRACT}(\mathrm{EMB}r\theta)\rceil \end{aligned}$$

But $\mathrm{EXTRACT}(\mathrm{EMB}r\theta) =_\alpha r\theta$, by Lemma 2(c) and since substitution is determined only up to α-equivalence, we may write $\mathrm{EXTRACT}(\mathrm{EMB}r\theta) = r\theta$.

2. $\rho \to \sigma$, $r = \lambda x^\rho.s^\sigma$ (w.l.o.g. $\theta x^\rho = x^\rho$ and x^ρ not free in θy for any $y \neq x^\rho$ free in s^σ):
$\Phi_{\rho\to\sigma}(|\lambda x.s|\eta_\theta) = \mathrm{fam}(a)$, where $a \in M^{\iota\to o}$ is s.t. (using the same abbreviations as in the definition of $\Phi_{\rho\to\sigma}$)

$$ap = \mathrm{abst}_\rho p(\mathrm{fun}(\Phi_\sigma(|s|\eta_\theta[x \mapsto \psi[p]]))(p0))$$

for all $p \in M^\iota$. We have to show $\mathrm{fam}(a) = \mathrm{emb}\lceil(\lambda x.s)\theta\rceil$. By virtue of lemma 4 (d) it will suffice to show $a\lceil k\rceil = \lceil\mathrm{EMB}((\lambda x.s)\theta)k\rceil$ for all $k \in \{0,1\}^*$:

$$\begin{aligned} & a\lceil k\rceil \\ = \ & \mathrm{abst}_\rho\lceil k\rceil(\mathrm{fun}(\Phi_\sigma(|s|\eta_\theta[x \mapsto \psi[\lceil k\rceil]]))(\lceil k\rceil * \lceil 0\rceil)) \\ = \ & \mathrm{abst}_\rho\lceil k\rceil(\mathrm{fun}(\Phi_\sigma(|s|\eta_{\theta[x\mapsto x_k^\rho]}))\lceil k * 0\rceil) \\ = \ & \mathrm{abst}_\rho\lceil k\rceil(\mathrm{fun}(\mathrm{emb}\lceil s(\theta[x \mapsto x_k^\rho])\rceil)\lceil k * 0\rceil) \\ = \ & \mathrm{abst}_\rho\lceil k\rceil\lceil\mathrm{EMB}(s(\theta[x \mapsto x_k]))(k * 0)\rceil \\ = \ & \lceil\lambda x_k^\rho.\mathrm{EMB}(s\theta[x_k/x])(k * 0)\rceil \\ = \ & \lceil\mathrm{EMB}(\lambda x.(s\theta))k\rceil \end{aligned}$$

Because $\lambda x.(s\theta) = (\lambda x.s)\theta$ we are done. □

For every $b \in \lceil\Lambda\rceil$ let $b^\sim \in \Lambda$ be any term s.t. $\lceil b^\sim\rceil = b$. By (emb), we then have $\lceil r\rceil^\sim =_\alpha r$ for every $r \in \Lambda$. Now define partial functions $\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho : M^\rho \to \Lambda^\rho$ by

$$\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a) \simeq (\text{f-extract}\Phi_\rho(a))^\sim$$

Call $a \in M^\rho$ λ-definable if there is a closed term $r \in \Lambda^\rho$ s.t. $|r| = a$.

Theorem 1. *Let M be an admissible λ-model.*

1. **Inversion:** *If $a \in M^\rho$ is λ-definable then $\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a)$ is a closed term of type ρ s.t. $|\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a)| = a$.*

2. **Normalization and substitution:** *For every term $r \in \Lambda^\rho$ and every substitution θ*

$$\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(|r|\eta_\theta) =_\alpha r^*\theta$$

3. **Completeness:** *If $M \models r = s$ then r and s are provably equal in the $\beta\eta$-calculus.*

Proof: 1.: Let $a = |r|$ with $r \in \Lambda^\rho$ closed and in long β-normalform. By Lemma 5 and Lemma 4(a)

$$\begin{aligned} & \text{f-extract}(\Phi_\rho a) \\ = \ & \text{f-extract}(\Phi_\rho|r|) \\ = \ & \text{f-extract}(\mathrm{emb}\lceil r\rceil) \\ = \ & \lceil\mathrm{EXTRACT}(\mathrm{EMB}r)\rceil \end{aligned}$$

Hence $\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a) = \lceil\mathrm{EXTRACT}(\mathrm{EMB}r)\rceil^\sim =_\alpha r$.

