Parigot's Second Order $\lambda\mu$-Calculus and Inductive Types

Ralph Matthes*

Institut für Informatik der Ludwig-Maximilians-Universität München
Oettingenstraße 67, D-80538 München, Germany
matthes@informatik.uni-muenchen.de

October 10, 2000

Abstract. A new proof of strong normalization of Parigot's (second order) $\lambda\mu$-calculus is given by a reduction-preserving embedding into system F (second order polymorphic λ-calculus). The main idea is to use the least stable supertype for any type. These non-strictly positive inductive types and their associated iteration principle are available in system F, and allow to give a translation vaguely related to CPS translations (corresponding to the Kolmogorov embedding of classical logic into intuitionistic logic). However, they simulate Parigot's μ-reductions whereas CPS translations hide them.
As a major advantage, this embedding does not use the idea of reducing stability ($\neg\neg\phi \to \phi$) to that for atomic formulae. Therefore, it even extends to non-interleaving fixed-point types. As a non-trivial application, strong normalization of $\lambda\mu$-calculus, extended by primitive recursion on monotone inductive types, is established.

1 Introduction

$\lambda\mu$-calculus [12] essentially is the extension of natural deduction by "reductio ad absurdum" (RAA), i.e., by a term formation rule corresponding to stability ($\neg\neg\rho \to \rho$, also called "duplex negatio affirmat") and by rewrite rules for the simplification of the application of elimination rules to RAA (in the case of $\to$-elimination this corresponds to the fact that the stability of $\rho \to \sigma$ is derivable from that of σ).

In [14]—besides a direct proof of strong normalization via saturated sets—we find a reduction of the proof of strong normalization of $\lambda\mu$-calculus to the well-known strong normalization of system F. The proof is quite intricated since the considered CPS translation maps the RAA redexes and their contracta to the same term, and hence needs additional arguments on RAA reductions alone to guarantee strong normalization of the whole calculus.

* I am grateful for an invitation to present a preliminary version of the present results at the "Séminaire Preuves, Programmes et Systèmes" at Paris VII.

Since these μ-reductions have the flavour of iteration (the application occurs in the contractum in a controlled fashion), it has been tempting to explain μ-reductions via inductive types. This is achieved by studying the "stabilization" $\sharp\rho$ of any type ρ. It is the least stable type (i. e., there is a constant of type $\neg\neg\sharp\rho \to \sharp\rho$) such that ρ is included in $\sharp\rho$. From the minimality, we get an iteration principle which in fact simulates μ-reductions and therefore illustrates nicely what can be achieved with non-strictly positive inductive types.

Since there are other embeddings into system F [6], the question arises what else can be proved with this kind of embedding. It turns out that also type concepts can be treated where the elimination does not deconstruct the type to be eliminated. This is exemplified with the addition of non-interleaving positive fixed-point types. It is very unlikely that they can be embedded into system F [15]. Therefore, we cannot expect to get an embedding into system F but into system F with those fixed-point types (but of course without the μ-reductions). μ-reductions for fixed-point types still have a connection between the type eliminated and the result type. This is not true of primitive recursion. The most general formulation of primitive recursion is introduced: primitive recursion on monotone inductive types. However, since an embedding into non-interleaving positive fixed-point types can be established, which moreover interacts nicely with μ-reductions, we can even prove that $\lambda\mu$-calculus with primitive recursion on monotone inductive types (and μ-reductions for every type construct) is strongly normalizing.

The next section recalls system F, in section 3 we review $\lambda\mu$-calculus and discuss variations on the definition. Section 4 presents the stabilization $\sharp\rho$ and the associated iteration principle and explains it via an impredicative encoding, whereas in section 5 this system becomes the target of an embedding of $\lambda\mu$-calculus. Substitution lemmas are proven in great detail in order to give a feeling for the embedding. Fixed-point types are introduced in section 6, and the embedding of the previous section extended to cover fixed-point types (in any of the systems studied before). This extension is astonishingly straightforward. In the final section 7 monotone inductive types with primitive recursion are described. A new embedding of monotone inductive types with primitive recursion into non-interleaving positive fixed-point types gives the final normalization theorem.

2 System F

We consider system F (see e. g. [5]) only with function types and universal types, i. e., we have infinitely many type variables (denoted by α, β, ...) and with types ρ and σ we also have the function type $\rho \to \sigma$. Moreover, given a variable α and a type ρ we form the universal type $\forall\alpha\rho$. The $\forall$ binds α in ρ. The renaming convention for bound variables is adopted. Let $\mathrm{FV}(\rho)$ be the set of type variables occurring free in ρ.

The terms of F are presented as in [5], i. e., without contexts and with fixed types (see [1, p. 159] for comments on this original typing à la Church). We have infinitely many term variables with types (denoted e. g. by x^ρ), lambda abstraction $(\lambda x^\rho r^\sigma)^{\rho\to\sigma}$ for terms binding x^ρ in r^σ, term application $(r^{\rho\to\sigma}s^\rho)^\sigma$, lambda abstraction $(\Lambda\alpha r^\rho)^{\forall\alpha\rho}$ for types (under the usual proviso that $\alpha\notin\mathrm{FV}(\sigma)$ for any x^σ free in r^ρ) and type application $(r^{\forall\alpha\rho}\sigma)^{\rho[\alpha:=\sigma]}$. We freely use the renaming convention for bound term and type variables of terms, and moreover omit type superscripts of the terms. We do this even in case of typed variables which are in fact pairs of variable names and types, but only if the type can be reconstructed from a lambda abstraction binding a variable with the same name. (The interested reader may consult the discussion of these issues in [7, sections 2.1.2 and 2.2.6.].) Instead of r^ρ we often write that r has type ρ or even $r:\rho$. We let application associate to the left and write $\lambda x^\rho.r$ to avoid parenthesizing of r.

Beta reduction $\rhd$ for system F is as usual given by

$$\begin{array}{ll}(\beta_\to) & (\lambda x^\rho r)s \rhd r[x^\rho := s]\\ (\beta_\forall) & (\Lambda\alpha r)\sigma \rhd r[\alpha := \sigma]\ .\end{array}$$

Here, we used the result $r[x^\rho := s]$ of the capture-free substitution of the typed variable x^ρ by the term s of type ρ in r and the result $r[\alpha:=\sigma]$ of the capture-free substitution of the type variable α by the type σ in r. The reduction relation $\to$ is defined as the term closure of $\rhd$. The main theorem on F (due to Girard) is that $\to$ is strongly normalizing, i. e., there are no infinite reduction sequences $r_1\to r_2\to r_3\to\cdots$ (In [7] all the definitions and lemmas are given quite carefully so as to make sure that the main theorem can also be proved in this presentation without contexts.)

