

An arithmetical proof of the strong normalization for the λ -calculus with recursive equations on types

René David & Karim Nour*

Université de Savoie

Abstract. We give an arithmetical proof of the strong normalization of the λ -calculus (and also of the $\lambda\mu$ -calculus) where the type system is the one of simple types with recursive equations on types.

The proof using candidates of reducibility is an easy extension of the one without equations but this proof cannot be formalized in Peano arithmetic. The strength of the system needed for such a proof was not known. Our proof shows that it is not more than Peano arithmetic.

1 Introduction

The λ -calculus is a powerful model for representing functions. In its un-typed version, every recursive function can be represented. But, in this model, a term can be applied to itself and a computation may not terminate. To avoid this problem, types are used. In the simplest case, they are built from atomic types with the arrow and the typing rules say that a function of type $U \rightarrow V$ may only be applied to an argument of type U . This discipline ensures that every typed term is strongly normalizing, i.e. a computation always terminate.

In this system (the simply typed λ -calculus), Church numerals, i.e. the terms of the form $\lambda f \lambda x (f (f \dots (f x)))$, are codes for the integers. They are the only terms (in normal form) of type $(o \rightarrow o) \rightarrow (o \rightarrow o)$. Thus, functions on the integers can be represented but Schwichtenberg [38] has shown that very few functions are so. He showed that the extended polynomials (i.e. polynomials with positive coefficients together with a conditional operator) are the only functions that can be represented there. Other type systems were then designed to allow the representation of more functions. They are built in different ways.

The first one consists in extending the set of terms. For example, in Gödel system T , the terms use the usual constructions of the λ -calculus, the constant 0, the constructor S and an operator for recursion. The types are built from the atomic type N with the arrow. This system represents exactly the functions whose totality can be shown in Peano first order arithmetic.

The second one consists in keeping the same terms but extending the type system. This is, for example, the case of Girard system F where the types can use a second order universal quantifier. There, the type of the integers is given by $\forall X ((X \rightarrow X) \rightarrow (X \rightarrow X))$. This system represents exactly the functions whose totality can be shown in Peano second order arithmetic.

A third way consists in extending the *logic*. In the Curry-Howard correspondence, the previous systems correspond to *intuitionistic* logic. Other systems correspond to *classical* logic. There, again, new constructors for terms are introduced. This is, for example, the case of Parigot's $\lambda\mu$ -calculus [35].

* Université de Savoie, Campus Scientifique, 73376 Le Bourget du Lac, France.
Email : {david, nour}@univ-savoie.fr

Since the introduction of Girard system F for intuitionistic logic and Parigot's $\lambda\mu$ -calculus for classical logic, many others, more and more powerful, type systems were introduced. For example, the calculus of constructions (Coquand & Huet [7]) and, more generally, the Pure Type Systems.

It is also worth here to mention the system TTR of Parigot [33] where some types are defined as the least fixed point of an operator. This system was introduced, not to represent more functions, but to represent more *algorithms*. For example, to be able to represent the integers in such a way that the predecessor can be computed in constant time, which is not the case for the previous systems.

These systems all satisfy the subject reduction (i.e. the fact that the type is preserved by reduction), the strong normalization (i.e. every computation terminates) and, for the systems based on simple types, the decidability of type assignment.

We study here other kinds of extension of the simply typed λ -calculus, i.e. systems where *equations* on types are allowed. These types are usually called *recursive types*. For more details see, for example, [3]. They are present in many languages and are intended to be able to be *unfolded* recursively to match other types. The subject reduction and the decidability of type assignment are preserved but the strong normalization may be lost. For example, with the equation $X = X \rightarrow T$, the term $(\delta \delta)$ where $\delta = \lambda x (x x)$ is typable but is not strongly normalizing. With the equation $X = X \rightarrow X$, every term can be typed.

By making some natural assumptions on the recursive equations the strong normalization can be preserved. The simplest condition is to accept the equation $X = F$ (where F is a type containing the variable X) only when the variable X is positive in F . For a set $\{X_i = F_i \mid i \in I\}$ of mutually recursive equations, Mendler [29] has given a very simple and natural condition that ensures the strong normalization of the system. He also showed that the given condition is necessary to have the strong normalization. His proof is based on the reducibility method. The condition ensures enough monotonicity to have fixed point on the candidates. But this proof (using candidates of reducibility) cannot be formalized in Peano arithmetic and the strength of the system needed for a proof of the strong normalization of such systems was not known.

In this paper, we give an *arithmetical* proof of the strong normalization of the simply typed λ -calculus (and also of the $\lambda\mu$ -calculus) with recursive equations on types satisfying Mendler's condition.

This proof is an extension of the one given by the first author for the simply typed λ -calculus. It can be found either in [8] (where it appears among many other things) or as a simple unpublished note on the web page of the first author [9]. Apparently, proof methods similar to that used here were independently invented by several authors (Levy, van Daalen, Valentini and others). The proof for the $\lambda\mu$ -calculus is an extension of the ones given in [11] or [12].

The paper is organized as follows. In section 2 we define the simply typed λ -calculus with recursive equations on types. To help the reader and show the main ideas, we first give, in section 3, the proof of strong normalization for the λ -calculus. We generalize this proof to the $\lambda\mu$ -calculus in section 4. In section 5, we give two examples of applications of systems with recursive types. We conclude in section 6 with some open questions.

2 The typed λ -calculus

Definition 1. Let \mathcal{V} be an infinite set of variables.

1. The set \mathcal{M} of λ -terms is defined by the following grammar

$$\mathcal{M} ::= \mathcal{V} \mid \lambda \mathcal{V} \mathcal{M} \mid (\mathcal{M} \mathcal{M})$$

2. The relation \triangleright on \mathcal{M} is defined as the least relation (compatible with the context) containing the rule $(\lambda x M N) \triangleright M[x := N]$. As usual, \triangleright^* (resp. \triangleright^+) denotes the reflexive and transitive (resp. transitive) closure of \triangleright .

Definition 2. Let \mathcal{A} be a set of atomic constants and $\mathcal{X} = \{X_i / i \in I\}$ be a set of type variables.

1. The set \mathcal{T} of types is defined by the following grammar

$$\mathcal{T} ::= \mathcal{A} \mid \mathcal{X} \mid \mathcal{T} \rightarrow \mathcal{T}$$

2. When $E = \{F_i / i \in I\}$ is a set of types, the congruence \approx generated by E is the least congruence on \mathcal{T} such that $X_i \approx F_i$ for each $i \in I$.

Definition 3. Let \approx be a congruence on \mathcal{T} . The typing rules of the typed system are given below where Γ is a context, i.e. a set of declarations of the form $x : U$ where $x \in \mathcal{V}$ and $U \in \mathcal{T}$.

$$\begin{array}{c} \frac{}{\Gamma, x : U \vdash x : U} \text{ ax} \quad \frac{\Gamma \vdash M : U \quad U \approx V}{\Gamma \vdash M : V} \approx \\ \frac{\Gamma, x : U \vdash M : V}{\Gamma \vdash \lambda x M : U \rightarrow V} \rightarrow_i \quad \frac{\Gamma \vdash M_1 : U \rightarrow V \quad \Gamma \vdash M_2 : U}{\Gamma \vdash (M_1 M_2) : V} \rightarrow_e \end{array}$$

Lemma 1. Let \approx be a congruence generated by a set of types.

1. If $U \approx V_1 \rightarrow V_2$, then $U \in \mathcal{X}$ or $U = U_1 \rightarrow U_2$.
2. If $U_1 \rightarrow V_1 \approx U_2 \rightarrow V_2$, then $U_1 \approx U_2$ and $V_1 \approx V_2$.
3. If $\Gamma \vdash x : T$, then $x : U$ occurs in Γ for some $U \approx T$.
4. If $\Gamma \vdash \lambda x M : T$, then $\Gamma, x : U \vdash M : V$ for some U, V such that $U \rightarrow V \approx T$.
5. If $\Gamma \vdash (MN) : T$, then $\Gamma \vdash M : U \rightarrow V$, $\Gamma \vdash N : U$ for some $V \approx T$ and U .
6. If $\Gamma, x : U \vdash M : T$ and $U \approx V$, then $\Gamma, x : V \vdash M : T$.
7. If $\Gamma, x : U \vdash M : T$ and $\Gamma \vdash N : U$, then $\Gamma \vdash M[x := N] : T$.