2.:

$$\begin{aligned} & \text{f-extract}(\Phi_\rho|r|\eta_\theta) \\ = \ & \text{f-extract}(\Phi_\rho|r^*|\eta_\theta) \\ = \ & \text{f-extract}(\text{emb}\lceil r^*\theta\rceil) \\ = \ & \lceil \text{EXTRACT}(\text{EMB}(r^*\theta))\rceil \end{aligned}$$

Hence $\text{p}\!\rightharpoonup\!\text{e}_\rho(|r|\eta_\theta) = \lceil \text{EXTRACT}(\text{EMB}r^*\theta)\rceil =_\alpha r^*\theta$.
3.: If $M \models r = s$ then $|r|\eta_{\text{id}} = |s|\eta_{\text{id}}$ and therefore, by 2., $r^* =_\alpha s^*$. □

Let us reformulate this theorem in terms of the more natural pr–models (see Section 3). Recall that with every pr–model M we associated an injection $\nu\colon\omega \to M^o$ and that $\lceil r\rceil = \nu n$ whenever $r = r_n$ in the (fixed) effective, repetition free numbering $r_1, r_2, \ldots$ of Λ. We call $\lceil r\rceil$ the *canonical code* of r in M.

Inversion and normalization theorem

In a pr–model M there exists for every ρ an object $\text{inv}_\rho \in M^{\rho\to o}$ s.t., for every λ–definable $a \in M^\rho$, $\text{inv}_\rho a$ is the canonical code of a closed term $r \in \Lambda^\rho$ in long β–normalform s.t. $|r| = a$. Hence, for every term s of type ρ, $\text{inv}_\rho|s|$ is the canonical code of the long β–normalform of s.

Proof: By Lemma 3, M is admissible. Hence, by the previous theorem, we may define inv_ρ by $\text{inv}_\rho a = \text{f-extract}(\Phi_\rho a)$. We could define even simpler $\text{inv}_\rho a = \Phi_\rho a$ because in pr–models $\text{f-extract}(\text{emb}\lceil r\rceil) = \text{emb}\lceil r\rceil$ and therefore, for closed $r \in \Lambda^\rho$, $\text{f-extract}(\Phi_\rho|r|) = \text{f-extract}(\text{emb}\lceil r\rceil) = \text{emb}\lceil r\rceil = \Phi_\rho|r|$. □

Completeness Theorem

If r and s are λ–terms of type ρ s.t. there exists a pr–model M in which $M \models r = s$ holds, then r and s are provably equal in the $\beta\eta$–calculus.

Proof: By Lemma 5 and Theorem 1.3. □

Note that the conditions on a λ–model for being a pr–model refer only to the types o and $o \to o \to o$. For higher types nothing is required. Thus this completenss result is considerably stronger than that obtained by Friedman [1, Theorem 3] because there the models are required to be full type structures T_B over an infinite set B. Statman proves another completeness theorem [3, Theorem 2]: For every closed λ–term r there is a finite set E, such that for all closed s, $\text{T}_E \models r = s \Leftrightarrow r =_{\beta\eta} s$. Let us briefly indicate how this may be generalized by our method: Call a λ–model admissible below a natural number n, if it has the same properties as an admissible model but everything is restricted to indices of length $< n$ and terms of length $< n$ built up from n variables. If any syntactical operation exeeds this finite set of indices and terms, then an error element should be retourned. Clearly this determines an increasing sequence of finite sets of terms Λ_n having Λ as their union s.t. for every model M admissible below n and every closed r in long normal form, $\text{p}\!\rightharpoonup\!\text{e}_M(|r|) = r$ if $r \in \Lambda_n$ and $\text{p}\!\rightharpoonup\!\text{e}_M(|r|) =$ error otherwise. Now let M be admissible below n and let r be closed s.t. $r^* \in \Lambda_n$. Then for every closed s s.t. $M \models r = s$ we have $\text{p}\!\rightharpoonup\!\text{e}_M(|s|) = r^* \neq$ error and hence $s^* \in \Lambda_n$ and $r^* = s^*$. Because clearly for every n there is a finite set E s.t. T_E is admissible below n we have generalized Statmans theorem.

