Discriminating coded lambda terms

Henk Barendregt

Computing Science Institute
Catholic University Nijmegen
and
Department of Software Technology
CWI, Amsterdam

Dedicated in friendship to *Cor Baayen*
on the occasion of his retirement

Abstract

A *coding* for a (type-free) lambda term M is a lambda term $\ulcorner M \urcorner$ in normal form such that M (and its parts) can be reconstructed from $\ulcorner M \urcorner$ in a lambda definable way. Kleene[1936] defined a coding $\ulcorner M \urcorner^K$ and a *self-interpreter* $\mathsf{E}^K \in \Lambda^\circ$ such that

$$\forall M \in \Lambda^\circ \ \mathsf{E}^K \ulcorner M \urcorner^K = M. \tag{1}$$

In this style one can construct a *discriminator* $\Delta^K \in \Lambda^\circ$ such that

$$\forall M, N \in \Lambda \ \ \Delta^K \ulcorner M \urcorner^K \ulcorner N \urcorner^K = \begin{cases} \texttt{true} & (\equiv \lambda xy.x) \quad \text{if } M \equiv N; \\ \texttt{false} & (\equiv \lambda xy.y) \quad \text{else.} \end{cases} \tag{2}$$

The terms E^K and Δ^K are complicated. They depend on the lambda definability of functions on the integers dealing with coded syntactic properties. Inspired by a construction of P. de Bruin (see Barendregt [1991]) Mogensen [1992] constructed a different coding $\ulcorner M \urcorner$ and an efficient self-interpreter $\mathsf{E} \in \Lambda^\circ$ such that

$$\forall M \in \Lambda \ \mathsf{E} \ulcorner M \urcorner = M. \tag{3}$$

This construction does not use an encoding of syntax as numbers but directly as lambda terms. This results in a much less complex E. Mogensen's construction was simplified even further in Böhm et al. [1994]. In this paper we construct a simple discriminator $\Delta \in \Lambda^\circ$ such that

$$\forall M, N \in \Lambda^\circ \ \ \Delta \ulcorner M \urcorner \ulcorner N \urcorner = \begin{cases} \texttt{true} & \text{if } M \equiv_\alpha N; \\ \texttt{false} & \text{else.} \end{cases} \tag{4}$$

Note that in (1) and (4) the statement is only about closed lambda terms, while that in (2) and (3) is about all lambda terms. It will become clear why this is so.

1. Introduction

The most important notations for the type-free lambda calculus will be given here. Background can be found in Barendregt [1984].

1.1. DEFINITION. Variables and terms of the lambda calculus are defined by the following abstract syntax.

$$\begin{array}{lcl} \texttt{var} & = & a \mid \texttt{var}' \\ \texttt{term} & = & \texttt{var} \mid \texttt{term term} \mid \lambda\, \texttt{var term} \end{array}$$

NOTATION. (i) $M, N, \ldots, P, Q, \ldots$ range over λ-terms. The letters $x, y, z, \ldots$ range over variables. Note that the variables are $\{a, a', a'', \ldots, a^{(n)}, \ldots\}$.

(ii) Λ is the set of lambda terms. $\mathrm{FV}(M)$ is the set of free variables of M.The set of closed terms is $\Lambda^\circ = \{M{\in}\Lambda \mid \mathrm{FV}(M) = \emptyset\}$.

(iii) The relation $\equiv$ denotes syntactic equality; the relation $\equiv_\alpha$ denotes syntactic equality up to a change of names of the bound variables. For example

$$\lambda x.x \equiv_\alpha \lambda y.y \not\equiv \lambda x.x.$$

(iv) The relation $=$ denotes β-convertibility, axiomatized by

$$(\lambda x.M)N = M[x := N].$$

Here $[x := n]$ denotes substitution of N in the free occurrences of x. E.g.

$$(x(\lambda x.x))[x := a] \equiv a(\lambda x.x).$$

(v) $\mathbb{N}$ is the set of natural numbers. For $n{\in}\mathbb{N}$ the terms $\mathbf{c}_n \equiv \lambda f x.f^n x$, where $f^0 x \equiv x$ and $f^{n+1}x \equiv f(f^n x)$, denote the so called Church numerals. Note that the $\mathbf{c}_n$ are distinct normal forms; hence

$$\mathbf{c}_n = \mathbf{c}_m \;\Rightarrow\; n = m$$

by the Church-Rosser theorem.

