A self-interpreter of lambda calculus having a normal form

Alessandro Berarducci *
Università di Pisa, Dipartimento di Matematica
Via Buonarroti 2, 56100 Pisa, Italy.

Corrado Böhm †
Università di Roma, Dipartimento di Scienze dell'Informazione,
Via Salaria 113, 00198 Roma, Italy.

Abstract

We formalize a technique introduced by Böhm and Piperno to solve systems of recursive equations in lambda calculus without the use of the fixed point combinator and using only normal forms. To this aim we introduce the notion of a canonical algebraic term rewriting system, and we show that any such system can be interpreted in the lambda calculus by the Böhm - Piperno technique in such a way that strong normalization is preserved. This allows us to improve some recent results of Mogensen concerning efficient gödelizations $\lceil\ \rceil : \Lambda \to \Lambda$ of lambda calculus. In particular we prove that under a suitable gödelization there exist two lambda terms $\mathbf{E}$ (self-interpreter) and $\mathbf{R}$ (reductor), both having a normal form, such that for every (closed or open) lambda term M $\mathbf{E}\lceil M\rceil \to M$ and if M has a normal form N, then $\mathbf{R}\lceil M\rceil \to \lceil N\rceil$.

1 Introduction and summary

Λ is the set of all lambda terms. Λ_0 is the set of all closed lambda terms. The sign "$=$" between lambda terms denotes beta-convertibility (also written "$=_\beta$"), $\to$ is beta-reduction (possibly multistep), and "$\equiv$" is alpha convertibility, namely identity up to renamings of bound variables. $NF \subseteq \Lambda$ is the set of normal (i.e. irreducible) lambda terms, and $SN \subseteq \Lambda$ is the set of all strong normalizing lambda terms (i.e. all the reduction paths are finite).

In the first part of the paper we study the problem of interpreting an equational theory inside lambda calculus (section 2), and of interpreting a term

*Partially supported by MURST Research projects 40% "Modelli della computazione e dei linguaggi di programmazione", and 60% "Specifiche, Concorrenza e Logica computazionale", while the first author was holding a research position at the University of L'Aquila.

†Partially supported by MURST Research projects 40% "Fondamenti dei linguaggi funzionali e logici", and 60% "Progetto Ateneo".

rewriting system (TRS) inside lambda calculus (section 3). In the second part of the paper we apply our results to find a "self-interpreter" and a "reductor" of lambda calculus having a normal form. This improves some results of [1] and [9]. We consider equational theories and TRS's which are not necessarily first order, namely they involve some lambda abstractions and applications besides the usual rules for the formation of first order terms (on a given signature Σ). Interpreting an equational theory corresponds to solving a system of equations inside lambda calculus involving certain unknown lambda terms to be found (represented by the symbols of the given signature Σ). Statman proved that there is no algorithm which, given $F, G \in \Lambda$ decides whether there is $X \in \Lambda$ such that $FX = G$ (cf. [6]). So the general problem of solving equations in Λ is undecidable. In section 2 we define the notion of a *canonical* set (or "system") of equations and we prove (Theorem 2.5) that every canonical set of equations can be interpreted inside lambda calculus. This is essentially a formalization of a technique to solve recursive equations inside lambda calculus due to Böhm and Piperno, namely we prove that the Böhm-Piperno technique captures at least the canonical systems of equations. We are particularly interested in studying those systems of equations which admit a normal form solution (i.e. all the unknowns must be replaced by lambda terms in normal form). We show that if a canonical set of equations $\mathcal{E}$ is *algebraic* (i.e. first order), then the Böhm-Piperno technique yields a normal form solution of $\mathcal{E}$ (Theorem 2.5 part 2). Such canonical-algebraic systems are quite powerful since they can be used to find lambda representations of any partial recursive function (even on non-numerical data structures). Note that a normal form solution excludes the use of the fixed point combinator. Solvability problems of systems of equations have been extensively studied in [5, 6, 7]. Here we deal with a more restricted class of systems of equations suitable for functional programming. So our approach is more in the spirit of [4]. Our motivation is to represent programs and data-structures inside lambda calculus in such a way that the lambda-representation of every partial recursive function on such data-structures assumes a very simple form and can be automatically derived from an algebraic specification. In section 3 we strengthen the results of section 2 by considering TRS's instead of equational theories. The general problem of which TRS can be intepreted inside lambda calculus was raised in [8]. We show that every canonical TRS can be intepreted and every canonical algebraic TRS can be intepreted in a nice way, namely preserving strong normalization (Theorem 3.4). Starting with section 4 we begin the second part of the paper in which we study "efficient" gödelizations of lambda calculus. Given a lambda term M, let $\lceil M \rceil$ be the Church numeral of the Gödel number of M (with respect to a suitable Gödel numbering). Note that: 1. if $M \not\equiv N$, then $\lceil M \rceil \neq \lceil N \rceil$ (i.e. $\lceil M \rceil$ and $\lceil N \rceil$ are not beta-convertible). This seems to be an obvious requirement for any reasonable gödelization. From the representability of partial recursive functions in the lambda calculus, it follows that there is a lambda term $\mathbf{R}$ ("reductor"), such that: 2. if $M \in \Lambda_0$ has a normal form N, then $\mathbf{R}(\lceil M \rceil) = \lceil N \rceil$ (cf. [2]). Moreover there exists a lambda term $\mathbf{E}$ ("self-interpreter" or "enumerator") such that: 3. $\mathbf{E}(\lceil M \rceil) = M$ for

all $M \in \Lambda_0$ (cf. [1]). We can think of $\mathbf{E}$ as the analogue of a universal Turing machine for the lambda calculus: $\mathbf{E}(\ulcorner M \urcorner)x = Mx$. Any reasonable gödelization should satisfy the second fixed point theorem, namely the analogue of the recursion theorem in the theory of recursive functions: 4. $\forall F \in \Lambda\ \exists t \in \Lambda : F\ulcorner t \urcorner = t$ (cf. [1]). We are not aware of any attempt to give an axiomatic definition of the notion "gödelization of lambda calculus", namely what properties should a map $\ulcorner\ \urcorner : \Lambda \to \Lambda$ satisfy to be considered a gödelization? It seems to us that the fact that the range of $\ulcorner\ \urcorner$ is a set of numerals is not to be included among the desired properties, however it is crucial that if M_1 and M_2 are not identical (i.e. not α-convertible), then $\ulcorner M_1 \urcorner$ and $\ulcorner M_2 \urcorner$ are not β-convertible. Mogensen [9] defined an innovative "gödelization" (there called "representation") $\ulcorner\ \urcorner : \Lambda \to \Lambda$ in which $\ulcorner M \urcorner$, far from beeing a numeral, is a lambda term with the same free variables as M. Mogensen's gödelization satisfies the above properties 1, 2, 3, 4 and has several advantages over the conventional gödelizations, for instance property 3, i.e. $\mathbf{E}(\ulcorner M \urcorner) = M$, holds for any term $M \in \Lambda$, not necessarily closed. Mogensen's gödelization commutes both with substitutions of variables and α-conversion: $\ulcorner M \urcorner[x := y] \equiv \ulcorner M[x := y] \urcorner$ and α-convertible terms have α-convertible gödelizations. Even more strikingly, a lambda abstraction "λx" goes unchanged under the gödelization, i.e. $\ulcorner \lambda x.M \urcorner \equiv U(\lambda x.\ulcorner M \urcorner)$ where U is a suitable left-invertible lambda term. We push Mogensen's approach to its extreme consequences by letting the godelization of a bound variable to be the variable itself. This almost paradoxical move, together with an application of Theorem 2.5, will greatly simplify all the theorems concerning gödelizations, like the second fixed point theorem, the existence a reductor $\mathbf{R}$ and of an enumerator (or "self-interpreter") $\mathbf{E}$, etc. In particular we will define $\mathbf{E}$ and $\mathbf{R}$ as lambda terms having a normal form. This is quite surprising because it means that we can solve the necessarily involved recursive definitions needed to define $\mathbf{R}$ and $\mathbf{E}$ without using any fixed point combinator. In conclusion we have:

