Explicit Substitutions for the Lambda-Mu-Calculus.

Philippe Audebaud

Laboratoire de l'Informatique du Parallélisme

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Explicit Substitutions for the Lambda-Mu Calculus

Philippe Audebaud

October 1994

Research Report N$^{\text{o}}$ 94-26

Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Explicit Substitutions for the Lambda-Mu Calculus

Philippe Audebaud

October 1994

Abstract

We present a confluent rewrite system which extents a previous calculus of explicit substitutions for the lambda-calculus [HaLe89] to Parigot's untyped lambda-mu-calculus [Par91]. This extension embeds the lambda-mu-calculus as a sub-theory, and provides the basis for a theoretical framework to study the abstract properties of implementations of functional programming languages enriched with control structures. This study gets also some interesting feedback on lambda-mu-calculus on both the syntactical and semantics levels.

Keywords: Rewrite systems, Lambda-Mu Calculus

Résumé

Nous proposons un système de récriture confluent qui étend un précédent système avec substitutions explicites pour le lambda-calcul [HaLe89] au lambda-mu-calcul non typé de Parigot [Par91]. Cette extension contient le lambda-mu-calcul comme sous-théorie, et fournit une cadre théorique de base pour l'étude des propriétés abstraites des implantations des langages fonctionnels étendus avec des structures de contrôle. Cette étude fournit également un éclairage nouveau sur le lambda-mu-calcul à la fois au plan syntaxique et sémantique.

Mots-clés: Systèmes de Récriture, Lambda-Mu Calcul

Explicit Substitutions for the Lambda-Mu Calculus *

Philippe Audebaud
Ecole Normale Supérieure de Lyon
LIP-IMAG, URA CNRS 1398
46 Allée d'Italie, 69364 Lyon cedex 07, France
e-mail : paudebau@lip.ens-lyon.fr

September 7, 1994

Abstract

We present a confluent rewrite system which extents a previous calculus of explicit substitutions for the λ-calculus [HaLe89] to Parigot's untyped $\lambda\mu$-calculus [Par91]. This extension embeds the $\lambda\mu$-calculus as a sub-theory, and provides the basis for a theoretical framework to study the abstract properties of implementations of functional programming languages enriched with control structures. This study gets also some interesting feedback on $\lambda\mu$-calculus on both the syntactical and semantics levels.

1 Introduction

The correspondence between proofs and programs plays a major role in Computer Science, because it provides solid mathematical models for the study of functional programming languages. The starting point is the Curry-Howard correspondence for the intuitionistic proofs; their contructive nature allows an easy computational interpretation. The resulting programs can be coded into the (pure) lambda-calculus. Therefore, the lambda-calculus appears as the paradigm for the (pure) functional programming languages. A functional program consists in a set of applications of functions to arguments. The replacement of the formal parameters of functions is represented by the beta-reduction within the lambda-calculus. However, the substitution mechanism is not simple, as it is necessary to take into account the scope of functions, and to take care of the possible clashes between names. These practical questions are hidden behind the *implicit* operation of substitution of variables by lambda-terms in the Church's lambda-calculus. Thus, in order to allow modelisation of implementations of the β-reductions, it is necessary to explain the substitution process, that is to say, to make it *explicit*. Of course, other approaches are possible: the Combinatory Logic for example, which eliminates bound variables, hence the problem. We choose to work in the λ-calculus itself, to take advantage of the intuitive clarity of the λ-notation, and of the power that this syntax conveys at the mathematical level as well as the practical one.

In [HaLe89], a calculus of explicit substitutions, named $\boldsymbol{\lambda\mu}$**env**, is proposed. It is a confluent rewrite system, strongly normalizing on the sub-system dealing with the substitution process. This system is an improvement of [ACCL90] and is also based on the crucial distinction between terms and substitutions. The lambda-theory (with lambda-terms coded using de Bruijn's notation) is embedded as a sub-theory into the extended calculus. All these results make the $\boldsymbol{\lambda\mu}$**env**-calculus a nice theoretical framework for the study of the abstract properties of implementations of functional languages: correctness, optimization for example.