3 Second Order $\lambda\mu$-Calculus

We present $\lambda\mu$-calculus [12] not with sequents but in the same style as system F. Although $\lambda\mu$-calculus extends system F by the classical law of *reductio ad absurdum*, there is neither falsity nor negation in the type system. Hence, we keep the type system of F. However, there is a second kind of variables (called μ-variables in [12]), denoted by $a,b,c,\ldots$ They are also paired with types, e. g. a^ρ, whose interpretation is the assumption of the *negation* of ρ.

The term system is essentially that of F, extended by the rule of *reductio ad absurdum*: $\mu a^\sigma.[b^\rho]r^\rho$ is a term of type σ, binding a^σ in $[b^\rho]r^\rho$. The latter is no term but will be called a named term as in [12].

The named term $[b^\rho]r^\rho$ is to be understood as the application of the μ-variable b^ρ—assuming the negation of ρ—to the term r proving ρ. Hence, b^ρ is free in $[b^\rho]r^\rho$. Moreover, a named term would prove falsity (if it were a legal term and falsity were included into the type system), hence $[b^\rho]r^\rho$ gives falsity under the assumption of the negation of σ, expressed by the bound variable a^σ. *Reductio ad absurdum* yields σ, hence justifying the type assignment for $\mu a^\sigma.[b^\rho]r^\rho$.

As a word of caution, a^ρ is *not* a term and not legal for lambda abstraction. (Although we allow renaming of bound variables without explicit mention, we have to keep to the same sort: either normal variable—called λ-variable in [12]—or μ-variable.)

The beta reduction relation $\rhd$ of system F is extended by μ-reductions as follows:

$$
\begin{array}{ll}
(\mu_{\to}) & (\mu a^{\rho\to\sigma}.r)s \rhd \mu b^{\sigma}.r\Big[x^{\rho\to\sigma}.[a^{\rho\to\sigma}]x := [b](xs)\Big] \\
(\mu_{\forall}) & (\mu a^{\forall\alpha\rho}.r)\sigma \rhd \mu b^{\rho[\alpha:=\sigma]}.r\Big[x^{\forall\alpha\rho}.[a^{\forall\alpha\rho}]x := [b](x\sigma)\Big] \ .
\end{array}
$$

Here, r is always a named term, and $r\Big[x^{\rho}.[a^{\rho}]x := [b^{\sigma}]t^{\sigma}\Big]$ denotes the result of replacing inductively (bottom-up) in r each subexpression of the form $[a^{\rho}]u^{\rho}$ by $[b^{\sigma}](t[x^{\rho} := u])$ for any term u of type ρ ("x^{ρ}." binds x^{ρ} in this substitution notation, especially in t). In fact, this is only another notation for the same substitution concept in [12].

Remark 1. For our present purposes it seems that this unusual notion of substitution cannot be avoided. If we had type $\bot$, we could formulate *reductio ad absurdum* without reference to μ-variables: Write $\neg\rho$ for $\rho \to \bot$ and extend F only by terms $\mu x^{\neg\rho}.r^{\bot}$ of type ρ which bind $x^{\neg\rho}$ in r. (We do not need the extra concept of named terms since we simply have $x^{\neg\rho}r^{\rho}$ of type $\bot$ instead of some named term $[a^{\rho}]r^{\rho}$.) Now, we can extend $\rhd$ of system F by

$$
\begin{array}{ll}
(\mu_{\to})' & (\mu x^{\neg(\rho\to\sigma)}.r)s \rhd \mu y^{\neg\sigma}.r[x^{\neg(\rho\to\sigma)} := \lambda z^{\rho\to\sigma}.y(zs)] \\
(\mu_{\forall})' & (\mu x^{\neg\forall\alpha\rho}.r)\sigma \rhd \mu y^{\neg\rho[\alpha:=\sigma]}.r[x^{\neg\forall\alpha\rho} := \lambda z^{\forall\alpha\rho}.y(z\sigma)] \ .
\end{array}
$$

This formulation only needs standard substitution and is obviously slightly more general than $\lambda\mu$ ($\lambda\mu$ imposes a stronger discipline on term formation). By the method of saturated sets, it is possible to show strong normalization of the term closure $\to$ of $\rhd$. Of course, this requires an appropriate formulation of the notion of saturation. Moreover, we would reprove strong normalization of F instead of using this fact via an embedding into system F. Unfortunately, our embedding shown below does not extend to this reformulation which in some sense abuses the function types (via negation) for the explanation of *reductio ad absurdum*. Therefore, we abandon this reformulation.

Another approach to avoid the peculiar substitution is taken in [16, pp.17–20] and [2]: Instead of named terms $[a]r$, we have responses/commands $[C]r$ (resp. $\langle r \mid C\rangle$) where C is a stack of terms with a μ-variable a at the bottom. This more general view of continuations/contexts allows to define the equality relation associated with μ-reduction by help of ordinary substitution. While this is sufficient for the study of equality in [16], we are interested in strong normalization. Hence, also the call-by-value and call-by-name formulations in [2] which are possible with ordinary substitution do not suffice.

Second order $\lambda\mu$-calculus is strongly normalizing [13].

4 Extension of F by Iteration on Stabilization

If $\neg\neg\rho \to \rho$ is provable, then ρ is called stable. We consider an extension of system F by a least stable supertype $\sharp\rho$ for any type ρ. This is expressed as follows: We add a type constant $\bot$ for falsity (and set $\neg\rho := \rho \to \bot$) and for every type ρ we assume the type $\sharp\rho$ ($\sharp$ is a unary type former) called the stabilization of ρ.