The completeness theorem may be formulated equivalently as follows: In every pr–model M and every type $\rho = \rho_1 \to \ldots \to \rho_n \to o$

$$\forall r, s \in \Lambda^\rho(r \neq_{\beta\eta} s \Rightarrow \exists \eta \in \text{ENV}, \vec{a} : |r|\eta\vec{a} \neq |s|\eta\vec{a})$$

holds where η and $\vec{a} := a_1, \ldots, a_n \in M^{\rho_1} \times \ldots \times M^{\rho_n}$ may depend on r and s. But in fact η and $\vec{a}$ may be chosen independently from r and s because we can take $\eta = \eta_{\text{id}}$ and $a_i = \Psi_{\rho_i}(\text{emb}(x^{\rho_i}_{\underbrace{0\ldots0}_{i-1}}))$. Therefore the stronger formula

$$\exists \eta \in \text{ENV} \exists \vec{a} \forall r, s \in \Lambda^\rho(r \neq_{\beta\eta} s \Rightarrow |r|\eta\vec{a} \neq |s|\eta\vec{a})$$

is true in every pr–model.

5 An efficient normalization algorithm

So far we were studying the *theoretical* consequences of our inversion method. Now we will discuss the question if our results also are of *practical* interest. In particular we will examine the normalization procedure defined in the previous section. This procedure is based on Φ and Ψ and hence on EMB and EXTRACT which are rather expensive operations. Now if we work in the naive model described in Section 3, emb = EMB and extract = f-extract = EXTRACT and hence we have not gained very much on the computational side. In pr–models the algorithms are even worse. Therefore, if we want an efficient normalization procedure, we must look for a more sophisticated model where emb and extract are as simple as possible. At this point the generality of admissible models pays out: We can find an admissible λ–model where emb = extract = identity. Before we define such a model, let us look how Φ and Ψ may then be computed.

Assume M to be an admissible λ–model s.t. $\text{f} = o$, $M^o = \Lambda^{\{0,1\}^*}$, $M^{\rho\to\sigma} = (M^\rho)^{M^\sigma}$, $A_{\rho,\sigma}$ = function application, $\lceil r\rceil^o = \text{EMB}r$ and emb = f-extract = identity. Then $\Phi_o = \Psi_o =$ identity and by Lemma 5 for $r = \lambda x.s \in \Lambda^{\rho\to\sigma}$ in long β–normalform, any substitution θ and any $k \in \{0,1\}^*$

$$\begin{aligned} & \Phi_{\rho\to\sigma}(|r|\eta_\theta)k \\ = \ & \text{emb}(\lceil r\theta\rceil)k \\ = \ & \text{EMB}(\lambda x.s(\theta[x \mapsto x]))k \\ = \ & \lambda x^\rho_k.\text{EMB}(s(\theta[x \mapsto x_k]))(k*0) \\ = \ & \lambda x^\rho_k.\Phi_\rho(|s|\eta_{\theta[x\mapsto x_k]})(k*0) \\ = \ & \lambda x^\rho_k.\Phi_\rho(|s|\eta_\theta[x \mapsto \Psi_\rho(\text{emb}\lceil x_k\rceil)])(k*0) \\ = \ & \lambda x^\rho_k.\Phi_\rho(|s|\eta_\theta(\Psi_\rho(\text{EMB}x_k)))(k*0) \end{aligned}$$

i.e. for all $a \in M^{\rho\to\sigma}$ of the form $a = |r|\eta_\theta$

$$\Phi_{\rho\to\sigma} ak = \lambda x^\rho_k.\Phi_\sigma(a(\Psi_\rho(\mathrm{EMB}x_k)))(k*0).$$

