Normalization by evaluation

Ulrich Berger, Matthias Eberl and Helmut Schwichtenberg

Mathematisches Institut der Universität München

Abstract. We extend normalization by evaluation (first presented in [4]) from the pure typed λ-calculus to general higher type term rewrite systems. This work also gives a theoretical explanation of the normalization algorithm implemented in the MINLOG system.

1 Introduction

In interactive proof systems it is crucial to have a term rewriting machinery available, in order to ease the burden of equational reasoning. Quite often term rewriting can be reduced to normalization and therefore it is essential to implement normalization of terms efficiently. By the same token, one then can also effectively normalize whole proofs (which can be written as derivation terms, using the CURRY-HOWARD correspondence). Normalization is used when extracting terms from formal proofs. For an application concerning circuits cf. [12].

It is well known that implementing normalization of λ-terms in the usual recursive fashion is quite inefficient. However, it is possible to compute the long normal form of a λ-term by evaluating it in an appropriate model (cf. [4]). When using for that purpose the built-in evaluation mechanism of e.g. SCHEME (a pure LISP dialect) one obtains an amazingly fast algorithm called "normalization by evaluation". The essential idea is to find an inverse to evaluation, converting a semantic object into a syntactic term. This normalization procedure is used and tested in the proof system MINLOG developed in Munich (cf. [2]).

Obviously, for applications pure typed λ-terms are not sufficient, but it is necessary to have constants in it. These were not considered in [4], but will be treated in this paper.

Let us begin with a short explanation of the essence of the method for normalizing typed λ-terms by means of an evaluation procedure of some functional programming language such as SCHEME. For simplicity we return to the simplest case, simply typed λ-calculus without constants.

Simple types are built from ground types τ by $\rho \to \sigma$ (later also products $\rho \times \sigma$ will be included). The set Λ of terms is given by x^τ, $(\lambda x^\rho M^\sigma)^{\rho\to\sigma}$, $(M^{\rho\to\sigma} N^\rho)^\sigma$. The set LNF of terms in long normal form (i.e. normal w.r.t. β-reduction and η-expansion) is defined inductively by $(xM_1 \dots M_n)^\tau$, $\lambda x M$ (we abbreviate $xM_1 \dots M_n$ by $x\boldsymbol{M}$ and similar a list $M_1 \dots M_n$ by $\boldsymbol{M}$). By $\mathsf{lnf}(M)$ we denote the long normal form of M, i.e. the unique term in long normal form $\beta\eta$-equal to M.

Now we have to choose our model. A simple solution is to take terms of ground type as ground type objects, and all functions as possible function type objects:

$$[\![\tau]\!] := \Lambda_\tau, \quad [\![\rho \to \sigma]\!] := [\![\sigma]\!]^{[\![\rho]\!]} \quad \text{(the full function space)}.$$

It is crucial that all terms (of ground type) are present, not just the closed ones. Next we need an assignment $\uparrow$ lifting a variable to an object, and a function $\downarrow$ giving us a normal term from an object. They should meet the following condition, which might be called "correctness of normalization by evaluation":

$$\downarrow([\![M]\!]_\uparrow) = \mathsf{lnf}(M)$$

where $[\![M^\rho]\!]_\uparrow \in [\![\rho]\!]$ denotes the value of M under the assignment $\uparrow$. Two such functions $\downarrow$ and $\uparrow$ can be defined simultaneously, by induction on the type. It is convenient to define $\uparrow$ on all terms (not just on variables). Hence for every type ρ we define $\downarrow_\rho : [\![\rho]\!] \to \Lambda_\rho$ and $\uparrow_\rho : \Lambda_\rho \to [\![\rho]\!]$ (called reify and reflect) by

$$\begin{aligned} &\downarrow_\tau(M) := M, & &\uparrow_\tau(M) := M, \\ &\downarrow_{\rho\to\sigma}(a) := \lambda x \downarrow_\sigma(a(\uparrow_\rho(x))) \quad \text{"}x\text{ new"}, & &\uparrow_{\rho\to\sigma}(M)(a) := \uparrow_\sigma(M \downarrow_\rho(a)). \end{aligned}$$

Here a little difficulty appears: what does it mean that x is new? We will solve this problem by slightly modifying the model and defining $[\![\tau]\!]$ to be the set of families of terms of type τ (instead of single terms) and setting $\downarrow_{\rho\to\sigma}(a)(k) := \lambda x_k(\downarrow_\sigma(a(\uparrow_\rho(x_k^\infty)))(k{+}1))$, where x_k^∞ is the constant family x_k. The definition of $\uparrow_{\rho\to\sigma}$ has to be modified accordingly. This idea corresponds to a representation of terms in the style of de Bruijn [9]. An advantage of this approach is that we get the same normal form even if the terms are only equal up to renaming of bound variables.

The proof of correctness is easy: Since for the typed lambda calculus without constants we have preservation of values, i.e. $[\![M]\!]_\xi = [\![\mathsf{lnf}(M)]\!]_\xi$ for all terms M and environments ξ, we only have to verify $\downarrow([\![N]\!]_\uparrow) = N$ for normal terms N, which is straightforward. The situation is different when we add constants together with rewrite rules, since then preservation of values (in our model) is false in general. However, correctness of normalization by evaluation still holds, but needs to be proven by a different method.

The structure of the paper is as follows. In section 2 we present the simply typed λ-calculus with constants and pairing and give some examples of higher order rewrite systems. Then we inductively define a relation $M \longrightarrow Q$, with the intended meaning that M is normalizable with long normal form Q, and prove in section 4 the correctness of normalization by evaluation by showing that $M \longrightarrow Q$ implies $\downarrow([\![M]\!]_\uparrow) = Q$. Hence the mapping $M \mapsto \downarrow([\![M]\!]_\uparrow)$ is a normalization function. In order to define the semantics $[\![M]\!]$ of a term M properly we use domain theory. This is described briefly in section 3.

In subsection 4.2 we show how to interpret a constant c more efficiently if all its rules are of some special forms. In fact, most of the rewrite rules in the

MINLOG system have one of these forms. Again, normalization by evaluation is shown to remain correct.

The final section 5 contains a review of the relevant literature [1, 5, 7, 8], and also a comparison of the run-times of our algorithm with those coming from the other setups. The results clearly support our claim that the present approach (with function spaces as higher type domains) leads to a much faster implementation.

Acknowledgements. The present work has benefitted considerably from ideas of Felix Joachimski and Ralph Matthes concerning strategies for normalization proofs, including η-expansion and primitive recursion. In particular, the idea to employ the inductive definition of the relation $M \longrightarrow Q$ is essentially due to them. We also want to thank Holger Benl for illuminating discussions.

2 A simply typed $\boldsymbol{\lambda}$-calculus with constants

2.1 Types and terms; rewrite rules

We start from a given set of *ground types*. *Types* are inductively generated from ground types τ by $\rho \to \sigma$ and $\rho \times \sigma$. *Terms* are

x^ρ	typed variables,
c^ρ	constants,
$(\lambda x^\rho M^\sigma)^{\rho\to\sigma}$	abstractions,
$(M^{\rho\to\sigma} N^\rho)^\sigma$	applications,
$\langle M_0^\rho, M_1^\sigma\rangle^{\rho\times\sigma}$	pairing,
$\pi_0(M^{\rho\times\sigma})^\rho$, $\pi_1(M^{\rho\times\sigma})^\sigma$	projections.

Ground types will always be denoted by τ. We sometimes write $M0$ for $\pi_0(M)$ and $M1$ for $\pi_1(M)$. Two terms M and N are called α*-equal* – written $M =_\alpha N$ – if they are equal up to renaming of bound variables. Λ_ρ denotes the set of all terms of type ρ. $M\boldsymbol{N}$ denotes $(\dots(MN_1)N_2\dots)N_n$, where some of the N_i's may be 0 or 1. By $\mathsf{FV}(M)$ we denote the set of variables occurring free in M. By $M_x[N]$ we mean substitution of every free occurrence of x in M by N, renaming bound variables if necessary. Similarly $M_{\boldsymbol{x}}[\boldsymbol{N}]$ denotes simultaneous substitution. Finally $\boldsymbol{\rho} \to \sigma$ stands for $\rho_1 \to (\rho_2 \to \dots(\rho_n \to \sigma)\dots)$ and $\lambda\boldsymbol{x}r$ abbreviates $\lambda x_1 \dots \lambda x_n r$.

For the constants c^ρ we assume that some rewrite rules of the form $c\boldsymbol{K} \longmapsto N$ are given, where $\mathsf{FV}(N) \subseteq \mathsf{FV}(\boldsymbol{K})$ and $c\boldsymbol{K}$, N have the same type (not necessarily a ground type). Moreover, for any two c-rules $c\boldsymbol{K} \longmapsto N$ and $c\boldsymbol{K}' \longmapsto N'$ with equal projection markers 0, 1 we require that $\boldsymbol{K}$ and $\boldsymbol{K}'$ are of the same length. For example, if c is of type $(\tau \to \tau \to \tau) \times (\tau \to \tau)$, then the rules $c0x_1x_1 \longmapsto a$ and $c1x \longmapsto b$ are admitted.