1. If $M_1 \not\equiv M_2$, then $\ulcorner M_1 \urcorner$ and $\ulcorner M_2 \urcorner$ are not beta-convertible.
2. If $M \in \Lambda$, then $\mathbf{E}(\ulcorner M \urcorner) \to M$.
3. If $M \in \Lambda$ has a normal form N, then $\mathbf{R}(\ulcorner M \urcorner) \to \ulcorner N \urcorner$.
4. $\mathbf{E}$ and $\mathbf{R}$ have a normal form.
5. If M is strong normalizing, then $\mathbf{E}(\ulcorner M \urcorner)$ and $\mathbf{R}(\ulcorner M \urcorner)$ are strong normalizing.
6. The second fixed point theorem holds for $\ulcorner\ \urcorner$.

It should be remarked that points 2 and 3 hold even when M is an open lambda term. Under Mogensen's approach, points 1, 2 and 6 hold, points 4 and 5 fail, and 3 holds restricted to closed lambda terms. We also consider a variant of our gödelization, closer to Mogensen's one, in which there is a self-interpreter given by $\mathbf{E} = \lambda x.x < \mathbf{K}, \mathbf{S}, \mathbf{C} >$ where $< \mathbf{K}, \mathbf{S}, \mathbf{C} >$ is the Church triple of the well known combinators $\mathbf{K} = \lambda xy.x$, $\mathbf{S} = \lambda xyz.xz(yz)$, $\mathbf{C} = \lambda xyz.xzy$. This is

a quite simple solution for a lambda term which plays the role of a universal Turing machine.

2 Solving equations inside lambda calculus

Let Σ be a set of function symbols from a given signature. $\Lambda(\Sigma)$ denotes the set of *extended lambda terms* with symbols from the signature Σ. To be precise $\Lambda(\Sigma)$ can be defined by adding the following clause to the usual clauses for the formation of lambda terms: if $t_1, \ldots, t_n \in \Lambda(\Sigma)$ and $f \in \Sigma$ is an n-ary function symbols, then $f(t_1, \ldots, t_n) \in \Lambda(\Sigma)$. Note that $\mathrm{Ter}(\Sigma) \subseteq \Lambda(\Sigma)$ where $\mathrm{Ter}(\Sigma)$ is the set of first order terms with signature Σ. A *representation* of the signature Σ in Λ is a function

$$\phi : \Sigma \to \Lambda$$

Any such representation ϕ induces a map $\hat{\phi} : \Lambda(\Sigma) \to \Lambda$ in the obvious way, namely $\hat{\phi}(\lambda x.M) \equiv \lambda x.\hat{\phi}(M)$, $\hat{\phi}(MN) \equiv \hat{\phi}(M)\hat{\phi}(N)$ and for $f \in \Sigma$, $\hat{\phi}(f(M_1, \ldots, M_n)) \equiv \phi(f)\hat{\phi}(M_1)\ldots\hat{\phi}(M_n)$, abbreviated as $f^\phi M_1^\phi \ldots M_n^\phi$ (as usual in lambda calculus the omitted parenthesis are associated to the left). So if $\Sigma = \{0, f, g\}$, $(\lambda x.xf(x, g(0, x)))^\phi \equiv \lambda x.x(f^\phi x(g^\phi 0^\phi x))$. Note that $xf(x, g(0, x))$ denotes unambiguosly x applied to $f(x, g(0, x))$. Given a set of equations $\mathcal{E} = \{a_i = b_i | i \in J\}$ between extended lambda terms $a_i, b_i \in \Lambda(\Sigma)$, we say that a representation ϕ *satisfies* (or *solves*) $\mathcal{E}$ if for each equation $a_i = b_i$ in $\mathcal{E}$ we have $\hat{\phi}(a_i) =_\beta \hat{\phi}(b_i)$. If there exists a representation ϕ which satisfies $\mathcal{E}$ we say that $\mathcal{E}$ can be *interpreted* (or *represented* or *solved*) inside lambda calculus. We will only consider representations ϕ with range included in the set of closed lambda terms.

Definition 2.1 Let $\mathcal{E}$ be a set of equations in $\Lambda(\Sigma)$. We say that $\mathcal{E}$ is *canonical* if the function symbols in Σ can be partitioned in two disjoint subsets $\Sigma = \Sigma_0 \cup \Sigma_1$ so that, letting $\Sigma_0 = \{c_1, \ldots, c_r\}$ and $\Sigma_1 = \{f_1, \ldots, f_k\}$, each equation $t = t'$ of $\mathcal{E}$ has the form $f_i(c_j(x_1, \ldots, x_m), y_1, \ldots, y_n) = b_{i,j}$ where $f_i \in \Sigma_1$, $c_j \in \Sigma_0$, $b_{i,j} \in \Lambda(\Sigma)$ is a term depending on i and j, $n, m \geq 0$, the variables $x_1, \ldots, x_m$, $y_1, \ldots, y_n$ are all distinct and the free variables of $b_{i,j}$ are included among $x_1, \ldots, x_m, y_1, \ldots, y_n$. We call the elements of Σ_0 *constructors* and those of Σ_1 *programs*. We say that $\mathcal{E}$ is *complete* if for all $f_i \in \Sigma_1$ and $c_j \in \Sigma_0$, $\mathcal{E}$ contains an equation of the form $f_i(c_j(x_1, \ldots, x_m), y_1, \ldots, y_n) = b_{i,j}$.

Note that we allow some lambda abstractions and applications to appear on the right-hand-sides $b_{i,j}$ of a canonical system but not on the left-hand-sides.

Definition 2.2 A system of equations $\mathcal{E}$ is called *algebraic* if for each equation $a = b$ in $\mathcal{E}$, a and b belong to $\mathrm{Ter}(\Sigma)$ (note that for a canonical system we only have $a \in \mathrm{Ter}(\Sigma)$ and $b \in \Lambda(\Sigma)$). So an algebraic system of equations is simply an equational theory in the usual sense of first order logic, and a solution $\phi : \Sigma \to \Lambda$ of an algebraic system is an interpretation of $\mathcal{E}$ inside lambda calculus. A *canonical algebraic* system is a system which is both canonical and algebraic.