A lambda term can be seen as an executable: the redexes want to be evaluated. In this sense a normal form is not executable anymore. For a lambda term M its *code* $\ulcorner M \urcorner$ is a normal form such that M is reconstructible from M. Kleene [1936] defined a code $\ulcorner M \urcorner^K$ essentially as follows.

1.2. DEFINITION. (i) By induction on the structure of M we define $\#M$.

$$\begin{array}{rcl} \#(a^{(n)}) & = & <0, i>; \\ \#(PQ) & = & <1, <\#(P), \#(Q)>>; \\ \#(\lambda x.P) & = & <2, <\#(x), \#(P)>> . \end{array}$$

Here $< -, - >$ denotes a recursive pairing function on $\mathbb{N}$ with the recursive projections $(-)_0, (-)_1$:

$$(< n_0, n_1 >)_i = n_i.$$

(ii) The map $\ulcorner - \urcorner^K : \Lambda \to \Lambda$ is defined by

$$\ulcorner M \urcorner^K = \mathsf{c}_{\# M}.$$

Note that for all $M \in \Lambda$ the term $\ulcorner M \urcorner^K$ is in normal form. Moreover,

$$\ulcorner M \urcorner^K = \ulcorner N \urcorner^K \Rightarrow M \equiv N.$$

1.3. PROPOSITION. *There is no lambda term Q such that for all $M \in \Lambda^{(o)}$ one has*

$$QM = \ulcorner M \urcorner^K.$$

PROOF. Suppose Q exists. Then for $\mathsf{I} \equiv \lambda x.x$ one has

$$Q(\mathsf{II}) = \ulcorner \mathsf{II} \urcorner^K = \mathsf{c}_{\#(\mathsf{II})} = \mathsf{c}_{<1,<\#\mathsf{I},\#\mathsf{I}>>}.$$

But also

$$Q(\mathsf{II}) = Q\mathsf{I} = \ulcorner \mathsf{I} \urcorner^K = \mathsf{c}_{\#\mathsf{I}} = \mathsf{c}_{<2,<\#(x),\#(x)>>}.$$

Hence $< 1, < \#(\mathsf{I}), \#(\mathsf{I}) >>=< 2, < \#(x), \#(x) >>$, a contradiction. ∎

In spite of this fact that the 'quote' Q does not exists, the inverse 'evaluation' E can be constructed.

1.4. THEOREM (Kleene [1936]). *There exists an $\mathsf{E}^K \in \Lambda^\circ$ such that for all $M \in \Lambda^\circ$ one has*

$$\mathsf{E}^K \ulcorner M \urcorner^K = M.$$

PROOF. See Kleene [1936] or Barendregt [1984], theorem 8.1.16. ∎

The self-interpreter E can work only for closed terms M (or terms having at most a fixed finite set of free variables). The reason is that if

$$\mathsf{E}^K \ulcorner M \urcorner^K = M,$$

then

$$\mathrm{FV}(M) \subseteq \mathrm{FV}(\mathsf{E}^K \ulcorner M \urcorner^K) = \mathrm{FV}(\mathsf{E}^K).$$

Therefore if E^K is closed, then the M have to be closed as well. This causes one difficulty in the construction of E^K. The closed terms do not form a context-free language. Kleene solved this problem by constructing E first for the set of combinatory terms $\mathcal{C}^\circ$ built from the basis $\{\mathsf{K}, \mathsf{S}\}$ using application only; then the real self-interpreter can be obtained by translations between Λ° and $\mathcal{C}^\circ$.

A different construction of a self-interpreter was given by a former student of mine, using ideas from denotational semantics.

1.5. THEOREM (P. de Bruin). *There exists an $\mathsf{E}_0 \in \Lambda^\circ$ such that for all $M \in \Lambda$ and all $F \in \Lambda$ one has*

$$\mathsf{E}_0 \ulcorner M \urcorner F = M[x_1, \ldots, x_n := F \ulcorner x_1 \urcorner, \ldots, F \ulcorner x_n \urcorner] \qquad (5)$$

(simultaneous substitution), where $\{x_1, \ldots, x_n\} = \mathrm{FV}(M)$.