The term system reflects that $\sharp\rho$ is stable, that ρ embeds into $\sharp\rho$ and that $\sharp\rho$ is the least type with these properties. We assume contants $S_{\sharp\rho}$ of type $\neg\neg\sharp\rho \to \sharp\rho$ and add two term formation rules: If r is of type ρ then $I_{\sharp\rho}r$ is of type $\sharp\rho$. If r is of type $\sharp\rho$, τ is a type and $s_1 : \rho \to \tau$ and $s_2 : \neg\neg\tau \to \tau$ then $rE_\tau s_1 s_2$ is a term of type τ.

$$\frac{r : \rho}{I_{\sharp\rho}r : \sharp\rho} \qquad \frac{}{S_{\sharp\rho} : \neg\neg\sharp\rho \to \sharp\rho} \qquad \frac{r : \sharp\rho \qquad s_1 : \rho \to \tau \qquad s_2 : \neg\neg\tau \to \tau}{rE_\tau s_1 s_2 : \tau}$$

Extend $\rhd$ of system F by

$$\begin{array}{ll} (\sharp I) & (I_{\sharp\rho}r)E_\tau s_1 s_2 \rhd s_1 r \\ (\sharp S) & (S_{\sharp\rho}r)E_\tau s_1 s_2 \rhd s_2\Big(\lambda y^{\neg\tau}.r(\lambda z^{\sharp\rho}.y(zE_\tau s_1 s_2))\Big) \ . \end{array}$$

In the second rule, we assume that $y^{\neg\tau}$ and $z^{\sharp\rho}$ do not occur free in s_1 or s_2. Let $\to$ be the term closure of $\rhd$. Clearly, it enjoys subject reduction, i. e., if $r^\rho \to s^\sigma$ then $\rho = \sigma$.

Definition 1. *A type-respecting reduction-preserving embedding (embedding for short) of a typed term rewrite system $\mathcal{S}$ into a typed term rewrite system $\mathcal{S}'$ is a function $-'$ (the $-$ sign represents the indefinite argument of the function $'$) which assigns to every type ρ of $\mathcal{S}$ a type ρ' of $\mathcal{S}'$ and to every term r of type ρ of $\mathcal{S}$ a term r' of the (image) type ρ' of $\mathcal{S}'$ such that the following implication holds: If $r \to s$ in $\mathcal{S}$, then $r' \to^+ s'$ in $\mathcal{S}'$. ($\to^+$ denotes the transitive closure of $\to$.)*

Obviously, if there is an embedding of $\mathcal{S}$ into $\mathcal{S}'$, then strong normalization of $\mathcal{S}'$ is inherited by $\mathcal{S}$.

Lemma 1. *There is an embedding of F with iteration on stabilization into F, extended by the type $\bot$.*

Proof. By the very description, $\sharp\rho$ is nothing but the non-strictly positive inductive type $\mu\alpha.\rho + \neg\neg\alpha$ with $\alpha \notin \mathrm{FV}(\rho)$, formulated without the sum type (do not mix up the notation $\mu\alpha\rho$ for inductive types with μar for *reductio ad absurdum*). Its canonical polymorphic encoding would be $\forall\alpha.((\rho + \neg\neg\alpha) \to \alpha) \to \alpha$ which can be simplified to $\forall\alpha.(\rho \to \alpha) \to (\neg\neg\alpha \to \alpha) \to \alpha$.

By iteration on the type ρ of F with stabilization define the type ρ' of F with $\bot$. This shall be done homomorphically in all cases except:

$$(\sharp\rho)' := \forall\alpha.(\rho' \to \alpha) \to (\neg\neg\alpha \to \alpha) \to \alpha$$

with $\alpha \notin \mathrm{FV}(\rho)$. Clearly, $\mathrm{FV}(\rho') = \mathrm{FV}(\rho)$ and $(\rho[\alpha := \sigma])' = \rho'[\alpha := \sigma']$.

By iteration on the term $r : \rho$ of F with stabilization define the term $r' : \rho'$ of F with $\bot$. (Simultaneously, one has to show that the free variables of r' are the $x^{\rho'}$ with x^ρ free in r.) We only consider the non-homomorphic cases.

$$\begin{aligned}
(I_{\sharp\rho} r)' &:= \Lambda\alpha\lambda x_1^{\rho'\to\alpha}\lambda x_2^{\neg\neg\alpha\to\alpha}.x_1 r' \\
(S_{\sharp\rho})' &:= \lambda x^{\neg\neg(\sharp\rho)'}\Lambda\alpha\lambda x_1^{\rho'\to\alpha}\lambda x_2^{\neg\neg\alpha\to\alpha}.x_2\Big(\lambda y^{\neg\alpha}.x(\lambda z^{(\sharp\rho)'}.y(z\alpha x_1 x_2))\Big) \\
(rE_\tau s_1 s_2)' &:= r'\tau' s_1' s_2' \ .
\end{aligned}$$

Since $(r[x^\rho := s])' = r'[x^{\rho'} := s']$ and $(r[\alpha := \rho])' = r'[\alpha := \rho']$, this translation respects $(\beta_\to)$ and $(\beta_\forall)$. It is a trivial calculation to prove that

$$((I_{\sharp\rho} r)E_\tau s_1 s_2)' \to^3 (s_1 r)' \qquad \text{and}$$

$$\Big((S_{\sharp\rho} r)E_\tau s_1 s_2\Big)' \to^4 \Big(s_2\Big(\lambda y^{\neg\tau}.r(\lambda z^{\sharp\rho}.y(zE_\tau s_1 s_2))\Big)\Big)'.$$

(We use $\to^n$ to express n steps of $\to$.) Therefore, $-'$ is indeed an embedding. □

5 Embedding Second Order $\lambda\mu$-Calculus into F with Iteration on Stabilization

In contrast to the proof of strong normalization of second order $\lambda\mu$-calculus in [14] by a CPS translation which maps the terms to the left and the right side of $(\mu_\to)$ and $(\mu_\forall)$ to the same term, respectively, and therefore needs an additional argument for the strong normalization of $(\mu_\to)$ and $(\mu_\forall)$ alone (without $(\beta_\to)$ and $(\beta_\forall)$), our translation given below also simulates those μ-reductions and hence is an *embedding*. Note, however, that [6] presents an embedding which is even easier—the only non-homomorphic rule is $(\forall\alpha\rho)' := \forall\alpha.(\neg\neg\alpha \to \alpha) \to \rho'$—but which heavily uses the fact that stability may be proved for those translated types from stability of their free type variables. This will rule out the extension to fixed-point types to be studied in the next section.