Since we allowed almost arbitrary rewrite rules, it may happen that a term can be rewritten by different rules. In order to obtain a deterministic procedure we assume that for every constant $c^{\boldsymbol{\rho}\to\sigma}$ we are given a function sel_c computing for every tuple of terms $\boldsymbol{M}^{\boldsymbol{\rho}}$ either a rule $c\boldsymbol{K} \longmapsto N$, in which case $\boldsymbol{M}$ is an instance of $\boldsymbol{K}$, i.e. $\boldsymbol{M} = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]$, or else the message "$\mathsf{no\text{-}match}$", in which case $\boldsymbol{M}$ doesn't match any rule, i.e. there is no rule $c\boldsymbol{K} \longmapsto N$ such that $\boldsymbol{M}$ is an instance of $\boldsymbol{K}$. Clearly sel_c should be compatible with α-equality.

For readability a rule $c\boldsymbol{K} \longmapsto \lambda \boldsymbol{y} N$ with $\boldsymbol{y}$ distinct variables not free in $\boldsymbol{K}$ will be written $c\boldsymbol{K}\boldsymbol{y} \longmapsto N$.

2.2 Examples

(a) Usually we have the ground type ι of natural numbers available, with constructors 0^ι, $\textsc{Suc}^{\iota\to\iota}$ and *recursion operators* $R_\rho^{\iota\to\rho\to(\iota\to\rho\to\rho)\to\rho}$. The rewrite rules for R are

$$\begin{aligned} R0yz &\longmapsto y, \\ R(\textsc{Suc}\, x)yz &\longmapsto zx(Rxyz), \end{aligned}$$

i.e. $R0 \longmapsto \lambda y \lambda z y$ and $R(\textsc{Suc}\, x) \longmapsto \lambda y \lambda z(zx(Rxyz))$. A simplified scheme of the same form gives a cases construct:

$$\begin{aligned} \mathsf{if}\; 0yz &\longmapsto y, \\ \mathsf{if}\,(\textsc{Suc}\, x)yz &\longmapsto z. \end{aligned}$$

We may also add a rewrite rule (due to McCarthy [11])

$$\mathsf{if}\,(\mathsf{if}\; xyz)uv \longmapsto \mathsf{if}\; x(\mathsf{if}\; yuv)(\mathsf{if}\; zuv).$$

Moreover we can write down rules according to the usual recursive definitions of addition and multiplication, and then also the rewrite rule

$$\mathsf{mult}(\mathsf{add}\; xy)z \longmapsto \mathsf{add}(\mathsf{mult}\; xz)(\mathsf{mult}\; yz).$$

Simultaneous recursion may be treated as well, e.g.

$$\begin{aligned} \mathsf{odd}\; 0 &\longmapsto \textsc{Suc}\, 0 & \mathsf{even}\; 0 &\longmapsto 0, \\ \mathsf{odd}(\textsc{Suc}\, x) &\longmapsto \mathsf{even}\; x & \mathsf{even}(\textsc{Suc}\, x) &\longmapsto \mathsf{odd}\; x. \end{aligned}$$

(b) We can also deal with infinitely branching trees like the Brouwer ordinals of type $\mathcal{O}$. We have constructors $0^{\mathcal{O}}$ and $\textsc{Sup}^{(\iota\to\mathcal{O})\to\iota}$, and for recursion constants $\textsc{Rec}_\rho^{\mathcal{O}\to\rho\to((\iota\to\mathcal{O})\to(\iota\to\rho)\to\rho)\to\rho}$. The rewrite rules for $\textsc{Rec}$ are:

$$\begin{aligned} \textsc{Rec}0yz &\longmapsto y, \\ \textsc{Rec}(\textsc{Sup}\; x)yz &\longmapsto zx(\lambda u\, \textsc{Rec}(xu)yz). \end{aligned}$$

(c) It is well known that by the Curry-Howard correspondence natural deduction proofs can be written as λ-terms with formulas as types. To use normalization by evaluation for normalizing proofs we may also introduce a ground type ex with constructors and destructors

$$(\exists^+_{\rho_0,\rho_1})^{\rho_0\to\rho_1\to\mathsf{ex}} \quad \text{and} \quad (\exists^-_{\rho_0,\rho_1,\sigma})^{\mathsf{ex}\to(\rho_0\to\rho_1\to\sigma)\to\sigma};$$

these are called *existential constants*. The rewrite rule for $\exists^-$ is

$$\exists^-(\exists^+ x_0 x_1) y \longmapsto y x_0 x_1.$$

The (constructive) existential quantifier can then be dealt with conveniently by means of axioms

$$\begin{aligned}&\exists^+ : \forall x(A \to \exists x A),\\ &\exists^- : \exists x A \to \forall x(A \to B) \to B \quad \text{with } x \notin \mathsf{FV}(B).\end{aligned}$$

If x has type ρ_0 and the formulas A and B are associated with the types ρ_1 and σ, respectively, the rewrite rule above is clear. It seems that the existential type ex could be replaced by $\rho_0 \times \rho_1$ and the constants $\exists^+_{\rho_0,\rho_1}$ and $\exists^-_{\rho_0,\rho_1,\sigma}$ by the terms $\lambda x_0 \lambda x_1 \langle x_0, x_1\rangle$ and $\lambda z \lambda f(f\pi_0(z)\pi_1(z))$ respectively. However, the latter term does not correspond to a derivation in first order logic, since it is impossible to pass from an arbitrary derivation d (possibly with free assumptions) of $\exists x A$ to a term $\pi_0(d)$ and a derivation $\pi_1(d)$ of $A_x[\pi_0(d)]$.

2.3 Normalizable terms and their normal forms

We inductively define a relation $M \longrightarrow Q$ for terms M, Q. The intended meaning of $M \longrightarrow Q$ is that M is normalizable with (long) normal form Q. Here it is convenient to identify α-equal terms.

Eta.

$$\frac{My \longrightarrow Q}{M^{\rho\to\sigma} \longrightarrow \lambda y Q} \quad \text{for } y \notin \mathsf{FV}(M) \qquad \frac{\pi_0(M) \longrightarrow Q_0 \quad \pi_1(M) \longrightarrow Q_1}{M^{\rho\times\sigma} \longrightarrow \langle Q_0, Q_1\rangle}$$

For the next rules it is enough that they all have a conclusion $M \longrightarrow Q$ with M, Q of a ground type τ.

Beta.

$$\frac{M_x[N]\boldsymbol{P} \longrightarrow Q}{(\lambda x M)N\boldsymbol{P} \longrightarrow Q} \qquad \frac{M_i\boldsymbol{P} \longrightarrow Q}{\pi_i(\langle M_0, M_1\rangle)\boldsymbol{P} \longrightarrow Q} \quad \text{for } i \in \{0,1\}$$

VarApp.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{M}'}{x\boldsymbol{M} \longrightarrow x\boldsymbol{M}'}$$

$\boldsymbol{M} \longrightarrow \boldsymbol{M}'$ abbreviates the list $M_1 \longrightarrow M_1', \dots, M_n \longrightarrow M_n'$ of assumptions. Moreover, for every constant c we have the following rules.

Red.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{M'} \qquad N_{\boldsymbol{x}}[\boldsymbol{L}]\boldsymbol{P} \longrightarrow Q}{c\boldsymbol{M}\boldsymbol{P} \longrightarrow Q} \quad \text{if } \mathsf{sel}_c(\boldsymbol{M'}) = c\boldsymbol{K} \longmapsto N \text{ and } \boldsymbol{M'} = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]$$

PassApp.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{M}' \quad \boldsymbol{P} \longrightarrow \boldsymbol{P}'}{c\boldsymbol{M}\boldsymbol{P} \longrightarrow c\boldsymbol{M}'\boldsymbol{P}'} \quad \text{if } \mathsf{sel}_c(\boldsymbol{M'}) = \text{no-match}$$

For readability we will write Red in the following form and always assume that $c\boldsymbol{K} \longmapsto N$ is the selected rule.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}] \qquad N_{\boldsymbol{x}}[\boldsymbol{L}]\boldsymbol{P} \longrightarrow Q}{c\boldsymbol{M}\boldsymbol{P} \longrightarrow Q} \quad \text{if } c\boldsymbol{K} \longmapsto N$$

The set Lnf of terms in long normal form is defined as follows: $\lambda x M$, $\langle M, N\rangle$, $x\boldsymbol{M}$ and $c\boldsymbol{M}\boldsymbol{N}$ are in Lnf if $M, N, \boldsymbol{M}, \boldsymbol{N}$ are and $\mathsf{sel}_c(c\boldsymbol{M}) = \mathsf{no\text{-}match}$. It is obvious that $M \longrightarrow Q$ implies that $Q \in$ Lnf. Furthermore it can be shown that in this case M normalizes to Q in the usual sense w.r.t. β-reduction, η-expansion and the rewrite rules for the constants. Conversely, if M is strongly normalizable w.r.t. these reductions (i.e. every reduction sequence terminates) then $M \longrightarrow Q$ for some Q. We omit the proofs of these facts since they will not be needed in the sequel. Although $M \longrightarrow Q$ implies that Q is a normal form of M, the converse is not true in general. To see this, consider the non-terminating rewrite rules $\mathsf{mult}x^{\iota}0 \longmapsto 0$ and $\bot^{\iota} \longmapsto \bot$. Then 0 is a normal form of $\mathsf{mult}\bot 0$, but $\mathsf{mult}\bot 0 \longrightarrow 0$ does not hold. Moreover, the relation $M \longrightarrow Q$ clearly is not closed under substitution. However, it is closed under substitution of variables.