Example 2.3 The Ackermann's function can be defined by the following algebraic system (where 0 and s represent the constant zero and the successor function). 1. $Ack(0,y) = s(y)$, 2. $Ack(s(x),0) = Ack(x,s(0))$, 3. $Ack(s(x),s(y)) = Ack(x,Ack(s(x),y))$. The above system is not canonical, but it can be reduced to a canonical (and algebraic) system by enlarging the signature with a new function symbol f as follows. 1. $Ack(0,y) = s(y)$, 2. $Ack(s(x),y) = f(y,x)$, 3. $f(0,x) = Ack(x,s(0))$, 4. $f(s(z),x) = Ack(x,Ack(s(x),z))$. Similarly every partial recursive function can be defined by a canonical algebraic system (it is enough to verify closure under minimalization).

Remark 2.4 We can always enlarge a canonical system of equations by additional equations which force the constructors to by injective. For instance in the case of an n-ary constructor c we can add n program symbols $p_1,\ldots,p_n$ and the (canonical) equations $p_i(c(x_1,\ldots,x_n)) = x_i$. In all the applications we will give the constructors will always be interpreted as injective functions.

Theorem 2.5 *1. Any canonical system of equations $\mathcal{E}$ has a solution $\phi : \Sigma \to \Lambda$ inside lambda calculus.*

2. Any canonical algebraic system $\mathcal{E}$ has a solution $\phi : \Sigma \to \Lambda$ with range included in SN.

3. Moreover we can choose ϕ so that the restriction $\phi|\Sigma_0$ depends only on Σ_0 and not on $\mathcal{E}$, namely there is a fixed representation of the constructors.

The representation of the constructors $\phi|\Sigma_0$ is due to Böhm and Piperno and is defined as follows.

Definition 2.6 Suppose $\Sigma_0 = \{c_1,c_2,\ldots,c_r\}$. For $1 \le j \le r$ we define $\phi(c_j) \equiv \lambda x_1 \ldots x_n e.eU_j^r x_1 \ldots x_n e$ where n is the arity of c_j and $U_j^r \equiv \lambda x_1 \ldots x_r.x_j$. We call this representation the *canonical* representation of the constructors. Note that c_j^ϕ is injective.

It remains to define $\phi|\Sigma_1$, namely the representation of the programs. Without loss of generality we can assume that $\mathcal{E}$ is complete (otherwise adjoin more equations to make it complete).

Definition 2.7 Suppose $\Sigma_1 = \{f_1,\ldots,f_k\}$ and $\Sigma_0 = \{c_1,\ldots,c_r\}$. Consider $k \times r$ lambda terms $t_{i,j}$, $1 \le i \le k$, $1 \le j \le r$ to be defined later. Recall the definition of *Church r-tuple*: $< M_1,\ldots,M_r > \equiv \lambda x.xM_1 \ldots M_r$. For $1 \le i \le k$ let

$$t_i \equiv < t_{i,1},\ldots,t_{i,r} >$$

and define:

$$\phi(f_1) \equiv < t_1,t_2,\ldots,t_k >$$

and for $i > 1$

$$\phi(f_i) \equiv < t_i, t_{i+1}, \ldots, t_{i-1} >$$

where $< t_i, t_{i+1}, \ldots, t_{i-1} >$ is the cyclic permutation of $< t_1, t_2, \ldots, t_k >$ beginning with t_i. By abuse of notation when $i = 1$ $< t_i, t_{i+1}, \ldots, t_{i-1} >$ denotes $< t_1, t_2, \ldots, t_k >$. Thus $\phi(f_i)$ is a Church k-tuple of Church r-tuples of lambda terms. The lambda terms $t_{i,j}$ are chosen in the only natural way which makes ϕ a solution of the canonical system of equations $\mathcal{E}$. More precisely consider a canonical equation $f_i(c_j(x_1, \ldots, x_m), y_1, \ldots, y_n) = b_{i,j}$ belonging to $\mathcal{E}$ ($b_{i,j} \in \Lambda(\Sigma)$). After applying ϕ the equation becomes:

$$< t_i, t_{i+1}, \ldots, t_{i-1} > (c_j^\phi \vec{x})\vec{y} = b_{i,j}^\phi.$$

By definition of Church tuple, this simplifies to:

$$c_j^\phi \vec{x} t_i t_{i+1} \ldots t_{i-1} \vec{y} = b_{i,j}^\phi.$$

Recalling the definition of c_j^ϕ we have $c_j^\phi \vec{x} t_i = t_i U_j^r \vec{x} t_i = t_{i,j} \vec{x} t_i$. Hence the equation becomes:

$$t_{i,j} \vec{x} t_i t_{i+1} \ldots t_{i-1} \vec{y} = b_{i,j}^\phi.$$

We can now solve this equation for $t_{i,j}$ by replacing on both sides all the occurrences of $t_1, \ldots, t_k$ by fresh variables $v_1, \ldots, v_k$ and abstracting with respect to these variables. More precisely define:

$$t_{i,j} \equiv \lambda \vec{x} v_i v_{i+1} \ldots v_{i-1} \vec{y}.(b_{i,j}^\psi)^\phi$$

where $\psi : \Sigma_1 \to \Lambda$ is defined by $\psi(f_s) = < v_s, v_{s+1}, \ldots, v_{s-1} >$ $(s = 1, \ldots, k)$. With this definition $t_{i,j} \vec{x} t_i t_{i+1} \ldots t_{i-1} \vec{y} \to b_{i,j}^\phi$ and all the equations will be satisfied.

For an illustration of the above definitions with $k = 1$ and $r = 3$ see the computations following Theorem 4.5. We have thus proved that every canonical system has a solution inside lambda calculus. To complete the proof of Theorem 2.5 it remains to prove that if $\mathcal{E}$ is canonical algebraic, then ϕ has range included in SN. A useful tool to prove strong normalization is the notion of perpetual reduction.

Definition 2.8 $\infty(M)$ means that M is not strong normalizing, namely $M \notin SN$. A *perpetual reduction* is a reduction $M \to N$, such that $\infty(M)$ implies $\infty(N)$.

In [1] we find the related notion of *perpetual redex*. A perpetual redex is a lambda term $\Delta = (\lambda x.P)Q$ such that any beta-reduction of the form $C[(\lambda x.P)Q] \to C[P[x := Q]]$ (for an arbitrary context $C[\,]$) is perpetual, namely $\infty(C[(\lambda x.P)Q]) \to \infty(C[P[x := Q]])$. It is known that I-redexes are always perpetual (see [1] Theorem 13.4.12). A theorem of Bergstra and Klop [3] characterizes the perpetual K-redexes (see also [1] Theorem 13.4.15). In particular the result of Bergstra and Klop implies the following:

Proposition 2.9 *Let $\Delta \equiv (\lambda x.P)Q$ where Q is a strong normalizing closed lambda term. Then Δ is a perpetual redex.*

Now we proceed to prove part 2 of Theorem 2.5.