Define the type ρ^* of F with stabilization by iteration on the type ρ of second order $\lambda\mu$-calculus as follows:

$$\begin{aligned}
\alpha^* &:= \alpha \\
(\rho \to \sigma)^* &:= \sharp\rho^* \to \sharp\sigma^* \\
(\forall\alpha\rho)^* &:= \forall\alpha\,\sharp\rho^*
\end{aligned}$$

By induction on ρ one verifies that $(\rho[\alpha := \sigma])^* = \rho^*[\alpha := \sigma^*]$ and $\mathrm{FV}(\rho^*) = \mathrm{FV}(\rho)$.
The type translation of our embedding is given by $\rho' := \sharp\rho^*$. Therefore,

$$\begin{aligned}
\alpha' &= \sharp\alpha \\
(\rho \to \sigma)' &= \sharp(\rho' \to \sigma') \\
(\forall\alpha\rho)' &= \sharp(\forall\alpha\rho')
\end{aligned}$$

and $(\rho[\alpha := \sigma])' = \rho'[\alpha := \sigma^*]$, and also $\mathrm{FV}(\rho') = \mathrm{FV}(\rho)$.

Of course, ρ' could have been defined directly by iteration on ρ without reference to ρ^*. However, the substitution property would not have looked so natural. Note that if we had used $\neg\neg\rho$ instead of $\sharp\rho$ everywhere in this definition, we would have arrived at Kolmogorov's negative translation used in [3, 14] (the corresponding term translation would have been a translation in continuation-passing style).

By iteration on the term $r:\rho$ of second order $\lambda\mu$-calculus define the term $r':\rho'$ of F with stabilization. (Simultaneously, one has to show that the free variables of r' are the $x^{\rho'}$ with x^ρ free normal variable in r and the $a^{\neg\sigma'}$ with a^σ free μ-variable in r. Hence, we assume that the names of the μ-variables are also names of variables of our F with stabilization.)

$$
\begin{aligned}
x^\rho &:= x^{\rho'}\\
(\lambda x^\rho r^\sigma)' &:= I_{(\rho\to\sigma)'}(\lambda x^{\rho'} r')\\
(r^{\rho\to\sigma}s^\rho)' &:= r' E_{\sigma'}(\lambda z^{\rho'\to\sigma'}.zs')S_{\sigma'}\\
(\Lambda\alpha r^\rho)' &:= I_{(\forall\alpha\rho)'}(\Lambda\alpha r')\\
(r^{\forall\alpha\rho}\sigma)' &:= r' E_{(\rho[\alpha:=\sigma])'}(\lambda z^{\forall\alpha\rho'}.z\sigma^*)S_{(\rho[\alpha:=\sigma])'}\\
(\mu a^\sigma.[b^\rho]r^\rho)' &:= S_{\sigma'}(\lambda a^{\neg\sigma'}.b^{\neg\rho'}r')
\end{aligned}
$$

In the third clause, we assume that $z^{\rho\to\sigma}$ is not free in s. Note that the fourth clause is legal since the proviso on the formation of $\Lambda\alpha r'$ is fulfilled by our statement which is proved simultaneously with the definition. The substitution property $(\rho[\alpha:=\sigma])' = \rho'[\alpha:=\sigma^*]$ is heavily used in the fifth clause.

Lemma 2. $(r[x^\rho:=s])' = r'[x^{\rho'}:=s']$ *and* $(r[\alpha:=\rho])' = r'[\alpha:=\rho^*]$.

Proof. Induction on r. □

Corollary 1. $((\lambda x^\rho r)s)' \to^3 (r[x^\rho:=s])'$ *and* $((\Lambda\alpha r)\sigma)' \to^3 (r[\alpha:=\sigma])'$.

Let $\to^*$ denote the reflexive transitive closure of $\to$.

Lemma 3. $r'[a^{\neg\rho'}:=\lambda x^{\rho'}.b^{\neg\sigma'}t'] \to^* \big(r[x^\rho.[a^\rho]x:=[b^\sigma]t]\big)'$ *for t of type σ.*

Proof. By induction on *terms* r. We show the only non-trivial case where r has the form $\mu d^\tau.[a^\rho]s^\rho$, hence with the same μ-variable a^ρ. The left-hand side becomes

$$S_{\tau'}\Big(\lambda d^{\neg\tau'}.(\lambda x^{\rho'}.b^{\neg\sigma'}t')s'[a^{\neg\rho'}:=\lambda x^{\rho'}.b^{\neg\sigma'}t']\Big).$$

One beta reduction step yields

$$S_{\tau'}\Big(\lambda d^{\neg\tau'}.b^{\neg\sigma'}t'\Big[x^{\rho'}:=s'[a^{\neg\rho'}:=\lambda x^{\rho'}.b^{\neg\sigma'}t']\Big]\Big).$$

By induction hypothesis,

$$s'[a^{\neg\rho'}:=\lambda x^{\rho'}.b^{\neg\sigma'}t'] \to^* \big(s[x^\rho.[a^\rho]x:=[b^\sigma]t]\big)'.$$

Hence, $\to^*$ leads to

$$S_{\tau'}\Big(\lambda d^{\neg\tau'}.b^{\neg\sigma'}t'\Big[x^{\rho'}:=\big(s[x^\rho.[a^\rho]x:=[b^\sigma]t]\big)'\Big]\Big).$$

By the previous lemma,

$$t'\Big[x^{\rho'} := \big(s[x^\rho.[a^\rho]x := [b^\sigma]t]\big)'\Big] = \Big(t\Big[x^\rho := s[x^\rho.[a^\rho]x := [b^\sigma]t]\Big]\Big)'.$$