Lemma 1. *If $M \longrightarrow Q$, then $M_x[z] \longrightarrow Q_x[z]$ with a derivation of the same height.*

Proof. We use induction on the height of the derivation of $M \longrightarrow Q$, and only treat the rule Eta

$$\frac{My \longrightarrow Q}{M \longrightarrow \lambda y Q} \quad \text{for } y \notin \mathsf{FV}(M)$$

with M of type $\rho \to \sigma$. In case $x = y$ there is nothing to show, since then x does not occur free in the conclusion $M \longrightarrow \lambda y Q$. So assume $x \neq y$.

Subcase $y \neq z$. We must show $M_x[z] \longrightarrow \lambda y Q_x[z]$. By induction hypothesis $M_x[z]y \longrightarrow Q_x[z]$, since $x \neq y$. Therefore by Eta $M_x[z] \longrightarrow \lambda y Q_x[z]$, since $y \notin \mathsf{FV}(M_x[z])$ (because of $y \neq z$).

Subcase $y = z$. We must show $M_x[y] \longrightarrow (\lambda y Q)_x[y] = \lambda u Q_{x,y}[y, u]$ with a new variable u. By induction hypothesis $Mu \longrightarrow Q_y[u]$ with a derivation of the same height as that for $My \longrightarrow Q$, hence again by induction hypothesis $M_x[y]u \longrightarrow Q_{x,y}[y, u]$. Therefore by Eta $M_x[y] \longrightarrow \lambda u Q_{x,y}[y, u]$.

2.4 Term families

Since normalization by evaluation needs to create bound variables when "reifying" abstract objects of higher type, it is useful to follow de Bruijn's [9] style of representing bound variables in terms. This is done here – as in [4] – by means of *term families*. A term family is a parametrized version of a given term M. The idea is that the term family of M at index k reproduces M with bound variables renamed starting at k. For example, for

$$M := \lambda u \lambda v.c(\lambda x.vx)(\lambda y \lambda z.zu)$$

the associated term family M^∞ at index 3 yields

$$M^\infty(3) := \lambda x_3 \lambda x_4 . c(\lambda x_5 . x_4 x_5)(\lambda x_5 \lambda x_6 . x_6 x_3).$$

We denote terms by $M, N, K, \ldots$, and term families by $r, s, t, \ldots$.

To every term M^ρ we assign a term family $M^\infty \colon \mathbb{N} \to \Lambda_\rho$ by

$$\begin{aligned} x^\infty(k) &:= x, \\ (\lambda y M)^\infty(k) &:= \lambda x_k (M_y[x_k]^\infty(k+1)), & \langle M_0, M_1\rangle^\infty(k) &:= \langle M_0^\infty(k), M_1^\infty(k)\rangle, \\ (MN)^\infty(k) &:= M^\infty(k) N^\infty(k), & \pi_i(M)^\infty(k) &:= \pi_i(M^\infty(k)). \end{aligned}$$

Application of a term family $r \colon \mathbb{N} \to \Lambda_{\rho\to\sigma}$ to a term family $s \colon \mathbb{N} \to \Lambda_\rho$ is the family $rs \colon \mathbb{N} \to \Lambda_\sigma$ defined by $(rs)(k) := r(k)s(k)$, and similarly for pairing $\langle r_0, r_1\rangle(k) := \langle r_0(k), r_1(k)\rangle$ and projections $\pi_i(r)(k) := \pi_i(r(k))$. Hence e.g. $(MN)^\infty = M^\infty N^\infty$.

Note that in contrast to our convention to consider terms up to bound renaming, the definition of $(\lambda y M)^\infty$ refers to a particular choice of the bound variable y. Part a of the following lemma shows that nevertheless this definition is compatible with our convention. In the rest of this subsection we drop the convention to identify α-equal terms. Hence $M = N$ means that M and N are literally identical as opposed to $M =_\alpha N$, which means equality up to bound renaming.

We let $k > \mathsf{FV}(M)$ mean that k is greater than all i such that $x_i^\rho \in \mathsf{FV}(M)$ for some type ρ.

Lemma 2. *a. If $M =_\alpha N$, then $M^\infty = N^\infty$.*
b. If $k > \mathsf{FV}(M)$, then $M^\infty(k) =_\alpha M$.

Proof. a. Induction on the height of M. Only the case where M and N are abstractions is critical. So assume $\lambda y^\rho M =_\alpha \lambda z^\rho N$. Then $M_y[P] =_\alpha N_z[P]$ for all terms P^ρ. In particular $M_y[x_k] =_\alpha N_z[x_k]$ for arbitrary $k \in \mathbb{N}$. Hence $M_y[x_k]^\infty(k+1) = N_z[x_k]^\infty(k+1)$, by induction hypothesis. Therefore

$$(\lambda y M)^\infty(k) = \lambda x_k (M_y[x_k]^\infty(k+1)) = \lambda x_k (N_z[x_k]^\infty(k+1)) = (\lambda z N)^\infty(k).$$

b. Induction on the height of M. We only consider the case $\lambda y M$. The assumption $k > \mathsf{FV}(\lambda y M)$ implies $x_k \notin \mathsf{FV}(\lambda y M)$ and hence $\lambda y M =_\alpha \lambda x_k (M_y[x_k])$. Furthermore $k+1 > \mathsf{FV}(M_y[x_k])$, and hence $M_y[x_k]^\infty(k+1) =_\alpha M_y[x_k]$, by induction hypothesis. Therefore

$$(\lambda y M)^\infty(k) = \lambda x_k (M_y[x_k]^\infty(k+1)) =_\alpha \lambda x_k (M_y[x_k]) =_\alpha \lambda y M. \qquad \square$$

Let $\mathsf{ext}(r) := r(k)$, where k is the least number greater than all i such that some variable of the form x_i^ρ occurs (free or bound) in $r(0)$.

Lemma 3. $\mathsf{ext}(M^\infty) =_\alpha M$.

Proof. Use part b of the lemma above, and the fact that $\mathsf{ext}(M^\infty) =_\alpha M^\infty(k)$ where $k > \mathsf{FV}(M)$.

3 Domain theoretic semantics of simply typed $\boldsymbol{\lambda}$-calculi

In this section we shall discuss the domain theoretic semantics of simply typed lambda calculi in general. Although the constructions below are standard (see e.g. the books of LAMBEK/SCOTT [10] or CROLE [6]), we discuss them in some detail in order to make the paper accessible also for readers not familiar with this subject. Most constructions make sense in an arbitrary cartesian closed category (ccc). However we will confine ourselves to the domain semantics and will only occasionally comment on the categorical aspects.

It is well-known that SCOTT-domains with continuous functions form a cartesian closed category DOM. The product $D \times E$ is the set-theoretic product with the component-wise ordering. The exponential $[D \to E]$ is the continuous function space with the pointwise ordering. The terminal object is the one point space $\mathbf{1} := \{\bot\}$ (there is no initial object and there are no coproducts). In order to cope with the categorical interpretation we will identify an element x of a domain D with the mapping from $\mathbf{1}$ to D with value x.

Besides the cartesian closedness we also use the fact that DOM is closed under infinite products and that there is a fixed point operator $\mathrm{FIX}\colon (D \to D) \to D$ assigning to every continuous function $f\colon D \to D$ its least fixed point $\mathrm{FIX}(f) \in D$. Furthermore we will use that partial families of terms form a domain and some basic operations on terms and term families are continuous and hence exist as morphisms in the category. Any other ccc with these properties would do as well.

Elements of a product domain $D_1 \times \cdots \times D_n$ are written $[a_1, \ldots, a_n]$. If $f \in [D_1 \to [D_2 \to \ldots [D_n \to E] \ldots]]$ and $a_i \in D_i$, then $f(a_1, \ldots, a_n)$ or $f(\boldsymbol{a})$ stands for $f(a_1) \ldots (a_n)$.