Proof Easy. □

Theorem 1. If $\Gamma \vdash M : T$ and $M \triangleright^* M'$, then $\Gamma \vdash M' : T$.

Proof It is enough to show that if $\Gamma \vdash (\lambda x M N) : T$, then $\Gamma \vdash M[x := N] : T$. Assume $\Gamma \vdash (\lambda x M N) : T$. By lemma 1, $\Gamma \vdash \lambda x M : U \rightarrow V$, $\Gamma \vdash N : U$ and $V \approx T$. Thus, $\Gamma, x : U' \vdash M : V'$ and $U' \rightarrow V' \approx U \rightarrow V$. By lemma 1, we have $U' \approx U$ and $V' \approx V$. Thus, $\Gamma, x : U \vdash M : V$. Since $\Gamma \vdash N : U$ and $V \approx T$, the result follows immediately. □

Definition 4. Let $X \in \mathcal{X}$. We define the subsets $\mathcal{T}^+(X)$ and $\mathcal{T}^-(X)$ of \mathcal{T} as follows.

- $X \in \mathcal{T}^+(X)$
- If $U \in (\mathcal{X} - \{X\}) \cup \mathcal{A}$, then $U \in \mathcal{T}^+(X) \cap \mathcal{T}^-(X)$.
- If $U \in \mathcal{T}^-(X)$ and $V \in \mathcal{T}^+(X)$, then $U \rightarrow V \in \mathcal{T}^+(X)$ and $V \rightarrow U \in \mathcal{T}^-(X)$.

Definition 5. We say that a congruence \approx is good if the following property holds: for each $X \in \mathcal{X}$, if $X \approx T$, then $T \in \mathcal{T}^+(X)$.

Examples

In each of the following cases, the congruence generated by the given equations is good.

1. $X_1 \approx (X_1 \rightarrow X_2 \rightarrow Y) \rightarrow Y$ and $X_2 \approx (X_2 \rightarrow X_1 \rightarrow Y) \rightarrow Y$.
2. $X_1 \approx X_2 \rightarrow X_1$ and $X_2 \approx X_1 \rightarrow X_2$.

3. The same equations as in case 2 and $X_3 \approx F(X_1, X_2) \rightarrow X_3$ where F is any type using only the variables X_1, X_2 .
4. The same equations as in case 3 and $X_4 \approx X_5 \rightarrow G(X_1, X_2, X_3) \rightarrow X_4$, $X_5 \approx X_4 \rightarrow H(X_1, X_2, X_3) \rightarrow X_5$ where G, H are any types using only the variables X_1, X_2, X_3 .

In the rest of the paper, we fix a finite set $E = \{F_i \mid i \in I\}$ of types and we denote by \approx the congruence generated by E . We assume that \approx is good.

Notations and remarks

- We have assumed that the set of equations that we consider is finite. This is to ensure that the order on I given by definition 6 below is well founded. It should be clear that this is not a real constraint. Since to type a term, only a finite number of equations is used, we may consider that the other variables are constant and thus the general result follows immediately from the finite case.
- If M is a term, $cxy(M)$ will denote the structural complexity of M .
- We denote by SN the set of strongly normalizing terms. If $M \in SN$, we denote by $\eta(M)$ the length of the longest reduction of M and by $\eta c(M)$ the pair $\langle \eta(M), cxy(M) \rangle$.
- We denote by $M \preceq N$ the fact that M is a sub-term of a reduct of N .
- As usual, some parentheses are omitted and, for example, we write $(M P Q)$ instead of $((M P) Q)$. More generally, if \vec{O} is a finite sequence O_1, \dots, O_n of terms, we denote by $(M \vec{O})$ the term $((\dots(M O_1) \dots O_{n-1}) O_n)$ and by $\vec{O} \in SN$ the fact that $O_1, \dots, O_n \in SN$.
- If σ is the substitution $[x_1 := N_1, \dots, x_n := N_n]$, we denote by $dom(\sigma)$ the set $\{x_1, \dots, x_n\}$, by $Im(\sigma)$ the set $\{N_1, \dots, N_n\}$ and by $\sigma \in SN$ the fact that $Im(\sigma) \subset SN$.
- If σ is a substitution, $z \notin dom(\sigma)$ and M is a term, we denote by $[\sigma + z := M]$ the substitution σ' defined by $\sigma'(x) = \sigma(x)$ for $x \in dom(\sigma)$ and $\sigma'(z) = M$.
- In a proof by induction, IH will denote the induction hypothesis. When the induction is done on a tuple of integers, the order always is the lexicographic order.

3 Proof of the strong normalization

3.1 The idea of the proof

We give the idea for one equation $X \approx F$. The extension for the general case is given at the beginning of section 3.4.

It is enough to show that, if M, N are in SN , then $M[x := N] \in SN$. Assuming it is not the case, the interesting case is $M = (x P)$ with $(N P_1) \notin SN$ where $P_1 = P[x := N] \in SN$. This implies that $N \triangleright^* \lambda y N_1$ and $N_1[y = P_1] \notin SN$. If we know that the type of N is an arrow type, we get a similar situation to the one we started with, but where the type of the substituted variable has decreased. Repeating the same argument, we get the desired result, at least for N whose type does not contain X . If it is not the case, since, by repeating the same argument, we cannot come to a constant type (because such a term cannot be applied to something), we come to X . Thus, it remains to show that, if M, N are in SN and the type of x is X , then $M[x := N] \in SN$.

To prove this, we prove something a bit more general. We prove that, if $M, \sigma \in SN$ where σ is a substitution such that the types of its image are in $\mathcal{T}^+(X)$, then $M[\sigma] \in SN$. The proof is done, by induction on $\eta c(M)$ as follows. As before, the interesting case is $M = (x P)$, $\sigma(x) = N \triangleright^* \lambda y N_1$, $P_1 = P[\sigma] \in SN$ and

$N_1[y = P_1] \notin SN$. Thus, there is a sub-term of a reduct of N_1 of the form $(y N_2)$ such that $(P_1 N_2[y := P_1]) \notin SN$ but $N_2[y := P_1] \in SN$. Thus P_1 must reduce to a λ .

This λ cannot come from some $x' \in \text{dom}(\sigma)$, i.e. $P \triangleright^* (x' \overrightarrow{Q})$. Otherwise, the type of P would be both positive (since $P \triangleright^* (x' \overrightarrow{Q})$ and the type of x' is positive) and negative (since, in M , P is an argument of x whose type also is positive). Thus the type of P_1 (the same as the one of P) does not contain X . But since N_1, P_1 are in SN , we already know that $N_1[y = P_1]$ must be in SN . A contradiction. Thus, $P \triangleright^* \lambda x_1 M_1$ and we get a contradiction from the induction hypothesis since we have $M_1[\sigma'] \notin SN$ for M_1 strictly less than M . The case when y has more than one argument is intuitively treated by “repeat the same argument” or, more formally, by lemma 8 below.

As a final remark, note that many lemmas are stated in a negative style and thus may seem to hold only classically. This has been done in this way because we believe that this presentation is closer to the intuition. However, it is not difficult to check that the whole proof can be presented and done in a constructive way.

3.2 Some useful lemmas on the un-typed calculus

Lemma 2. Assume $M, N, \overrightarrow{O} \in SN$ and $(M N \overrightarrow{O}) \notin SN$. Then, for some term M' , $M \triangleright^* \lambda x M'$ and $(M'[x := N] \overrightarrow{O}) \notin SN$.