Proof. By the representability of computable functions and the fixedpoint theorem there is a term $\mathsf{E}_0 \in \Lambda^\circ$ such that

$$\begin{array}{rcl} \mathsf{E}_0 \ulcorner x \urcorner^K F & = & F \ulcorner x \urcorner^K; \\ \mathsf{E}_0 \ulcorner PQ \urcorner^K F & = & (\mathsf{E}_0 \ulcorner P \urcorner^K F)(\mathsf{E}_0 \ulcorner Q \urcorner^K F); \\ \mathsf{E}_0 \ulcorner \lambda x.P \urcorner^K F & = & \lambda x.(\mathsf{E}_0 \ulcorner P \urcorner^K F_{[\ulcorner x \urcorner \mapsto x]}), \end{array}$$

where $F_{[\ulcorner x \urcorner \mapsto x]} = F'_x$ with

$$\begin{array}{rcl} F'_x \ulcorner x \urcorner & = & x; \\ F'_x \ulcorner y \urcorner & = & F \ulcorner y \urcorner, \text{ if } y \not\equiv x. \end{array}$$

Note that F'_x can be written as GFx, with G closed. By induction on the structure of $M \in \Lambda$ one can show that the statement holds. ■

1.6. Corollary. *There exists an* $\mathsf{E}^{dB} \in \Lambda^\circ$ *such that for all* $M \in \Lambda^\circ$ *one has*

$$\mathsf{E} \ulcorner M \urcorner^K = M.$$

Proof (P. de Bruin). We can take

$$\mathsf{E}^{dB} \equiv \lambda m.\mathsf{E}_0 m \mathsf{I}.$$

Indeed, for closed terms M it follows from (5) that

$$\mathsf{E}^{dB} \ulcorner M \urcorner = \mathsf{E}_0 \ulcorner M \urcorner \mathsf{I} = M. \blacksquare$$

2. Representing data types

After seeing the method of P. de Bruin, Mogensen [1992] gave an improved version of it by representing data types directly (i.e. not using the natural numbers) in lambda calculus as done in e.g. Böhm and Berarducci [1985]. This approach was improved later by Böhm et al. [1994] by constructing a new representation of data types into type-free lambda calculus. This new representation will be treated in a slightly modified form in this section.

2.1. Definition. Write

$$\begin{array}{rcl} \langle M_1, \ldots, M_n \rangle & = & \lambda z.zM_1 \ldots M_n; \\ \mathsf{U}^n_i & = & \lambda x_1 \ldots x_n.x_i; \\ \texttt{true} & = & \mathsf{U}^2_1; \\ \texttt{false} & = & \mathsf{U}^2_2. \end{array}$$

Note that

$$\begin{array}{rcl} \langle M_1, \ldots, M_n \rangle \mathsf{U}^n_i & = & M_i; \\ \texttt{true}\, PQ & = & P; \\ \texttt{false}\, PQ & = & Q. \end{array}$$

In particular we have $\langle M \rangle = \lambda z.zM$ and $\langle\ \rangle = \lambda x.x = \mathsf{I}$. Now we define the notion of lists inspired by the language LISP, McCarthy et al. [1961].

2.2. Definition. (i) Write

$$\begin{aligned} \texttt{nil} &= \langle\ \rangle; \\ \texttt{cons} &= \lambda xy.\langle x, y\rangle; \\ \texttt{car} &= \langle \mathbf{U}^2_1\rangle; \\ \texttt{cdr} &= \langle \mathbf{U}^2_2\rangle; \\ \texttt{null}_? &= \langle \mathbf{U}^3_3, \mathbf{U}^2_1, \texttt{false}, \texttt{true}\rangle. \end{aligned}$$

(ii) Define

$$\begin{aligned} [\,] &= \langle\ \rangle; \\ [M_1, \ldots, M_{n+1}] &= \texttt{cons}\, M_1[M_2, \ldots, M_{n+1}]. \end{aligned}$$

So for example

$$[M_1, M_2, M_3] = \langle M_1, \langle M_2, \langle M_3, \langle\ \rangle\rangle\rangle\rangle.$$