Lemma 2.10 *Let $\mathcal{E}$ be a canonical set of equations and let ϕ and $t_{i,j}$ be as in the proof of Theorem 2.5. Then ϕ has range included in SN iff $t_{i,j} \in SN$ for all i's and j's.*

Proof. First note that for $c \in \Sigma_0$, c^ϕ is always a normal form. On the other hand for $f \in \Sigma_1$, f^ϕ is a Church k-tuple of Church r-tuples of the $t_{i,j}$'s. So if all the $t_{i,j}$'s are in SN, so are all the f^ϕ's (and conversely). QED

Lemma 2.11 *If $\mathcal{E}$ is a complete canonical algebraic system of equations and ϕ is the solution of $\mathcal{E}$ constructed in Theorem 2.5, then the terms $t_{i,j}$ defined in the proof of Theorem 2.5 are strong normalizing and closed.*

Proof. The fact that $t_{i,j}$ is a closed lambda term follows easily from the fact that in a canonical system of equations the free variables of the right-hand-sides are included among the free variables of the corresponding left-hand-sides. Recalling the definition of $t_{i,j}$, in order to prove $t_{i,j} \in SN$ it suffices to prove that $(b_{i,j}^\psi)^\phi \in SN$ ($\psi : \Sigma_1 \to \Lambda$ and $\phi : \Sigma \to \Lambda$ are as in the proof of Theorem 2.5). We will prove the stronger claim that $(P^\psi)^\phi \in SN$ for every $P \in \mathrm{Ter}(\Sigma)$. Write $P^{\psi\phi}$ for $(P^\psi)^\phi$. The proof is by induction on the length of P distinguishing the cases in which the outermost function symbol of P is in Σ_0 or in Σ_1. For the induction to work we need to prove a stronger result, namely we prove that: 1) $P^{\psi\phi} \in SN$ and 2) the normal form of $P^{\psi\phi}$ is either of the form $\lambda x.xM_1 \dots M_u$ or of the form $xM_1 \dots M_u$ and $P^{\psi\phi}$ can be reduced to its normal form by reducing I-redexes only (so perpetual redexes).

Case 1: P is $c(P_1, \dots, P_n)$ where $c \in \Sigma_0$. Say $c = c_j$. Then $P^{\psi\phi} \equiv c_j^\phi P_1^{\psi\phi} \dots P_n^{\psi\phi} \to \lambda e.eU_j^r P_1^{\psi\phi} \dots P_n^{\psi\phi}$. Note that the displayed reduction contracts only I-redexes. By induction hypothesis all the $P_i^{\psi\phi}$'s belong to SN and satisfy (2). Thus clearly $P^{\psi\phi} \in SN$ and $P^{\psi\phi}$ satisfies (2).

Case 2. P is $f(P_1, \dots, P_n)$ where $f \in \Sigma_1$. Say $f = f_1$ (the other cases are similar). Then $P^{\psi\phi} \equiv f^\psi P_1^{\psi\phi} \dots P_n^{\psi\phi} \equiv < v_1, v_2, \dots, v_k > P_1^{\psi\phi} \dots P_n^{\psi\phi} \to P_1^{\psi\phi} v_1 v_2 \dots v_k P_2^{\psi\phi} \dots P_n^{\psi\phi}$ where the displayed reduction contracts an I-redex. By induction hypothesis $P_1^{\psi\phi}$ belongs to SN and satisfies (2). Thus $P_1^{\psi\phi}$ reduces, by a sequence of I-reductions, either to a term of the form $xM_1 \dots M_u$ or to a term of the form $\lambda x.xM_1 \dots M_u$. In the first case $P^{\psi\phi} \to xM_1 \dots M_u v_1 v_2 \dots v_k P_2^{\psi\phi} \dots P_n^{\psi\phi}$ by a sequence of I-reductions. In the second case $P^{\psi\phi} \to v_1 M_1[x := v_1] \dots M_u[x := v_1] v_2 \dots v_k P_2^{\psi\phi} \dots P_n^{\psi\phi}$ by a sequence of I-reductions. So in either case $P^{\psi\phi}$ satisfies (1) and (2). QED

The proof of Theorem 2.5 is now complete.

Remark 2.12 In a "real life" situation the signature of constructors Σ_0 will in general contain the constructors of several data structures (numbers, lists, booleans, etc.). According to Definition 2.6 the lambda representation of a constructor depends not only on the data structure to which the given constructor belongs, but also on the cardinality of the whole set of constructors Σ_0. This might be a disadvantage if we later want to expand the signature of constructors by adjoining a new data structure, because we would have to change the lambda representation of the old data structures as well. This problem can be solved by applying Definition 2.6 to each data structure separately rather then to the whole set of constructors. If we do so it might happen that the constructors of two different data structures are represented by the same lambda term. However this will cause no harm if we assume that the programs specified by the equations $\mathcal{E}$ "respect the types" in the sense that if $\mathcal{E}$ contains two equations of the form $f_i(c_j(\vec{x}), \vec{y})$ and $f_i(c_{j'}(\vec{x}), \vec{y})$, then c_j and $c_{j'}$ both belong to the same data structure (this might be taken as a definition of "data structure"). Note that it is allowed that the codomain of f_i belong to a different data structure. In this setting the proof of Theorem 2.5 goes through with minor changes. For instance if the first argument of the program f is of type "integer" (with constructors $\{s, 0\}$), then the lambda representation of f will involve a 2-tuple (since 2 is the cardinality of $\{s, 0\}$) rather than a r-tuple (where $r = |\Sigma_0|$).

Remark 2.13 We say that $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \ldots \cup \mathcal{E}_n$ is *stratified* if $\mathcal{E}_i$ does not use any program symbol mentioned in $\mathcal{E}_{i+1}$ (although $\mathcal{E}_{i+1}$ may use program symbols in $\mathcal{E}_i$). In such a case instead of solving directly $\mathcal{E}$, it is simpler to solve the various $\mathcal{E}_i$'s consecutively. (Of course before solving $\mathcal{E}_{i+1}$ we need to substitute those program symbols of $\mathcal{E}_{i+1}$ which were mentioned in some previous $\mathcal{E}_j$ by the corresponding lambda terms.)

3 Interpreting a term rewriting system inside lambda calculus

In this section we consider the problem of interpreting a *term rewriting system* (TRS) inside lambda calculus. We strengthen the results of the previous section by taking into account reductions and not only equalities. First note that the notion of β-reduction "$\rightarrow$" makes sense also for extended lambda terms $t \in \Lambda(\Sigma)$. So for example $(\lambda x.f(x, y))z \rightarrow f(z, y)$. As usual α-convertible terms are considered identical.