At this point, one has at one's disposal a mathematical model taking into account the dynamic aspects of the previous correspondence between intuitionistic proofs and programs. But, for Π^0_2 statements, classical and intuitionistic provability coincide. Therefore, classical proofs are also candidates for being programs. What is the right counterpart from the point of view of the programming languages? Classical logic appears to be an adequate framework for modeling the imperative features of these programming

*This research was partly supported by ESPRIT Basic Research Action "Types for Proofs and Programs" and by the Programme de Recherche Coordonnées and CNRS GDR "Programmation".

languages. The link between classical logic and functional languages has been established few years ago by Griffin in [Gri90], where Felleisen's generic control operator is given the type $\neg\neg A \rightarrow A$. This correspondence is however more difficult to establish, in this wider setting. This is explained in [Par93], where Parigot advocates the interest of his lambda-mu-calculus in this area, and also the difficulties encountered. This calculus is an extension of the λ-calculus, and shares the same properties of confluence and strong normalization - when this point makes sense. It provides the computational interpretation for classical proofs developed in a natural deduction system with multiple conclusions [Par91, Par93]. Actually, "Mu" comes from the introduction of a new kind of variables, introduced precisely for dealing with the labeling of the different formulae on the right side of a judgment. We do not go into full details, but insist on the fact that this system is strongly justified from the logical point of view. From our point of view, it can be considered as an serious candidate for studying and perhaps establishing completely the correspondence we are looking for between classical proofs and programs using control structures.

On this basis, we propose a rewrite system $\boldsymbol{\lambda\mu}$**env**, which extends the $\boldsymbol{\lambda}$**env** system [HaLe89] for the purpose of providing a system of explicit substitutions for Parigot's $\lambda\mu$-calculus. We get the same properties: confluence and termination for the sub-system dealing with the (more complex) process of substitution. The $\lambda\mu$-calculus is proved being embedded as a sub-theory into the rewrite system. So, we are in position to develop theoretical and practical issues suggested above, but now for the study of functional programming languages extended with control structures; this is undertaken in [Au94].

Section 2 gives a brief overview of both the λ-calculus in the de Bruijn setting and $\boldsymbol{\lambda}$**env** rewrite system. This will help introducing and understanding most of the notations. Section 3 presents Parigot's $\lambda\mu$-calculus. We give only a short description of the calculus, mainly from the computational point of view. A first translation using de Bruijn's notation is presented. The presentation of the $\boldsymbol{\lambda\mu}$**env**-calculus is given in section 4. Informal explanations are provided, and the properties of the calculus are established. Section 5 states some simulation results which entails the embedding of Parigot's $\lambda\mu$-calculus as a sub-theory of $\boldsymbol{\lambda\mu}$**env**.

2 A quick overview of the $\boldsymbol{\lambda}$env-calculus

We introduce the λ-calculus with de Bruijn's notation. Then a short presentation of Hardin-Lévy $\boldsymbol{\lambda}$**env**-calculus is given, with its main properties.

2.1 Church's λ-calculus in de Bruijn's notation

de Bruijn's idea is to replace each variable occurrence by an integer measuring its binding height to the corresponding λ. For example, $\boldsymbol{\lambda} x.(\boldsymbol{\lambda} y.x\ x)$ is represented as $\boldsymbol{\lambda}(\boldsymbol{\lambda}\mathtt{2}\ \mathtt{1})$, since x has two different occurrences, with different binding heights. This way, each free variable occurring in a term M can be interpreted as a depth in a finite fixed stack Δ.

This notation provides a mechanical treatment of α-conversion: informally, such terms can be considered as canonical representative of the class of all terms identified modulo renaming of the bound variables.

Formally, the set $\boldsymbol{\Lambda}$ of de Bruijn λ-terms is defined by the grammar $M ::= \mathtt{n} \mid (M\ M) \mid \boldsymbol{\lambda} M$ where $\mathtt{n} \in \mathbb{N}$. In this new setting, the β-reduction is described precisely as:

$$\boldsymbol{\beta} \quad (\boldsymbol{\lambda} M\ N) \quad \rightarrow \quad M\{\mathtt{1} \leftarrow N\}$$

The substitution process $M\{\mathtt{n} \leftarrow P\}$, introduced by the reduction step, is defined inductively by:

$$\begin{array}{rcl} \mathtt{p}\{\mathtt{n} \leftarrow P\} & = & \left\{ \begin{array}{ll} \mathtt{p-1} & \text{if } \mathtt{p} > \mathtt{n} \\ t_0^{\mathtt{n}}(P) & \text{if } \mathtt{p} = \mathtt{n} \\ \mathtt{p} & \text{if } \mathtt{p} < \mathtt{n} \end{array} \right. \\ (M\ N)\{\mathtt{n} \leftarrow P\} & = & (M\{\mathtt{n} \leftarrow P\}\ N\{\mathtt{n} \leftarrow P\}) \\ (\boldsymbol{\lambda} M)\{\mathtt{n} \leftarrow P\} & = & \boldsymbol{\lambda} M\{\mathtt{n+1} \leftarrow P\} \end{array}$$