An *interpretation* for a given system of ground types is a mapping $\mathcal{I}$ assigning to every ground type τ a domain $\mathcal{I}(\tau)$. Given such an interpretation we define domains $[\![\rho]\!]^{\mathcal{I}}$ for every type ρ by

$$[\![\tau]\!]^{\mathcal{I}} := \mathcal{I}(\tau), \quad [\![\rho \to \sigma]\!]^{\mathcal{I}} := [[\![\rho]\!]^{\mathcal{I}} \to [\![\sigma]\!]^{\mathcal{I}}], \quad [\![\rho \times \sigma]\!]^{\mathcal{I}} := [\![\rho]\!]^{\mathcal{I}} \times [\![\sigma]\!]^{\mathcal{I}}.$$

We write $[\![\rho_1, \ldots, \rho_n]\!]^{\mathcal{I}} := [\![\rho_1 \times \cdots \times \rho_n]\!]^{\mathcal{I}} = [\![\rho_1]\!]^{\mathcal{I}} \times \cdots \times [\![\rho_n]\!]^{\mathcal{I}} =: [\![\boldsymbol{\rho}]\!]^{\mathcal{I}}$. An *interpretation of a typed lambda calculus* (specified by a set of ground types and a set of constants) is a mapping $\mathcal{I}$ assigning to every ground type τ a domain $\mathcal{I}(\tau)$ (hence $\mathcal{I}$ is an interpretation of ground types), and assigning to every constant c^ρ a value $\mathcal{I}(c) \in [\![\rho]\!]^{\mathcal{I}}$ (i.e. a morphism from $\mathbf{1}$ to $[\![\rho]\!]^{\mathcal{I}}$).

In order to extend such an interpretation to all terms we use the following continuous functions, i.e. morphisms (in the sequel a continuous function will be called morphism if its role as a morphism in the ccc DOM is to be emphasized):

$$\begin{aligned}
&!_D\colon D \to \mathbf{1}, \quad !_D(d) := \bot \\
&\pi_i\colon D_1 \times \cdots \times D_n \to D_i, \quad \pi_i([\boldsymbol{a}]) := a_i, \\
&\mathsf{curry}\colon [D \times E \to F] \to [D \to [E \to F]], \quad \mathsf{curry}(f, a, b) := f([a, b]), \\
&\mathsf{eval}\colon [D \to E] \times D \to E, \quad \mathsf{eval}(f, a) := f(a).
\end{aligned}$$

Furthermore we use the fact that morphisms are closed under composition $\circ$ and (since DOM is a ccc) under pairing $\langle .,. \rangle$, where for $f \colon D \to E$ and $g \colon D \to F$ the function $\langle f, g \rangle \colon D \to E \times F$ maps a to $[f(a), g(a)]$. For every type ρ and every list of distinct variables $\boldsymbol{x}^{\boldsymbol{\rho}} = x_1^{\rho_1}, \ldots, x_n^{\rho_n}$ we let $\Lambda_\rho(\boldsymbol{x})$ denote the set of terms of type ρ with free variables among $\{\boldsymbol{x}\}$. Let $\mathcal{I}$ be an interpretation. Then for every $M \in \Lambda_\rho(\boldsymbol{x}^{\boldsymbol{\rho}})$ we define a morphism $[\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}} \colon [\![\boldsymbol{\rho}]\!] \to [\![\rho]\!]$ by

$$\begin{aligned} [\![c]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \mathcal{I}(c) \circ !_{[\![\boldsymbol{\rho}]\!]}, \\ [\![x_i]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \pi_i, \\ [\![\lambda x M]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \mathsf{curry}([\![M]\!]^{\mathcal{I}}_{\boldsymbol{x},x}), \\ [\![MN]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \mathsf{eval} \circ \langle [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}, [\![N]\!]^{\mathcal{I}}_{\boldsymbol{x}} \rangle, \\ [\![\langle M, N \rangle]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \langle [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}, [\![N]\!]^{\mathcal{I}}_{\boldsymbol{x}} \rangle, \\ [\![\pi_i(M)]\!]^{\mathcal{I}}_{\boldsymbol{x}} &:= \pi_i \circ [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}. \end{aligned}$$

This definition works in any ccc. For our purposes it will be more convenient to evaluate a term in a global environment and not in a local context. Let

$$\mathrm{ENV} := \prod_{x^\sigma \in \mathrm{VAR}} [\![\sigma]\!]^{\mathcal{I}} \in \mathrm{DOM}.$$

For every term $M \in \Lambda_\rho(x_1, \ldots, x_n)$ we define a continuous function

$$[\![M]\!]^{\mathcal{I}} \colon \mathrm{ENV} \to [\![\rho]\!]^{\mathcal{I}}, \quad [\![M]\!]^{\mathcal{I}}_\xi := [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}([\xi(x_1), \ldots, \xi(x_n)]).$$

Formally this definition depends on a particular choice of the list of variables $x_1, \ldots, x_n$. However, because of the well-known coincidence property in fact it does not.

Lemma 4. *If $M \in \Lambda_\rho(y_1, \ldots, y_m)$ then*

$$[\![M]\!]^{\mathcal{I}}_\xi = [\![M]\!]^{\mathcal{I}}_{\boldsymbol{y}}(\xi(y_1), \ldots, \xi(y_m)).$$

From this we easily get the familiar equations

$$\begin{aligned} [\![c]\!]^{\mathcal{I}}_\xi &= \mathcal{I}(c), \\ [\![x]\!]^{\mathcal{I}}_\xi &= \xi(x), \\ [\![\lambda x M]\!]^{\mathcal{I}}_\xi(a) &= [\![M]\!]^{\mathcal{I}}_{\xi[x \mapsto a]}, \\ [\![MN]\!]^{\mathcal{I}}_\xi &= [\![M]\!]^{\mathcal{I}}_\xi([\![N]\!]^{\mathcal{I}}_\xi), \\ [\![\langle M, N \rangle]\!]^{\mathcal{I}}_\xi &= [[\![M]\!]^{\mathcal{I}}_\xi, [\![N]\!]^{\mathcal{I}}_\xi], \\ [\![\pi_i(M)]\!]^{\mathcal{I}}_\xi &= \pi_i([\![M]\!]^{\mathcal{I}}_\xi). \end{aligned}$$

In many cases the interpretation $\mathcal{I}$ of the constants will have to be defined recursively, by e.g. referring to $[\![M]\!]^{\mathcal{I}}$ for several terms M. This causes no problem, since the functionals $[\![M^\rho]\!]^{\mathcal{I}} \colon \mathrm{ENV} \to [\![\rho]\!]$ depend continuously on $\mathcal{I}$, where $\mathcal{I}$ is to be considered as an element of the infinite product $\Pi_{c^\rho}[\![\rho]\!]$. This can be seen as follows: Looking at their definitions we see that the functions $[\mathcal{I}, \boldsymbol{a}] \mapsto [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}(\boldsymbol{a})$ are built by composition from the continuous functions

$$\begin{aligned}
&\pi_{c^\sigma} : \Pi_{c^\rho}[\![\rho]\!] \to [\![\sigma]\!], \quad \pi_{c^\sigma}(\mathcal{I}) := \mathcal{I}(c),\\
&\cdot \circ \cdot : [E \to F] \times [D \to E] \to [D \to F],\\
&\langle \cdot, \cdot \rangle : [D \to E] \times [D \to F] \to [D \to E \times F],
\end{aligned}$$

as well as the functions $!_D, \pi_i$, curry and eval above. Hence $[\mathcal{I}, \boldsymbol{a}] \mapsto [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}(\boldsymbol{a})$ is continuous. But then also $[\mathcal{I}, \xi] \mapsto [\![M]\!]^{\mathcal{I}}_{\xi}$ is continuous, since we have $[\![M]\!]^{\mathcal{I}}_{\xi} = [\![M]\!]^{\mathcal{I}}_{\boldsymbol{x}}([\pi_{x_1}(\xi), \ldots, \pi_{x_n}(\xi)])$, where $\pi_{x^\rho} : \mathrm{E{\scriptsize NV}} \to [\![\rho]\!]$, $\pi_x(\xi) := \xi(x)$.

Hence the value $\mathcal{I}(c)$ may be defined as a least fixed point of a continuous function on the domain $\Pi_{c^\rho}[\![\rho]\!]$. – In the sequel we will omit the superscript $\mathcal{I}$ when it is clear from the context.

The following facts hold in any ccc.

Lemma 5.

$$\begin{aligned}
[\![M_{\boldsymbol{x}}[\boldsymbol{N}]]\!]_\xi &= [\![M]\!]_{\xi[\boldsymbol{x} \mapsto [\![\boldsymbol{N}]\!]_\xi]} && \textit{(substitution lemma)}\\
[\![(\lambda x M)N]\!]_\xi &= [\![M_x[N]]\!]_\xi && \textit{(beta 1)}\\
[\![\pi_i(\langle M_0, M_1\rangle)]\!]_\xi &= [\![M_i]\!]_\xi && \textit{(beta 2)}\\
[\![M]\!]_\xi &= [\![\lambda y(My)]\!]_\xi \quad (y^\rho \notin \mathsf{FV}(M^{\rho\to\sigma})) && \textit{(eta 1)}\\
[\![M]\!]_\xi &= [\![\langle \pi_0(M), \pi_1(M)\rangle]\!]_\xi && \textit{(eta 2)}
\end{aligned}$$

Lemma 6. *If $[\![P]\!]_\xi = [\![Q]\!]_\xi$ for all environments ξ, and M is transformed into N by replacing an occurrence of P in M by Q, then $[\![M]\!]_\xi = [\![N]\!]_\xi$ for all environments ξ.*

Proof. Induction on M.

