Math. Struct. in Comp. Science (2003), *vol.* 13, *pp.* 5–13. © 2003 Cambridge University Press
DOI: 10.1017/S096012950200381X Printed in the United Kingdom

A simple proof of the undecidability of strong normalization

PAWEŁ URZYCZYN[†]

Institute of Informatics, University of Warsaw
Banacha 2, 02-097 Warsaw, Poland
Email: urzy@mimuw.edu.pl

Received 20 November 2000; revised 15 December 2001

The purpose of this note is to give a methodologically simple proof of the undecidability of strong normalization in the pure lambda calculus. For this we show how to represent an arbitrary partial recursive function by a term whose application to any Church numeral is either strongly normalizable or has no normal form. Intersection types are used for the strong normalization argument.

1. Introduction

It is a well known fact that normalization of pure lambda terms is undecidable. The standard method of proving this fact is *via* lambda representability of all (partial) recursive functions. It is also well known that strong normalization is undecidable too. However, even people familiar with lambda calculus are often confused about the proper way of proving this fact. First of all, it does not follow from Rice's theorem, because strong normalization is not closed under equality. In addition, the standard way of representing a recursive function in lambda calculus, the one we find in most textbooks, uses the fixed point combinator **Y**. Since expressions involving **Y** do not strongly normalize, this method is useless for our purpose.

But there is also another possibility, based on an idea that is apparently due to Kleene. It is to 'delay' the action of the fixpoint combinator in such a way that one can avoid the loop, which is inherent in **Y**. This trick was used in Barendregt (1984, Chapter 9), to prove that all partial recursive functions are representable in the $\lambda\mathbf{I}$ calculus. It was necessary, because for $\lambda\mathbf{I}$ terms normalization implies strong normalization, so the standard approach with the non-normalizable fixed point simply would not work. Also, recursive function representability in both CL and lambda-calculus is shown without the help of **Y** in the book Hindley and Seldin (1986).

Undecidability of strong normalization was first shown as a consequence of the representability of partial recursive functions in $\lambda\mathbf{I}$. See Leivant (1983) and Barendregt (1984, Exercise 9.5.17). In Henglein and Mairson (1994) there is a partial argument explicitly based on the 'delayed fixpoint' $\overline{\mathbf{Y}} \equiv \lambda f.(\lambda x.f(\lambda y.yxx))(\lambda x.f(\lambda y.yxx))$, which is applied to a Turing Machine simulation.

[†] Partly supported by KBN Grant 8 T11C 035 14.

With a similar technique, one can obtain a simple representation of every k-ary partial recursive function f by a lambda term F, with the following property. For all $n_1, \ldots, n_k$:

(a) If $f(n_1, \ldots, n_k)$ is defined, then $F\mathbf{n}_1 \ldots \mathbf{n}_k$ is strongly normalizable;
(b) If $f(n_1, \ldots, n_k) = n$, then $F\mathbf{n}_1 \ldots \mathbf{n}_k =_\beta \mathbf{n}$;
(c) If $f(n_1, \ldots, n_k)$ is undefined, then $F\mathbf{n}_1 \ldots \mathbf{n}_k$ does not have a normal form.

Here, the boldface $\mathbf{n}$ is the Church numeral corresponding to n.

The definition of F is no more complicated than the ordinary one, and showing that F represents f (that is, that conditions (b) and (c) hold) is not very hard (see Proposition 1). But proving condition (a) above is not immediate. After some 'simple proof' attempts led into complicated syntactic considerations, the author of the present note discovered that perhaps the simplest way to prove strong normalization is *via* typing in a type system satisfying the property of strong normalization. In fact, such a typing has already been done in the higher-order polymorphic lambda calculus $\mathbf{F}_\omega$ (Urzyczyn 1997). This construction, aimed at proving the undecidability of typability in $\mathbf{F}_\omega$, is, however, extremely complicated. This complication is unnecessary if all we want is condition (a) above, because we can use a more flexible type system of intersection types and still retain the strong normalization (Barendregt *et al.* 1983).

In this note we adopt the approach of Urzyczyn (1997) to work with intersection types. We use the ω-free variant of the system CDV (Coppo *et al.* 1981). Although the main proof follows exactly the same pattern as that of Urzyczyn (1997), the use of intersection types makes it incomparably simpler and, hopefully, much more understandable.