where the purpose of the operation $t_{\mathtt{i}}^{\mathtt{m}}(\cdot)$ is to lift the variable occurrences in order to make them adequate: $t_{\mathtt{i}}^{\mathtt{m}}(P)$ means that $\mathtt{i}$ binders have been crossed currently, and that the context in which M is now embedded makes its free variables referenced `m-1` units deeper in the stack. This operation is defined by:

$$\begin{aligned} t^{\mathtt{m}}_{\mathtt{i}}(\mathtt{p}) &= \begin{cases} \mathtt{p+m-1} & \text{if } \mathtt{p} > \mathtt{i+1} \\ \mathtt{p} & \text{otherwise} \end{cases} \\ t^{\mathtt{m}}_{\mathtt{i}}(M\ N) &= (t^{\mathtt{m}}_{\mathtt{i}}(M)\ t^{\mathtt{m}}_{\mathtt{i}}(N)) \\ t^{\mathtt{m}}_{\mathtt{i}}(\boldsymbol{\lambda} M) &= \boldsymbol{\lambda} t^{\mathtt{m}}_{\mathtt{i+1}}(M) \end{aligned}$$

This is well known, and we do not go into further details. Let us emphasize that this special definition for the β-reduction and substitution should be proved equivalent to the usual ones. See [ACCL90, CHL92] for details.

Beta	$(\boldsymbol{\lambda} M\ N)$	$\rightarrow$	$M\langle N \cdot Id\rangle$
LamTerm	$(\boldsymbol{\lambda} M)\langle s\rangle$	$\rightarrow$	$\boldsymbol{\lambda} M\langle \Uparrow(s)\rangle$
AppTerm	$(M\ N)\langle s\rangle$	$\rightarrow$	$(M\langle s\rangle\ N\langle s\rangle)$
Closure	$M\langle s\rangle\langle t\rangle$	$\rightarrow$	$M\langle s \circ t\rangle$
IdEnv	$M\langle Id\rangle$	$\rightarrow$	M
RefShift1	$\mathtt{n}\langle\uparrow\rangle$	$\rightarrow$	$\mathtt{n+1}$
RefShift2	$\mathtt{n}\langle\uparrow \circ s\rangle$	$\rightarrow$	$\mathtt{n+1}\langle s\rangle$
FRefLift1	$\mathtt{1}\langle\Uparrow(s)\rangle$	$\rightarrow$	$\mathtt{1}$
FRefLift2	$\mathtt{1}\langle\Uparrow(s) \circ t\rangle$	$\rightarrow$	$\mathtt{1}\langle t\rangle$
RRefLift1	$\mathtt{n+1}\langle\Uparrow(s)\rangle$	$\rightarrow$	$\mathtt{n}\langle s \circ \uparrow\rangle$
RRefLift2	$\mathtt{n+1}\langle\Uparrow(s) \circ t\rangle$	$\rightarrow$	$\mathtt{n}\langle s \circ (\uparrow \circ t)\rangle$
FRefMap	$\mathtt{1}\langle M \cdot s\rangle$	$\rightarrow$	M
RRefMap	$\mathtt{n+1}\langle M \cdot s\rangle$	$\rightarrow$	$\mathtt{n}\langle s\rangle$
LiftId	$\Uparrow(Id)$	$\rightarrow$	Id
MapEnv	$M \cdot s \circ t$	$\rightarrow$	$M\langle t\rangle \cdot (s \circ t)$
LiftLift1	$\Uparrow(s) \circ \Uparrow(t)$	$\rightarrow$	$\Uparrow(s \circ t)$
LiftLift2	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\rightarrow$	$\Uparrow(s \circ t) \circ u$
LiftMap	$\Uparrow(s) \circ N \cdot t$	$\rightarrow$	$M\langle N \cdot t\rangle \cdot (s \circ t)$
ShiftMap	$\uparrow \circ M \cdot s$	$\rightarrow$	s
ShiftLift1	$\uparrow \circ \Uparrow(s)$	$\rightarrow$	$s \circ \uparrow$
ShiftLift2	$\uparrow \circ (\Uparrow(s) \circ t)$	$\rightarrow$	$s \circ (\uparrow \circ t)$
IdL	$Id \circ s$	$\rightarrow$	s
IdR	$s \circ Id$	$\rightarrow$	s
AssEnv	$(s \circ t) \circ u$	$\rightarrow$	$s \circ (t \circ u)$

Table 1: The **λenv** rewrite system

2.2 Hardin-Lévy λenv-calculus

The rewrite system **λenv** distinguishes **terms** and **substitutions**. Terms are built using de Bruijn's notation. The action of the substitution s on the term M is written $M\langle s\rangle$. The basic idea for understanding the interaction between terms and substitutions is to think about the usual contraction of a β-redex. Such a contraction *creates* (or emits) a specific substitution s: let us write $\mathtt{Redex} \rightarrow M\langle s\rangle$. Then depending on the structure of M, this action is *propagated* towards the variable positions where it is *absorbed* (or received). So the action of s depends on the stack Δ which includes the free variables of M, and the result is defined on the basis of a particular stack Γ, which is only dependent on s. Therefore, s can be seen as a morphism; in that case $s : \Gamma \rightarrow \Delta$. Composition is a partial operation on substitutions, written $s \circ t$, with Id as neutral element for this operation.