Proof Since $M, N, \overrightarrow{O} \in SN$, an infinite reduction of $P = (M N \overrightarrow{O})$ looks like $P \triangleright^* (\lambda x M' N' \overrightarrow{O'}) \triangleright (M'[x := N'] \overrightarrow{O'}) \triangleright \dots$ and the result immediately follows from the fact that $(M'[x := N] \overrightarrow{O}) \triangleright^* (M'[x := N'] \overrightarrow{O'})$. \square

Lemma 3. Let M be a term and σ be a substitution. Assume $M, \sigma \in SN$ and $M[\sigma] \notin SN$. Then $(\sigma(x) \overrightarrow{P[\sigma]}) \notin SN$ for some $(x \overrightarrow{P}) \preceq M$ such that $\overrightarrow{P[\sigma]} \in SN$.

Proof A sub-term M' of a reduct of M such that $\eta c(M')$ is minimum and $M'[\sigma] \notin SN$ has the desired form. \square

Lemma 4. Let M be a term and σ be a substitution such that $M[\sigma] \triangleright^* \lambda z M_1$. Then
- either $M \triangleright^* \lambda z M_2$ and $M_2[\sigma] \triangleright^* M_1$
- or $M \triangleright^* (x \overrightarrow{N})$ for some $x \in \text{dom}(\sigma)$ and $(\sigma(x) \overrightarrow{N[\sigma]}) \triangleright^* \lambda z M_1$.

Proof This is a classical (though not completely trivial) result in λ -calculus. Note that, in case $M \in SN$ (and we will only use the lemma in this case), it becomes easier. The proof can be done by induction on $\eta c(M)$ by considering the possibility for M : either $\lambda y M_1$ or $(\lambda y M_1 P \overrightarrow{Q})$ or $(x \overrightarrow{N})$ (for x in $\text{dom}(\sigma)$ or not). \square

3.3 Some useful lemmas on the congruence

Definition 6. We define on I the following relations

- $i \leq j$ iff $X_i \in \text{var}(T)$ for some T such that $X_j \approx T$.
- $i \sim j$ iff $i \leq j$ and $j \leq i$.
- $i < j$ iff $i \leq j$ and $j \not\sim i$

It is clear that \sim is an equivalence on I .

Definition 7. 1. Let $\mathcal{X}_i = \{X_j / j \leq i\}$ and $\mathcal{X}'_i = \{X_j / j < i\}$.
2. For $\mathcal{Y} \subseteq \mathcal{X}$, let $\mathcal{T}(\mathcal{Y}) = \{T \in \mathcal{T} / \text{var}(T) \subseteq \mathcal{Y}\}$ where $\text{var}(T)$ is the set of type variables occurring in T .
3. For $i \in I$, we will abbreviate by \mathcal{T}_i the set $\mathcal{T}(\mathcal{X}_i)$ and by \mathcal{T}'_i the set $\mathcal{T}(\mathcal{X}'_i)$.

4. If $\varepsilon \in \{+, -\}$, $\bar{\varepsilon}$ will denote the opposite of ε . The opposite of $+$ is $-$ and conversely.

Lemma 5. *Let $i \in I$. The class of i can be partitioned into two disjoint sets i^+ and i^- satisfying the following properties.*

1. If $\varepsilon \in \{+, -\}$, $j \in i^\varepsilon$ and $X_j \approx T$, then for each $k \in i^\varepsilon$, $T \in \mathcal{T}^\varepsilon(X_k)$ and for each $k \in i^{\bar{\varepsilon}}$, $T \in \mathcal{T}^{\bar{\varepsilon}}(X_k)$.
2. Let $j \sim i$. Then, if $j \in i^+$, $j^+ = i^+$ and $j^- = i^-$ and if $j \in i^-$, $j^+ = i^-$ and $j^- = i^+$.

Proof This follows immediately from the following observation. Let $i \sim j$ and $X_i \approx T \approx U$. Choose an occurrence of X_j in T and in U . Then, these occurrences have the same polarity. This is because, otherwise, since $i \leq j$, there is a V such that $X_j \approx V$ and X_i occurs in V . But then, replacing the mentioned occurrences of X_j by V in T and U will contradict the fact that \approx is good. \square

Definition 8. *Let $i \in I$ and $\varepsilon \in \{+, -\}$. We denote $\mathcal{T}_i^\varepsilon = \{T \in \mathcal{T}_i \mid \text{for each } j \in i^\varepsilon, T \in \mathcal{T}^\varepsilon(X_j) \text{ and for each } j \in i^{\bar{\varepsilon}}, T \in \mathcal{T}^{\bar{\varepsilon}}(X_j)\}$.*

Lemma 6. *Let $i \in I$ and $\varepsilon \in \{+, -\}$.*

1. $\mathcal{T}_i^\varepsilon \cap \mathcal{T}_i^{\bar{\varepsilon}} \subseteq \mathcal{T}_i'$.
2. If $U \in \mathcal{T}_i^\varepsilon$ and $U \approx V$, then $V \in \mathcal{T}_i^\varepsilon$.
3. If $U \in \mathcal{T}_i^\varepsilon$ and $U \approx U_1 \rightarrow U_2$, then $U_1 \in \mathcal{T}_i^{\bar{\varepsilon}}$ and $U_2 \in \mathcal{T}_i^\varepsilon$.

Proof Immediate. \square

Notations, remarks and examples

- If the equations are those of the case 4 of the examples given above, we have $1 \sim 2 < 3 < 4 \sim 5$ and, for example, $1^+ = \{1\}$ and $1^- = \{2\}$, $3^+ = \{3\}$, $3^- = \emptyset$, $4^+ = \{4\}$ and $4^- = \{5\}$.
- If T is a type, we denote by $lg(T)$ the size of T . Note that the size of a type is, of course, not preserved by the congruence. The size of a type will only be used in lemma 7 and the only property that we will use is that $lg(U_1)$ and $lg(U_2)$ are less than $lg(U_1 \rightarrow U_2)$.
- By the typing rules, the type of a term can be freely replaced by an equivalent one. However, for $i \in I$ and $\varepsilon \in \{+, -\}$, the fact that $U \in \mathcal{T}_i^\varepsilon$ does not change when U is replaced by V for some $V \approx U$. This will be used extensively in the proofs of the next sections.

3.4 Proof of the strong normalization

To give the idea of the proof, we first need a definition.

Definition 9. *Let \mathcal{E} be a set of types. Denote by $H[\mathcal{E}]$ the following property:*

Let $M, N \in SN$. Assume $\Gamma, x : U \vdash M : V$ and $\Gamma \vdash N : U$ for some Γ, U, V such that $U \in \mathcal{E}$. Then $M[x := N] \in SN$.

To get the result, it is enough to show $H[\mathcal{T}]$. The proof that any typed term is in SN is then done by induction on $cxtty(M)$. The only non trivial case is $M = (M_1 M_2)$. But $M = (x M_2)[x := M_1]$ and the result follows from $H[\mathcal{T}]$ and the IH.

We first show the following (see lemma 7). Let $\mathcal{Y} \subseteq \mathcal{X}$. To prove $H[\mathcal{T}(\mathcal{Y})]$, it is enough to prove $H[\{X\}]$ for each $X \in \mathcal{Y}$.

It is thus enough to prove of $H[\{X_i\}]$ for each $i \in I$. This is done by induction on i . Assume $H[\{X_j\}]$ for each $j < i$. Thus, by the previous property, we know

$H[\mathcal{T}'_i]$. We show $H[\{X_i\}]$ essentially as we said in section 3.1. The only difference is that, what was called there “ X is both positive and negative in T ” here means T is both in \mathcal{T}_i^+ and \mathcal{T}_i^- . There we deduced that X does not occur in T . Here we deduce $T \in \mathcal{T}'_i$ and we are done since we know the result for this set.

Lemma 7. *Let $\mathcal{Y} \subseteq \mathcal{X}$ be such that $H[\{X\}]$ holds for each $X \in \mathcal{Y}$. Then $H[\mathcal{T}(\mathcal{Y})]$ holds.*

Proof Let M, N be terms in SN . Assume $\Gamma, x : U \vdash M : V$ and $\Gamma \vdash N : U$ and $U \in \mathcal{T}(\mathcal{Y})$. We have to show $M[x := N] \in SN$.