To sum up, the left-hand side is in relation $\to^*$ to

$$S_{\tau'}\Big(\lambda d^{\neg\tau'}.b^{\neg\sigma'}\Big(t\Big[x^\rho := s[x^\rho.[a^\rho]x := [b^\sigma]t]\Big]\Big)'\Big)$$

which is equal to

$$\Big(\mu d^\tau.[b^\sigma]t\Big[x^\rho := s[x^\rho.[a^\rho]x := [b^\sigma]t]\Big]\Big)'$$

which is the right-hand side by the definition of the "special" substitution. □

Theorem 1. $-'$ *is an embedding.*

Proof. We only check that μ-reduction steps give rise to at least one rewrite step of F with stabilization. Since we already convinced ourselves that there are no problems with types, we will neglect the types altogether. As an additional benefit, we may treat both μ-reductions uniformly (and will later profit from this uniformity in the extension by fixed-point types). Write R for a term or a type. Define the term or type $\hat{R}$ as follows: $\hat{r} := r'$ and $\hat{\rho} := \rho^*$. The μ-reduction rules now both become:

$$(\mu a.[c]r)R \rhd \mu b.([c]r)[x.[a]x := [b](xR)].$$

$$((\mu a.[c]r)R)' = (S(\lambda a.cr'))E(\lambda z.z\hat{R})S$$

and one application of $(\sharp S)$ leads to

$$S\Big(\lambda b.(\lambda a.cr')\Big(\lambda x.b(xE(\lambda z.z\hat{R})S)\Big)\Big) = S\Big(\lambda b.(\lambda a.cr')(\lambda x.b(xR)')\Big).$$

If $a = c$ then one beta reduction step yields

$$S\Big(\lambda b.(\lambda x.b(xR)')r'[a := (\lambda x.b(xR)')]\Big) \rhd S\Big(\lambda b.b(r'[a := (\lambda x.b(xR)')]E(\lambda z.z\hat{R})S)\Big).$$

By the previous lemma, $r'[a := (\lambda x.b(xR)')] \to^* (r[x.[a]x := [b](xR)])'$, hence

$$((\mu a.[c]r)R)' \to^* S\Big(\lambda b.b(r[x.[a]x := [b](xR)]R)'\Big) = \Big(\mu b.([a]r)[x.[a]x := [b](xR)]\Big)'.$$

If $a \neq c$ then one beta reduction step yields

$$S\Big(\lambda b.cr'[a := \lambda x.b(xR)']\Big) \to^* S\Big(\lambda b.c(r[x.[a]x := [b](xR)])'\Big)$$

again by the previous lemma. This is $\Big(\mu b.([c]r)[x.[a]x := [b](xR)]\Big)'$. □

Hence, second order $\lambda\mu$-calculus has been proven to be strongly normalizing once more since also F with iteration on stabilization has been embedded into the strongly normalizing system F. (More precisely, it has been embedded into F with $\bot$ which is of course also strongly normalizing since we did not consider any rewrite rule for $\bot$.)

6 Extension to Fixed-Point Types

We extend each system by non-interleaving positive fixed-point types: system F, system F with iteration on stabilization and second order $\lambda\mu$-calculus. Our aim is to extend the embedding of second order $\lambda\mu$-calculus into system F to the variants with those fixed-point types.

System F with non-interleaving positive fixed-point types essentially has been studied in [4] under the name F_{ret}, and its strong normalization shown by an embedding into Mendler's system [10]. A direct proof of strong normalization by saturated sets has been given in [9] under the name NPF. There is strong evidence that no embedding into system F exists [15].

We now extend system F by types $f\alpha\rho$ which are supposed to describe arbitrary fixed-points of $\lambda\alpha\rho$, i. e., of the operation $\sigma \mapsto \rho[\alpha := \sigma]$. We confine ourselves to (non-strictly) positive dependencies which moreover have to be non-interleaved, i. e., $f\alpha\rho$ may only be formed when every occurrence of α in ρ is "to the left of an even number of $\to$" and not free in some subexpression $f\beta\sigma$ of $f\alpha\rho$. The last clause may be rephrased as follows: If fixed-point types $f\beta\sigma$ are formed with a free parameter α then the formation of a fixed-point type $f\alpha\rho$—hence w. r. t. that parameter α—is forbidden.[1] More formally:

Definition 2. *We inductively define the set $\mathcal{T}_{npf}$ of non-interleaving positive fixed-point types and simultaneously for every $\rho \in \mathcal{T}_{npf}$ the sets $N_+(\rho)$ and $N_-(\rho)$ of type variables which occur only positively or occur only negatively, respectively, and moreover do not occur in the scope of a fixed-point type formation (the set $FV(\rho)$ of free type variables is defined as before with the additional $FV(f\alpha\rho) := FV(\rho) \setminus \{\alpha\}$). Let always range p over the set $\{+,-\}$ of polarities and set $-+ := -$ and $-- := +$. Let TV be the set of type variables.*

$\alpha \in \mathcal{T}_{npf}$. $N_+(\alpha) := TV$. $N_-(\alpha) := TV \setminus \{\alpha\}$.
If $\rho, \sigma \in \mathcal{T}_{npf}$ then $\rho \to \sigma \in \mathcal{T}_{npf}$ and $N_p(\rho \to \sigma) := N_{-p}(\rho) \cap N_p(\sigma)$.
If $\rho \in \mathcal{T}_{npf}$ then $\forall\alpha\rho \in \mathcal{T}_{npf}$ and $N_p(\forall\alpha\rho) := N_p(\rho) \cup \{\alpha\}$.
If $\rho \in \mathcal{T}_{npf}$ and $\alpha \in N_+(\rho)$ (the only place where the $N_p(\rho)$ enter the conditions) then $f\alpha\rho \in \mathcal{T}_{npf}$ and $N_p(f\alpha\rho) := TV \setminus FV(f\alpha\rho)$.

Note the change of the polarity in the second rule which substantiates the slogan that α's occurrences may only be to the left of an even number of $\to$. In the last rule we achieve non-interleavedness by removing any free variable of $f\alpha\rho$.

It is somewhat awkward to prove that $\mathcal{T}_{npf}$ is closed under substitution [9, p. 303]. Since it nevertheless holds, we may extend system F to $\mathcal{T}_{npf}$ and moreover add the following term formation rules: If $t : \rho[\alpha := f\alpha\rho]$ then $C_{f\alpha\rho}t : f\alpha\rho$. If $r : f\alpha\rho$ then $rE_f : \rho[\alpha := f\alpha\rho]$.

$$\frac{t : \rho[\alpha := f\alpha\rho]}{C_{f\alpha\rho}t : f\alpha\rho} \qquad \frac{r : f\alpha\rho}{rE_f : \rho[\alpha := f\alpha\rho]}$$

[1] Note that otherwise there would be a very high degree of freedom in the interpretation of $f\alpha\rho$ since $f\beta\sigma$ is intended only to model an arbitrary fixed-point.

Beta reduction for fixed-point types will extend $\rhd$ of F by

$$(\beta_f) \qquad (C_{f\alpha\rho}t)E_f \rhd t \ .$$

It has been mentioned above that the term closure of $\rhd$ is strongly normalizing.