(In Barendregt [1984] this term is written as $[M_1, M_2, M_3, \mathbf{I}]$. At the time of writing that book we did not yet see the usefulness of terminating a list with a special constructor.) Note that

$$\begin{aligned} \texttt{car}\,(\,\texttt{cons}\, PQ) &= P; \\ \texttt{cdr}\,(\,\texttt{cons}\, PQ) &= Q; \\ \texttt{null}_?\,\texttt{nil} &= \texttt{true}\,; \\ \texttt{null}_?(\,\texttt{cons}\,\texttt{PQ}) &= \texttt{false}\,. \end{aligned}$$

2.3. Proposition. *There exists lambda definable functions* $(\ )_i$ *such that for* $1 \leq i \leq n$ *one has*

$$([M_1, \ldots, M_n])_i = M_i.$$

Proof. Take

$$\begin{aligned} (l)_1 &= \texttt{car}\, l; \\ (l)_{i+1} &= (\,\texttt{cdr}\, l)_i. \blacksquare \end{aligned}$$

2.4. Definition. An *(algebraic) signature* s consists of a number $n \in \mathbb{N}$ (thought of as the list of symbols $[f_1, \ldots, f_n]$) together with a list of numbers $[s_1, \ldots, s_n]$ (thought of as the *arity* of the respective f_i's). We write $s = [s_1, \ldots, s_n]$.

For example a field has signature $s = [2, 2, 1, 1, 0, 0]$ (thought of as the arities of the functionsymbols $[+, \times, -, ^{-1}, 0, 1]$; so $f_1 = +, f_2 = \times$ etcetera).

2.5. Definition. If s is a signature then $\texttt{term}_s$, the set of *terms of signature* s, is defined as follows.

$$\begin{aligned} x \in \texttt{var} &\Rightarrow x \in \texttt{term}_\texttt{s}; \\ t_1, \ldots, t_{s_i} \in \texttt{term}_s &\Rightarrow f_i(t_1, \ldots, t_{s_i}) \in \texttt{term}_s. \end{aligned}$$

For example in the signature of fields the term $f_1(f_1(x, f_3(f_2(y, f_4(z))), f_6)$ is usually written as $x - yz^{-1} + 1$.

2.6. Definition. Let $s = [s_1, \ldots, s_n]$ be a signature.

(i) A *lambda interpretation* of s is a list of 'constructors' $C_1, \ldots, C_n \in \Lambda$.

(ii) Let $C_1, \ldots, C_n$ be a lambda interpretation of s. Then we define a map

$$T : \mathtt{term}_s \to \Lambda$$

as follows.

$$\begin{aligned} T_x &= x; \\ T_{f_i(t_1,\ldots,t_{s_i})} &= C_i[T_{t_1}, \ldots, T_{t_{s_i}}], \end{aligned}$$

where $[T_{t_1}, \ldots, T_{t_{s_i}}]$ is the list operation on lambda terms defined in 2.2.

Example. The signature of *binary trees* is $[0, 2]$. The term $t = f_2(f_2(f_1, f_1), f_1)$ denotes a simple tree and $t' = f_2(f_1, f_2(f_1, f_1))$ its miror image. Can we find a lambda interpretation for this signature in such a way that mirroring becomes lambda definable, i.e. for some $F \in \Lambda^\circ$ one has $FT_t = T_{t'}$? The following result, due to Böhm et al. [1994], will affirm this. We present the result in a modified form that will be useful for §4.

2.7. Theorem. *For every algebraic signature $s = [s_1, \ldots, s_n]$ there exists a lambda interpretation $C_1, \ldots, C_n$ such that the following hold.*

(i) $\forall A_1 \ldots A_n \, \exists F$

$$FT_{f_i(t_1,\ldots,t_{s_i})} = A_i[T_{t_1}, \ldots, T_{t_{s_i}}]F, \qquad 1 \le i \le n. \tag{6}$$

(ii) *The $C_1, \ldots, C_n$ only depend on n, not on the $[s_1, \ldots, s_n]$. In (6) we can take $F \equiv \langle\langle A_1, \ldots, A_n\rangle\rangle$.*

Proof. Define $C_{f_i} \equiv \lambda le.e\mathbf{U}_i^n l\langle e\rangle$.