This is done by induction on $lg(U)$. Assume $M[x := N] \notin SN$. By lemma 3, let $(x P \vec{Q}) \preceq M$ be such that $P_1, \vec{Q}_1 \in SN$ and $(N P_1 \vec{Q}_1) \notin SN$ where $P_1 = P[x := N]$ and $\vec{Q}_1 = \vec{Q}[x := N]$. By lemma 2, $N \triangleright^* \lambda x_1 N_1$ and $(N_1[x_1 := P_1] \vec{Q}_1) \notin SN$.

If U is a variable (which is in \mathcal{Y} since $U \in \mathcal{T}(\mathcal{Y})$), we get a contradiction since we have assumed that $H[\{X\}]$ holds for each $X \in \mathcal{Y}$.

The type U cannot be a constant since, otherwise x could not be applied to some arguments.

Thus $U = U_1 \rightarrow U_2$. In the typing of $(N P_1 \vec{Q}_1)$, the congruence may have been used and thus, by lemma 1, there are $W_1 \approx U_1$, $W_2 \approx U_2$, $U \approx W_1 \rightarrow W_2$ and $\Gamma, x_1 : W_1 \vdash N_1 : W_2$ and $\Gamma \vdash P_1 : W_1$. But then, we also have $\Gamma, x_1 : U_1 \vdash N_1 : U_2$ and $\Gamma \vdash P_1 : U_1$. Now, by the *IH*, we have $N_1[x_1 := P_1] \in SN$ since $lg(U_1) < lg(U)$. Since $\Gamma, z : U_2 \vdash (z \vec{Q}_1) : V'$ for some V' and $\Gamma \vdash N_1[x_1 := P_1] : U_2$, by the *IH* since $lg(U_2) < lg(U)$, we have $(N_1[x_1 := P_1] \vec{Q}_1) = (z \vec{Q}_1)[z = N_1[x_1 := P_1]] \in SN$. Contradiction. \square

For now on, we fix some i and we assume $H[\{X_j\}]$ for each $j < i$. Thus, by lemma 7, we know that $H[\mathcal{T}'_i]$ holds. It remains to prove $H[\{X_i\}]$ i.e. proposition 1.

Definition 10. *Let M be a term, σ be a substitution, Γ be a context and U be a type. Say that (σ, Γ, M, U) is adequate if the following holds.*

- $\Gamma \vdash M[\sigma] : U$ and $M, \sigma \in SN$.
- For each $x \in \text{dom}(\sigma)$, $\Gamma \vdash \sigma(x) : V_x$ and $V_x \in \mathcal{T}_i^+$.

Lemma 8. *Let n, m be integers, \vec{S} be a sequence of terms and (δ, Δ, P, B) be adequate. Assume that*

1. $B \in \mathcal{T}_i^- - \mathcal{T}'_i$ and $\Delta \vdash (P[\delta] \vec{S}) : W$ for some W .
2. $\vec{S} \in SN$, $P \in SN$ and $\eta c(P) < \langle n, m \rangle$.
3. $M[\sigma] \in SN$ for every adequate (σ, Γ, M, U) such that $\eta c(M) < \langle n, m \rangle$.

Then $(P[\delta] \vec{S}) \in SN$.

Proof By induction on the length of \vec{S} . If \vec{S} is empty, the result follows from (3) since $\eta c(P) < \langle n, m \rangle$. Otherwise, let $\vec{S} = S_1 \vec{S}_2$ and assume that $P[\delta] \triangleright^* \lambda z R$. By lemma 4, there are two cases to consider:

- $P \triangleright^* \lambda z R'$. We have to show that $Q = (R'[\delta + z := S_1] \vec{S}_2) \in SN$. Since $B \in \mathcal{T}_i^-$, by lemmas 1 and 6, there are types B_1, B_2 such that $B \approx B_1 \rightarrow B_2$ and $\Delta, z : B_1 \vdash R' : B_2$ and $\Delta \vdash S_1 : B_1$ and $B_1 \in \mathcal{T}_i^+$ and $B_2 \in \mathcal{T}_i^-$. Since $\eta c(R') < \langle n, m \rangle$ and $([\delta + z = S_1], \Delta \cup \{z : B_1\}, R', B_2)$ is adequate, it follows from (3) that $R'[\delta + z := S_1] \in SN$.
 - Assume first $B_2 \in \mathcal{T}'_i$. Since $(z' \vec{S}_2) \in SN$ and $Q = (z' \vec{S}_2)[z' := R'[\delta + z := S_1]]$, the result follows from $H[\mathcal{T}'_i]$.
 - Otherwise, the result follows from the *IH* since $([\delta + z = S_1], \Delta \cup \{z : B_1\}, R', B_2)$ is adequate and the length of \vec{S}_2 is less than the one of \vec{S} .
- If $P \triangleright^* (y \vec{T})$ for some $y \in \text{dom}(\delta)$. Then $\Delta \vdash (\delta(y) \vec{T}[\vec{\delta}]) : B$. By the definition of adequacy, the type of y is in \mathcal{T}_i^+ and $B \in \mathcal{T}_i^- \cap \mathcal{T}_i^+ \subseteq \mathcal{T}'_i$. Contradiction. \square

Lemma 9. Assume (σ, Γ, M, A) is adequate. Then $M[\sigma] \in SN$.

Proof By induction on $\eta c(M)$. The only non trivial case is $M = (x \ Q \ \vec{O})$ for some $x \in \text{dom}(\sigma)$. Let $N = \sigma(x)$.

By the IH, $Q[\sigma], \vec{O}[\sigma] \in SN$. By lemma 1, we have $V_x \approx W_1 \rightarrow W_2$, $\Gamma \vdash Q[\sigma] : W_1$ and $\Gamma \vdash (N \ Q[\sigma]) : W_2$. Moreover, by lemma 6, $W_1 \in \mathcal{T}_i^-$ and $W_2 \in \mathcal{T}_i^+$. Since $M[\sigma] = (z \ \vec{O})[\sigma + z := (N \ Q[\sigma])]$, $\eta((z \ \vec{O})) \leq \eta(M)$, $\text{cxtty}((z \ \vec{O})) < \text{cxtty}(M)$ and $W_2 \in \mathcal{T}_i^+$, it is enough, by the IH, to show that $(N \ Q[\sigma]) \in SN$. Assume that $N \triangleright^* \lambda y \ N'$. We have to show that $N'[y := Q[\sigma]] \in SN$.

- Assume first $W_1 \in \mathcal{T}_i'$. The result follows from $H[\mathcal{T}_i']$.

- Otherwise, assume $N'[y := Q[\sigma]] \notin SN$. Since $N', Q[\sigma] \in SN$, by lemma 3, $(y \ \vec{L}) \preceq N'$ for some \vec{L} such that $\vec{L}[y := Q[\sigma]] \in SN$ and $(Q[\sigma] \ \vec{L}[y := Q[\sigma]]) \notin SN$. But this contradicts lemma 8. Note that, by the IH, condition (3) of this lemma is satisfied. \square

Proposition 1. Assume $\Gamma, x : X_i \vdash M : U$ and $\Gamma \vdash N : X_i$ and $M, N \in SN$. Then $M[x := N] \in SN$.

Proof This follows from lemma 9 since $([x := N], \Gamma, M, U)$ is adequate. \square

4 The typed $\lambda\mu$ -calculus

Definition 11. 1. Let \mathcal{W} be an infinite set of variables such that $\mathcal{V} \cap \mathcal{W} = \emptyset$. An element of \mathcal{V} (resp. \mathcal{W}) is said to be a λ -variable (resp. a μ -variable). We extend the set of terms by the following rules

$$\mathcal{M} ::= \dots \mid \mu \mathcal{W} \mathcal{M} \mid (\mathcal{W} \mathcal{M})$$

2. We add to the set \mathcal{A} the constant symbol \perp and we denote by $\neg U$ the type $U \rightarrow \perp$.
3. We extend the typing rules by

$$\frac{\Gamma, \alpha : \neg U \vdash M : \perp}{\Gamma \vdash \mu \alpha M : U} \perp_e \quad \frac{\Gamma, \alpha : \neg U \vdash M : U}{\Gamma, \alpha : \neg U \vdash (\alpha M) : \perp} \perp_i$$

where Γ is now a set of declarations of the form $x : U$ and $\alpha : \neg U$ where x is a λ -variable and α is a μ -variable.