Lemma 7. *If $M \xrightarrow{1}_\beta N$ or $M \xrightarrow{1}_\eta N$, then $[\![M]\!]_\xi = [\![N]\!]_\xi$.*

4 Normalization by evaluation

We now consider a special model, whose ground type objects are made from syntactic material. We let $\mathbb{N} \xrightarrow{\cup} \Lambda_\rho$ denote the set of partial term families, i.e. partial functions from the integers to the set of terms of type ρ. $\mathbb{N} \xrightarrow{\cup} \Lambda_\rho$ partially ordered by inclusion of graphs is a domain. We interpret the ground types by

$$\mathcal{I}(\tau) := \mathbb{N} \xrightarrow{\cup} \Lambda_\tau.$$

This gives us an interpretation $[\![\rho]\!] = [\![\rho]\!]^{\mathcal{I}} \in \mathrm{D{\scriptsize OM}}$ for every type ρ. In order to define the interpretation of the constants, some preparations are necessary.

4.1 Reification and reflection; interpretation of the constants

For every type ρ we define two continuous functions

$$\downarrow_\rho \colon [\![\rho]\!] \to (\mathbb{N} \xrightarrow{\cup} \Lambda_\rho) \quad (\text{“reify”}) \qquad \uparrow_\rho \colon (\mathbb{N} \xrightarrow{\cup} \Lambda_\rho) \to [\![\rho]\!] \quad (\text{“reflect”}).$$

These functions will be used when defining the interpretation of the constants as well as in our final goal, normalization by evaluation. $\downarrow_\rho$ and $\uparrow_\rho$ are defined simultaneously by induction on the type ρ.

$$\begin{aligned} \downarrow_\tau(r) &:= r, & \uparrow_\tau(r) &:= r,\\ \downarrow_{\rho\to\sigma}(a)(k) &:= \lambda x_k^\rho \downarrow_\sigma\big(a(\uparrow_\rho(x_k^\infty))\big)(k{+}1), & \uparrow_{\rho\to\sigma}(r)(b) &:= \uparrow_\sigma(r\downarrow_\rho(b)),\\ \downarrow_{\rho\times\sigma}([a,b]) &:= \langle \downarrow_\rho(a), \downarrow_\sigma(b)\rangle, & \uparrow_{\rho\times\sigma}(r) &:= [\uparrow_\rho(\pi_0(r)), \uparrow_\sigma(\pi_1(r))]. \end{aligned}$$

Note that for $a_i \in [\![\rho_i]\!]$ we have $\uparrow_{\boldsymbol{\rho}\to\sigma}(r)(a_1,\dots a_n) = \uparrow_\sigma(\downarrow_{\rho_1}(a_1)\dots\downarrow_{\rho_n}(a_n))$, to which we refer by

$$\uparrow(r)(\boldsymbol{a}) = \uparrow(r\downarrow(\boldsymbol{a})). \tag{1}$$

We now define the values of the constants c in our special model. Note that we can view $\uparrow$ as an environment: it assigns to every variable x of type ρ the value $\uparrow(x^\infty) \in [\![\rho]\!]$.

$$\mathcal{I}(c)(\boldsymbol{a}) := \begin{cases} [\![N]\!]^{\mathcal{I}}_{[\boldsymbol{x}\mapsto[\![\boldsymbol{L}]\!]^{\mathcal{I}}_{\uparrow}]} & \text{if } \mathsf{sel}_c(\mathsf{ext}(\downarrow(\boldsymbol{a}))) = c\boldsymbol{K}\longmapsto N \text{ and } \mathsf{ext}(\downarrow(\boldsymbol{a}))=\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]\\ \uparrow(c^\infty\downarrow(\boldsymbol{a})) & \text{if } \mathsf{sel}_c(\mathsf{ext}(\downarrow(\boldsymbol{a}))) = \mathsf{no\text{-}match}\\ \bot & \text{otherwise, i.e. } \mathsf{ext}(\downarrow(\boldsymbol{a})) \text{ is undefined.} \end{cases}$$

Note that this in general is a recursive definition, since the terms N and $\boldsymbol{L}$ may contain c. We now obtain the correctness of normalization by evaluation:

Theorem 1. *If $M \longrightarrow Q$, then $\downarrow([\![M]\!]_\uparrow) = Q^\infty$.*

Proof. By induction on the height of the derivation of $M \longrightarrow Q$. For brevity we leave out the rules concerning product types, since their treatment does not bring up any new issues.

Case ETA, i.e.

$$\frac{My \longrightarrow Q}{M \longrightarrow \lambda y Q} \quad \text{for } y \notin \mathsf{FV}(M)$$

with M of type $\rho \to \sigma$. By lemma 1 we then have $Mx_k \longrightarrow Q_y[x_k]$ with a derivation of the same height as that of the given derivation of $My \longrightarrow Q$, hence by induction hypothesis $\downarrow([\![Mx_k]\!]_\uparrow) = Q_y[x_k]^\infty$. We obtain

$$\begin{aligned} \downarrow([\![M]\!]_\uparrow)(k) &= \lambda x_k\Big(\downarrow([\![M]\!]_\uparrow(\uparrow(x_k^\infty)))(k+1)\Big)\\ &= \lambda x_k\Big(\downarrow([\![Mx_k]\!]_\uparrow)(k+1)\Big)\\ &= \lambda x_k\Big(Q_y[x_k]^\infty(k+1)\Big) && \text{by IH}\\ &= (\lambda y Q)^\infty(k). \end{aligned}$$

In the following cases the rules are of ground type τ, so we use that $\downarrow([\![M]\!]_\uparrow) = [\![M]\!]_\uparrow$ for $M \in \Lambda_\tau$.

Case BETA, i.e.

$$\frac{M_x[N]\boldsymbol{P} \longrightarrow Q}{(\lambda x M)N\boldsymbol{P} \longrightarrow Q}$$

$$\begin{aligned} [\![(\lambda x M)N\boldsymbol{P}]\!]_\uparrow &= [\![M_x[N]\boldsymbol{P}]\!]_\uparrow && \text{by lemma 7} \\ &= Q^\infty && \text{by IH.} \end{aligned}$$

Case RED, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{K_x}[\boldsymbol{L}] \qquad N_{\boldsymbol{x}}[\boldsymbol{L}]\boldsymbol{P} \longrightarrow Q}{c\boldsymbol{M}\boldsymbol{P} \longrightarrow Q}$$

where $c\boldsymbol{K} \longmapsto N$ is the selected rule, i.e. $\mathsf{sel}_c(\boldsymbol{K_x}[\boldsymbol{L}]) = c\boldsymbol{K} \longmapsto N$. Recall

$$[\![c\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow = \mathcal{I}(c)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow).$$

By induction hypothesis $\downarrow\big([\![\boldsymbol{M}]\!]_\uparrow\big) = \boldsymbol{K_x}[\boldsymbol{L}]^\infty$. By definition of $\mathcal{I}(c)$ we have to compute $\mathsf{sel}_c(\mathsf{ext}(\downarrow([\![\boldsymbol{M}]\!]_\uparrow)))$. By lemma 3 $\mathsf{ext}\big(\boldsymbol{K_x}[\boldsymbol{L}]^\infty\big) = \boldsymbol{K_x}[\boldsymbol{L}]$. Hence $\mathsf{sel}_c(\mathsf{ext}(\downarrow([\![\boldsymbol{M}]\!]_\uparrow))) = \mathsf{sel}_c(\boldsymbol{K_x}[\boldsymbol{L}]) = c\boldsymbol{K} \longmapsto N$, and therefore

$$\begin{aligned} [\![c\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow &= \mathcal{I}(c)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow) \\ &= [\![N]\!]_{[\boldsymbol{x}\mapsto[\![\boldsymbol{L}]\!]_\uparrow]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{by definition of } \mathcal{I}(c) \\ &= [\![N_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow([\![\boldsymbol{P}]\!]_\uparrow) && \text{by the substitution lemma} \\ &= [\![N_{\boldsymbol{x}}[\boldsymbol{L}]\boldsymbol{P}]\!]_\uparrow \\ &= Q^\infty && \text{by IH.} \end{aligned}$$

Case VARAPP, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{M}'}{x\boldsymbol{M} \longrightarrow x\boldsymbol{M}'}$$

$$\begin{aligned} [\![x\boldsymbol{M}]\!]_\uparrow &= \uparrow(x^\infty)\,([\![\boldsymbol{M}]\!]_\uparrow) \\ &= x^\infty\downarrow([\![\boldsymbol{M}]\!]_\uparrow) && \text{by (1)} \\ &= x^\infty\,(\boldsymbol{M}')^\infty && \text{by IH} \\ &= (x\boldsymbol{M}')^\infty. \end{aligned}$$