Likewise extend system F with iteration on stabilization by non-interleaving positive fixed-point types: We also write $\mathcal{T}_{npf}$ for its set of types which additionally has the rules:

$\perp \in \mathcal{T}_{npf}$. $N_p(\perp) := \mathrm{TV}$.
If $\rho \in \mathcal{T}_{npf}$ then $\sharp\rho \in \mathcal{T}_{npf}$ and $N_p(\sharp\rho) := N_p(\rho)$.

We may now add the same term formation rules and (β_f) as above. It is clear that the embedding of F with stabilization into F with $\perp$ extends immediately to the extensions by non-interleaving positive fixed-point types: one only has to add homomorphic clauses for the new type former and the new term formation rules. (Note that $(\rho[\alpha := f\alpha\rho])' = \rho'[\alpha := (f\alpha\rho)'] = \rho'[\alpha := f\alpha\rho']$ indicates that the new type former does not pose any problem with the embedding.)

Second order $\lambda\mu$-calculus may as well be extended to the types $\mathcal{T}_{npf}$ (the original definition). We again add the two term formation rules and (β_f) but also a μ-rule pertaining to the fixed-point types. It strictly follows the pattern given in the proof of Theorem 1 if we also consider E_f as a possible value of R:

$$(\mu_f) \qquad (\mu a^{f\alpha\rho}.r)E_f \rhd \mu b^{\rho[\alpha:=f\alpha\rho]}.r\Big[x^{f\alpha\rho}.[a^{f\alpha\rho}]x := [b](xE_f)\Big]$$

with r a named term, hence it is also an instance of (the untyped pattern)

$$(\mu a.[c]r)R \rhd \mu b.([c]r)[x.[a]x := [b](xR)].$$

Let us extend the embedding of section 5. Define the type ρ^* of F with stabilization and non-interleaving fixed-point types by iteration on the type ρ of second order $\lambda\mu$-calculus with non-interleaving fixed-point types as follows (and simultaneously prove that $N_p(\rho^*) = N_p(\rho)$ and $\mathrm{FV}(\rho^*) = \mathrm{FV}(\rho)$):

$$\begin{aligned} \alpha^* &:= \alpha \\ (\rho \to \sigma)^* &:= \sharp\rho^* \to \sharp\sigma^* \\ (\forall\alpha\rho)^* &:= \forall\alpha\,\sharp\rho^* \\ (f\alpha\rho)^* &:= f\alpha\,\sharp\rho^* \end{aligned}$$

The last clause is legal since $\alpha \in N_+(\rho) = N_+(\rho^*) = N_+(\sharp\rho^*)$ by the simultaneously proved statement.

By induction on ρ one again verifies that $(\rho[\alpha := \sigma])^* = \rho^*[\alpha := \sigma^*]$.

Again set $\rho' := \sharp\rho^*$ which implies that $(f\alpha\rho)' = \sharp(f\alpha\rho')$. Also we still have $(\rho[\alpha := \sigma])' = \rho'[\alpha := \sigma^*]$ and $\mathrm{FV}(\rho') = \mathrm{FV}(\rho)$.

The previous translation $-'$ of the terms is extended by clauses for the new term formation rules. If we set $\hat{E}_f := E_f$, then the crucial rule follows the usual (untyped) pattern:

$$(rR)' := r'E(\lambda z.z\hat{R})S$$

The new rules are

$$(C_{f\alpha\rho}t^{\rho[\alpha:=f\alpha\rho]})' := I_{(f\alpha\rho)'}(C_{f\alpha\rho'}t')$$
$$(r^{f\alpha\rho}E_f)' := r'E_{(\rho[\alpha:=f\alpha\rho])'}(\lambda z^{f\alpha\rho'}.zE_f)S_{(\rho[\alpha:=f\alpha\rho])'}$$

Notice that the first term is well-typed since $t' : (\rho[\alpha := f\alpha\rho])'$ and

$$(\rho[\alpha := f\alpha\rho])' = \rho'[\alpha := (f\alpha\rho)^*] = \rho'[\alpha := f\alpha\rho'].$$

The crucial equation $(\rho[\alpha := f\alpha\rho])' = \rho'[\alpha := f\alpha\rho']$ also justifies the second definition.

Theorem 2. *$-'$ is an embedding of second order $\lambda\mu$-calculus with non-interleaving positive fixed-point types into system F with iteration on stabilization, $\perp$ and non-interleaving positive fixed-point types.*

Proof. Lemma 2 and Lemma 3 clearly still hold, $((C_{f\alpha\rho}t)E_f)' \to^2 (C_{f\alpha\rho'}t')E_f \to t'$, and the treatment of (μ_f) is already captured by the uniform proof of Theorem 1. $\square$

Corollary 2. *Second order $\lambda\mu$-calculus with non-interleaving positive fixed-point types is strongly normalizing.*

Remark 2. There seems to be a widespread belief that in some sense μ-reductions are nothing but an exploitation of the fact that stability of a type may be proved from the stability of its atoms. This immediately works for first-order $\lambda\mu$-calculus (only function types) since the stability of $\rho \to \sigma$ is derivable from stability of σ. Since also the stability of $\forall\alpha\rho$ is derivable from that of ρ, one may expect that the universal quantifier also is well-behaved. But notice that the set of atoms (type variables) varies with quantification. Nevertheless, it is possible to give an embedding on grounds of this view [6]. As remarked in the introduction to section 5, the crucial clause is $(\forall\alpha\rho)' := \forall\alpha.(\neg\neg\alpha \to \alpha) \to \rho'$, hence a relativization which neatly solves the problem.

What is the problem with fixed-points? It is again possible to derive the stability of $f\alpha\rho$ from that of $\rho[\alpha := f\alpha\rho]$:

$$\lambda u^{\neg\neg\rho[\alpha:=f\alpha\rho]\to\rho[\alpha:=f\alpha\rho]}\lambda x^{\neg\neg f\alpha\rho}.C_{f\alpha\rho}(u(\lambda y^{\neg\rho[\alpha:=f\alpha\rho]}.x(\lambda z^{f\alpha\rho}.y(zE_f)))).$$

But the latter type is usually more complex than the former! Therefore, I do not see a way to extend the idea of reducing the proof of stability to that of the free type variables. How could one define stability proofs for ρ' by iteration on ρ with some easy translation $-'$? Our embedding shown above does not at all care about such a definition since every ρ' is stable by construction.