4. We add to \triangleright the following reduction rule $(\mu \alpha M \ N) \triangleright \mu \alpha M[\alpha = N]$ where $M[\alpha = N]$ is obtained by replacing each sub-term of M of the form $(\alpha \ P)$ by $(\alpha \ (P \ N))$. This substitution will be called a μ -substitution whereas the (usual) substitution $M[x := N]$ will be called a λ -substitution.

Remarks

- Note that we adopt here a more liberal syntax (also called de Groote's calculus [13]) than in the original calculus since we do not ask that a $\mu \alpha$ is immediately followed by a $(\beta \ M)$ (denoted $[\beta]M$ in Parigot's notation).
- We also have changed Parigot's typing notations. Instead of writing $M : (A_1^{x_1}, \dots, A_n^{x_n} \vdash B, C_1^{\alpha_1}, \dots, C_m^{\alpha_m})$ we have written $x_1 : A_1, \dots, x_n : A_n, \alpha_1 : \neg C_1, \dots, \alpha_m : \neg C_m \vdash M : B$ but, since the first introduction of the $\lambda\mu$ -calculus, this is now quite common.
- Unlike for a λ -substitution where, in $M[x := N]$, the variable x has disappeared it is important to note that, in a μ -substitution, the variable α has not disappeared. Moreover its type has changed. If the type of N is U and, in M , the type of α is $\neg(U \rightarrow V)$ it becomes $\neg V$ in $M[\alpha = N]$.

- The definition of good congruence is the same as before. As a consequence, we now have the following facts. If $U \approx \perp$, then $U = \perp$ and, if $\neg U \approx \neg V$, then $U \approx V$.
- We also extend all the notations given in section 2. Finally note that lemma 1 remains valid. Moreover, they are easily extended by lemma 10 below.

Lemma 10. 1. If $\Gamma \vdash \mu\alpha M : U$, then $\Gamma, \alpha : \neg V \vdash M : \perp$ for some V such that $U \approx V$.

2. If $\Gamma, \alpha : \neg U \vdash (\alpha M) : T$, then $\Gamma, \alpha : \neg U \vdash M : U$ and $T = \perp$.

3. If $\Gamma, \alpha : \neg(U \rightarrow V) \vdash M : T$ and $\Gamma \vdash N : U$, then $\Gamma, \alpha : \neg V \vdash M[\alpha = N] : T$.

Theorem 2. If $\Gamma \vdash M : T$ and $M \triangleright^* M'$, then $\Gamma \vdash M' : T$.

Proof It is enough to show that, if $\Gamma \vdash (\mu\alpha M N) : T$, then $\Gamma \vdash \mu\alpha M[\alpha = N] : T$. Assume $\Gamma \vdash (\mu\alpha M N) : T$. By lemma 1, $\Gamma \vdash \mu\alpha M : U \rightarrow V$, $\Gamma \vdash N : U$ and $V \approx T$. Thus, $\Gamma, \alpha : \neg T' \vdash M : \perp$ and $T' \approx U \rightarrow V$. By lemma 1, we have $\Gamma, \alpha : \neg(U \rightarrow V) \vdash M : \perp$. Since $\Gamma \vdash N : U$ and $V \approx T$, $\Gamma, \alpha : \neg V \vdash M[\alpha = N] : \perp$. Then $\Gamma \vdash \mu\alpha M[\alpha = N] : V$ and $\Gamma \vdash \mu\alpha M[\alpha = N] : T$. \square

4.1 Some useful lemmas on the un-typed calculus

Lemma 11. Let M be a term and $\sigma = \sigma_1 \cup \sigma_2$ where σ_1 (resp. σ_2) is λ (resp. μ) substitution. Assume $M[\sigma] \triangleright^* \mu\alpha M_1$ (resp. $\lambda y M_1$). Then

- either $M \triangleright^* \mu\alpha M_2$ (resp. $\lambda y M_2$) and $M_2[\sigma] \triangleright^* M_1$
- or $(M \triangleright^* (x \vec{N}))$ for some $x \in \text{dom}(\sigma_1)$ and $(\sigma(x) \vec{N}[\sigma]) \triangleright^* \mu\alpha M_1$ (resp. $\lambda y M_1$).

Proof A μ -substitution cannot create a λ or a μ (see, for example, [11]) and thus, the proof is as in lemma 4. \square

Lemma 12. Assume $M, P, \vec{Q} \in SN$ and $(M P \vec{Q}) \notin SN$. Then either $(M \triangleright^* \lambda x M_1$ and $(M_1[x := P] \vec{Q}) \notin SN)$ or $(M \triangleright^* \mu\alpha M_1$ and $(\mu\alpha M_1[\alpha = P] \vec{Q}) \notin SN)$.

Proof As in lemma 2. \square

Lemma 13. Let M be a term and σ be a λ -substitution. Assume $M, \sigma \in SN$ and $M[\sigma] \notin SN$. Then $(\sigma(x) \vec{P}[\sigma]) \notin SN$ for some $(x \vec{P}) \preceq M$ such that $\vec{P}[\sigma] \in SN$.

Proof As in lemma 3. \square

Definition 12. A μ -substitution σ is said to be fair if, for each $\alpha \in \text{dom}(\sigma)$, $\alpha \notin Fv(\sigma)$ where $x \in Fv(\sigma)$ (resp. $\beta \in Fv(\sigma)$) means that $x \in Fv(N)$ (resp. $\beta \in Fv(N)$) for some $N \in \text{Im}(\sigma)$.

Lemma 14. Let σ be a fair μ -substitution, $\alpha \in \text{dom}(\sigma)$ and $x \notin Fv(\sigma)$ (resp. $\beta \notin Fv(\sigma)$), then $M[\sigma][x := \sigma(\alpha)] = M[x := \sigma(\alpha)][\sigma]$ (resp. $M[\sigma][\beta = \sigma(\alpha)] = M[\beta = \sigma(\alpha)][\sigma]$).

Proof Immediate. \square

Lemma 15. Let M, N be terms and σ be a fair μ -substitution. Assume $M[\sigma], N \in SN$ but $(M[\sigma] N) \notin SN$. Assume moreover that $M[\sigma] \triangleright^* \mu\alpha M_1$. Then, for some $(\alpha M_2) \preceq M$, we have $(M_2[\sigma'] N) \notin SN$ and $M_2[\sigma'] \in SN$ where $\sigma' = [\sigma + \alpha = N]$.

Proof By lemma 11, we know that $M \triangleright^* \mu\alpha M'_1$ for some M'_1 such that $M'_1[\sigma] \triangleright^* M_1$. Let M' be a sub-term of a reduct of M such that $\langle \eta(M'[\sigma]), \text{ctxty}(M') \rangle$ is minimum and $M'[\sigma'] \notin SN$. We show that $M' = (\alpha M_2)$ and has the desired properties. By minimality, M' cannot be of the form $\lambda x P$, $\mu\beta P$ nor (βP) for $\beta \neq \alpha$ or $\beta \notin \text{dom}(\sigma)$.

If $M' = (P_1 P_2)$. By the minimality of M' , $P_1[\sigma'], P_2[\sigma'] \in SN$. Thus, by lemma 11 and 12, $P_1 \triangleright^* \lambda x Q$ (resp. $P_1 \triangleright^* \mu\beta Q$) such that $Q[\sigma'][x := P_2[\sigma']] = Q[x :=$

$P_2[\sigma'] \notin SN$ (resp. $Q[\sigma'][\beta = P_2[\sigma']] = Q[\beta = P_2][\sigma'] \notin SN$) and this contradicts the minimality of M' .

If $M' = (\beta P)$ for some $\beta \in \text{dom}(\sigma)$. Then $(P[\sigma'] \sigma(\beta)) \notin SN$ and, by the minimality of M' , $P[\sigma'] \in SN$. Thus, by lemmas 11, 12 and 14, $P \triangleright^* \lambda x Q$ (resp. $P \triangleright^* \mu \gamma Q$) such that $Q[\sigma'][\gamma := \sigma(\beta)] = Q[\gamma := \sigma(\beta)][\sigma'] \notin SN$ (resp. $Q[\sigma'][\gamma = \sigma(\beta)] = Q[\gamma = \sigma(\beta)][\sigma'] \notin SN$) and this contradicts the minimality of M' .