(i) Given $A_1, \ldots, A_n$, we try whether $F \equiv \langle\langle A_1, \ldots, A_n\rangle\rangle$ works. Indeed,

$$\begin{aligned} FT_{f_i(t_1,\ldots,t_{s_i})} &= \langle\langle A_1, \ldots, A_n\rangle\rangle(C_{f_i}[T_{t_1}, \ldots, T_{t_{s_i}}]) \\ &= C_{f_i}[T_{t_1}, \ldots, T_{t_{s_i}}]\langle A_1, \ldots, A_n\rangle \\ &= \langle A_1, \ldots, A_n\rangle\mathbf{U}_i^n[T_{t_1}, \ldots, T_{t_{s_i}}]\langle\langle A_1, \ldots, A_n\rangle\rangle \\ &= A_i[T_{t_1}, \ldots, T_{t_{s_i}}]F. \end{aligned}$$

(ii) By the construction. ∎

2.8. Corollary. *Let $s = [s_1, \ldots, s_n]$ be an algebraic signature. Let $C_1, \ldots, C_n$ be the lambda interpretation of s constructed in theorem 2.7. Then for all $B_1 \ldots B_n$ there exists an F such that*

$$F(C_{f_i}[x_1, \ldots, x_{s_i}]) = B_i x_1 \ldots x_{s_i} F, \qquad 1 \le i \le n. \tag{7}$$

PROOF. Let $B_1, \ldots, B_n$ be given. Define $A_i = \lambda l.B_i(l)_1 \ldots (l)_{s_i}$. Then

$$A_i[x_1, \ldots, x_{s_i}] = B_i x_1, \ldots, x_{s_i}.$$

Then the F for the A_i found in theorem 2.7 is the F satisfying (7). ■

Now we can program the function that 'mirrors' trees. In the signature [0,2] for binary trees let

$$\begin{array}{rcccl} \texttt{leaf} & = & T_{f_1} & = & C_{f_1}\langle\ \rangle; \\ \texttt{tree} & = & \lambda ab.T_{f_2(a,b)} & = & \lambda ab.C_{f_2}\langle a, b\rangle. \end{array}$$

By corollary 2.8 there exists an F such that

$$\begin{array}{rcl} F\texttt{leaf} & = & \texttt{leaf}; \\ F(\texttt{tree}\, a\, b) & = & \texttt{tree}(fb)(fa). \end{array}$$

This F has the mirror effect. E.g. $F(f_2(f_2(f_1, f_1), f_1)) = f_2(f_1, f_2(f_1, f_1))$.

3. A simple self-interpreter

In Mogensen [1992] a simple coding and self-interpreter for lambda terms is defined, using the fact that data types (term algebras of a signature s) have a lambda interpretation. The method was simplified by Böhm et al. [1994] by making use of their lambda representation of algebraic signatures given in §2.

3.1. DEFINITION. Let s be the signature $[1, 2, 1]$. Define

$$\begin{array}{rcccl} \texttt{const} & = & C_{f_1} & \equiv & \lambda le.e\mathbf{U}^3_1 l\langle e\rangle; \\ \texttt{app} & = & C_{f_2} & \equiv & \lambda le.e\mathbf{U}^3_2 l\langle e\rangle; \\ \texttt{abs} & = & C_{f_3} & \equiv & \lambda le.e\mathbf{U}^3_3 l\langle e\rangle. \end{array}$$

3.2. DEFINITION. For $M \in \Lambda$ define $\ulcorner M \urcorner$ as follows.

$$\begin{array}{rcl} \ulcorner x \urcorner & = & \texttt{const}\,[x]; \\ \ulcorner PQ \urcorner & = & \texttt{app}\,[\ulcorner P \urcorner, \ulcorner Q \urcorner]; \\ \ulcorner \lambda x.P \urcorner & = & \texttt{abs}\,[\lambda x.\ulcorner P \urcorner]. \end{array}$$

Note that $\mathrm{FV}(\ulcorner M \urcorner) = \mathrm{FV}(M)$.