Case PASSAPP, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{M}' \quad \boldsymbol{P} \longrightarrow \boldsymbol{P}'}{c\boldsymbol{M}\boldsymbol{P} \longrightarrow c\boldsymbol{M}'\boldsymbol{P}'}$$

where there is no rule $c\boldsymbol{K} \longmapsto N$ such that $\boldsymbol{M}'$ is of an instance of $\boldsymbol{K}$, i.e. $\mathsf{sel}_c(\boldsymbol{M}') = \mathsf{no\text{-}match}$. We obtain by induction hypothesis $\downarrow([\![\boldsymbol{M}]\!]_\uparrow) = \boldsymbol{M}'^\infty$ and hence $\mathsf{sel}_c\big(\downarrow([\![\boldsymbol{M}]\!]_\uparrow)\big) = \mathsf{sel}_c\big(\mathsf{ext}(\boldsymbol{M'}^{\infty})\big) = \mathsf{sel}_c(\boldsymbol{M'}) = \textsf{no-match}$. Hence

$$\begin{aligned}
[\![c\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow &= \mathcal{I}(c)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow) \\
&= \uparrow\Big(c^\infty\downarrow([\![\boldsymbol{M}]\!]_\uparrow)\Big)([\![\boldsymbol{P}]\!]_\uparrow) && \text{by definition of } \mathcal{I}(c) \\
&= c^\infty\downarrow([\![\boldsymbol{M},\boldsymbol{P}]\!]_\uparrow) && \text{by (1)} \\
&= c^\infty(\boldsymbol{M}')^\infty(\boldsymbol{P}')^\infty && \text{by IH} \\
&= (c\boldsymbol{M}'\boldsymbol{P}')^\infty. && \square
\end{aligned}$$

4.2 Special rewrite rules

We will now consider special cases of our general form of rewrite rules – to be called *d*-, *e*- and *f*-rules –, where the interpretation function $\mathcal{I}$ and hence normalization by evaluation is more efficient. It will be shown that theorem 1 continues to hold.

***d*-rules.** One problem with the interpretation of the constants in 4.1 is a certain inefficiency inherent in it: after computing a syntactic reification of $\boldsymbol{a}$ by $\mathsf{ext}(\downarrow(\boldsymbol{a}))$, determining the relevant rule $c\boldsymbol{K} \longmapsto N$ as $\mathsf{sel}_c(\mathsf{ext}(\downarrow(\boldsymbol{a})))$ and reading off $\boldsymbol{L}$ such that $\mathsf{ext}(\downarrow(\boldsymbol{a})) = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]$, we must go back to semantics and compute $[\![\boldsymbol{L}]\!]_\uparrow$.

However, in many cases we have rules of the special form

$$d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}], \tag{2}$$

with $\boldsymbol{K}$ of ground types and $\mathsf{FV}(P) \subseteq \boldsymbol{z}$. Notice that $\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}]_{\boldsymbol{x}}[\boldsymbol{L}] = \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]$ and $P_{\boldsymbol{z}}[\boldsymbol{K}]_{\boldsymbol{x}}[\boldsymbol{L}] = P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]$. So for instance $R0 \longmapsto \lambda y,z.y$ and $R(\textsc{Suc}\,x) \longmapsto \lambda y,z.zx(Rxyz)$ are instances of this kind of rules. Then it is tempting to define

$$\mathcal{I}(d)(\boldsymbol{a}) := \begin{cases} [\![P]\!]_{[\boldsymbol{z}\mapsto\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty]} & \text{if } \mathsf{sel}_d(\mathsf{ext}(\downarrow(\boldsymbol{a}))) = d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}] \\ & \text{and } \mathsf{ext}(\downarrow(\boldsymbol{a})) = \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]] \\ \uparrow(d^\infty\downarrow(\boldsymbol{a})) & \text{if } \mathsf{sel}_d(\mathsf{ext}(\downarrow(\boldsymbol{a}))) = \textsf{no-match} \\ \bot & \text{otherwise.} \end{cases}$$

Here the necessity to evaluate $\boldsymbol{L}$ no longer exists; rather, we can work with the term family $\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty$ directly.

***e*-rules.** One may still not be satisfied with the *d*-rules, since each time $\mathcal{I}(d)(\boldsymbol{a})$ is evaluated we need to compute $\mathsf{sel}_d(\mathsf{ext}(\downarrow(\boldsymbol{a})))$ in order to determine the rule to be applied. Now for a_i of ground type each $r_i := \downarrow(a_i)$ is a term family, and the computation of $\mathsf{ext}(r_i) = r_i(k)$ involves computing k from the set of all free and bound variables of the term $r_i(0)$. However, in many cases it suffices to evaluate the term family r_i at a fixed index, say 0.

Consider e-rules of the form

$$e\boldsymbol{K} \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}] \tag{3}$$

with $\boldsymbol{K}$ of ground types and $\mathsf{FV}(P) \subseteq \boldsymbol{z}$, and assume that for normal terms M

$$\mathsf{sel}_e\big(M^{\infty}(0)\big) = \mathsf{sel}_e(M). \tag{4}$$

This is particularly likely to hold if the terms $\boldsymbol{K}$ are λ-free, since the formation of term families $\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^{\infty}$ only affects names of λ-bound variables. Now define

$$\mathcal{I}(e)(\boldsymbol{r}) := \begin{cases} [\![P]\!]_{[\boldsymbol{z}\mapsto\boldsymbol{r}]} & \text{if } \mathsf{sel}_e(\boldsymbol{r}(0)) = e\boldsymbol{K}\longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}] \\ \uparrow\!\big(e^{\infty}\boldsymbol{r}\big) & \text{if } \mathsf{sel}_e(\boldsymbol{r}(0)) = \mathsf{no\text{-}match} \\ \bot & \text{otherwise.} \end{cases}$$

Note that if we would interpret e according to the d-rules, we would have $[\![P]\!]_{[\boldsymbol{z}\mapsto\mathsf{ext}(\boldsymbol{r})^{\infty}]}$ in the first case. But $\mathsf{ext}(\boldsymbol{r})^{\infty}$ and $\boldsymbol{r}$ will be different in general.

Rules of the form (3) satisfying (4) are particularly possible if we work within the ground type ι of natural numbers and have the predecessor function available, as a fixed constant with a fixed interpretation.

***f*-rules.** There is a final type of rules – to be called f-rules – which we want to consider. Their usefulness comes up when we work with concrete data types like the type ι of natural numbers and want to employ e.g. the usual polynomial normal form of terms. For instance, the term $(n+3)(n^2+2n+5)$ should normalize to $n^3+5n^2+11n+15$. Abstractly, what we have here is a function $\mathsf{norm}\colon \Lambda_\tau \to \Lambda_\tau$ for a ground type τ, and it will turn out that all we need to prove theorem 1 are the following properties:

$$\mathsf{norm}^2 = \mathsf{norm} \tag{5}$$

$$\mathsf{norm}(M^{\infty}(k)) = \mathsf{norm}(M)^{\infty}(k) \tag{6}$$

$$\text{if } M \text{ is } cde\text{-normal, then } \mathsf{norm}(M) \text{ is } cde\text{-normal} \tag{7}$$

where cde-normal refers to the c-rules, the d-rules and the e-rules (and of course the β-rule). Given such a function norm, we add all rules of the form

$$f\boldsymbol{M} \longmapsto \mathsf{norm}(f\boldsymbol{M}) \quad \text{provided both are different.} \tag{8}$$

with $\boldsymbol{M}$ and $f\boldsymbol{M}$ of ground types; these should be the only rules for f. We define the sel_f function by

$$\mathsf{sel}_f(\boldsymbol{M}) := \begin{cases} f\boldsymbol{M} \longmapsto \mathsf{norm}(f\boldsymbol{M}) & \text{if } f\boldsymbol{M} \text{ and } \mathsf{norm}(f\boldsymbol{M}) \text{ are different,} \\ \mathsf{no\text{-}match} & \text{otherwise.} \end{cases}$$

The interpretation is defined by

$$\mathcal{I}(f)(\boldsymbol{r})(k) := \mathsf{norm}(f\boldsymbol{r}(k)).$$

Extension of theorem 1. We will show that theorem 1 (i.e. $M \longrightarrow Q$ implies $\downarrow(\llbracket M \rrbracket_{\uparrow}) = Q^\infty$) continues to hold for such constants d, e and f with interpretations as given above.

Lemma 8. *For normal terms M we have*

$$\downarrow(\llbracket M \rrbracket_{\uparrow}) = M^\infty.$$

Proof. Induction on the height of M.

Case $\lambda y^\rho M^\sigma$. This is similar to, but a little simpler than the case ETA in theorem 1.

$$\begin{aligned}
\downarrow(\llbracket \lambda y M \rrbracket_{\uparrow})(k) &= \lambda x_k \Big(\downarrow(\llbracket \lambda y M \rrbracket_{\uparrow}(\uparrow(x_k^\infty)))(k+1) \Big) \\
&= \lambda x_k \Big(\downarrow(\llbracket M_y[x_k] \rrbracket_{\uparrow})(k+1) \Big) \\
&= \lambda x_k \big(M_y[x_k]^\infty(k+1) \big) && \text{by IH} \\
&= (\lambda y M)^\infty(k).
\end{aligned}$$

Case $\langle M, N \rangle$. Easy.