The general structure of our argument consists of three implications. If f is the function represented by F, then for all arguments $n_1, \ldots, n_k$, we have:

(1) $f(n_1, \ldots, n_k)$ defined $\Rightarrow F\mathbf{n}_1 \ldots \mathbf{n}_k$ typable;
(2) $F\mathbf{n}_1 \ldots \mathbf{n}_k$ typable $\Rightarrow F\mathbf{n}_1 \ldots \mathbf{n}_k$ strongly normalizable;
(3) $F\mathbf{n}_1 \ldots \mathbf{n}_k$ strongly normalizable $\Rightarrow f(n_1, \ldots, n_k)$ defined.

Thus, we essentially prove the undecidability of two problems at the same time: strong normalization and typability in intersection types.

This is not so surprising, because typability in intersection types is in fact equivalent to strong normalization. See Pottinger (1980) for the first proof ever, and Amadio and Curien (1998) for the first correct proof published (though it was preceded by an unpublished one by Betti Venneri).

2. How to represent partial recursive functions

We work with the ordinary Church numerals, defined by

$$\mathbf{n} \equiv \lambda f x. f^n(x).$$

Our representation of the successor function is standard:

$$\mathbf{succ} \equiv \lambda n f x. f(n f x),$$

as are the obvious representations of zero, projections, and composition (of total functions). The representation of a function defined by primitive recursion is standard as well, but we must describe it in detail in order to be able to define the typing in Section 3. For this, we need pairing and projections defined routinely as:

$$\begin{aligned} \langle M, N\rangle &\equiv \lambda x.xMN; \\ \pi_i &\equiv \lambda x_1x_2.x_i, \text{ for } i = 1, 2. \end{aligned}$$

Suppose that $f : \omega^{m+1} \to \omega$ is defined by the equations:

$$\begin{aligned} f(0, n_1, \dots, n_m) &= g(n_1, \dots, n_m); \\ f(n+1, n_1, \dots, n_m) &= h(f(n, n_1, \dots, n_m), n, n_1, \dots, n_m), \end{aligned}$$

and let G and H represent g and h, respectively. Define auxiliary terms

$$\begin{aligned} \textbf{Step} &\equiv \lambda p.\langle \textbf{succ}(p\pi_1), H(p\pi_2)(p\pi_1)x_1 \dots x_m\rangle; \\ \textbf{Init} &\equiv \langle \mathbf{0}, Gx_1 \dots x_m\rangle. \end{aligned}$$

Then the representation of f is taken as

$$F \equiv \lambda xx_1 \dots x_m.x\,\textbf{Step}\,\textbf{Init}\pi_2.$$

Of course, this definition encodes an iterative algorithm to compute $f(n_1, \dots, n_m)$, for any numbers $n_1, \dots, n_m$. One generates a sequence of pairs of numbers

$$(0, a_0), (1, a_1), \dots, (n, a_n),$$

where we have $a_0 = g(n_1, \dots, n_m)$, each a_{i+1} is $h(a_i, i, n_1, \dots, n_m)$, and, finally, $a_n = f(n, n_1, \dots, n_m)$.

Finally, to represent minimization, first define:

- **true** $\equiv \lambda x\lambda y.x$;
- **false** $\equiv \lambda x\lambda y.y$;
- **zero** $\equiv \lambda x.x(\lambda y.\textbf{false})\textbf{true}$;
- **if** M **then** N **else** $P \equiv MNP$.

Now suppose that

$$f(n_1, \dots, n_m) = \ell(\mu n[g(n, n_1, \dots, n_m) = 0]),$$

where g and ℓ are total functions represented by G and L, respectively. (The ordinary minimum is the special case where ℓ is the identity.) We define an auxiliary term

$$W \equiv \lambda y.\textbf{if } \textbf{zero}(Gyx_1 \dots x_m) \textbf{ then } \lambda w.Ly \textbf{ else } \lambda w.w(\textbf{succ}\, y)w,$$

which is meant to represent a single step of the iteration. The function f is then represented by the term

$$F \equiv \lambda x_1 \dots x_m.W\mathbf{0}W.$$

To see that it works, assume that $\mu n[g(n, n_1, \dots, n_m) = 0] = n$, and $\ell(n) = r$. Then observe that we have the following reduction sequence:

$$F\mathbf{n}_1 \dots \mathbf{n}_m \twoheadrightarrow_\beta \overline{W}\mathbf{0}\overline{W} \twoheadrightarrow_\beta \overline{W}\mathbf{1}\overline{W} \twoheadrightarrow_\beta \cdots \twoheadrightarrow_\beta \overline{W}\mathbf{n}\overline{W} \twoheadrightarrow_\beta L\mathbf{n} \twoheadrightarrow_\beta \mathbf{r}, \tag{1}$$

where $\overline{W}$ is the result of substituting $\mathbf{n}_1, \dots, \mathbf{n}_m$ for $x_1, \dots, x_m$ in W.

On the other hand, if the minimum is undefined (this is the only way in which $f(n_1,\dots,n_m)$ can be undefined), we have an infinite reduction sequence

$$F\mathbf{n}_1\dots\mathbf{n}_m \twoheadrightarrow_\beta \overline{W}\mathbf{0}\overline{W} \twoheadrightarrow_\beta \overline{W}\mathbf{1}\overline{W} \twoheadrightarrow_\beta \cdots \twoheadrightarrow_\beta \overline{W}\mathbf{n}\overline{W} \twoheadrightarrow_\beta \cdots$$

which is *quasi leftmost* in the sense of Barendregt (1984, Chapter 13). Since quasi leftmost reductions are normalizing, the normal form does not exist.

Proposition 1. All partial recursive functions are representable with the help of the above definitions. Composition, primitive recursion and minimum need only be applied to total recursive functions.

Proof. By Kleene's normal form theorem (see Rogers (1967, Section 1.10), for example), every partial recursive function f can be written as

$$f(n_1,\dots,n_m) = \ell(\mu y[g(n_1,\dots,n_m,y)=0]),$$

where g is total recursive (in fact primitive recursive) and ℓ is the first inverse of a pairing function. □

3. How to type function representations

In what follows, intersection types, defined by the grammar

$$\langle\mathit{type}\rangle := \langle\mathit{type\ variable}\rangle \mid (\langle\mathit{type}\rangle \to \langle\mathit{type}\rangle) \mid (\langle\mathit{type}\rangle \cap \langle\mathit{type}\rangle)$$

are taken as associative, commutative and idempotent, and without any constant types or subsumption rules. That is, we deal with the ω-free variant of the system CDV (Coppo *et al.* 1981), denoted by $\vdash_\wedge$ in Cardonne and Coppo (1990), see Figure 1.

Notationally, intersection has priority over arrow, and the latter associates, as usual, to the right. A *straight* type of a Church numeral $\mathbf{n}$ is any type of the form

$$(\tau_0 \to \tau_1) \cap (\tau_1 \to \tau_2) \cap \cdots \cap (\tau_{n-1} \to \tau_n) \to \tau_0 \to \tau_n,$$

whenever $n > 0$, and any type of the form $(\tau_0 \to \tau_1) \to \tau_0 \to \tau_0$, for $n = 0$. A *fundamental* type of $\mathbf{n}$ is an arbitrary intersection of straight types for $\mathbf{n}$.

If τ is a fundamental type of $\mathbf{n}$, then clearly $\vdash \mathbf{n} : \tau$. In fact, a straight type of $\mathbf{n}$ in which all the τ_i's are distinct type variables, is the principal type of $\mathbf{n}$.

$$\text{(VAR)}\quad E \vdash x : \sigma \qquad \text{if } (x:\sigma) \text{ is in } E$$

$$(\mathrm{E}{\to})\quad \frac{E \vdash M : \tau \to \sigma,\ E \vdash N : \tau}{E \vdash (MN) : \sigma} \qquad\qquad (\mathrm{I}{\to})\quad \frac{E(x:\tau) \vdash M : \sigma}{E \vdash (\lambda x.M) : \tau \to \sigma}$$

$$(\mathrm{E}{\cap})\quad \frac{E \vdash M : \sigma \cap \tau}{E \vdash M : \sigma} \qquad\qquad (\mathrm{I}{\cap})\quad \frac{E \vdash M : \tau,\ E \vdash M : \sigma}{E \vdash M : \tau \cap \sigma}$$

Fig. 1. Type assignment rules.