Definition 3.1 Given a set of equations $\mathcal{E}$ on $\Lambda(\Sigma)$ we define a notion of reduction by orienting the equations of $\mathcal{E}$ from left to right. So $\mathcal{E}$ generates a binary relation $\rightarrow_{\mathcal{E}}$ on $\Lambda(\Sigma)$ defined as the least transitive relation containing $\rightarrow$ and the oriented equations of $\mathcal{E}$, and closed under substitutions and contexts. So if $\mathcal{E} = \{f(g(x), y) = x\}$, then $(\lambda x.f(x, y))g(\lambda v.v) \rightarrow_{\mathcal{E}} f(g(\lambda v.v), y) \rightarrow_{\mathcal{E}} \lambda v.v$.

Note that $(\mathcal{E}, \rightarrow_{\mathcal{E}})$ can be considered as a generalized TRS in which the terms to be rewritten are in $\Lambda(\Sigma)$ rather than in $\mathrm{Ter}(\Sigma)$. If $\mathcal{E}$ is a set of algebraic

equations and we restrict $\rightarrow_{\mathcal{E}}$ to $\mathrm{Ter}(\Sigma)$, we recover the usual notion of a TRS on $\mathrm{Ter}(\Sigma)$.

Definition 3.2 An *interpretation* of $(\mathcal{E}, \rightarrow_{\mathcal{E}})$ in Λ is a representation $\phi : \Sigma \rightarrow \Lambda$ which solves all the equations of $\mathcal{E}$ (in the sense of section 2) and preserves reduction, namely for all $M, N \in \Lambda(\Sigma)$, if $M \rightarrow_{\mathcal{E}} N$, then $M^{\phi} \rightarrow N^{\phi}$.

Lemma 3.3 *Let $M \in Ter(\Sigma_0)$, and $\phi : \Sigma_0 \rightarrow \Lambda$ be the canonical representation of the constructors (as in Definition 2.6). Then M^{ϕ} is strong normalizing.*

Proof. We reason by induction on the length of M. Suppose that $M \equiv c_j(M_1, \ldots, M_n)$ where $c_j \in \Sigma_0$. By inductive hypothesis $M_1^{\phi}, \ldots, M_n^{\phi}$ are strong normalizing. Recall that $c_j^{\phi} \equiv \lambda x_1 \ldots x_n e.eU_j^r x_1 \ldots x_n e$. Hence $M^{\phi} \equiv c_j^{\phi} M_1^{\phi} \ldots M_n^{\phi} \rightarrow \lambda e.eU_j^n M_1^{\phi} \ldots M_n^{\phi} e$. The latter term is strongly normalizing since each $M_i^{\phi} \in SN$. But then also M^{ϕ} is strongly normalizing since the above reduction is an I-reduction. QED

Theorem 3.4 *Let $\mathcal{E}$ be a canonical system and let $\phi : \Sigma \rightarrow \Lambda$ be the solution constructed in Theorem 2.5. Then:*

1. *ϕ is an interpretation of $(\mathcal{E}, \rightarrow_{\mathcal{E}})$ inside lambda calculus (i.e. if $M \rightarrow_{\mathcal{E}} N$, then $M^{\phi} \rightarrow N^{\phi}$).*
2. *Moreover if $\mathcal{E}$ is complete, canonical and algebraic, then for every closed term $M \in Ter(\Sigma)$, if M is strong normalizing relative to $\rightarrow_{\mathcal{E}}$, then M^{ϕ} is strong normalizing relative to $\rightarrow$ (i.e. ϕ preserves strong normalization of closed algebraic terms).*

Proof. To prove part 1 it is enough to follow the proof of Theorem 2.5 replacing everywhere equalities with reductions. To prove part 2 assume that $\mathcal{E}$ is complete, canonical and algebraic and $M \in \mathrm{Ter}(\Sigma)$ is a closed term which is strong normalizing relative to $\rightarrow_{\mathcal{E}}$. Then there is an integer n such that all the $\mathcal{E}$-reduction paths starting from M have length $\leq n$. We prove that M^{ϕ} is strong normalizing (relative to $\rightarrow$) by induction on n.

If $n = 0$, then it easily follows that $M \in \mathrm{Ter}(\Sigma_0)$ (here we use the assumption that M is closed and $\mathcal{E}$ is complete). Hence by Lemma 3.3 $M^{\phi} \in SN$.

If $n > 0$, then M contains at least one occurrence of a program symbol $f \in \Sigma_1$. By considering an innermost program symbol occurring in M we see M must contain an subterm of the form $f_i(c_j(\vec{X}), \vec{Y})$ where $f_i \in \Sigma_1$, $c_j \in \Sigma_0$ and $\vec{X} = X_1, \ldots, X_m$ and $\vec{Y} = Y_1, \ldots, Y_n$ are finite sequences of closed terms in $\mathrm{Ter}(\Sigma_0)$. Since $\mathcal{E}$ is complete, $\mathcal{E}$ contains an equation of the form $f_i(c_j(x_1, \ldots, x_m), y_1, \ldots, y_n) = b_{i,j}$, so the subterm $f_i(c_j(\vec{X}), \vec{Y})$ is an $(\rightarrow_{\mathcal{E}})$-redex. By contracting it we have $M \rightarrow_{\mathcal{E}} N$ where $N \in \mathrm{Ter}(\Sigma)$ since $\mathcal{E}$ is algebraic. We write $\vec{X}^{\phi}$ for the sequence $X_1^{\phi}, \ldots, X_m^{\phi}$ and similarly for $\vec{Y}^{\phi}$. We then have the following reductions (using the notations of the proof of Theorem 2.5):

$$
\begin{aligned}
&(f_i(c_j(\vec{X}),\vec{Y}))^\phi\\
&\equiv < t_i, t_{i+1}, \ldots, t_{i-1} > (c_j^\phi \vec{X}^\phi)\vec{Y}^\phi\\
&\to c_j^\phi \vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\to t_i U_j^r \vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\equiv < t_{i,1}, t_{i,2}, \ldots, t_{i,r} > U_j^r \vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\to U_j^r t_{i,1} t_{i,2} \ldots t_{i,r} \vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\to t_{i,j} \vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\equiv (\lambda \vec{x} v_i v_{i+1} \ldots v_{i-1} \vec{y}. b_{i,j}^{\psi\phi})\vec{X}^\phi t_i t_{i+1} \ldots t_{i-1}\vec{Y}^\phi\\
&\to b_{i,j}^\phi[\vec{x} := \vec{X}^\phi, \vec{y} := \vec{Y}^\phi]
\end{aligned}
$$

We claim that all the redexes reduced in the above reduction are perpetual. This is clear for the first three reductions since they reduce an I-redex. The last two reductions may reduce some K-redexes, but in any case they can only erase closed terms which are strong normalizing. In fact the terms X_i^ϕ and Y_j^ϕ are strong normalizing by Lemma 3.3 and all the various t_i's and $t_{i,j}$'s are strong normalizing by Lemma 2.11 because the system $\mathcal{E}$ is algebraic. Moreover all such terms are closed because M is closed and ϕ does not introduce free variables. Since N^ϕ is obtained from M^ϕ by the above sequence of reductions (inside a context), we can conclude that there is a perpetual reduction $M^\phi \to N^\phi$. By induction hypothesis N^ϕ is strong normalizing (since N certainly has a lower n). But since $M^\phi \to N^\phi$ is perpetual, also M^ϕ is strong normalizing. QED

Remark 3.5 The assumption $M \in \mathrm{Ter}(\Sigma)$ in Theorem 3.4 is necessary. Consider in fact the signature $\Sigma = \{s\}$ where s is a unary constructor. Take $\mathcal{E}$ to be the empty set of equations and $M \equiv s(\lambda x.xx)(s(\lambda x.xx)) \in \Lambda(\{s\})$. Then M is in normal form with respect to $\to_\mathcal{E}$ (for any $\mathcal{E}$), but M^ϕ is a lambda term without normal form (recall that according to Definition 2.6 $s^\phi \equiv \lambda xe.eU_1^1 xe$). We believe that by changing Definition 2.6 it is possible to strengthen Theorem 3.4 by allowing M to be an arbitrary term in $\Lambda(\Sigma)$.