Case $(x\boldsymbol{M})^\tau$.

$$\begin{aligned}
\llbracket x\boldsymbol{M} \rrbracket_{\uparrow} &= \uparrow(x^\infty)(\llbracket \boldsymbol{M} \rrbracket_{\uparrow}) \\
&= x^\infty \downarrow(\llbracket \boldsymbol{M} \rrbracket_{\uparrow}) && \text{by (1)} \\
&= x^\infty \boldsymbol{M}^\infty && \text{by IH} \\
&= \big(x\boldsymbol{M}\big)^\infty.
\end{aligned}$$

Case $(c\boldsymbol{M}\boldsymbol{P})^\tau$. Since $c\boldsymbol{M}\boldsymbol{P}$ is normal, we have $\mathsf{sel}_c(\boldsymbol{M}) = \mathsf{no\text{-}match}$ by our general requirements on sel_c, and by induction hypothesis and lemma 3 $\mathsf{sel}_c(\mathsf{ext}(\downarrow(\llbracket M \rrbracket_{\uparrow}))) = \mathsf{sel}_c(\mathsf{ext}(\boldsymbol{M}^\infty)) = \mathsf{sel}_c(\boldsymbol{M}) = \mathsf{no\text{-}match}$, hence

$$\begin{aligned}
\llbracket c\boldsymbol{M}\boldsymbol{P} \rrbracket_{\uparrow} &= \mathcal{I}(c)(\llbracket \boldsymbol{M} \rrbracket_{\uparrow})(\llbracket \boldsymbol{P} \rrbracket_{\uparrow}) \\
&= \uparrow\big(c^\infty \downarrow(\llbracket \boldsymbol{M} \rrbracket_{\uparrow})\big) \llbracket \boldsymbol{P} \rrbracket_{\uparrow} \\
&= c^\infty \downarrow(\llbracket \boldsymbol{M} \rrbracket_{\uparrow}) \downarrow(\llbracket \boldsymbol{P} \rrbracket_{\uparrow}) && \text{by (1)} \\
&= c^\infty \boldsymbol{M}^\infty \boldsymbol{P}^\infty && \text{by IH} \\
&= \big(c\boldsymbol{M}\boldsymbol{P}\big)^\infty.
\end{aligned}$$

Case $(d\boldsymbol{M}\boldsymbol{P})^\tau$. We have only used that the interpretation of the constants c satisfies $\mathcal{I}(c)(\boldsymbol{a}) = \uparrow(c^\infty \downarrow(\boldsymbol{a}))$ if $\mathsf{sel}_c(\mathsf{ext}(\downarrow(\boldsymbol{a}))) = \mathsf{no\text{-}match}$, and this also holds for the constants d.

Case $(e\boldsymbol{M}\boldsymbol{P})^\tau$. Since $e\boldsymbol{M}\boldsymbol{P}$ is normal, we again have $\mathsf{sel}_e(\boldsymbol{M}) = \mathsf{no\text{-}match}$. By induction hypothesis and (4) $\mathsf{sel}_e(\llbracket \boldsymbol{M} \rrbracket_{\uparrow}(0)) = \mathsf{sel}_e(\boldsymbol{M}^\infty(0)) = \mathsf{sel}_e(\boldsymbol{M}) = \mathsf{no\text{-}match}$. The proof then proceeds as before.

Case $(f\boldsymbol{M})^\tau$. Since $f\boldsymbol{M}$ is normal, we have $f\boldsymbol{M} = \mathsf{norm}(f\boldsymbol{M})$, hence

$$\begin{aligned}
[\![f\boldsymbol{M}]\!]_\uparrow(k) &= \mathcal{I}(f)([\![\boldsymbol{M}]\!]_\uparrow)(k) \\
&= \mathsf{norm}(f[\![\boldsymbol{M}]\!]_\uparrow(k)) \\
&= \mathsf{norm}(f\boldsymbol{M}^\infty(k)) && \text{by IH} \\
&= \mathsf{norm}((f\boldsymbol{M})^\infty(k)) \\
&= \mathsf{norm}(f\boldsymbol{M})^\infty(k) && \text{by (6)} \\
&= (f\boldsymbol{M})^\infty(k).
\end{aligned}$$

□

Theorem 2. *If $M \longrightarrow Q$ w.r.t. all types of rules above, then $\downarrow([\![M]\!]_\uparrow) = Q^\infty$.*

Proof. As for theorem 1; we only have to add an argument for the new rules in case Red.

Case $d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$ for Red, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]] \qquad P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\boldsymbol{P} \longrightarrow Q}{d\boldsymbol{M}\boldsymbol{P} \longrightarrow Q}$$

where $\mathsf{sel}_d(\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]) = d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$, i.e. $d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$ is the selected rule. Recall

$$[\![d\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow = \mathcal{I}(d)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow).$$

By induction hypothesis we have $\downarrow([\![\boldsymbol{M}]\!]_\uparrow) = \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]^\infty$. Now by lemma 3 $\mathsf{ext}(\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]^\infty) = \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]$, so

$$\mathsf{sel}_d(\mathsf{ext}(\downarrow([\![\boldsymbol{M}]\!]_\uparrow))) = \mathsf{sel}_d(\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]) = d\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}] \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}].$$

Hence

$$\begin{aligned}
[\![d\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow &= \mathcal{I}(d)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow) \\
&= [\![P]\!]_{[\boldsymbol{z}\mapsto \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{by definition of } \mathcal{I}(d) \\
&= [\![P]\!]_{[\boldsymbol{z}\mapsto [\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{see below} \\
&= [\![P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]]\!]_\uparrow([\![\boldsymbol{P}]\!]_\uparrow) && \text{by the substitution lemma} \\
&= [\![P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\boldsymbol{P}]\!]_\uparrow \\
&= Q^\infty && \text{by IH.}
\end{aligned}$$

It remains to show that $\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty = [\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow$. Now $\boldsymbol{M} \longrightarrow \boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}]_{\boldsymbol{x}}[\boldsymbol{L}]$, hence $\boldsymbol{N}_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]] \in$ Lnf, and so all subterms of ground type, in particular $\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]$, are also in Lnf. Hence $[\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow = \downarrow([\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow) = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty$ by lemma 8.

Case $e\boldsymbol{K} \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$ for Red, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}] \qquad P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\boldsymbol{P} \longrightarrow Q}{e\boldsymbol{M}\boldsymbol{P} \longrightarrow Q}$$

where $e\boldsymbol{K} \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$ is the selected rule, i.e. $\mathsf{sel}_e(\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]) = e\boldsymbol{K} \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$. By induction hypothesis we have $[\![\boldsymbol{M}]\!]_\uparrow = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty$, hence $\mathsf{sel}_e([\![M]\!]_\uparrow(0)) = \mathsf{sel}_e(\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty(0)) = \mathsf{sel}_e(\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]) = e\boldsymbol{K} \longmapsto P_{\boldsymbol{z}}[\boldsymbol{K}]$ by (4). Therefore

$$\begin{aligned}
[\![e\boldsymbol{M}\boldsymbol{P}]\!]_\uparrow &= \mathcal{I}(e)([\![\boldsymbol{M}]\!]_\uparrow)([\![\boldsymbol{P}]\!]_\uparrow) \\
&= [\![P]\!]_{[\boldsymbol{z}\mapsto[\![\boldsymbol{M}]\!]_\uparrow]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{by definition of } \mathcal{I}(e) \\
&= [\![P]\!]_{[\boldsymbol{z}\mapsto \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{by IH} \\
&= [\![P]\!]_{[\boldsymbol{z}\mapsto[\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow]}([\![\boldsymbol{P}]\!]_\uparrow) && \text{since } [\![\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\!]_\uparrow = \boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]^\infty \text{ by lemma 8} \\
&= [\![P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]]\!]_\uparrow([\![\boldsymbol{P}]\!]_\uparrow) && \text{by the substitution lemma} \\
&= [\![P_{\boldsymbol{z}}[\boldsymbol{K}_{\boldsymbol{x}}[\boldsymbol{L}]]\boldsymbol{P}]\!]_\uparrow \\
&= Q^\infty && \text{by IH.}
\end{aligned}$$

Case $f\boldsymbol{K} \longrightarrow \mathsf{norm}(f\boldsymbol{K})$ for RED, i.e.

$$\frac{\boldsymbol{M} \longrightarrow \boldsymbol{K} \qquad \mathsf{norm}(f\boldsymbol{K}) \longrightarrow Q}{f\boldsymbol{M} \longrightarrow Q}$$

Note that $f\boldsymbol{K}$ is cde-normal, hence by (7) $\mathsf{norm}(f\boldsymbol{K})$ is cde-normal as well. Moreover because of (5) we have $\mathsf{norm}^2(f\boldsymbol{K}) = \mathsf{norm}(f\boldsymbol{K})$, so $\mathsf{norm}(f\boldsymbol{K})$ is also normal w.r.t. the f-rules. Now by induction on the rules for $\longrightarrow$ one can see at once that for normal M, $M \longrightarrow Q$ implies $M = Q$. Therefore, in our instance of the rule RED above we actually have $\mathsf{norm}(f\boldsymbol{K}) = Q$. Now we obtain

$$\begin{aligned}
[\![f\boldsymbol{M}]\!]_\uparrow(k) &= \mathcal{I}(f)([\![\boldsymbol{M}]\!]_\uparrow)(k) \\
&= \mathsf{norm}(f[\![\boldsymbol{M}]\!]_\uparrow(k)) && \text{by definition of } \mathcal{I}(f) \\
&= \mathsf{norm}(f\boldsymbol{K}^\infty(k)) && \text{by IH} \\
&= \mathsf{norm}((f\boldsymbol{K})^\infty(k)) \\
&= \mathsf{norm}(f\boldsymbol{K})^\infty(k) && \text{by (6)} \\
&= Q^\infty(k).
\end{aligned}$$

□

5 Comparison

The difference of the present approach to those of other authors is mainly that $\to$ is modelled simply by the function space. This means that we only use the properties of application and abstraction given in the definition of a cartesian closed category. Due to this fact we can use an internal evaluation of a programming language like SCHEME. In other approaches [1, 5, 7] evaluation has to be defined by hand. A comparison between an internal and a hand made evaluation shows that the former is much more efficient. We tested this in SCHEME and present a table with the run-times for some appropriate examples below.