But note that not every type of $\mathbf{n}$ must be fundamental. For instance, the following is a non-fundamental type of $\mathbf{2}$:

$$(\alpha \to \beta) \cap (\delta \to \gamma) \cap (\beta \cap \gamma \to \varepsilon) \to \alpha \cap \delta \to \varepsilon$$

Note also that the type $\mathbf{int} = (\alpha \to \alpha) \to \alpha \to \alpha$ is a straight type of all $\mathbf{n}$.

The following definition is quite different from ordinary definitions of a function representable in a typed system. The difference is that we do not require a single integer type, but we allow different typings for different arguments. This is called 'non-uniform' representability. Daniel Leivant was the first to show that all recursive functions are non-uniformly representable in intersection types (Leivant 1983). In fact, we use a slightly stronger notion than Leivant's.

A partial function $f : \omega^m \to \omega$ is *properly representable* in intersection types by a closed term F, if the following hold:

(1) F represents f (that is, satisfies (b) and (c) in Section 1).
(2) If $f(n_1, \ldots, n_m) = n$ and τ is a straight type of $\mathbf{n}$, there are fundamental types $\sigma_1, \ldots, \sigma_m$ of $\mathbf{n}_1, \ldots, \mathbf{n}_m$, respectively, such that $\vdash F : \sigma_1 \to \cdots \to \sigma_m \to \tau$.

Unfortunately, in part (2) of the above definition we cannot require that $\sigma_1, \ldots, \sigma_m$ be straight types. But it can be strenghtened as follows.

Lemma 2. Let $f : \omega^m \to \omega$ be properly representable, and assume $f(n_1, \ldots, n_m) = n$. If τ is a fundamental type of $\mathbf{n}$, there are fundamental types $\sigma_1, \ldots, \sigma_m$ of $\mathbf{n}_1, \ldots, \mathbf{n}_m$, respectively, such that $\vdash F : \sigma_1 \to \cdots \to \sigma_m \to \tau$.

Proof. The proof is easy and omitted. □

Lemma 3. The base functions: zero, successor and projections are properly representable.

Proof. This is completely obvious for zero and projections. For the successor, observe that

$$\begin{aligned}\vdash \mathbf{succ} : ((\tau_0 \to \tau_1) \cap (\tau_1 \to \tau_2) \cap \cdots \cap (\tau_{n-1} \to \tau_n) \to \tau_0 \to \tau_n) \to \\ (\tau_0 \to \tau_1) \cap (\tau_1 \to \tau_2) \cap \cdots \cap (\tau_{n-1} \to \tau_n) \cap (\tau_n \to \tau_{n+1}) \to \tau_0 \to \tau_{n+1},\end{aligned}$$

for all $n > 0$. In addition, we have

$$\vdash \mathbf{succ} : ((\tau_0 \to \tau_1) \to \tau_0 \to \tau_0) \to (\tau_0 \to \tau_1) \to \tau_0 \to \tau_1.$$ □

Lemma 4. A composition of total, properly representable functions is properly representable.

Proof. The proof is easy and omitted. □

Lemma 5. Let $f : \omega^{m+1} \to \omega$ be defined by primitive recursion from properly representable total functions $g : \omega^m \to \omega$ and $h : \omega^{m+2} \to \omega$ as follows:

$$\begin{aligned} f(0, n_1, \ldots, n_m) &= g(n_1, \ldots, n_m); \\ f(n+1, n_1, \ldots, n_m) &= h(f(n, n_1, \ldots, n_m), n, n_1, \ldots, n_m). \end{aligned}$$

Then f is properly representable.

Proof. As noted in Section 2, the process of computing $f(n_1,\dots,n_m)$ according to the recursive definition of f can be represented by the sequence

$$(0,a_0),(1,a_1),\dots,(n,a_n),$$

where $a_0 = g(n_1,\dots,n_m)$, $a_{i+1} = h(a_i,i,n_1,\dots,n_m)$, for $i = 0,\dots,n-1$, and $a_n = f(n, n_1,\dots,n_m)$. Let a fundamental type τ of $\mathbf{a}_n$ be given. For $i = n,\dots,0$, we define by induction backwards a sequence of tuples of types $(\tau_i,\zeta_i,\sigma_i^1\dots,\sigma_i^m)$, so that τ_i, ζ_i and σ_i^j are fundamental types of $\mathbf{a}_i$, $\mathbf{i}$, and $\mathbf{n}_j$, respectively. We begin with $\tau_n = \tau$ and $\zeta_n = \sigma_n^j = \mathbf{int}$. Assume that $(\tau_{i+1},\zeta_{i+1},\sigma_{i+1}^1\dots,\sigma_{i+1}^m)$ is already defined. As h is properly representable by a term H, we may choose $(\tau_i,\zeta_i,\sigma_i^1\dots,\sigma_i^m)$ so that $\zeta_i = \zeta_i' \cap \zeta_i''$, and the following holds:

$$\vdash H : \tau_i \to \zeta_i' \to \sigma_i^1 \to \cdots \to \sigma_i^m \to \tau_{i+1},$$
$$\vdash \mathbf{succ} : \zeta_i'' \to \zeta_{i+1}.$$

Now g is properly representable by some G, and there are $\sigma_{-1}^1,\dots,\sigma_{-1}^m$, fundamental types of $n_1,\dots,n_m$, such that

$$\vdash G : \sigma_{-1}^1 \to \cdots \to \sigma_{-1}^m \to \tau_0.$$

For $j = 1,\dots,m$, let $\sigma^j = \sigma_{-1}^j \cap \sigma_0^j \cap \cdots \cap \sigma_n^j$.

It remains to define the type for the first argument of f. Let us use the following abbreviation:

$$\mathscr{A} \times \mathscr{B} = ((\mathscr{A} \to \mathscr{B} \to \mathscr{A}) \to \mathscr{A}) \cap ((\mathscr{A} \to \mathscr{B} \to \mathscr{B}) \to \mathscr{B}).$$

It should be readily seen that $\Gamma \vdash \langle M,N\rangle : \mathscr{A} \times \mathscr{B}$ holds whenever $\Gamma \vdash M : \mathscr{A}$ and $\Gamma \vdash N : \mathscr{B}$. Let ζ be the following type

$$((\zeta_0 \times \tau_0) \to (\zeta_1 \times \tau_1)) \cap \cdots \cap ((\zeta_{n-1} \times \tau_{n-1}) \to (\zeta_n \times \tau_n)) \to (\zeta_0 \times \tau_0) \to (\zeta_n \times \tau_n).$$

We leave it to the reader to verify that

$$\vdash F : \zeta \to \sigma^1 \to \cdots \to \sigma^m \to \tau.$$

□

Lemma 6. Let $\mathscr{A}$ and $\mathscr{B}$ be arbitrary types, and let

$$\rho_0(\mathscr{A},\mathscr{B}) = ((\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to(\mathscr{A}\to\mathscr{B}\to\mathscr{B}))\to(\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to\mathscr{A}\to\mathscr{B}\to\mathscr{A};$$
$$\rho_1(\mathscr{A},\mathscr{B}) = ((\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to(\mathscr{A}\to\mathscr{B}\to\mathscr{B}))\to(\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to\mathscr{A}\to\mathscr{B}\to\mathscr{B};$$
$$\rho_n(\mathscr{A},\mathscr{B}) = ((\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to(\mathscr{A}\to\mathscr{B}\to\mathscr{B}))\cap((\mathscr{A}\to\mathscr{B}\to\mathscr{B})\to(\mathscr{A}\to\mathscr{B}\to\mathscr{B}))\to (\mathscr{A}\to\mathscr{B}\to\mathscr{A})\to\mathscr{A}\to\mathscr{B}\to\mathscr{B}, \text{ when } n>1.$$

Clearly, $\rho_n(\mathscr{A},\mathscr{B})$ for each n is a straight type of $\mathbf{n}$. In addition, $\vdash \mathbf{zero} : \rho_0(\mathscr{A},\mathscr{B}) \to \mathscr{A} \to \mathscr{B} \to \mathscr{A}$ and, also, $\vdash \mathbf{zero} : \rho_n(\mathscr{A},\mathscr{B}) \to \mathscr{A} \to \mathscr{B} \to \mathscr{B}$, for $n \geqslant 1$. Note that $\mathscr{A} \to \mathscr{B} \to \mathscr{A}$ and $\mathscr{A} \to \mathscr{B} \to \mathscr{B}$ are types of **true** and **false**, respectively.