Furthermore for $t \in \Lambda^{\rho\to\sigma}$ and $r \in \Lambda^\rho$, defining $\text{f-APP}fgk = f(k*0)g(k*1)$,

$$\begin{aligned}
&\Psi_{\rho\to\sigma}(\mathrm{EMB}t)|r|\eta_\theta\\
=\ &\Psi_\sigma(\text{f-app}(\mathrm{EMB}t)|r|\eta_\theta)\\
=\ &\Psi_\sigma(\text{f-app}(\mathrm{emb}\lceil t\rceil)(\mathrm{emb}\lceil r\theta\rceil))\\
=\ &\Psi_\sigma(\mathrm{emb}\lceil t(r\theta)\rceil)\\
=\ &\Psi_\sigma(\mathrm{EMB}(t(r\theta)))\\
=\ &\Psi_\sigma(\text{f-APP}(\mathrm{EMB}t)(\mathrm{EMB}(r\theta)))\\
=\ &\Psi_\sigma(\text{f-APP}(\mathrm{EMB}t)(\Phi_\rho|r|\eta_\theta))
\end{aligned}$$

i.e. for $f \in \mathrm{EMB}(\Lambda^{\rho\to\sigma})$ and b of the form $b = |r|\eta_\theta$ we have

$$\Psi_{\rho\to\sigma} fb = \Psi(\text{f-APP}f(\Phi_\rho b)).$$

These very easy recursion equations for Φ and Ψ suggest to *define* $\overline{\Phi}_\rho \in M^{\rho\to o}$ and $\overline{\Psi}_\rho \in M^{\rho\to o}$ by

$$\overline{\Phi}_o f = \overline{\Psi}_o f = f$$

$$\overline{\Phi}_{\rho\to\sigma} ak = \lambda x^\rho_k.\overline{\Phi}_\rho(a(\overline{\Psi}_\rho(\mathrm{EMB}x_k)))(k*0)$$

$$\overline{\Psi}_{\rho\to\sigma} fb = \overline{\Psi}_\sigma(\text{f-APP}f(\overline{\Phi}_\rho b))$$

for *all* $f \in M^o$, $a \in M^{\rho\to\sigma}$ and $b \in M^\rho$.

Now it can be shown that for $r \in \Lambda^\rho$ in long β-normalform again the equation $\overline{\Phi}_\rho(|r|\overline{\eta}_\theta) = \mathrm{EMB}(r\theta)$ holds, where $\overline{\eta}_\theta(x^\rho) = \overline{\Psi}_\rho(\mathrm{EMB}x^\rho)$, with a proof much simpler than the proof for Φ in Lemma 5.
Thus for $r \in \Lambda^\rho$ containing no variables x_k free we get an efficient normalization algorithm by defining

$$\mathrm{norm}(r) = \overline{\Phi}_\rho(|r|\overline{\eta}_{\mathrm{id}})\epsilon.$$

The hard part in the computation of $\mathrm{norm}(r)$ is to compute $|r|\overline{\eta}_{\mathrm{id}}$ which is done by the compiler of a functional programming language. Therefore, this normalization procedure will be as efficient as the compiler is.

Note that the definitions of $\overline{\Phi}$ and $\overline{\Psi}$ are external, i.e. $\overline{\Phi}$ and $\overline{\Psi}$ are not defined as the values in M of certain λ-terms. The definition takes place in a different λ-model $\overline{M}$ over ground types o and ι with $\overline{M^o} = \Lambda$, $\overline{M^\iota} = \{0,1\}^*$ and $\overline{M}^{\rho\to\sigma} = (\overline{M}^\sigma)^{\overline{M}^\rho}$ (Hence $M^\rho = \overline{M}^{\rho[\iota\to o/o]}$).