Let us introduce some special substitutions. The effect of the **shift** substitution $\uparrow$ is to increase by one an integer variable, meaning that the free variables of $M\langle\uparrow\rangle$ are simply located one unit deeper in the stack. As a morphism, it can be considered as $\uparrow : \Delta, x \rightarrow \Delta$, for any stack Δ and new variable x.

The **lift** operator on the substitution $s : \Gamma \rightarrow \Delta$ provides a new substitution $\Uparrow(s) : \Gamma, x \rightarrow \Delta, x$, where x is a new variable. $\Uparrow(s)$ leaves unchanged the top of the stack. So $\mathtt{1}\langle\Uparrow(s)\rangle$ rewrites to `1`. On variables `n+1`, that is to say the ones in the stack Δ, the action is merely the same as the action of s. However, we must keep in mind that the free variables of the results are pushed one unit deeper in the stack Γ, x. Thus, $\texttt{n+1}\langle \Uparrow(s) \rangle$ rewrites to $\texttt{n}\langle s \circ \uparrow \rangle$.

The substitution created by the contraction of a redex is still undefined. Given $(\boldsymbol{\lambda} x.M\ N) \rightarrow M\langle s \rangle$, let us try to make s more precise. As a morphism, we have clearly $s : \Delta \rightarrow \Delta, x$. We can even see that s extends the identity substitution $Id : \Delta \rightarrow \Delta$, due to the left hand expression. The substitution s could therefore be written $N \cdot Id$. More generally, given $s : \Gamma \rightarrow \Delta$ and any term M whose free variables belong to Γ, we define the **cons** $M \cdot s : \Gamma \rightarrow \Delta, x$, where x is a fresh variable. Its action on variables should be clear: the top variable is replaced by M, and it behaves as s with respect to the other ones.

Last but not least, **λenv** allows for free term-variables and substitution-variables, not to be confused with the free variables of a term!

Formally, the set **Λenv** of terms and substitutions is defined inductively as follows:

$$\begin{array}{llcll} \textbf{Terms} & M & ::= & X \mid \texttt{n} \mid (M\ M) \mid \boldsymbol{\lambda} M \mid M\langle s \rangle & \text{where } \texttt{n} \in \mathbb{N} \\ \textbf{Substitutions} & s & ::= & x \mid Id \mid \uparrow \mid \Uparrow(s) \mid M \cdot s \mid s \circ s & \end{array}$$

where X is a variable for terms, and x a variable for substitutions.

The **λenv** rewrite system is defined by the rules given in table 1. The rewrite rules have been explained informally above. Let us notice that some rules are "duplicated": suffixes $1, 2$ are appended somewhere. The introduction of the pairs of rules with pattern like (XXX1) and (XXX2) is needed for ensuring the confluence property. Actually, the second one is introduced automatically by Knuth-Bendix completion process. The secondary rules can therefore be ignored at the first reading.

Since our study follows very closely Hardin-Lévy's, the main properties will appear as special cases of the results presented here. Therefore, we do not give them in full details, but rather refer to [HaLe89] for a complete presentation of the system and its own main properties.

3 Parigot's pure $\lambda\mu$-calculus

We briefly present Parigot's calculus, and give a de Bruijn translation for this extension of the usual λ-calculus. We recall the more usual reductions for this calculus, and their traduction in the de Bruijn setting.

The sets of $\lambda\mu$-terms and named terms, is inductively defined by:

$$\begin{array}{llcl} \textbf{Terms} & T & ::= & x \mid (T\ T) \mid \boldsymbol{\lambda} x.T \mid \boldsymbol{\mu}\alpha T' \\ \textbf{Named} & T' & ::= & [\alpha]T \end{array}$$

where x (resp. α) ranges over λ-variables (resp. μ-variables). As $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ is a binding operator: free occurrences of the μ-variable α in the (named) term T' become bound in $\boldsymbol{\mu}\alpha T'$.