4 The self-interpreter

Let us fix a signature of constructors $\Sigma_0 \supseteq \{\mathrm{Var}, \mathrm{App}, \mathrm{Abs}\}$ (mnemonic for "variable", "application", "abstraction") where Abs, Var are unary, and App is binary.

Definition 4.1 Define a map $\lfloor\ \rfloor : \Lambda \to \Lambda(\Sigma_0)$ as follows:

1. $\lfloor x \rfloor \equiv x$ if x is a variable,
2. $\lfloor MN \rfloor \equiv \mathrm{App}(\lfloor M \rfloor, \lfloor N \rfloor)$,
3. $\lfloor \lambda x.M \rfloor \equiv \mathrm{Abs}(\lambda x.\lfloor M \rfloor [Var(x) := x])$ where $[Var(x) := x]$ is the operation of replacing all the occurrences of $Var(x)$ by x.

The above definition is similar to the one in Mogensen [9] except that Mogensen defines $\lfloor \lambda x.M \rfloor \equiv \mathrm{Abs}(\lambda x.\lfloor M \rfloor)$. The difference between the two approaches is that Mogensen puts Var around each variable, while we put Var only around the free variables (in fact equation 3 says to take off the Var in the case of a bound variable x). So with our definition $\lfloor \lambda x.xy \rfloor \equiv \mathrm{Abs}(\lambda x.\mathrm{App}(x, \mathrm{Var}(y)))$. We abbreviate $\mathrm{Abs}(\lambda x.t)$ by $\lfloor \lambda x \rfloor.t$. So equation 3 reads: $\lfloor \lambda x.M \rfloor \equiv \lfloor \lambda x \rfloor.\lfloor M \rfloor[\mathrm{Var}(x) := x]$.

Definition 4.2 The gödelization $\lceil M \rceil \in \Lambda$ of $M \in \Lambda$, is defined by $\lceil M \rceil = \hat{\phi}(\lfloor M \rfloor)$ where $\phi : \Sigma_0 \to \Lambda$ is given in Definition 2.6. So if the signature of construcotrs Σ_0 contains no other symbols other than Var, App, Abs we have: 1. $\mathrm{Var}^\phi = \lambda xe.eU_1^3xe$, 2. $\mathrm{App}^\phi = \lambda xye.eU_2^3xye$, and 3. $\mathrm{Abs}^\phi = \lambda xe.eU_3^3xe$. (If other constructors are present we must replace the upper index 3 by the cardinality of Σ_0.)

Definition 4.3 Let $\Sigma = \Sigma_0 \cup \{E\}$. Consider the following set $\mathcal{E}_{\mathrm{E}}$ of equations in $\Lambda(\Sigma)$.

1. $E(\mathrm{Var}(x)) = x$,
2. $E(\mathrm{App}(x, y)) = E(x)E(y)$,
3. $E(\mathrm{Abs}(x)) = \lambda w.E(x\mathrm{Var}(w))$.

Lemma 4.4 *Let $\mathcal{E} = \mathcal{E}_E$. Then $E(\lfloor t \rfloor) \to_\mathcal{E} t$ for every $t \in \Lambda$.*

Proof. By induction on the length of the term t. Equation 1 takes care of the case when t is a variable. By equation 2 we have $E(\lfloor MN \rfloor) \to_\mathcal{E} E(\lfloor M \rfloor)E(\lfloor N \rfloor)$ and we can apply the induction hypothesis. By equation 3, $E(\lfloor \lambda x.M \rfloor) \equiv E(\mathrm{Abs}(\lambda x.\lfloor M \rfloor[Var(x) := x])) \to_\mathcal{E} \lambda w.E((\lambda x.\lfloor M \rfloor[Var(x) := x])\mathrm{Var}(w)) \to_\mathcal{E} \lambda w.E(\lfloor M \rfloor[Var(x) := x][x := \mathrm{Var}(w)]) \to_\mathcal{E} \lambda w.E(\lfloor M \rfloor[Var(x) := \mathrm{Var}(w)]) \equiv \lambda x.E(\lfloor M \rfloor)$ (by α-conversion) and we can apply the induction hypothesis. QED

Note that $\mathcal{E}_{\mathrm{E}}$ is canonical (but not algebraic), with set of constructors Σ_0 and set of programs $\Sigma_1 = \{E\}$.

Theorem 4.5 *There exists a lambda term $\mathbf{E}$ (enumerator) such that $\mathbf{E}(\lceil t \rceil) \to t$ for every $t \in \Lambda_0$. Moreover if $t \in SN$ (strong normalizing), then $\mathbf{E}(\lceil t \rceil) \in SN$.*

Proof. Let ϕ be given by Theorem 2.5 applied to the set of equations $\mathcal{E} = \mathcal{E}_{\mathrm{E}}$. Let $\mathbf{E} = E^\phi$. By Lemma 4.4 $E(\lfloor t \rfloor) \to_\mathcal{E} t$. Thus $(E(\lfloor t \rfloor))^\phi \to t^\phi$, that is $\mathbf{E}(\lceil t \rceil) \to t$ ($t^\phi \equiv t$ since $t \in \Lambda$). An explicit computation of E^ϕ shows that E^ϕ has a normal form. We leave to the reader the verification that if $t \in SN$, then $\mathbf{E}\lceil t \rceil \in SN$, namely the reduction $\mathbf{E}\lceil t \rceil \to t$ is perpetual. Note that we cannot invoke directly Theorem 3.4 since $\mathcal{E}_{\mathrm{E}}$ is not algebraic. QED

To illustrate the method in detail we compute explicitly the lambda term $E^\phi \equiv \mathbf{E}$ in the case $\Sigma_0 = \{\mathrm{Var}, \mathrm{App}, \mathrm{Abs}\}$. Since $\Sigma_1 = \{E\}$ has cardinality

1, and Σ_0 has cardinality 3, according to Theorem 2.5 E^ϕ is a Church 1-tuple whose only element is a Church 3-tuple. Thus we have $E^\phi \equiv \lambda x.xE_0$ where $E_0 \equiv < E_{\mathrm{Var}}, E_{\mathrm{App}}, E_{\mathrm{Abs}} >$ for suitable choices of the lambda terms $E_{\mathrm{Var}}, E_{\mathrm{App}}$, E_{Abs}. Applying ϕ to the equations $\mathcal{E}_{\mathrm{E}}$ we obtain:

1. $E^\phi(\mathrm{Var}^\phi x) = x$,
2. $E^\phi(\mathrm{App}^\phi xy) = E^\phi x(E^\phi y)$,
3. $E^\phi(\mathrm{Abs}^\phi x) = \lambda w.E^\phi(x(\mathrm{Var}^\phi w))$.