Proof. The proof is easy and omitted. □

Lemma 7. Let $f : \omega^m \to \omega$ be defined from total, properly representable functions $g : \omega^{m+1} \to \omega$ and $\ell : \omega \to \omega$ as follows:

$$f(n_1,\dots,n_m) = \ell(\mu n[g(n,n_1,\dots,n_m) = 0]).$$

Then f is properly representable.

Proof. Assume that g is properly representable by a term G, and let F be as in Section 2. Observe that the reduction (1) of Section 2 uses $2n+2$ copies of the term $\overline{W}$. We (informally) denote different copies by $\overline{W}_k$ and $\overline{W}^*_k$, where $k = 0, \ldots, n+1$, so that we can write our reduction sequence as follows:

$$F\mathbf{n}_1 \ldots \mathbf{n}_m \twoheadrightarrow_\beta \overline{W}_0\mathbf{0}\overline{W}^*_1 \twoheadrightarrow_\beta \overline{W}_1\mathbf{1}\overline{W}^*_2 \twoheadrightarrow_\beta \cdots \twoheadrightarrow_\beta \overline{W}_n\mathbf{n}\overline{W}^*_{n+1} \twoheadrightarrow_\beta L\mathbf{n} \twoheadrightarrow_\beta \mathbf{r}.$$

It is convenient to think that each $\overline{W}_k$ and $\overline{W}^*_k$ is obtained by an appropriate substitution from its own private copy of W, which is informally denoted by W_k and W^*_k, respectively. We will write F as $\lambda x_1 \ldots x_m.W_0\mathbf{0}W^*_1$. During our reduction we obtain many copies of W^*_1, and this copying process can be represented by the following scheme:

$$\begin{tikzcd} W^*_1 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & W^*_2 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & W^*_3 \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \cdots \arrow[r, Rightarrow] & W^*_n \arrow[r, Rightarrow] \arrow[d, Rightarrow] & W^*_{n+1} \\ W_1 & W_2 & W_3 & \cdots & W_n & \end{tikzcd}$$

Let v be a fixed straight type of $\mathbf{r}$. We will construct fundamental types $\sigma_1, \ldots, \sigma_m$ of $\mathbf{n}_1, \ldots, \mathbf{n}_m$ so that $\vdash F : \sigma_1 \to \cdots \to \sigma_m \to v$ will hold. First choose a fundamental type τ of $\mathbf{n}$, satisfying $L : \tau \to v$.

Our main task is to define an appropriate type for W. This type will be an intersection of all types that need to be assigned to all the copies of W along the reduction process. We describe types for these copies by induction, beginning with W^*_{n+1} and ending with W^*_1 and W_0.

We begin with W^*_{n+1}, the copy of W that is never applied to an argument. Let $[w] = \alpha \cap (\mathbf{int} \to \alpha \to \mathbf{int})$, where α is a fresh type variable. We define $\mathscr{A} = \mathscr{B} = [w] \to \mathbf{int}$. Suppose that $g(n+1, n_1, \ldots, n_m) = p$. Because g is properly defined by G, there are fundamental types ζ'_{n+1} and $\sigma^{n+1}_1, \ldots, \sigma^{n+1}_m$, of $\mathbf{n+1}$ and $\mathbf{n}_1, \ldots, \mathbf{n}_m$, respectively, such that $G : \zeta'_{n+1} \to \sigma^{n+1}_1 \to \cdots \to \sigma^{n+1}_m \to \rho_p(\mathscr{A}, \mathscr{B})$. (See Lemma 6.) Since $\mathscr{A} = \mathscr{B}$ anyway, this implies that $x_1 : \sigma^{n+1}_1, \ldots, x_m : \sigma^{n+1}_m, y : \zeta_{n+1} \vdash \mathbf{zero}(Gyx_1 \ldots x_m) : \mathscr{A} \to \mathscr{A} \to \mathscr{A}$. Now choose ζ''_{n+1} so that $L : \zeta''_{n+1} \to \mathbf{int}$, which guarantees that $y : \zeta''_{n+1} \vdash \lambda w.Ly : [w] \to \mathbf{int}$. We also have $y : \mathbf{int} \vdash \lambda w.w(\mathbf{succ}\, y)w : [w] \to \mathbf{int}$. Let us write $[n+1]$ for $\zeta'_{n+1} \cap \zeta''_{n+1} \cap \mathbf{int}$. It follows that (recall that $\mathscr{A} = [w] \to \mathbf{int}$):

$$\{x_j : \sigma^{n+1}_j\}_{j \leqslant m}, y : [n+1] \vdash \mathbf{if\ zero}(Gyx_1 \ldots x_m)\ \mathbf{then}\ \lambda w.Ly\ \mathbf{else}\ \lambda w.w(\mathbf{succ}\, y)w : \mathscr{A}.$$