Besides the usual notion of reduction $\boldsymbol{\beta}$, there is a *structural* notion of reduction $\boldsymbol{\mu}$; and we will also consider a *renaming* rule $\boldsymbol{\rho}$, defined as:

$$\begin{array}{lrcl} \boldsymbol{\mu} & (\boldsymbol{\mu}\alpha[\beta]M'\ N) & \rightarrow & \boldsymbol{\mu}(([\beta]M')[N/^{\star}\alpha]) \\ \boldsymbol{\rho} & [\delta]\boldsymbol{\mu}\alpha M' & \rightarrow & M'[\delta/\alpha] \end{array}$$

where the substitution $[N/x]$ of the term N for the variable x is well-known, and the substitution $[N/^{\star}\alpha]$ of the term N for the μ-variable α in the term M consists in replacing **recursively** each occurrence of a sub-named term $[\alpha]T$ in M by the sub-named term $[\alpha](T\ N)$. Again, we shall admit that these definitions agree with the usual ones. This fact can be proved, following [CHL92] for instance.

In [Par91, Par93] several examples are provided, and interesting properties of the calculus are shown when a type assignment is introduced. Moreover, [Gro94] gives an interesting link between this calculus and the former Felleisen's λ_c-calculus.

Our favorite example This example has no special significance from the point of view developed in this paper. One may notice, however, that this classical term is strongly related to the intuitionistic term $1 \equiv \boldsymbol{\lambda} x \boldsymbol{\lambda} f.(f\ x)$ [Par93]. We introduce the terms $T \equiv (\boldsymbol{\mu}\alpha[\alpha](f\ \boldsymbol{\mu}\delta[\alpha](f\ x))\ N)$ and $M \equiv \boldsymbol{\lambda} x \boldsymbol{\lambda} f.T$. We get:

$$\begin{array}{rcll} T & \equiv & (\boldsymbol{\mu}\alpha[\alpha](f\ \boldsymbol{\mu}\delta[\alpha](f\ x))\ N) & \boldsymbol{\mu} \text{ reduction} \\ & \rightarrow & \boldsymbol{\mu}\alpha([\alpha](f\ \boldsymbol{\mu}\delta[\alpha](f\ x)))[N/^{\star}\alpha] & \text{substitution step} \\ & \rightarrow & \boldsymbol{\mu}\alpha([\alpha](f\ \boldsymbol{\mu}\delta[\alpha](f\ x\ N)\ N) & \end{array}$$

de Bruijn coding consists in representing each occurrence of a variable by its distance to the related binder. In our particular case, the set $\mathbf{\Lambda M}$ of $\lambda\mu$-terms, in de Bruijn notation is:

$$\begin{array}{llll} \textbf{Terms} & T & ::= & \mathtt{n} \mid (T\ T) \mid \boldsymbol{\lambda} T \mid \boldsymbol{\mu} T' \qquad \text{where } \mathtt{n} \in \mathbb{N} \\ \textbf{Named} & T' & ::= & [\mathtt{n}]T \end{array}$$

We note that this obvious translation erases the distinction between λ-variables and μ-variables. For example $\boldsymbol{\lambda\mu}[2](\mathtt{2}\ \mathtt{1})$ is allowed; and it can be seen as a de Bruijn term for $\boldsymbol{\lambda} x \boldsymbol{\mu}\alpha[x](x\ \alpha)$, although it is not well formed in the $\lambda\mu$-calculus. However, this is not so important, since μ-variables can appear only in places such as $[\alpha]M$. And we can easily check for well formed terms informally. For the purpose of the embedding, we will need a formal treatment (see section 5).

We extend the previously given operations by:

$$\begin{array}{rcl} (\boldsymbol{\mu}[\mathtt{p}]M')\{\mathtt{n} \leftarrow P\} & = & \boldsymbol{\mu}[\mathtt{p}]M'\{\mathtt{n+1} \leftarrow P\} \\ t^{\mathtt{m}}_{\mathtt{i}}(\boldsymbol{\mu}[\mathtt{p}]M') & = & \boldsymbol{\mu}[t^{\mathtt{m}}_{\mathtt{i+1}}(\mathtt{p})]t^{\mathtt{m}}_{\mathtt{i+1}}(M') \end{array}$$

The $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ reductions are described precisely as follows:

$$\begin{array}{lrcl} \boldsymbol{\mu} & (\boldsymbol{\mu} M'\ N) & \rightarrow & \boldsymbol{\mu}(M'\{\mathtt{1} \Leftarrow N\}) \\ \boldsymbol{\rho} & [\mathtt{p}]\boldsymbol{\mu} M' & \rightarrow & M'\{\mathtt{1} \leftarrow \mathtt{p}\} \end{array}$$