Thus $M' = (\alpha M_2)$ and its minimality implies $M_2[\sigma'] \in SN$. \square

4.2 Proof of the strong normalization

We use the same notations as in section 3.

Lemma 16. *Let $\mathcal{Y} \subseteq \mathcal{X}$ be such that $H[\{X\}]$ holds for each $X \in \mathcal{Y}$. Then $H[\mathcal{T}(\mathcal{Y})]$ holds.*

Proof Assume that $H[\{X\}]$ holds for each $X \in \mathcal{Y}$. The result is a special case of the following claim.

Claim : Let M be a term, U, V be types such that $U \in \mathcal{T}(\mathcal{Y})$ and σ be a λ -substitution such that, for each x , $\sigma(x) = N_x[\tau_x]$ where τ_x is a fair μ -substitution such that $\text{dom}(\tau_x) \cap Fv(M[\sigma]) = \emptyset$. Assume $\Gamma \vdash M : V$ and for each $x \in \text{dom}(\sigma)$, $x : U \in \Gamma$. Assume finally that M and the $N_x[\tau_x]$ are in SN . Then, $M[\sigma] \in SN$.

Proof. By induction on $\langle lg(U), \eta c(M), \eta c(\sigma) \rangle$ where $\eta(\sigma) = \sum \eta(N_x)$ and $cxy(\sigma) = \sum cxy(N_x)$ and, in the sums, each occurrence of a variable counts for one. For example, if there are two occurrences of x_1 and three occurrences of x_2 , $cxy(\sigma) = 2 cxy(N_1) + 3 cxy(N_2)$. Note that we really mean $cxy(N_x)$ and not $cxy(N_x[\tau_x])$ and similarly for η .

The only non trivial case is when $M = (x Q \vec{O})$ for $x \in \text{dom}(\sigma)$. By the IH, $Q[\sigma], \vec{O}[\sigma] \in SN$. It is enough to show that $(N_x[\tau_x] Q[\sigma]) \in SN$ since $M[\sigma]$ can be written as $M'[\sigma']$ where $M' = (z \vec{O}[\sigma])$ and $\sigma'(z) = (N_x[\tau_x] Q[\sigma])$ and (since the size of the type of z is less than the one of U) the IH gives the result. By lemma 12, we have two cases to consider.

- $N_x[\tau_x] \triangleright^* \lambda y N_1$. By lemma 11, $N_x \triangleright^* \lambda y N_2$ and the proof is exactly the same as in lemma 7.
- $N_x[\tau_x] \triangleright^* \mu \alpha N_1$. By lemma 15, let $(\alpha N_2) \preceq N_x$ be such that $N_2[\tau'] \in SN$ and $R = (N_2[\tau'] Q[\sigma]) \notin SN$ where $\tau' = [\tau_x + \alpha = Q[\sigma]]$. But R can be written as $(y Q)[\sigma']$ where σ' is the same as σ except that $\sigma'(y) = N_2[\tau']$. Note that $(y Q)$ is the same as (or less than) M but one occurrence of x has been replaced by the fresh variable y . The substitution τ' is fair and $\text{dom}(\tau') \cap Fv((y Q)) = \emptyset$. The IH gives a contradiction since $\eta c(\sigma') < \eta c(\sigma)$. Note that the type condition on σ' is satisfied since N_x has type U , thus α has type $\neg U$ and thus N_2 also has type U . \square

For now on, we fix some i and we assume $H[\{X_j\}]$ for each $j < i$. Thus, by lemma 16, we know that $H[\mathcal{T}'_i]$ holds. It remains to prove $H[\{X_i\}]$ i.e. proposition 2.

Definition 13. *Let M be a term, $\sigma = \sigma_1 \cup \sigma_2$ where σ_1 (resp. σ_2) is a λ (resp. μ) substitution, Γ be a context and U be a type. Say that (σ, Γ, M, U) is adequate if the following holds:*

- $\Gamma \vdash M[\sigma] : U$ and $M, \sigma \in SN$.
- For each $x \in \text{dom}(\sigma_1)$, $\Gamma \vdash \sigma(x) : V_x$ and $V_x \in \mathcal{T}_i^+$.

Note that nothing is asked on the types of the μ -variables.

Lemma 17. *Let n, m be integers, \vec{S} be a sequence of terms and (δ, Δ, P, B) be adequate. Assume that*

1. $B \in \mathcal{T}_i^- - \mathcal{T}_i'$ and $\Delta \vdash (P[\delta] \vec{S}) : W$ for some W .
2. $\vec{S} \in SN$, $P \in SN$ and $\eta c(P) < \langle n, m \rangle$.
3. $M[\sigma] \in SN$ for every adequate (σ, Γ, M, U) such that $\eta c(M) < \langle n, m \rangle$.

Then $(P[\delta] \vec{S}) \in SN$.

Proof By induction on the length of \vec{S} . The proof is as in lemma 8. The new case is $P[\delta] \triangleright^* \mu \alpha R$ (when $\vec{S} = S_1 \vec{S}_2$). By lemma 11, we have two cases to consider.

- $P \triangleright^* \mu \alpha R'$. We have to show that $Q = (\mu \alpha R'[\delta + \alpha = S_1] \vec{S}_2) \in SN$. By lemma 10, the properties of \approx and since $B \in \mathcal{T}_i^-$, there are types B_1, B_2 such that $B \approx B_1 \rightarrow B_2$ and $\Delta \vdash \mu \alpha R'[\delta + \alpha = S_1] : B_2$ and $B_2 \in \mathcal{T}_i^-$. Since $\eta c(R') < \langle n, m \rangle$ and $([\delta + \alpha = S_1], \Delta \cup \{\alpha : \neg B_2\}, \mu \alpha R', B_2)$ is adequate, it follows from (3) that $R'[\delta + \alpha = S_1] \in SN$.
- Assume first $B_2 \in \mathcal{T}_i'$. Since $(z' \vec{S}_2) \in SN$ and $Q = (z' \vec{S}_2)[z' := \mu \alpha R'[\delta + \alpha = S_1]]$, the result follows from $H[\mathcal{T}_i']$.
- Otherwise, the result follows from the *IH* since $([\delta + \alpha = S_1], \Delta \cup \{\alpha : \neg B_2\}, \mu \alpha R', B_2)$ is adequate and the length of \vec{S}_2 is less than the one of \vec{S} .
- $P \triangleright^* (y \vec{T})$ for some λ -variable $y \in \text{dom}(\delta)$. As in lemma 8. \square

Lemma 18. *Assume (σ, Γ, M, A) is adequate. Then $M[\sigma] \in SN$.*

Proof As in the proof of the lemma 16, we prove a more general result. Assume that, for each $x \in \text{dom}(\sigma_1)$, $\sigma_1(x) = N_x[\tau_x]$ where τ_x is a fair μ -substitution such that $\text{dom}(\tau_x) \cap Fv(M[\sigma]) = \emptyset$. We prove that $M[\sigma] \in SN$.

By induction on $\eta c(M)$ and, by secondary induction, on $\eta c(\sigma_1)$ where $\eta(\sigma_1)$ and $\text{ctxy}(\sigma_1)$ are defined as in lemma 16. The proof is as in lemma 16. The interesting case is $M = (x Q \vec{O})$ for some $x \in \text{dom}(\sigma_1)$. The case when $N_x[\tau_x] \triangleright^* \lambda y N'$ is as in lemma 9. The new case is when $N_x[\tau_x] \triangleright^* \mu \alpha N'$. This is done as in lemma 16. Note that, for this point, the type was not used. \square

Proposition 2. *Assume $\Gamma, x : X_i \vdash M : U$ and $\Gamma \vdash N : X_i$ and $M, N \in SN$. Then $M[x := N] \in SN$.*

Proof This follows from lemma 18 since $([x := N], \Gamma, M, U)$ is adequate. \square

5 Some applications

5.1 Representing more functions

By using recursive types, some terms that cannot be typed in the simply typed λ -calculus become typable. For example, by using the equation $X \approx (X \rightarrow T) \rightarrow T$, it is possible to type terms containing both $(x y)$ and $(y x)$ as sub-terms. Just take $x : X$ and $y : X \rightarrow T$. By using the equation $X \approx T \rightarrow X$, it is possible to apply an unbounded number of arguments to a term.