3.3. THEOREM (Mogensen [1992]). *There exists an* $\mathbf{E} \in \Lambda^\circ$ *such that*

$$\forall M \in \Lambda\ \mathbf{E}\ulcorner M \urcorner = M.$$

PROOF (B"ohm et al. [1994]). By corollary 2.8 there exists a term $\mathbf{E} \in \Lambda^\circ$ such that

$$\begin{array}{rcl} \mathbf{E}(\,\texttt{const}\,[p]) & = & p; \\ \mathbf{E}(\,\texttt{app}\,[p, q]) & = & (\mathbf{E}p)(\mathbf{E}q); \\ \mathbf{E}(\,\texttt{abs}\,[p]) & = & \lambda x.\mathbf{E}(px). \end{array}$$

Then

$$\begin{array}{rcl}\mathsf{E}\ulcorner x\urcorner &=& x;\\ \mathsf{E}\ulcorner PQ\urcorner &=& (\mathsf{E}\ulcorner P\urcorner)(\mathsf{E}\ulcorner Q\urcorner);\\ \mathsf{E}\ulcorner \lambda x.P\urcorner &=& \lambda x.\mathsf{E}\ulcorner P\urcorner.\end{array}$$

Now the results follows by induction on the structure of M. ■

Using the constructions in §2 the self-interpreter becomes

$$\mathsf{E} \equiv \langle\langle \lambda lf.(l)_1, \lambda lf.f(l)_1(f(l)_2), \lambda lfx.f((l)_1 x)\rangle\rangle.$$

The construction in Böhm et al. [1994] is simpler. They take

$$\begin{array}{rcl}\mathtt{const} &=& \lambda xe.e\mathsf{U}^3_1 xe;\\ \mathtt{app} &=& \lambda xye.e\mathsf{U}^3_2 xye;\\ \mathtt{abs} &=& \lambda xe.e\mathsf{U}^3_3 xe.\end{array}$$

The resulting self-interpreter then becomes $\mathsf{E}^B = \langle\langle \mathsf{K}, \mathsf{S}, \mathsf{C}\rangle\rangle$. Here $\mathsf{K} \equiv \lambda xy.x$, $\mathsf{S} \equiv \lambda xyz.xz(yz)$ and $\mathsf{C} \equiv \lambda xyz.x(zy)$. For reasons of uniformity we have given the definition of $\mathtt{const}$, $\mathtt{app}$ and $\mathtt{abs}$ as in 3.1. This will be useful in §4.

4. A simple discriminator

In this section we will construct a simple term discriminating between coded closed lambda term. The discrimination is even modulo α-conversion. For open terms discrimination is possible only for the coding $\ulcorner\ \urcorner^K$ of Kleene.

4.1. Lemma. (i) *There exists a term* $\delta_{\mathbb{N}} \in \Lambda^\circ$ *such that*

$$\delta_{\mathbb{N}}\mathsf{c}_n\mathsf{c}_m = \begin{cases} \mathtt{true} & \textit{if } n = m;\\ \mathtt{false} & \textit{else.}\end{cases}$$

(ii) *There exists a term* $\mathtt{and} \in \Lambda^\circ$ *such that*

$$\begin{array}{rcl}\mathtt{and\ true\ true} &=& \mathtt{true};\\ \mathtt{and\ true\ false} &=& \mathtt{false};\\ \mathtt{and\ false\ true} &=& \mathtt{false};\\ \mathtt{and\ false\ false} &=& \mathtt{false}.\end{array}$$

Proof. (i) By the representability of the recursive functions.

(ii) Take $\mathtt{and} = \lambda ab.a\,\mathtt{true}\,b$. ■

4.2. PROPOSITION. *There exists a term $\delta\in\Lambda^\circ$ such that (writing δ_n for $\delta\mathbf{c}_n$) one has*

$$\begin{array}{lcl}
\delta_n\ulcorner x\urcorner\ulcorner x'\urcorner & = & \delta_{\mathbb{N}}xy;\\
\delta_n\ulcorner x\urcorner\ulcorner P'Q'\urcorner & = & \mathtt{false}\,;\\
\delta_n\ulcorner x\urcorner\ulcorner \lambda x'.P'\urcorner & = & \mathtt{false}\,;\\
\\
\delta_n\ulcorner PQ\urcorner\ulcorner x'\urcorner & = & \mathtt{false}\,;\\
\delta_n\ulcorner PQ\urcorner\ulcorner P'Q'\urcorner & = & \mathtt{and}\,(\delta_n\ulcorner P\urcorner\ulcorner P'\urcorner)(\delta_n\ulcorner Q\urcorner\ulcorner Q'\urcorner)\\
\delta_n\ulcorner PQ\urcorner\ulcorner \lambda x'.P'\urcorner & = & \mathtt{false}\,;\\
\\
\delta_n\ulcorner \lambda x.P\urcorner\ulcorner x'\urcorner & = & \mathtt{false}\,;\\
\delta_n\ulcorner \lambda x.P\urcorner\ulcorner P'Q'\urcorner & = & \mathtt{false}\,;\\
\delta_n\ulcorner \lambda x.P\urcorner\ulcorner \lambda x'.P'\urcorner & = & \delta_{n+1}(\ulcorner P\urcorner[x:=\mathbf{c}_n])(\ulcorner P'\urcorner[x':=\mathbf{c}_n]).
\end{array}$$