Since $E^\phi \equiv < E_0 > \equiv \lambda x.xE_0$ we obtain the equivalent set of equations:

1. $\mathrm{Var}^\phi xE_0 = x$,
2. $\mathrm{App}^\phi xyE_0 = < E_0 > x(< E_0 > y)$,
3. $\mathrm{Abs}^\phi xE_0 = \lambda w. < E_0 > (x(\mathrm{Var}^\phi w))$.

Recalling the definitions of Var^ϕ, App^ϕ, Abs^ϕ this simplifies as follows:

1. $E_0U_1^3xE_0 = x$,
2. $E_0U_2^3xyE_0 = < E_0 > x(< E_0 > y)$,
3. $E_0U_3^3xE_0 = \lambda w. < E_0 > (x(\mathrm{Var}^\phi w))$.

Since $E_0 \equiv < E_{\mathrm{Var}}, E_{\mathrm{App}}, E_{\mathrm{Abs}} >$ we obtain:

1. $E_{\mathrm{Var}}xE_0 = x$,
2. $E_{\mathrm{App}}xyE_0 = < E_0 > x(< E_0 > y)$,
3. $E_{\mathrm{Abs}}xE_0 = \lambda w. < E_0 > (x(\mathrm{Var}^\phi w))$.

We can now solve these equations in a natural way as follows.

1. $E_{\mathrm{Var}} \equiv \lambda xe.x$,
2. $E_{\mathrm{App}} \equiv \lambda xye. < e > x(< e > y) = \lambda xye.xe(ye)$,
3. $E_{\mathrm{Abs}} \equiv \lambda xew. < e > (x(\mathrm{Var}^\phi w)) = \lambda xew.x(\mathrm{Var}^\phi w)e$.

Thus:

$$\mathbf{E} = \lambda x.x < \mathbf{K}, \mathbf{S}, \lambda xew.x(\mathrm{Var}^\phi w)e >$$

where $\mathbf{K}, \mathbf{S}$ are the well known combinators and $\mathrm{Var}^\phi \equiv \lambda xe.eU_1^3xe$. This is a quite simple solution for a self-interpreter. Note in particular that $\mathbf{E}$ has a normal form.

Remark 4.6 If we modify equation 3 of Definition 4.1 by setting $\lfloor \lambda x.M \rfloor \equiv \mathrm{Abs}(\lambda x.\lfloor M \rfloor)$ we get the even simpler solution $\mathbf{E} = \lambda x.x < \mathbf{K}, \mathbf{S}, \mathbf{C} >$ where $\mathbf{C} \equiv \lambda xew.xwe$ is the well known combinator. However this alternative gödelization has the disadvantage that the reductor (see section 6) becomes much more complicated and does not work for closed terms. Moreover the proof of the second fixed point theorem (section 5) for this alternative gödelization would be quite difficult.

5 Second fixed point theorem

Any reasonable gödelization must satisfy the second fixed point theorem, namely for every $F \in \Lambda$ there exists $t \in \Lambda$ such that $F\lceil t \rceil = t$. To prove this fact for our gödelization note that since Abs is a constructor, Abs^ϕ has a left inverse (for instance $\lambda x.xU_2^3$ is a left inverse).

Lemma 5.1 *Let* **Sub** *be a left inverse of* Abs^ϕ. *Then* $\mathbf{Sub}\lceil \lambda x.M \rceil \lceil N \rceil \rightarrow \lceil M[x := N] \rceil$.

Proof. We have: $\mathbf{Sub}\lceil \lambda x.M \rceil \lceil N \rceil \equiv \mathbf{Sub}(\mathrm{Abs}^\phi(\lambda x.\lceil M \rceil [Var^\phi x := x])) \lceil N \rceil \rightarrow (\lambda x.\lceil M \rceil [Var^\phi x := x]) \lceil N \rceil \rightarrow \lceil M \rceil [Var^\phi x := x][x := \lceil N \rceil] \equiv \lceil M \rceil [Var^\phi x := \lceil N \rceil] \rightarrow \lceil M[x := N] \rceil$. QED

Theorem 5.2 *For every* $F \in \Lambda$ *there is* $t \in \Lambda$ *such that* $F\lceil t \rceil = t$.

Proof. Let $A \equiv \lambda x.F(\mathbf{Sub}xx)$, and $t \equiv F(\mathbf{Sub}\lceil A \rceil \lceil A \rceil)$. Then we have: $t \equiv F(\mathbf{Sub}\lceil \lambda x.F(\mathbf{Sub}xx) \rceil \lceil A \rceil) \rightarrow F\lceil F(\mathbf{Sub}\lceil A \rceil \lceil A \rceil) \rceil \equiv F\lceil t \rceil$. QED

6 The reductor

In this section we consider a signature of contructors $\Sigma_0 \supseteq \{\mathrm{Var}, \mathrm{App}, \mathrm{Abs}, \mathrm{Bou}, \mathrm{True}, \mathrm{False}\}$ and the signature of programs $\Sigma_1 = \{H, \mathrm{Test}, R, R'\}$. The symbol Bou (mnemonic for "bounded") plays a role similar to Var and it is used in the definition of R. The program symbol H represents a "single-step head reductor" defined as follows.

Definition 6.1 Let $\mathcal{E}_H$ be the following set of equations.

1. $H(\mathrm{Var}(x), y) = H(\mathrm{Bou}(x), y) = \mathrm{Error}$,
2. $H(\mathrm{App}(u, v), y) = \mathrm{App}(H(u, v), y)$,
3. $H(\mathrm{Abs}(x), y) = xy$
   where "Error" is an arbitrary term.

It follows from equation 3 and Definition 4.1, that $H(\lfloor \lambda x.M \rfloor, \lfloor N \rfloor) \to_{\mathcal{E}} \lfloor M[x := N] \rfloor$. Equations 2 and 3 imply: $H(\lfloor (\lambda x.M) Q_1 \dots Q_k \rfloor, \lfloor N \rfloor) \to_{\mathcal{E}} \lfloor M[x := Q_1] Q_2 \dots Q_k N \rfloor$, namely H performs a single head-reduction. We can now define a reductor R by iterating H is a suitable way. We need a test for the presence of head redexes:

Definition 6.2 Let $\mathcal{E}_T$ be the following set of equations:

1. $\mathrm{Test}(\mathrm{Var}(x)) = \mathrm{Test}(\mathrm{Bou}(x)) = \mathrm{False}$.
2. $\mathrm{Test}(\mathrm{App}(x, y)) = \mathrm{Test}(x)$,
3. $\mathrm{Test}(\mathrm{Abs}(x)) = \mathrm{True}$,

Note that $\mathrm{Test}(\lfloor x M_1 \dots M_k \rfloor) \to_{\mathcal{E}} \mathrm{False}$ and $\mathrm{Test}(\lfloor (\lambda x.M) M_1 \dots M_k \rfloor) \to_{\mathcal{E}} \mathrm{True}$, where $\mathcal{E} = \mathcal{E}_T$. We now give the canonical equations defining the reductor R.