It remains to define an admissible λ-model M with the properties postulated at the beginning of this section: $M^o = \Lambda^{\{0,1\}^*}$, $M^{\rho\to\sigma} = (M^\rho)^{M^\rho}$, $A_{\rho,\sigma}$ = function application, $\iota = \mathrm{f} = o$, $\lceil r\rceil^o = \mathrm{EMB}r$, $\lceil k\rceil = \mathrm{EMB}x^o_k$. For the definition of the remaining functions we use the auxiliary function $\tilde{\ }^\iota : M^o \to \{0,1\}^*$ defined by $\tilde{F}^\iota = k$ if $F\epsilon = x^o_k$, otherwise $\tilde{F}^\iota =$ anything. Now define $\mathrm{append}FG = \lceil\tilde{F}^\iota * \tilde{G}^\iota\rceil^\iota$, $\mathrm{mvar}_\rho F = \mathrm{EMB}(x^\rho_{\tilde{F}^\iota})$, app = f-APP, $\mathrm{abst}_\rho FG = \mathrm{EMB}(\lambda x^\rho_{\tilde{F}^\iota}.\mathrm{EXTRACT}(G))$, $\mathrm{emb}F = F$, $\mathrm{fun}FG = \mathrm{EMB}(F\tilde{G}^\iota)$, $\mathrm{extract}ak = a\lceil k\rceil k$.
Let's verify the laws for admissibility: (append), (mvar), (app), (abst), ($\lceil\cdot\rceil$) and (emb) clearly hold.
(fun): $\mathrm{fun}(\mathrm{emb}\lceil r\rceil)\lceil k\rceil = \mathrm{EMB}(\mathrm{emb}(\mathrm{EMB}r)\widetilde{\lceil k\rceil}^\iota) = \lceil\mathrm{EMB}rk\rceil$.
(extract): If $\lceil ak\rceil = \lceil\mathrm{EMB}rk\rceil$ then $\mathrm{extract}ak = a\lceil k\rceil k = \lceil\mathrm{EMB}rk\rceil k = \mathrm{EMB}(\mathrm{EMB}rk)k = \mathrm{EMB}rk$ by Lemma 2 (c). Because by assumption this is true for all k, we get $\mathrm{extract}a = \mathrm{EMB}r = \mathrm{EMB}(\mathrm{EXTRACT}(\mathrm{EMB}r)) = \lceil\mathrm{EXTRACT}(\mathrm{EMB}r)\rceil$.

6 Integrating constants

In this section we will extend our method from pure terms to terms containing constants. Suppose we are given a set $\mathcal{C}$ of typed constants equipped with an operational semantics $\mathrm{op}_\mathcal{C}$ given by *$\mathcal{C}$-conversions* $cr_1 \dots r_n \mapsto s$ for several $c \in \mathcal{C}$. These, together with β-conversion and η-expansion, induce a reduction relation $\to_\mathcal{C}$ on the set $\Lambda_\mathcal{C}$ of λ-terms possibly containing constants from $\mathcal{C}$. Call $r \in \Lambda_\mathcal{C}$ in *$\mathcal{C}$-normal form* if it is in normal form with respect to $\to_\mathcal{C}$, i.e., r is in long β-normal form and doesn't contain $\mathcal{C}$-convertible subterms. $\mathrm{op}_\mathcal{C}$ is *weakly normalizing* iff every term $\Lambda_\mathcal{C}$ is reducible by $\to_\mathcal{C}$, β-reduction and η-expansion to a term in $\mathcal{C}$-normalform.

A $\mathcal{C}$-model is a λ-model M together with an interpretation $c_M \in M^\rho$ for every constant $c \in \mathcal{C}$ of type ρ. This extends to an interpretation $|r|\eta \in \mathcal{C}$ for all $r \in \Lambda_\mathcal{C}$ and all $\eta \in \mathrm{ENV}$ in the obvious way. A $\mathcal{C}$-model M is called *admissible for* $\mathrm{op}_\mathcal{C}$ iff it is admissible (where of course $\lceil r\rceil$ has to be defined for all $r \in \Lambda_\mathcal{C}$) and for all substitution θ the following holds:

(C1) If $r \mapsto s$ by a $\mathcal{C}$-conversion then $|r|\eta_\theta = |s|\eta_\theta$.

(C2) For all terms of type o of the form $cs_1 \dots s_k$ (c a constant) in $\mathcal{C}$-normal form

$$|cs_1 \dots s_n| = \Psi(\mathrm{emb}\lceil c\rceil)|s_1|\eta_\theta \dots |s_n|\eta_\theta.$$

Now we may extend the results of Section 4:

Lemma 6 (Extended Main Lemma). *Let M be a $\mathcal{C}$-model admissible for $\mathrm{op}_\mathcal{C}$. Then for every $r \in \Lambda^\rho_\mathcal{C}$ in $\mathcal{C}$-normal form and every substitution θ*

$$\Phi_\rho(|r|\eta_\theta) = \mathrm{emb}(\lceil r\theta\rceil)$$

Moreover if $\rho = o$ then $|r|\eta_\theta = \lceil r\theta\rceil$.