It is thus natural to try to extend Schwichtenberg's result and to determine the class of functions that are represented in such systems and, in particular, to see whether or not they allow to represent more functions. Note that Doyen [15] and Fortune & all [16] have given extensions of Schwichtenberg's result.

Here is an example of function that cannot be typed (of the good type) in the simply typed λ -calculus.

Let $\text{Nat} = (X \rightarrow X) \rightarrow (X \rightarrow X)$ and $\text{Bool} = Y \rightarrow (Y \rightarrow Y)$ where X, Y are type variables. Let $\tilde{n} = \lambda f \lambda x (f (f \dots x) \dots)$ be the church numeral representing n

and $\mathbf{0} = \lambda x \lambda y y$, $\mathbf{1} = \lambda x \lambda y x$ be the terms representing *false* and *true*. Note that \tilde{n} has type Nat and $\mathbf{0}, \mathbf{1}$ have type Bool .

The term $\text{Inf} = \lambda x \lambda y (x M \lambda z \mathbf{1} (y M \lambda z \mathbf{0}))$ where $M = \lambda x \lambda y (y x)$ has been introduced by B.Maurey. It is easy to see that, for every $n, m \in \mathbb{N}$, the term $(\text{Inf } \tilde{m} \tilde{n})$ reduces to $\mathbf{1}$ if $m \leq n$ and to $\mathbf{0}$ otherwise. Krivine has shown in [24] that the type $\text{Nat} \rightarrow \text{Nat} \rightarrow \text{Bool}$ cannot be given to Inf in system F but, by adding the equation $X \approx (X \rightarrow \text{Bool}) \rightarrow \text{Bool}$, it becomes typable. Our example uses the same ideas.

Let \approx be the congruence generated by $X \approx (X \rightarrow \text{Bool}) \rightarrow \text{Bool}$. For each $n \in \mathbb{N}^*$, let $\text{Inf}_n = \lambda x (x M \lambda y \mathbf{1} (M^{n-1} \lambda y \mathbf{0}))$ where $(M^k P) = (M (M \dots (M P)))$.

Proposition 3. *For each $n \in \mathbb{N}^*$ we have $\vdash \text{Inf}_n : \text{Nat} \rightarrow \text{Bool}$.*

Proof We have $x : X \rightarrow \text{Bool}, y : X \vdash (y x) : \text{Bool}$, then $\vdash M : (X \rightarrow \text{Bool}) \rightarrow (X \rightarrow \text{Bool})$, thus $\vdash (\tilde{n} M) : (X \rightarrow \text{Bool}) \rightarrow (X \rightarrow \text{Bool})$. But $\vdash \lambda y \mathbf{0} : X \rightarrow \text{Bool}$, therefore $\vdash (\tilde{n} M \lambda y \mathbf{0}) : X \rightarrow \text{Bool}$.

We have $x : X, y : X \rightarrow \text{Bool} \vdash (y x) : \text{Bool}$, then $\vdash M : X \rightarrow X$, thus $x : \text{Nat} \vdash (x M) : X \rightarrow X$. But $\vdash \lambda y \mathbf{1} : (X \rightarrow \text{Bool}) \rightarrow \text{Bool}$, therefore $x : \text{Nat} \vdash (x M \lambda y \mathbf{1}) : X$.

We deduce that $x : \text{Nat} \vdash ((\tilde{n} M \lambda y \mathbf{0}) (x M \lambda y \mathbf{1})) : \text{Bool}$, then $x : \text{Nat} \vdash (x M \lambda y \mathbf{1} (M^{n-1} \lambda y \mathbf{0})) : \text{Bool}$ and thus $\vdash \text{Inf}_n : \text{Nat} \rightarrow \text{Bool}$. \square

Proposition 4. *For each $n \in \mathbb{N}^*$ and $m \in \mathbb{N}$, $(\text{Inf}_n \tilde{m})$ reduces to $\mathbf{1}$ if $m \leq n$ and to $\mathbf{0}$ otherwise.*

Proof

$(\text{Inf}_n \tilde{m}) \triangleright^* (M^m \lambda y \mathbf{1} (M^{n-1} \lambda y \mathbf{0})) \triangleright^* (M^{n-1} \lambda y \mathbf{0} (M^{m-1} \lambda y \mathbf{1})) \triangleright^* (M^{m-1} \lambda y \mathbf{1} (M^{n-2} \lambda y \mathbf{0})) \triangleright^* (M^{n-2} \lambda y \mathbf{0} (M^{m-2} \lambda y \mathbf{1})) \triangleright^* \dots$
 $\triangleright^* \mathbf{1}$ if $m \leq n$ and $\mathbf{0}$ otherwise. \square

Remarks

Note that for the (usual) simply typed λ -calculus we could have taken for X and Y the same variable but, for propositions 3 and 4, we cannot assume that $X = Y$ because then the condition of positivity would not be satisfied. This example is thus not completely satisfactory and it actually shows that the precise meaning of the question “which functions can be represented in such systems” is not so clear.

5.2 A translation of the $\lambda\mu$ -calculus into the λ -calculus

The strong normalization of a typed $\lambda\mu$ -calculus can be deduced from the one of the corresponding typed λ -calculus by using CPS translations. See, for example, [14] for such a translation. There is another, somehow simpler, way of doing such a translation. Add, for each atomic type X , a constant a_X of type $\neg\neg X \rightarrow X$. Using these constants, it is not difficult to get, for each type T , a λ -term M_T (depending on T) such that M_T has type $\neg\neg T \rightarrow T$. This gives a translation of the $\lambda\mu$ -calculus into the λ -calculus from which the strong normalization of the $\lambda\mu$ -calculus can be deduced from the one of the λ -calculus. This translation, quite different from the CPS translations, has been used by Krivine [26] to code the $\lambda\mu$ -calculus with second order types in the $\lambda\mathcal{C}$ -calculus.

With recursive equations, we do not have to add the constant a_X since we can use the equation $X \approx \neg\neg X$. We give here, without proof, the translation. We denote by S_{\approx} the simply typed λ -calculus where \approx is the congruence on \mathcal{T} (where $\mathcal{A} = \{\perp\}$) generated by $X \approx \neg\neg X$ for each X and by $S_{\lambda\mu}$ the usual (i.e. without recursive types) $\lambda\mu$ -calculus.

Definition 14. 1. We define, for each type T , a closed λ -term M_T such that $\vdash_{\approx} M_T : \neg\neg T \rightarrow T$ as follows. This is done by induction on T .

- $M_{\perp} = \lambda x (x I)$ where $I = \lambda x x$.
- If $X \in \mathcal{X}$, $M_X = I$.
- $M_{U \rightarrow V} = \lambda x \lambda y (M_V \lambda z (x \lambda t (z (t y))))$
- 2. We define a translation from $S_{\lambda\mu}$ to S_{\approx} as follows.
 - $x^* = x$.
 - $(\lambda x M)^* = \lambda x M^*$.
 - $(M N)^* = (M^* N^*)$.
 - $(\mu \alpha M)^* = (M_U \lambda \alpha M^*)$ if α has the type $\neg U$.
 - $(\alpha M)^* = (\alpha M^*)$.

For a better understanding, in the translation of $\mu \alpha M$ and (αM) , we have kept the same name to the variable α but it should be clear that the translated terms are λ -terms with only on kind of variables.