Definition 6.3 Let $\mathcal{E}_R$ be the following set of equations.

1. $R(\mathrm{Var}(x)) = \mathrm{Var}(x)$,
2. $R(\mathrm{Bou}(x)) = x$,
3. $R(\mathrm{App}(x, y)) = R'(\mathrm{Test}(x), x, y)$,
4. $R(\mathrm{Abs}(x)) = \mathrm{Abs}(\lambda w.R(x\mathrm{Bou}(w))$,
5. $R'(\mathrm{True}, x, y) = R(H(x, y))$,
6. $R'(\mathrm{False}, x, y) = \mathrm{App}(R(x), R(y))$.

Intuitively equations 3, 5 and 6 mean: $R(App(x, y)) =$ if $\mathrm{Test}(x)$, then $R(H(x, y))$, else $App(R(x), R(y))$. The purpose of Bou is to mark those variables which are bounded "externally" (namely the corresponding lambda abstractions are not in the scope of R).

Lemma 6.4 *Let $\mathcal{E} = \mathcal{E}_H \cup \mathcal{E}_T \cup \mathcal{E}_R$. If $M \in \Lambda$ has a normal form N, then $R(\lfloor M \rfloor) \to_{\mathcal{E}} \lfloor N \rfloor$.*

Proof. We must show that R finds the normal form of a term when it exists. Since the normal form of $\lambda x.t$ is λx followed by the normal form of t, R must somehow commute with $\lfloor \lambda x \rfloor$. By equation 4 we have: $R(\lfloor \lambda x.M \rfloor) \equiv R(\mathrm{Abs}(\lambda x.\lfloor M \rfloor[\mathrm{Var}(x) := x])) \to_{\mathcal{E}} \mathrm{Abs}(\lambda w.R((\lambda x.\lfloor M \rfloor[\mathrm{Var}(x) := x])\mathrm{Bou}(w)))$ $\to_{\mathcal{E}} \mathrm{Abs}(\lambda w.R(\lfloor M \rfloor[\mathrm{Var}(x) := \mathrm{Bou}(w)])) \equiv \mathrm{Abs}(\lambda x.R(\lfloor M \rfloor[\mathrm{Var}(x) := \mathrm{Bou}(x)]))$ $\equiv \lfloor \lambda x \rfloor.R(\lfloor M \rfloor[\mathrm{Var}(x) := \mathrm{Bou}(x)])$. So R commutes indeed with $\lfloor \lambda x \rfloor$, but at the same time replaces x with $\mathrm{Bou}(x)$ as a reminder that x has been bounded "externally". This will cause no harm since the protection will be eventually removed by equation 2. (Note that there is no danger that something will be substituted inside $\mathrm{Bou}(x)$ because we never have two nested occurrences of

R.) Equations 1 and 2 ensure that the process stops correctly when it has arrived at the level of the variables. Consider now $R(\mathrm{App}(M,N))$. Equation 3 says that we must perform the test "Test" to verify the presence of head-redexes. If there is such an head redex, then equations 3 and 5 ensure that $R(\mathrm{App}(M,N)) \to_{\mathcal{E}} R(H(M,N))$, namely first we contract the head redex by H, and then we search the normal form of the contracted term. If there is no head redex we must instead normalize the subterms coded by M and N. This is ensured by equations 3 and 6 which entail $R(\mathrm{App}(M,N)) \to_{\mathcal{E}} \mathrm{App}(R(M),R(N))$ as desired. QED

Applying Theorem 2.5 and the techniques of Section 3 to the above set of equations we easily obtain:

Theorem 6.5 *There is a lambda term* $\mathbf{R}$ *(reductor) such that* $\mathbf{R}$ *has a normal form and for every* $M \in \Lambda_0$*, if* M *has a normal form* N*, then* $\mathbf{R}\lceil M\rceil \to \lceil N\rceil$*. Moreover if* M *is strong normalizing* $\mathbf{R}\lceil M\rceil$ *is strong normalizing.*

Proof. Let $\mathbf{R} = R^{\phi}$ be the lambda representatio of R given by Theorem 2.5. $\mathbf{R}\lceil M\rceil \to \lceil N\rceil$ follows from $R(\lfloor M\rfloor) \to_{\mathcal{E}} \lfloor N\rfloor$. The fact that $\mathbf{R}$ has a normal form follows by a direct computation of R^{ϕ}. We leave to the reader the verification of the normalization properties. The idea is to reason as in the proof of Theorem 2.5 part 2. Note that we cannot invoke Theorem 2.5 directly since the third equation of the system defining H is not algebraic. QED

Notice that in general $\lceil N\rceil$ is not in normal form (although it is strongly normalizing). If we verify Theorem 6.5 on a computer using a standard lambda reducer, it may happen that $\lceil N\rceil$ be hardly recognizable from its normal form. Such a drawback will be eliminated in a forthcoming paper of Böhm and Piperno.

References

[1] H. P. Barendregt, The Lambda Calculus, Revised edition, North-Holland, Amsterdam 1984.

[2] H. P. Barendregt, Self interpretation in lambda calculus, Journal of functional programming 1 (2), 229 - 239, April 1991.

[3] J. A. Bergstra and J. W. Klop, Strong normalization and perpetual reductions in the lambda calculus, J. Inform. Process. Cybernet. 18 (718), 403 - 417, 1982.

[4] C. Böhm and A. Berarducci, Automatic synthesis of typed Λ-programs on term algebras, Theoretical Computer Science 39 (1985), 135 - 154.

[5] C. Böhm and A. Piperno, Characterizing X-separability and one-side-Invertibility in $\lambda - \beta - \Omega$-calculus, LICS 88, Edimburgh, Computer Soc. of the IEEE, 1988, 91-101.

[6] C. Böhm, A. Piperno and E. Tronci, Solving equations in Lambda-Calculus, in "Logic Colloquium '88, Proceedings of the Colloquium held in Padova, Italy, August 22-31, 1988", North-Holland 1989.

[7] C. Böhm and E. Tronci, About systems of equations in lambda calculus, Information and Computation, vol. 90, 1 (1991) 1 - 32.

[8] N. Deschowitz, J.-P. Jouannaud and J. W. Klop, Open problems in rewriting, in "Proceedings of RTA '91", Springer Lectures Notes in Computer Science, 488, 1991.

[9] Torben Æ. Mogensen, Efficient self interpretation in lambda calculus, to be published in J. of functional programming.

[10] A. Piperno and E. Tronci, Regular systems of equations in λ-calculus, International Journal of Foundations of Computer Science, Vol. 1, No. 3 (1990) 325-339.