Proof: Copy the proof of Lemma 5. There is only one additional case, namely when $\rho = o$ and r has the form $cs_1 \dots s_n$. But then (C2) applies. □

Theorem 2. *Let $\mathrm{op}_{\mathcal{C}}$ be weakly normalizing and let M be a $\mathcal{C}$-model admissible for $\mathrm{op}_{\mathcal{C}}$.*

1. **Inversion:** *If $a \in M^\rho$ is λ-definable from $\mathcal{C}$ then $\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a)$ is a closed term in $\Lambda^\rho_{\mathcal{C}}$ s.t. $|\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(a)| = a$.*
2. **Normalization and confluence:** *The reduction relation $\rightarrow_{\mathcal{C}}$ is confluent. In particular every term $r \in \Lambda^\rho_{\mathcal{C}}$ reduces to a unique $\mathcal{C}$-normal form r^* which my be computed by*
$$\mathrm{p}\!\rightarrowtail\!\mathrm{e}_\rho(|r|\eta_{\mathrm{id}}) = r^*$$
3. **Completeness:** *If $M \models r = s$ then r and s are provably equal in the $\beta\eta\mathcal{C}$-calculus.*

Proof: Clearly, by (C1), $|r|\eta_\theta = |s|\eta_\theta$ if $r \rightarrow_{\mathcal{C}} s$. Hence, if $r \rightarrow^*_{\mathcal{C}} s$ with an s in $\mathcal{C}$-normal form, then $|r|\eta_\theta = |s|\eta_\theta$. Now we may prove the theorem in the same manner as Theorem 1, using Lemma 6 instead of Lemma 5. □

7 Normalization of proofs

By the Curry–Howard correspondence every proof in the $\rightarrow \forall$-fragment of Gentzens natural deduction calculus may be represented as a typed λ-term. The type of a term (derivation) d^φ is the formula φ being derived, variables correspond to assumptions and λ-abstraction resp. application correspond to the introduction resp. elimination rules for $\rightarrow$ and $\forall$. This correspondence is the basis for the use of proofs as programs, where execution of programs is performed by normalizing proofs [2].

We now show how the normalization procedure derived in the previous sections may be used for the normalization of proofs. At first glance there seems to be a difficulty because we only considered terms typed by o and $\rho \rightarrow \sigma$ and not by arbitrary $\rightarrow \forall$-formulas. But note that the normal form of a typed term is completely determined by the underlying type free term (obtained by erasing all types). Therefore we only need to define for every $\rightarrow \forall$-formula φ a type $\tau(\varphi)$ (built up from o and $\rightarrow$) such that if d is a derivation then $\tau(d)$ is a wellformed typed term, where $\tau(d)$ is obtained from d by replacing all formulas φ by $\tau(\varphi)$. The definition of τ is obvious: $\tau(\pi) = o$ for atomic formulas π, $\tau(\varphi \rightarrow \psi) = \tau(\varphi) \rightarrow \tau(\psi)$ and $\tau(\forall x^\rho \varphi) = \tau(\rho) \rightarrow \tau(\varphi)$, where for types ρ over ground types other than o, $\tau(\rho)$ is defined by replacing all ground types by o. Hence a derivation d of a formula φ may be normalized by computing $\mathrm{p}\!\rightarrowtail\!\mathrm{e}_{\tau(\varphi)}(|\tau(d)|\eta_{\mathrm{id}})$. At least from this we obtain the type free term underlying the normal form of d. At the end of this section it is indicated how we can recover therefrom the complete normalized derivation.