Lemma 19. *If $\Gamma \vdash_{\lambda\mu} M : U$ then $\Gamma \vdash_{\approx} M^* : U$.*

Lemma 20. *Let M, N be typed $\lambda\mu$ -terms. If $M \triangleright N$, then $M^* \triangleright^+ N^*$.*

Proof It is enough to check that $(\mu \alpha M N)^* \triangleright^+ (\mu \alpha M[\alpha = N])^*$. □

Theorem 3. *The strong normalization of S_{\approx} implies the one of $S_{\lambda\mu}$.*

Proof By lemmas 19 and 20. □

Remark

Note that the previous translation cannot be used to show that the $\lambda\mu$ -calculus with recursive types is strongly normalizing since having two equations (for example $X \approx \neg \neg X$ and $X \approx F$) is problematic.

6 Remarks and open questions

1. The proof of the strong normalization of the system D of intersection types [6] is exactly the same as the one for simple types. Is it possible to extend our proof to such systems with equations ? Note that the sort of constraints that must be given on the equations is not so clear. For example, what does that mean to be positive in $A \wedge B$? To be positive both in A and B ? in one of them ? It will be interesting to check precisely because, for example, it is known that the system¹ given by system D and the equations $X \approx (Y \rightarrow X) \wedge (X \rightarrow X)$ and $Y \approx X \rightarrow Y$ is strongly normalizing (but the proof again is not formalized in Peano arithmetic) though the positivity condition is violated.
2. We could add other typing rules and constructors to ensure that, intuitively, X represents the *least fixed point* of the equation $X \approx F$. This kind of thing is done, for example, in *TTR*. What can be said for such systems?
3. There are many translations from, for example, the $\lambda\mu$ -calculus into the λ -calculus that allows to deduce the strong normalization of the former by the one of the latter. These CPS transformations differ from the one given in section 5.2 by the fact that the translation of a term does not depend on its type. What is the behavior of such translations with recursive equations ?

Acknowledgments

We would like to thank P Urzyczyn who has mentioned to us the question solved here and has also indicated some errors appearing in previous versions of our proofs. Thanks also to the referees and their valuable remarks.

¹ This example appears in a list of open problems of the working group Gentzen, Utrecht 1993.

References

1. H.P. Barendregt, *The Lambda Calculus, Its Syntax and Semantics*. North-Holland, 1985.
2. H.P. Barendregt, *Lambda Calculi with types*. In Abramsky & al. pp. 117-309, 1992.
3. H.P. Barendregt, W. Dekkers and R. Statman *Typed lambda calculus*. To appear.
4. U. Berger and H. Schwichtenberg, *An Inverse of the Evaluation Functional for Typed lambda-calculus*. LICS, pp. 203-211, 1991.
5. A. Church, *A Formulation of the Simple Theory of Types*. JSL 5, 1940.
6. M. Coppo and M. Dezani, *A new type assignment for lambda terms* Archiv. Math. Logik (19) pp. 139-156, (1978).
7. T. Coquand and G. Huet, *A calculus of constructions*. Information and Computation (76), pp. 95-120, 1988.
8. R. David, *Normalization without reducibility*. Annals of Pure and Applied Logic (107), pp. 121-130, 2001.
9. R. David, *A short proof of the strong normalization of the simply typed lambda calculus*. (www.lama.univ-savoie.fr/~david)
10. R. David and K. Nour, *A short proof of the strong normalization of the simply typed $\lambda\mu$ -calculus*. Schedae Informaticae (12), pp. 27-34, 2003.
11. R. David and K. Nour, *Arithmetical proofs of strong normalization results for the symmetric lambda-mu-calculus*. TLCA'2005, LNCS 3461, pp. 162-178, 2005.
12. R. David and K. Nour, *Arithmetical proofs of strong normalization results for symmetric λ -calculi*. To appear in Fundamenta Informaticae.
13. P. de Groote, *On the Relation between the Lambda-Mu-Calculus and the Syntactic Theory of Sequential Control*. LPAR, pp. 31-43, 1994.
14. P. de Groote, *A CPS-Translation of the $\lambda\mu$ -Calculus*. Proceedings 19th Intl. Coll. on Trees in Algebra and Programming, CAAP'94, Edinburgh, LNCS 787, pp. 85-99, 1994.
15. J. Doyen, *Quelques propriétés du typage des fonctions des entiers dans les entiers*. C.R. Acad. Sci. Paris, t.321, Série I, pp. 663-665, 1995.
16. S. Fortune, D. Leivant and M.O'Donnell. *Simple and Second order Types Structures*. JACM 30-1, pp. 151-185, 1983.
17. H. Friedman, *Equality between functionals*. Logic Coll'73, pp. 22-37, LNM 453, 1975.
18. J.-Y. Girard, Y. Lafont and P. Taylor, *Proofs and Types*. Cambridge University Press, 1989.
19. Kurt Gödel, *Über eine bisher noch nicht benützte Erweiterung des finiten Standpunkts*. Dialectica 12, pp. 280-287, 1958.
20. W. D. Goldfarb, *The undecidability of the 2nd order unification problem*. TCS (13), pp. 225- 230, 1981.
21. G.P. Huet, *The Undecidability of Unification in Third Order Logic* Information and Control 22(3) pp. 257-267, 1973.
22. F. Joachimski and R. Matthes, *Short proofs of normalization for the simply-typed lambda-calculus, permutative conversions and Gödel's T*. Archive for Mathematical Logic, 42(1), pp. 59-87, 2003.
23. A. Jung and J. A. Tiuryn, *A New Characterization of Lambda Definability*. TLCA'1993, LNCS 664 pp. 245-257. 1993.
24. J.-L. Krivine, *Un algorithme non typable dans le système F*. C. R. Acad. Sci. Paris 304 (5), 1987.
25. J.-L. Krivine, *Lambda Calcul : types et modèles*, Masson, Paris, 1990.
26. J.-L. Krivine, *Classical Logic, Storage Operators and Second-Order lambda-Calculus*. Ann. Pure Appl. Logic 68(1), pp. 53-78, 1994.
27. J. Lambek, *Cartesian Closed Categories and Typed Lambda-calculi*. Combinators and Functional Programming Languages, pp. 136-175, 1985.
28. R. Loader, *The Undecidability of λ -definability*. In "Essays in memory of A. Church", pp. 331-342, 2001.
29. N. P. Mendler, *Recursive Types and Type Constraints in Second-Order Lambda Calculus*. LICS, pp. 30-36, 1987.
30. N. P. Mendler, *Inductive Types and Type Constraints in the Second-Order Lambda Calculus*. Ann. Pure Appl. Logic 51(1-2), pp. 159-172, 1991.

31. M. Parigot, *Programming with proofs: a second order type theory*. ESOP'88, LNCS 300, (145-159), 1988.
32. M. Parigot, *On representation of data in lambda calculus*. CSL pp. 309-321, 1989.
33. M. Parigot, *Recursive programming with proofs*. Theoretical Computer Science, 94 (335-356), 1992.
34. M. Parigot, *Strong Normalization for Second Order Classical Natural Deduction*. LICS, pp. 39-46, 1993.
35. M. Parigot, $\lambda\mu$ -calculus: *An algorithmic interpretation of classical natural deduction*. Journal of symbolic logic (62-4), pp 1461-1479, 1997.
36. M. Parigot, *Proofs of Strong Normalisation for Second Order Classical Natural Deduction*. J. Symb. Log. 62(4), pp. 1461-1479, 1997.
37. G.D. Plotkin, *Lambda-definability and logical relations*. Technical report, 1973.
38. H. Schwichtenberg, *Functions definable in the simply-typed lambda calculus*. Arch. Math Logik 17, pp. 113-114, 1976.
39. R. Statman, *The Typed lambda-Calculus is not Elementary Recursive*. FOCS, pp. 90-94, 1977.
40. R. Statman, λ -definable functionals and $\beta\eta$ -conversion. Arch. Math. Logik 23, pp. 21-26, 1983.
41. R. Statman, *Recursive types and the subject reduction theorem*. Technical report 94-164, Carnegie Mellon University, March 1994.
42. W.W.Tait, *Intensional Interpretations of Functionals of Finite Type I*. JSL 32(2), 1967.
43. A. Weiermann, *A proof of strongly uniform termination for Gödel's T by methods from local predicativity*. Archive for Mathematical Logic 36, pp. 445-460, 1997.