Let $[W^*_{n+1}]$ stand for the type $[n+1] \to [w] \to \mathbf{int}$. We have obtained the following typing:

$$x_1 : \sigma^{n+1}_1, \ldots, x_m : \sigma^{n+1}_m \vdash W : [W^*_{n+1}].$$

The next step is to define a type $[W_n]$ for W_n. Take a fresh variable β, and define

$$\begin{aligned} \mathscr{A} &= [W^*_{n+1}] \to v; \\ \mathscr{B} &= ([n+1] \to \beta \to v) \cap \beta \to v. \end{aligned}$$

Take a fundamental type ζ' of $\mathbf{n}$, such that $\vdash \mathbf{succ} : \zeta' \to [n+1]$. Then we have

$$y : \tau \vdash \lambda w.Ly : \mathscr{A} \quad \text{and} \quad y : \zeta' \vdash \lambda w.w(\mathbf{succ}\, y)w : \mathscr{B}.$$

We know that $g(n, n_1, \ldots, n_m) = 0$. Take fundamental types ζ and $\xi_1^n, \ldots, \xi_m^n$ of $\mathbf{n}$ and $\mathbf{n}_1, \ldots, \mathbf{n}_m$, respectively, so that

$$y : \zeta, x_1 : \xi_1^n, \ldots, x_m : \xi_m^n \vdash Gyx_1 \ldots x_m : \rho_0(\mathscr{A}, \mathscr{B}),$$

and define $[n]$ as $\zeta \cap \zeta' \cap \tau$. Then we obtain

$$y{:}[n], \{x_j{:}\xi_j^n\}_{j \leqslant m} \vdash \textbf{if zero}(Gyx_1 \ldots x_m) \textbf{ then } \lambda w.Ly \textbf{ else } \lambda w.w(\textbf{succ}\, y)w : \mathscr{A}.$$

Thus, we can define $[W_n]$ as $[n] \to [W_{n+1}^*] \to v$, and we have the typing

$$x_1 : \xi_1^n, \ldots, x_m : \xi_m^n \vdash W : [W_n].$$

Now we define $[W_n^*]$ to be $[W_n] \cap [W_{n+1}^*]$. If we take σ_j^n as $\xi_j^n \cap \sigma_j^{n+1}$, for $j = 1, \ldots m$, we conclude with

$$x_1 : \sigma_1^n, \ldots, x_m : \sigma_m^n \vdash W : [W_n^*].$$

We proceed to the induction step for $k = n-1, n-2, \ldots, 1, 0$. We define types $[W_k]$ and $[W_k^*]$ so that the following holds:

(a) $[W_k] = [k] \to [W_{k+1}^*] \to v$, where $[k]$ is a fundamental type of k;
(b) $[W_k^*] = [W_k] \cap [W_{k+1}^*]$;
(c) $x_1 : \sigma_1^k, \ldots, x_m : \sigma_m^k \vdash W : [W_k^*]$, for some fundamental types $\sigma_1^k, \ldots, \sigma_m^k$ of $\mathbf{n}_1, \ldots, \mathbf{n}_m$.

Choose fundamental types ζ' and ζ'' of k, satisfying $L : \zeta' \to \textbf{int}$ and $\textbf{succ} : \zeta'' \to [k+1]$. Then take $\mathscr{A} = [W_{k+1}^*] \to \textbf{int}$ and $\mathscr{B} = [W_{k+1}^*] \to v$. By the induction hypothesis, we have $[W_{k+1}^*] = [W_{k+1}] \cap [W_{k+2}^*] = ([k+1] \to [W_{k+2}^*] \to v) \cap [W_{k+2}^*]$. Thus we have

$$y : \zeta' \vdash \lambda w.Ly : \mathscr{A} \quad \text{and} \quad y : \zeta'' \vdash \lambda w.w(\textbf{succ}\, y)w : \mathscr{B}.$$