One important point is still missing: In the $\forall$-elimination rule, which may be written as the term construction rule $(d^{\forall x \varphi} r)^{\varphi[r/x]}$ the derived formula is obtained by a substitution. Therefore if r and s are of functional type then $\varphi[r/x]$ may contain non-normal object terms which we may want to normalize. These substitutions and normalizations may be performed by our method if we represent formulas as λ-terms of a new ground type formula. Predicate symbols for predicates over objects of type $\rho_1, \ldots, \rho_n$ are then constants of type $\rho_1 \rightarrow \ldots \rightarrow \rho_n \rightarrow$ formula, implication is a constant of type formula $\rightarrow$ formula $\rightarrow$ formula and quantification over objects of type ρ is represented by a constant $\forall_\rho$ of type $(\rho \rightarrow \text{formula}) \rightarrow$ formula. $\forall x^\rho \varphi$ then stands for $\forall_\rho \lambda x^\rho . \varphi$. Now the normal form of $\varphi[r/x]$ may be computed by normalizing $(\lambda x.\varphi)r$. Since we have no conversion rules for the logical constants $\rightarrow$ and $\forall_\rho$, we have, by (C2) of the previous section, to interpret them by $\mathrm{mse}_{o\rightarrow o\rightarrow o}(\rightarrow)$ and $\mathrm{mse}_{(\tau(\rho)\rightarrow o)\rightarrow o}(\forall_\rho)$ respectively, where for $r \in \Lambda^\rho$ we define $\mathrm{mse}_\rho(r) = \Psi_\rho(\mathrm{emb}\lceil r \rceil)$ (make self evaluating, see introduction).

So far we have only considered pure logic. To normalize proofs in (higher order) arithmetic we have to deal with further constants: At the object level constants 0 for zero, $\sharp$t, $\sharp$f for the boolean objects, S for the successor and R_ρ for recursion operators of type
$$\mathrm{rec}(\rho) := \rho \rightarrow (\mathrm{nat} \rightarrow \rho \rightarrow \rho) \rightarrow \mathrm{nat} \rightarrow \rho.$$
At the proof level constants $\mathrm{I}^{\mathrm{nat}}_{\forall x \varphi(x)}$ for induction over the natural numbers
$$\mathrm{ind}(\forall x \varphi(x)) := \varphi(0) \rightarrow \forall x(\varphi(x) \rightarrow \varphi(\mathrm{S}x)) \rightarrow \forall x \varphi(x),$$
constants $\mathrm{I}^{\mathrm{boole}}_{\forall x \varphi(x)}$ for boolean induction (case analysis)
$$\varphi(\sharp\mathrm{t}) \rightarrow \varphi(\sharp\mathrm{f}) \rightarrow \forall x \varphi(x)$$
and a truth constant T of type atom($\sharp$t), where atom is a constant of type boole $\rightarrow$ formula, i.e., a predicate symbol (in fact it suffices to have atom as the only predicate symbol).

Note that we can define proofs of the 'stability axioms' $\neg\neg\varphi \rightarrow \varphi$ for every arithmetical formula φ by induction on φ ($\neg\varphi := \varphi \rightarrow \mathrm{atom}(\sharp\mathrm{f})$). For prime formulas atom(b), stability is proved by boolean induction. Hence, if $\exists$ and $\vee$ are defined as usual, we get full classical arithmetic.

In order to use arithmetical proofs as programs, the purely logical β-conversion has to be supplemented by conversion rules for the recursion and induction constants. With a recursion constant R_ρ we associate the usual conversions

$$\mathrm{R}_\rho rs0 \mapsto r$$

$$\mathrm{R}_\rho rs(\mathrm{S}t) \mapsto st(\mathrm{R}_\rho rst)$$

Similary for induction over the natual numbers $\mathrm{I}^{\mathrm{nat}}_{\forall x \varphi(x)}$ (omitting formulas)

$$\mathrm{I}^{\mathrm{nat}} de0 \mapsto d$$

$$\mathrm{I}^{\mathrm{nat}} de(\mathrm{S}t) \mapsto et(\mathrm{I}^{\mathrm{nat}} det)$$

and for case analysis $I^{boole}_{\forall x\varphi(x)}$

$$I^{boole} de\sharp t \mapsto d$$

$$I^{boole} de\sharp f \mapsto e$$

Note that $\tau(\mathrm{ind}(\forall x\varphi(x))) = \tau(\mathrm{rec}(\rho))$ provided $\tau(\varphi(x)) = \tau(\rho)$. Hence for the purpose of normalization, recursion and induction over the natural numbers may be identified.