Remark 3. One could ask for other type constructions where stability is not inherited from the constituent types. It is well-known that sum types (disjunction) provide an example of this phenomenon. The proof of strong normalization by saturated sets alluded to in Remark 1 can be extended to cover also sum types with permutative conversions. Unfortunately, the embedding into F with stability does not extend to sum types with permutative conversions but only without them. A solution could be to introduce permutative conversions for the stabilization types. At present, we could as well take the impredicative encoding of sums and also get the respective μ-reduction for free (inside second order $\lambda\mu$-calculus). This idea will be demonstrated in the next section for the more interesting case of monotone inductive types with primitive recursion.

7 Second Order $\lambda\mu$-calculus with Primitive Recursion on Monotone Inductive Types is Strongly Normalizing

Inductive types are a syntactic representation of least pre-fixed-points of operations $\sigma \mapsto \rho[\alpha := \sigma]$. Typically, they are studied as long as α only occurs positively in ρ. (Often, even interleaving is ruled out.) Nevertheless, it turned out [7] that the only ingredient for a useful notion of inductive type is a monotonicity witness, i. e., a term of type $\forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$. If those terms are not given beforehand but incorporated into the term system they even do not need to be closed in order to guarantee strong normalization of the rewrite rules associated with them. There is a choice whether the monotonicity witnesses are attached to the introduction rule or to the elimination rule which is primitive recursion. In both cases one has strong normalization [7], in the second case this even has been shown by an embedding into system F with non-interleaving positive fixed-point types [8]. However, for practical purposes the first variant seems more adequate. Fortunately, it also embeds into system F with non-interleaving positive fixed-point types which will be the key to this section's result.

Recall that the product type $\rho \times \sigma$ may be impredicatively encoded in system F by $\rho \times \sigma := \forall\alpha.(\rho \to \sigma \to \alpha) \to \alpha$ for $\alpha \notin \mathrm{FV}(\rho) \cup \mathrm{FV}(\sigma)$. If $r : \rho$ and $s : \sigma$ then $\langle r, s\rangle := \Lambda\alpha\lambda z^{\rho\to\sigma\to\alpha}.zrs : \rho \times \sigma$.

Second order $\lambda\mu$-calculus with primitive recursion on monotone inductive types is defined as an extension of second order $\lambda\mu$-calculus by arbitrary types $\mu\alpha\rho$ (μ binds α in ρ) and by the following term formation rules:
If $m : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta]$ and $t : \rho[\alpha := \mu\alpha\rho]$ then $C_{\mu\alpha\rho}mt : \mu\alpha\rho$.
If $r : \mu\alpha\rho$ and $s : \rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma$ then $rE_\mu\sigma s : \sigma$.

$$\frac{m : \forall\alpha\forall\beta.(\alpha \to \beta) \to \rho \to \rho[\alpha := \beta] \qquad t : \rho[\alpha := \mu\alpha\rho]}{C_{\mu\alpha\rho}mt : \mu\alpha\rho}$$

$$\frac{r : \mu\alpha\rho \qquad s : \rho[\alpha := \mu\alpha\rho \times \sigma] \to \sigma}{rE_\mu\sigma s : \sigma}$$

The associated beta reduction rule of primitive recursion is

$$(\beta_\mu) \qquad (C_{\mu\alpha\rho}t)E_\mu\sigma s \rhd s\Big(m(\mu\alpha\rho)(\mu\alpha\rho \times \sigma)\Big(\lambda x^{\mu\alpha\rho}.\langle x, (\lambda x^{\mu\alpha\rho}.xE_\mu\sigma s)x\rangle\Big)t\Big) \ .$$

The μ-reduction rule (μ in the sense of Parigot) follows the standard pattern if we now even allow R to be $E_\mu\sigma s$:

$$(\mu_\mu) \qquad (\mu a^{\mu\alpha\rho}.r)E_\mu\sigma s \rhd \mu b^\sigma.r\Big[x^{\mu\alpha\rho}.[a^{\mu\alpha\rho}]x := [b](xE_\mu\sigma s)\Big]$$

Again, r denotes a named term in this rule. Note that σ is not at all related to $\mu\alpha\rho$.

Theorem 3. *There is an embedding of second order $\lambda\mu$-calculus with primitive recursion on monotone inductive types into second order $\lambda\mu$-calculus with non-interleaving positive fixed-point types.*

Proof. $\rho' \in \mathcal{T}_{npf}$ is defined by iteration on ρ. The only non-homomorphic clause is that for $\mu\alpha\rho$:

$$(\mu\alpha\rho)' := f\beta\forall\gamma.\Big(\Big(\forall\alpha.(\beta\times\gamma\to\alpha)\to\rho'\Big)\to\gamma\Big)\to\gamma\ .$$

We assume that $\beta,\gamma\notin\{\alpha\}\cup\mathrm{FV}(\rho)$. In fact, the only occurrence of β is 6 times to the left of $\to$ (do not forget that the coding of $\beta\times\gamma$ provides 2 of them). It is easy to see that $(\rho[\alpha:=\sigma])' = \rho'[\alpha:=\sigma']$ and $\mathrm{FV}(\rho') = \mathrm{FV}(\rho)$.

The translation of the terms is also defined homomorphically, with the exception of the two clauses for monotone inductive types:

$$(C_{\mu\alpha\rho}mt)' := C_{(\mu\alpha\rho)'}\Big(\varLambda\gamma\lambda z^{(\forall\alpha.((\mu\alpha\rho)'\times\gamma\to\alpha)\to\rho')\to\gamma}.z\Big(\varLambda\alpha\lambda u^{(\mu\alpha\rho)'\times\gamma\to\alpha}.$$
$$m'(\mu\alpha\rho)'\alpha\Big(\lambda x^{(\mu\alpha\rho)'}.u\langle x,(\lambda x^{(\mu\alpha\rho)'}.xE_f\gamma z)x\rangle\Big)t\Big)\Big)$$

and

$$(rE_\mu\sigma s)' := r'E_f\sigma'\Big(\lambda z^{\forall\alpha.((\mu\alpha\rho)'\times\sigma'\to\alpha)\to\rho'}.s'\Big(z((\mu\alpha\rho)'\times\sigma')(\lambda x^{(\mu\alpha\rho)'\times\sigma'}x)\Big)\Big)$$

Since $(r[x^\rho := s])' = r'[x^{\rho'} := s']$ and $(r[\alpha := \rho])' = r'[\alpha := \rho']$, also this translation respects $(\beta_\to)$ and $(\beta_\forall)$. It is an interesting exercise to prove that also (β_μ) is simulated.