For the new substitution mechanism involved, we get:

$$\begin{array}{rcl} \mathtt{p}\{\mathtt{n} \Leftarrow P\} & = & \mathtt{p} \\ (M\ N)\{\mathtt{n} \Leftarrow P\} & = & (M\{\mathtt{n} \Leftarrow P\}\ N\{\mathtt{n} \Leftarrow P\}) \\ (\boldsymbol{\lambda} M)\{\mathtt{n} \Leftarrow P\} & = & \boldsymbol{\lambda} M\{\mathtt{n+1} \Leftarrow P\} \\ (\boldsymbol{\mu}[\mathtt{p}]M')\{\mathtt{n} \Leftarrow P\} & = & \begin{cases} \boldsymbol{\mu}[\mathtt{p}](M'\{\mathtt{n+1} \Leftarrow P\}\ t^{\mathtt{n+1}}_{\mathtt{0}}(P)) & \text{if } \mathtt{p} = \mathtt{n} \\ \boldsymbol{\mu}[\mathtt{p}]M'\{\mathtt{n+1} \Leftarrow P\} & \text{if } \mathtt{p} \neq \mathtt{n} \end{cases} \end{array}$$

An example Let us consider how this applies to the preceding example, where $t^{\mathtt{n+1}}_{\mathtt{0}}(N) \equiv N^{\mathtt{n}}$:

$$\begin{array}{rcll} T & \equiv & (\boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\ N) & \boldsymbol{\mu}\text{-reduction rule} \\ & \rightarrow & \boldsymbol{\mu}([\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4})))\{\mathtt{1} \Leftarrow N\} & \text{substitution rules} \\ & \rightarrow & \boldsymbol{\mu}[\mathtt{1}]((\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\{\mathtt{1} \Leftarrow N\}\ t^{\mathtt{1+1}}_{\mathtt{0}}(N)) & \\ & \rightarrow & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\{\mathtt{1} \Leftarrow N\}\ (\boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\{\mathtt{1} \Leftarrow N\}\ N^{1}) & \\ & \rightarrow & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ (\boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\{\mathtt{2} \Leftarrow N\}\ N^{1}) & \\ & \rightarrow & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}]((\mathtt{3}\ \mathtt{4})\{\mathtt{2} \Leftarrow N\}\ t^{\mathtt{2+1}}_{\mathtt{0}}(N))\ N^{1}) & \\ & \rightarrow & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}\ N^{2})\ N^{1}) & \end{array}$$

4 The $\boldsymbol{\lambda\mu}$env rewrite system

In this section, we give the formal presentation of our calculus and state its main properties: confluence of the full system and termination of the subsystem related to substitutions.

4.1 The full system

Before presenting the $\boldsymbol{\lambda\mu}$**env** rewrite system, let us underline the fact that we wanted to preserve two features proposed for $\boldsymbol{\lambda}$**env**, of some particular interest. First, it is presented as a full rewrite system, with a simple polynomial interpretation which easily ensures termination. The confluence property owes much to the introduction of the special operator **lift** in presence of λ-abstractions. Thus, our choices for the new rules and new constructions has been made to keep as much as possible the spirit of the $\boldsymbol{\lambda}$**env** calculus.

Now, some explanations about new term (resp. substitution) constructors, and the rules added in the new calculus. Both $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ reduction rule create substitutions. However, the one produced by the $\boldsymbol{\rho}$ is already present from $\boldsymbol{\lambda}$**env**. So, we concentrate on the substitution created through a μ-reduction.

Let us write informally $s \equiv N/^{\star}\alpha$. We consider the action $([\beta]M)\langle s\rangle$. The substitution s applies recursively, thus simultaneously to the sub-term $[\beta]M$ and to all its strict sub-named terms. We want to write that $([\beta]M)\langle s\rangle$ rewrites to $[\beta](M\langle s\rangle N)$, if $\beta = \alpha$, and to $[\beta]M\langle s\rangle$ otherwise. In a rewrite system,