By the results of the previous section we have to interpret the arithmetical constants in an admissible model M in such a way that (C1) and (C2) are satisfied. Hence we define $0_M = \mathrm{mse}_o(0)$ $(= \lceil 0 \rceil)$ etc., $S_M = \mathrm{mse}_{o\to o}(S)$ and for a recursion constant R_ρ, regarded as a constant of type $\tau(\mathrm{rec}(\rho))$, we define (ommitting types)

$$R_M ab\lceil 0 \rceil = a,$$

$$R_M ab\lceil St \rceil = b\lceil t \rceil (R_M ab\lceil t \rceil),$$

$$R_M abc = \mathrm{mseR}abc \quad \text{otherwise.}$$

Case analysis is interpreted similary. Of course we must have chosen a model M in which R_M exists (For the model defined in Section 5 this is certainly the case). The properties (C1) and (C2) are rather obvious. (C1) immediately follows from Lemma 6(a) (which doesn't use (C1)!). To prove (C2), Lemma 6(a) is needed too. But Lemma 6 in turn needs (C2). Therefore one has to prove (C2) and Lemma 6 simoultaneously by induction on the length of the term.

Now, because of the well known fact that the operative semantics defined by the conversions above is (even strongly) normalizing, we can apply Theorem 2.2 which gives us a normalization algorithm.

Remark: Our arithmetical system includes the calculus of primitive recursive functionals in all finite types (Gödels T, R-λ-calculus [1]). Because the full type structure over the naturals T_ω clearly is admissible, we can conclude from Theorem 2.3 that for closed primitive recursive terms (of finite type) $T_\omega \models r = s$ holds if and only if r and s have the same normal form. Since normal forms may be computed, this means that equality between prim. rec. terms in T_ω is decidable. This sharply contrasts Friedmans result that, for prim. rec. terms r, s, the relation $T_\omega \models r = s$ is complete Π^1_1 [1, Theorem 6]. The point is that the arithmetical constants are interpreted differently: In [1] they operate on the natural numbers as usual whereas we coded all (possibly open) terms into ω. Therefore $T_\omega \models r = s$ in our sense is much stronger than in the usual sense.

Presently we experiment with an implementation of proofs along these lines. There are two technical points in this implementation which are worth noticing: 1. In an implication elimination $(d^{\varphi\to\psi} e^{\chi})^{\psi}$ we don't require the formulas φ and χ to be identical. It suffices if they have α-identical normal forms. This means that a big deal of (in most cases) uninteresting equational reasoning is shifted from the proof into the evaluation mechaism of the programming language. 2. We don't save complete proofs but only three characteristic components: a list of the free object and assumption variables together with their types, the derived formula and the underlying type-free λ-term of the proof. If the proof is in normalform then the completely typed proof expression may be recovered from these data. This yields a very compact representation of proofs making it possible to handle even very large proofs. In [2] this has been done for proofs using the full strength of Peano arithmetic.

8 Summary and conclusions

It has been shown how to invert the evaluation functional for typed λ-calculus in a large class of models. The main consequences where an efficient normalization procedure for typed λ-terms and a completeness theorem for the $\beta\eta$-calculus. Furthermore we used the normalization procedure to normalize natural deduction proofs of higher order arithmetic.

We think that the normalization method presented here is an excellent example for the use of the abstract concept of functionals of higher type in constructing and analyzing concrete algorithms: The normalization algorithm takes and returns concrete objects (terms) but it uses the evaluation procedure which returns functionals in all higher types.

Our technique may be extended to infinite terms and terms containing full recursion, i.e., fixed point operators. We also plan to extend it to second order λ-calculus.

References

[1] H. FRIEDMAN, Equality between functionals, **Lecture Notes in Mathematics**(R. Parikh, Editor), vol. 453, Springer-Verlag, Berlin and New York, 1975, pp. 22-37.

[2] H. SCHWICHTENBERG, Proofs as Programs, Leeds: Proof theory '90 (P. Aczel, H. Simmons, Editors), 1991.

[3] R. STATMAN, Completeness, invariance and λ-definability, **The Journal of Symbolic Logic**, Vol. 47, no. 1, 1982, pp. 17-26.

[4] A. S. TROELSTRA, Metamathematical investigation of intuitionistic arithmetic and analysis, **Lecture Notes in Mathematics**, vol. 344, Springer-Verlag, Berlin and New York, 1973.