PROOF. We introduce the following ad hoc notation.

(i) Let $A_1,\ldots,A_n\in\Lambda$. Then we write

$$\lambda\vec{x}![A_1,\ldots,A_n]\equiv\langle\langle\lambda\vec{x}.A_1,\ldots,\lambda\vec{x}.A_n\rangle\rangle.$$

(ii) If $B_i\equiv[A_{i1},\ldots,A_{in}]$, then we write

$$\lambda\vec{x}!![B_1,\ldots,B_n]\equiv\langle\langle\lambda\vec{x}!B_1,\ldots,\lambda\vec{x}!B_n\rangle\rangle.$$

(iii) Let for $1\le i\le n, 1\le j\le n$ be given $A_{ij}\in\Lambda$. Then

$$\begin{array}{rcl}
[A_{ij}] & \equiv & [[A_{11},\ldots,A_{1n}],\\
& & \;[A_{21},\ldots,A_{2n}],\\
& & \qquad\cdots\\
& & \;[A_{n1},\ldots,A_{nn}]].
\end{array}$$

If $n=3$ we may write $[A_{ij}]$ as

$$\begin{bmatrix} A_{11} & A_{12} & A_{13}\\ A_{21} & A_{22} & A_{23}\\ A_{31} & A_{32} & A_{33}\end{bmatrix}.$$

Now define $\delta\equiv\lambda ntt'$.

$$\left(\lambda td!\lambda t'd'n!!\begin{bmatrix} \delta_{\mathbb{N}}(t)_1(t')_1 & \mathtt{false} & \mathtt{false}\\ \mathtt{false} & \mathtt{and}\,(d(t)_1(t')_1n)(d(t)_2(t')_2n) & \mathtt{false}\\ \mathtt{false} & \mathtt{false} & d(tn)(t'n)(\mathbf{S}^+n)\end{bmatrix}\right)tt'n,$$

where $\mathbf{S}^+$ lambda defines the successor function. This δ satisfies the specification. ■

4.3. PROPOSITION. *For all $M,M'\in\Lambda$ such that* $\mathrm{FV}(MM')\subseteq\{x_1,\ldots,x_n\}$ *and for substitutions $*=[x_1:=\mathbf{c}_{k_1}]\ldots[x_n:=\mathbf{c}_{k_n}]$ with $k_i\ne k_j$ (for $1\le i<j\le n$) one has for $p>k_i$ (for all $1\le i\le n$) that*

$$\delta_p\ulcorner M\urcorner\ulcorner M'\urcorner * = \begin{cases}\mathtt{true} & \textit{if } M\equiv_\alpha M';\\ \mathtt{false} & \textit{else.}\end{cases}$$

PROOF. By induction on the structure of M, in each case making distinctions according to the structure of M'. We treat four instructive cases.

Case $M \equiv x, M' \equiv x'$. Then

$$\delta_p \ulcorner M \urcorner \ulcorner M' \urcorner * = \delta_{\mathbb{N}} \mathbf{c}_{k_i} \mathbf{c}_{k_{i'}},$$

where $x \equiv x_i, x' \equiv x_{i'}$. This is true or false depending on whether $x \equiv x'$ (so $i = i'$) or $x \not\equiv x'$ (so $i \neq i'$).