Mu	$(\boldsymbol{\mu} M\ N)$	$\rightarrow$	$\boldsymbol{\mu} M\langle N\langle\uparrow\rangle\ /\ Id\rangle$
Rho	$[L](\boldsymbol{\mu} M)$	$\rightarrow$	$M\langle L\cdot Id\rangle$
SeqApp	$[N\mid L]M$	$\rightarrow$	$[L](M\ N)$
MuTerm	$(\boldsymbol{\mu} M)\langle s\rangle$	$\rightarrow$	$\boldsymbol{\mu} M\langle\Uparrow(s)\rangle$
NamTerm	$([L]M)\langle s\rangle$	$\rightarrow$	$[L\langle s\rangle]M\langle s\rangle$
SeqTerm	$(M\mid L)\langle s\rangle$	$\rightarrow$	$M\langle s\rangle\mid L\langle s\rangle$
FRefArg	$\mathtt{1}\langle M\ /\ s\rangle$	$\rightarrow$	$M\mid\mathtt{1}\langle s\rangle$
RRefArg	$\mathtt{n+1}\langle M\ /\ s\rangle$	$\rightarrow$	$\mathtt{n+1}\langle s\rangle$
ArgEnv	$M\ /\ s\circ t$	$\rightarrow$	$M\langle t\rangle\ /\ (s\circ t)$
ArgMap	$M\ /\ (L\cdot s)$	$\rightarrow$	$(M\mid L)\cdot s$
LiftArg	$\Uparrow(s)\circ N\ /\ t$	$\rightarrow$	$M\ /\ (\Uparrow(s)\circ t)$
ShiftArg	$\uparrow\circ M\ /\ s$	$\rightarrow$	$\uparrow\circ s$

Table 2: The additional rewrite rules

these two operations will be completely separated. Hence, an intermediate data structure is required, able to store the possible argument N until M possibly recovers it. Towards a convenient solution, we can observe that a μ-binder never disappears, hence any list of arguments, say $N_1\cdots N_p$ ($p\geq 0$) may be passed to places $[\alpha]T$ occuring in $[\beta]M$. Hence the effect of the derivation

$$(\boldsymbol{\mu}\alpha[\beta]M\ N_1\cdots N_p)\xrightarrow{\boldsymbol{\mu}^\star}\boldsymbol{\mu}\alpha([\beta]M)[N_1/^\star\alpha]\cdots[N_p/^\star\alpha]$$

consists in replacing **recursively** each occurrence of a sub-term $[\alpha]T$ of $[\beta]M$ by $[\alpha](T\ N_1\cdots N_p)$. Thus, the intermediate structure should store a list of terms as well. Since an empty list of arguments corresponds to the sole symbol α, our solution will consist in extending the set of $\lambda\mu$-terms, with terms of the form $\boldsymbol{\mu}[L]M$ (in de Bruijn notation) where L is built as a "list" of terms, say $N_1\cdots N_p$ *continued* with μ-variable β. Let us note $L\equiv N_1\mid\cdots\mid N_p\mid\beta$. Such terms will be called **sequences**. The substitution mechanism extends trivially to sequences by $(A\mid L)\langle s\rangle\equiv A\langle s\rangle\mid L\langle s\rangle$. Let us consider again the *empty* sequence case. It is twofold: $\beta[N/^\star\alpha]=\beta$ if $\beta\neq\alpha$, and $N\mid\alpha$ otherwise. The corresponding rules will be easy to express now. In order to complete the substitution process, it remains to communicate the waiting arguments of the sequences to the term. This is done through the new rule $[A\mid L]M\rightarrow[L](M\ A)$.

We still have to find a convenient solution for the creation of the substitution $s\equiv[N/^\star\alpha]$ itself. When a β-redex is contracted, the λ-binder vanishes. This is not the case for μ-redexes, as already noticed: $(\boldsymbol{\mu}\alpha M'\ N)$ is reduced into $\boldsymbol{\mu}\alpha M\langle s\rangle$ for some substitution s. So we get $s:\Delta,\alpha\rightarrow\Delta,\alpha$. But s should not be identified with a lifted substitution, since it depends also on N. Therefore, we must propose a new notation. Let us write $s\equiv N\ /\ Id$; or $s\equiv N\langle\uparrow\rangle\ /\ Id$ if we choose to relocate by anticipation, the free variables of N. The latter form has been chosen. The general form of this new substitution is $M\ /\ s:\Gamma,\alpha\rightarrow\Delta,\alpha$, where $s:\Gamma,\alpha\rightarrow\Delta,\alpha$ and the free variables of M belong to Γ,α. It remains to give some details about its action on variables. The result must be a sequence, anyway. Following the above explanations, $\mathtt{1}\langle M\ /\ s\rangle$ will rewrite to $M\mid\mathtt{1}\langle s\rangle$ and $\mathtt{n+1}\langle M\ /\ s\rangle$ to $\mathtt{n+1}\langle s\rangle$. In accordance with this definition, $\mathtt{1}\langle N_1\ /\cdots/\ N_p\ /\ s\rangle$ is reduced to $N_1\mid\cdots\mid N_p\mid(\mathtt{1}\langle s\rangle)$. Therefore, $([\mathtt{1}]T)\langle N_1\ /\cdots/\ N_p\ /\ s\rangle$ rewrites into $[\mathtt{1}](T\langle N_1\ /\cdots/\ N_p\ /\ s\rangle\ N_1\ \cdots\ N_p)$, as expected.