Since the other term formation rules are translated homomorphically, we may set $([b^\rho]r)' := [b^{\rho'}]r'$ and hence have a translation also for named terms. Consequently, $(\mu a^\sigma.r)' = \mu a^{\sigma'}.r'$ with r a named term. For the μ-reduction rules we need $(r[x.[a]x := [b]t])' = r'[x.[a]x := [b]t']$ for named terms r, but this is proved for named terms and for ordinary terms r simultaneously by induction on the size of r.

By induction on natural numbers n, one easily gets (only with μ-reductions) that

$$(\mu a.r)R_1\dots R_n \to^n \mu b.r[x.[a]x := [b](xR_1\dots R_n)]$$

(where $R_1,\dots,R_n$ are terms or types or objects of the form $E_\mu\sigma s$). Of course, this is compatible with the typing requirements.

Therefore, also (μ_μ) is simulated:
$((\mu a.r)E_\mu\sigma s))' = (\mu a.r')E_f\sigma'(\lambda z^{\cdots}.s'\dots) \to^3$
$\mu b.r'[x.[a]x := [b](xE_f\sigma'(\lambda z^{\cdots}.s'\dots))] = \mu b.r'[x.[a]x := [b](xE_\mu\sigma s)'] =$
$\mu b.(r[x.[a]x := [b](xE_\mu\sigma s)])' = (\mu b.r[x.[a]x := [b](xE_\mu\sigma s)])'$.
The other μ-reductions are treated slightly easier. $\square$

Remark 4. There is no hope for an embedding in the style of section 5. Firstly, the function spaces are overly used for the purpose of defining the typing for $rE_\mu\sigma s$. Secondly, if we replaced it by

$$\frac{r : \mu\alpha\rho \qquad s_0 : \sigma}{rE_\mu\sigma(x^{\rho[\alpha:=\mu\alpha\rho\times\sigma]}.s_0) : \sigma}$$

where $x^{\rho[\alpha:=\mu\alpha\rho\times\sigma]}$ is bound in s_0, we cannot use $(\rho[\alpha := \mu\alpha\rho \times \sigma])'$ since $\times$ is encoded. Assume it were explicitly included into the system. Then we would get $(\rho[\alpha := \mu\alpha\rho \times \sigma])' = \rho'[\alpha := (\mu\alpha\rho \times \sigma)^*] = \rho'[\alpha := (\mu\alpha\rho)' \times \sigma']$. But we would need $\mu\alpha\rho'$ instead of $(\mu\alpha\rho)'$. And this would have to be lifted with a monotonicity witness for ρ' w.r.t. α. But we do not even have a monotonicity witness for $\mu\alpha\rho$ at hand since those only come with the introduction rule for $\mu\alpha\rho$.

Corollary 3. *Second order $\lambda\mu$-calculus with primitive recursion on monotone inductive types is strongly normalizing.*

References

1. Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North–Holland, Amsterdam, second revised edition, 1984.
2. Pierre-Louis Curien and Hugo Herbelin. The duality of computation. In *Proceedings of the fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montréal*, pages 233–243. ACM Press, 2000.
3. Philippe de Groote. A CPS-translation of the $\lambda\mu$-calculus. In Sophie Tison, editor, *Trees in Algebra and Programming - CAAP'94, 19th International Colloquium*, volume 787 of *Lecture Notes in Computer Science*, pages 85–99, Edinburgh, 1994. Springer Verlag.
4. Herman Geuvers. Inductive and coinductive types with iteration and recursion. In Bengt Nordström, Kent Pettersson, and Gordon Plotkin, editors, *Proceedings of the 1992 Workshop on Types for Proofs and Programs, Båstad, Sweden, June 1992*, pages 193–217, 1992. Only published via `ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z`.
5. Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.
6. Thierry Joly. Un plongement de la logique classique du 2nd ordre dans AF_2. Unpublished manuscript. In French, 5 pp. Autor contact: `joly@logique.jussieu.fr`, January 1996.
7. Ralph Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types.* Doktorarbeit (PhD thesis), University of Munich, 1998. Available via the homepage `http://www.tcs.informatik.uni-muenchen.de/~matthes/`.
8. Ralph Matthes. Monotone (co)inductive types and positive fixed-point types. *Theoretical Informatics and Applications*, 33(4/5):309–328, 1999.
9. Ralph Matthes. Monotone fixed-point types and strong normalization. In Georg Gottlob, Etienne Grandjean, and Katrin Seyr, editors, *Computer Science Logic,*

12th International Workshop, Brno, Czech Republic, August 24–28, 1998, Proceedings, volume 1584 of *Lecture Notes in Computer Science*, pages 298–312. Springer Verlag, 1999.
10. Nax P. Mendler. Recursive types and type constraints in second-order lambda calculus. In *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science, Ithaca, N.Y.*, pages 30–36. IEEE Computer Society Press, 1987. Forms a part of [11].
11. Paul F. Mendler. Inductive definition in type theory. Technical Report 87-870, Cornell University, Ithaca, N.Y., September 1987. Ph.D. Thesis (Paul F. Mendler = Nax P. Mendler).
12. Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning, International Conference LPAR'92, St. Petersburg, Russia*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer Verlag, 1992.
13. Michel Parigot. Strong normalization for second order classical natural deduction. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 39–46, Montreal, Canada, 1993. IEEE Computer Society Press.
14. Michel Parigot. Proofs of strong normalisation for second order classical natural deduction. *The Journal of Symbolic Logic*, 62(4):1461–1479, 1997.
15. Zdzisław Spławski and Paweł Urzyczyn. Type Fixpoints: Iteration vs. Recursion. *SIGPLAN Notices*, 34(9):102–113, 1999. Proceedings of the 1999 International Conference on Functional Programming (ICFP), Paris, France.
16. Thomas Streicher and Bernhard Reus. Classical logic, continuation semantics and abstract machines. *Journal of Functional Programming*, 8(6):543–572, 1998.