We know that $g(k, n_1, \ldots, n_m) = p \neq 0$. We choose fundamental types ζ and $\xi_1^k, \ldots, \xi_m^k$ of $\mathbf{k}$ and $\mathbf{n}_1, \ldots, \mathbf{n}_m$, respectively, satisfying

$$y : \zeta, x_1{:}\xi_1^k, \ldots, x_m{:}\xi_m^k \vdash Gyx_1 \ldots x_m : \rho_p(\mathscr{A}, \mathscr{B}).$$

Finally, we take $[k] = \zeta \cap \zeta' \cap \zeta''$ and $\sigma_j^k = \xi_j^k \cap \sigma_j^{k+1}$, for $j = 1, \ldots, m$. Now $[W_k]$ and $[W_k^*]$ are defined according to conditions (a) and (b), respectively. It is routine to verify condition (c), and thus our induction step is completed.

Applying conditions (a)–(c) to $k = 0, 1$ we obtain that the desired fundamental types of $\mathbf{n}_1, \ldots, \mathbf{n}_m$ are $\sigma_1^0, \ldots, \sigma_m^0$. □

Theorem 8. Every partial recursive function is properly representable in intersection types.

Proof. The proof follows from Lemmas 3, 4, 5 and 7. □

Corollary 9. Strong normalization of pure lambda terms is undecidable.

Proof. The halting problem is reducible to strong normalization. Indeed, let a partial recursive function $f : \omega \to \omega$ be (effectively) represented by F. For a given n, ask whether $F\mathbf{n}$ is strongly normalizable. If it is, then $f(n)$ is defined, otherwise it is not. $\square$

Corollary 10. Strong normalization of combinatory terms is undecidable.

Proof. Yohji Akama (Akama 1997) gives a simple translation $(\ )_a$ from lambda calculus to Combinatory Logic, which preserves strong normalization. That is, a lambda term M is strongly normalizable if and only if the combinatory term $(M)_a$ is strongly normalizable. The translation is effective, and thus strong normalization of lambda terms reduces to strong normalization for combinatory terms. $\square$

Acknowledgement

I should like to thank Morten Heine Sørensen, who encouraged me to write down this proof. Also, thanks are due to Silvia Ghilezan and two anonymous referees for their comments.

References

Akama, Y. (1997) A lambda-to-CL translation for strong normalization. In: de Groote, P. (ed.) Proc. Typed Lambda Calculi and Applications. *Springer-Verlag Lecture Notes in Computer Science* **1210** 1–10.

Amadio, R.M. and Curien, P.-L. (1998) *Domains and Lambda Calculi*, Cambridge University Press.

Barendregt, H.P. (1984) *The Lambda Calculus: Its Syntax and Semantics*, 2nd edition, North-Holland.

Barendregt, H.P., Coppo, M. and Dezani-Ciancaglini, M. (1983) A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic* **48** (4) 931–940.

Cardone, F. and Coppo, M. (1990) Two extensions of Curry's type inference system. In: Oddifreddi, P. (ed.) *Logic and Computer Science*, Academic Press 19–75.

Coppo, M., Dezani-Ciancaglini, M. and Venneri, B. (1981) Functional character of solvable terms. *Z. Math. Log. Grund. Math.* **27** 45–58.

Henglein, F. and Mairson, H.G. (1994) The complexity of type inference for higher-order typed lambda calculi. *Journal of Functional Programming* **4** (4) 435–477.

Hindley, J.R. and Seldin, J.P. (1986) *Introduction to Combinators and λ-calculus*, Cambridge University Press.

Leivant, D. (1983) Polymorphic type inference. *Proc. of 10-th ACM Symposium on Principles of Programming Languages* 88–98.

Pottinger, G. (1980) A type assignment for the strongly normalizable λ-terms. In: Seldin, J.P. and Hindley, J.R. (eds.) *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press 561–577.

Rogers, H.Jr. (1967) *Theory of Recursive Functions and Effective Computability*, McGraw Hill.

Urzyczyn, P. (1997) Type reconstruction in $\mathbf{F}_\omega$. *Mathematical Structures in Computer Science* **7** 329–358. (Preliminary version in: Proc. TLCA '93, *Springer-Verlag Lecture Notes in Computer Science* **664** 418–432.)