Case $M \equiv x, M' \equiv P'Q'$. Then

$$\delta_p \ulcorner M \urcorner \ulcorner M' \urcorner * = \texttt{false}\,.$$

Case $M \equiv PQ, M' \equiv P'Q'$. Then

$$\begin{array}{rcl} \delta_p \ulcorner M \urcorner \ulcorner M' \urcorner * & = & \texttt{and}\,(\delta_p \ulcorner P \urcorner \ulcorner P' \urcorner *)(\delta_p \ulcorner Q \urcorner \ulcorner Q' \urcorner *) \\ & =_{IH} & \texttt{and}\,(\,\texttt{true}\,/\,\texttt{false}\,)(\,\texttt{true}\,/\,\texttt{false}\,) \\ & = & \texttt{true}\,/\,\texttt{false}\,, \end{array}$$

as it should ($=$ `true` only if $PQ \equiv P'Q'$ i.e. if both $P \equiv P'$ and $Q \equiv Q'$).

Case $M \equiv \lambda x.P, M' \equiv \lambda x'.P'$. Then

$$\begin{array}{rcl} \delta_p \ulcorner M \urcorner \ulcorner M' \urcorner * & = & \delta_{p+1}(\ulcorner P \urcorner [x := \mathbf{c}_p])(\ulcorner P' \urcorner [x' := \mathbf{c}_p]) * \\ & = & \delta_{p+1} \ulcorner P \urcorner \ulcorner P'[x' := x] \urcorner [x := \mathbf{c}_p] * \\ & = & \delta_{p+1} \ulcorner P \urcorner \ulcorner P'[x' := x] \urcorner *', \end{array}$$

with $*' = *[x := \mathbf{c}_p]$ being an admissible substitution. So

$$\delta_p \ulcorner M \urcorner \ulcorner M' \urcorner * =_{IH} \begin{cases} \texttt{true} & \text{if } P \equiv_\alpha P'[x' := x]; \\ \texttt{false} & \text{else.} \end{cases}$$

Now $M \equiv_\alpha M'$ iff $\lambda x.P \equiv_\alpha \lambda x'.P'$ ($\equiv_\alpha \lambda x.P'[x' := x]$) iff $P \equiv_\alpha P'[x' := x]$. Hence we are done. ∎

4.4. COROLLARY. *Write $\Delta \equiv \delta_0$. Then for all $M, M' \in \Lambda^\circ$ one has*

$$\Delta \ulcorner M \urcorner \ulcorner M' \urcorner = \begin{cases} \texttt{true} & \textit{if } M \equiv_\alpha M'; \\ \texttt{false} & \textit{else.} \end{cases}$$

PROOF. Immediate from the proposition. ∎

Note that this corollary cannot hold for arbitrary $M, M' \in \Lambda$. For example, it is impossible to discriminate $\ulcorner x \urcorner$ and $\ulcorner x' \urcorner$. Indeed take $x \not\equiv x'$ and make a substitution:

$$\Delta \ulcorner x \urcorner \ulcorner x' \urcorner = \texttt{false} \;\Rightarrow\; \Delta \ulcorner x \urcorner \ulcorner x \urcorner = \texttt{false}\,,$$

a contradiction.

References

BARENDREGT, H.P.

[1984] *The Lambda Calculus, its Syntax and Semantics*, revised edition, Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam.

[1991] Theoretical pearls: Self-interpretation in lambda calculus, *J. Funct. Programming,* **1**(2), 229–233.

BÖHM, C., and A. BERARDUCCI

[1985] Automatic synthesis of typed λ-programs on term algebras, Theor. Comput. Sci. **39**, 135-154.

BÖHM, C., A. PIPERNO and S. GUERRINI

[1994] λ-definitions of function(al)s by normal forms, in: *ESOP '94*, (ed. D. Sannella), LNCS 788, Springer, Berlin, 135–149

CHURCH, A.

[1941] *The calculi of lambda conversion*, Princeton University Press, Princeton. Reprinted by Kraus reprint corporation, New York, 1965.

KLEENE, S.C.

[1936] λ-definability and recursiveness, *Duke Math. J.* **2**, 340-353.

MCCARTHY, J., P.W. ADAMS, D.J. EDWARDS, T.P. HART and M.I. LEVIN

[1962] *LISP 1.5 Programmer's manual*, MIT Press.

MOGENSEN, T.Æ.

[1992] Efficient self-interpretation in lambda calculus, *J. Funct. Programming,* **2**(3), 345–364.