5.1 Related work

In [1] ALTENKIRCH, HOFMANN and STREICHER give a categorical explanation of normalization by evaluation. Therefore they describe terms as morphisms from a context to their types. The model is not an arbitrary cartesian closed category but a presheaf over a category W of weakenings with contexts as objects. Roughly speaking weakenings are projections on finite sequences of variables and a presheaf can be seen as a variable set or a proof relevant KRIPKE structure. In the interpretation of $\Lambda_{\rho\to\sigma}$ objects depend on $[\![\Lambda_\rho]\!]$ as well as on the morphisms of W (cf. the implication in a KRIPKE structure). In their proof they make use of a category that has both, a semantical and a syntactical component, called glueing model. They also describe a correspondence between intuitionistic completeness proofs and normalization.

CUBRIC, DYBJER and SCOTT [7] have a similar aim, namely to give a normalization proof with as little syntactic properties as possible. They also describe terms as morphisms and use a presheaf model over these terms, but here categories are equipped with a partial equivalence relation. So the function $\downarrow$ (called quote in [1, 5, 7]) appears as natural isomorphism between the YONEDA embedding and an interpretation, that interprets atoms by YONEDA. The function quote and its inverse are used when the universal property of the category of terms is shown. In the appendix they investigate some λ-theories, but it seems that these are too restricted for practical purposes.

COQUAND and DYBJER [5] also interpret terms in a model and reconstruct the normal forms via the function quote. The λ-calculus is given by S- and K-combinators, so they get slightly different normal forms. The model is a glueing model, therefore it is not extensional and application is application of the semantic part. They emphasize that they use an intuitionistic metalanguage and have implemented the algorithm in ALF.

DANVY [8] successfully uses the $\downarrow$ and $\uparrow$ functions for partial evaluation, in particular for compiler construction. His setting is the two-level λ-calculus.

5.2 Comparison of algorithms

We tested different ways to normalize simply typed λ-terms. The first is normalization by evaluation using an internal eval function of the given programming language. The second is normalization by evaluation with a user defined eval function. The third is normalization by means of a user defined β-conversion. In situations where application is not the usual one, e.g. in presheaf models or in glueing models, and for typed programming language like Standard ML, whose eval is not accessible to the user, the first way is excluded.

To test the different normalization algorithms we used iterated functions, i.e. $M_{nm} := (\mathsf{it}_n^3 \mathsf{it}_m^2)(\lambda x^0 x)^1$, where $\mathsf{it}_n^{i+2} := \lambda f^{i+1} \lambda x^i (f \ldots (fx) \ldots)$ with n occurrences of f. Here the type 0 is any ground type and $i+1 := i \to i$. So it_n^i is of type i and M_{nm} is of type 1 with $\lambda x^0 x$ as its normal form. The point in these examples is that the result is small (always $\lambda x^0 x$), but due to iterated function applications it takes many steps to reach the normal form. The table shows that normalization by evaluation with an internal evaluation is about twenty times faster than the version with self-made evaluation, and this again is faster than a recursively defined normalization. The run-times are given in seconds resp.

minutes and seconds:

	normalization by an internal evaluation	normalization by a defined evaluation	recursive normalization
M_{45}	0	1	3
M_{55}	0	2	14
M_{56}	0	6	stack overflow
M_{66}	2	36	
M_{67}	4	1:27	
M_{76}	10	3:34	
M_{77}	31	10:13	
M_{78}	1:16	25:28	
M_{88}	10:12	207:59	

Now we give the main definitions of an implementation in SCHEME. We restrict ourselves to the case of closed terms since by λ-abstraction we can bind all free variables. First our $\downarrow$ and $\uparrow$-functions:

```
(define (reify type)
  (if (ground-type? type)
      (lambda (x) x)
      (let ((reflect-rho (reflect (arg-type type)))
            (reify-sigma (reify (val-type type))))
        (lambda (a)
          (lambda (k)
            (let ((xk (mvar k)))
              (abst xk ((reify-sigma
                         (a (reflect-rho (lambda (l) xk))))
                        (+ k 1)))))))))

(define (reflect type)
  (if (ground-type? type)
      (lambda (x) x)
      (let ((reify-rho (reify (arg-type type)))
            (reflect-sigma (reflect (val-type type))))
        (lambda (r)
          (lambda (a)
            (reflect-sigma
             (lambda (k)
               (app (r k) ((reify-rho a) k)))))))))
```

Normalization with internal evaluation:

```
(define (ev1 x) (eval x (the-environment)))
```

```
(define (norm1 r rho) (((reify rho) (ev1 r)) 0))
```

Normalization with defined evaluation:

```
(define (ev2 M)
  (lambda (env)
    (cond ((variable? M) (cadr (assq M env)))
          ((application? M)
           (((ev2 (operator M)) env) ((ev2 (argument M)) env)))
          ((abstraction? M)
           (lambda (a) ((ev2 (kernel M))
                        (cons (list (abstvar M) a) env))))
          (else #f))))

(define (norm2 r rho) (((reify rho) ((ev2 r) ())) 0))
```

Finally a recursive normalization:

```
(define (norm3 r)
  (cond ((variable? r) r)
        ((application? r)
         (let ((op (norm3 (operator r)))
               (arg (norm3 (argument r))))
           (if (abstraction? op)
               (let ((x (abstvar op))
                     (s (kernel op)))
                 (norm3 (substitute s x arg)))
               (app op arg))))
        ((abstraction? r)
         (abst (abstvar r) (norm3 (kernel r))))))
```

The auxiliary definitions are not mentioned and hopefully obvious. So for instance `mvar` produces variables x_k from k, `abst` constructs a λ-abstraction $\lambda x_k M$ from x_k and M, `app` constructs an application MN from M and N, and `abstvar` resp. `kernel` gets out x_k resp. M of $\lambda x_k M$.

References

1. Thorsten Altenkirch, Martin Hofmann, and Thomas Streicher. Categorical reconstruction of a reduction free normalization proof. In *CTCS'95, Cambridge*, volume 953 of *Lecture Notes in Computer Science*, pages 182–199. Springer Verlag, Berlin, Heidelberg, New York, 1995.

2. Holger Benl, Ulrich Berger, Helmut Schwichtenberg, Monika Seisenberger, and Wolfgang Zuber. Proof theory at work: Program development in the Minlog system. To appear: Automated Deduction – A Basis for Applications (eds W. Bibel and P. Schmitt), Volume I: Foundations, Kluwer Academic Publishers, 1998.
3. Ulrich Berger. Continuous functionals of dependent and transfinite types. Habilitationsschrift, Mathematisches Institut der Universität München, 1997.
4. Ulrich Berger and Helmut Schwichtenberg. An inverse of the evaluation functional for typed λ–calculus. In R. Vemuri, editor, *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 203–211. IEEE Computer Society Press, Los Alamitos, 1991.
5. Thierry Coquand and Peter Dybjer. Intuitionistic model constructions and normalization proofs. *Mathematical Structures in Computer Science*, 7:73–94, 1997.
6. Roy L. Crole. *Categories for Types*. Cambridge University Press, 1993.
7. Djordje Cubric, Peter Dybjer, and Philip Scott. Normalization and the Yoneda embedding. To appear: Mathematical Structures in Computer Science, 1998.
8. Olivier Danvy. Pragmatics of type-directed partial evaluation. In O. Danvy, R. Glück, and P. Thiemann, editors, *Partial Evaluation*, volume 1110 of *LNCS*, pages 73–94. Springer Verlag, Berlin, Heidelberg, New York, 1996.
9. N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Math.*, 34:381–392, 1972.
10. J. Lambek and P. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.
11. John McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195, 1960.
12. Helmut Schwichtenberg and Karl Stroetmann. From higher order terms to circuits. In M.L. Dalla Chiara, K. Doets, D. Mundici, and J. van Benthem, editors, *Logic and Scientific Methods. Proceedings (Vol. 1) of the Tenth International Congress of Logic, Methodology and Philosophy of Science, Florence, August 1995*, volume 259 of *Synthese Library*, pages 209–220, Dordrecht, Boston, London, 1997. Kluwer Academic Publishers.