The second rule we retain from the $\lambda\mu$-calculus is $\boldsymbol{\rho}$, called a *renaming* rule. However, at the light of the previous analysis, the $\boldsymbol{\rho}$ rule has also the meaning of connecting two "channels" on which sequences of arguments are communicated. In our "enriched" $\lambda\mu$-calculus this reduction rule will substitute, in the usual sense, a μ-variable α by any sequence L. Therefore, the reduction $[L]\boldsymbol{\mu}\alpha M\rightarrow M[L/\alpha]$ is taken into account through the rewrite rule $[L]\boldsymbol{\mu} M\rightarrow M\langle L\cdot Id\rangle$.

Definition 4.1 *The set* $\boldsymbol{\Lambda M}$***env*** *of terms and substitutions is inductively defined by:*

Terms	M	$::=$	$X\mid\boldsymbol{n}\mid(M\ M)\mid\boldsymbol{\lambda} M\mid\boldsymbol{\mu} M\mid[M]M\mid(M\mid M)\mid M\langle s\rangle$	*where* $\boldsymbol{n}\in\mathbb{N}$
Substitutions	s	$::=$	$x\mid Id\mid\uparrow\mid\Uparrow(s)\mid M\cdot s\mid M\ /\ s\mid s\circ s$	

where X is a variable for terms, and x a variable for substitutions. The full $\boldsymbol{\lambda\mu}$***env*** *rewrite system is defined by the rules given in table 4. New rules are shown in table 2.*

The language is an extension of the language $\mathbf{\Lambda env}$ from [HaLe89]. It is called $\mathbf{\Lambda Menv}$ although it actually contains more terms than the ones corresponding to terms of the $\lambda\mu$-calculus: terms are coded using integers, with the confusing effect already pointed out.

An example Let N be any term. We note N^+ for $N\langle\uparrow\rangle$, N^{++} for $N\langle\uparrow\circ\uparrow\rangle$, etc... Only one rule is applicable to your term. We trace the main steps of the rewriting of T:

$$
\begin{array}{lllr}
T & \equiv & (\boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\ N) & \text{(Mu)} \\
 & \to & \boldsymbol{\mu}([\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4})))\langle N^+ \ /\ Id\rangle & \text{(NamedTerm), where } s \equiv N^+ \ /\ Id \\
 & \to & \boldsymbol{\mu}[\mathtt{1}\langle N^+ \ /\ Id\rangle](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\langle s\rangle & \text{(FRefArg)} \\
 & \to & \boldsymbol{\mu}[N^+ \mid \mathtt{1}\langle Id\rangle](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\langle s\rangle & \text{(IdEnv),(SeqApp),(AppTerm)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\langle s\rangle\ (\boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}))\langle s\rangle\ N^+) & \text{(RRefArg),(IdEnv),(MuTerm),(NamedTerm)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}\langle\Uparrow(s)\rangle](\mathtt{3}\ \mathtt{4})\langle\Uparrow(s)\rangle\ N^+) & \text{(AppTerm),(RRefLift1)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{1}\langle s\circ\uparrow\rangle](\mathtt{3}\ \mathtt{4})\langle\Uparrow(s)\rangle\ N^+) & \text{(FRefArg),(IdEnv),(RefShift1)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[N^+\langle\uparrow\rangle \mid \mathtt{2}](\mathtt{3}\ \mathtt{4})\langle\Uparrow(s)\rangle\ N^+) & \text{(SeqApp)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}]((\mathtt{3}\ \mathtt{4})\langle\Uparrow(s)\rangle\ N^{++})\ N^+) & \text{(AppTerm),(RRefLift1),(RRefArg),(IdL),(RefShift1)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}\langle\Uparrow(s)\rangle\ N^{++})\ N^+) & \text{(RRefLift1),(RRefArg),(IdL),(RefShift1)} \\
 & \to & \boldsymbol{\mu}[\mathtt{1}](\mathtt{2}\ \boldsymbol{\mu}[\mathtt{2}](\mathtt{3}\ \mathtt{4}\ N^{++})\ N^+) & \text{That's all folks.}
\end{array}
$$

4.2 The Subst subsystem

We restrict our attention to the sub-rewrite system related to substitutions. The following results imply the confluence property as a corollary.

Definition 4.2 *Subst is the rewrite system obtained from* $\boldsymbol{\lambda\mu}\mathbf{env}$ *by removing the reduction rules: Beta, Mu and Rho.*

Proposition 4.3 *The Subst system is weakly confluent and terminating.*

Proof The proof follows closely [HaLe89]. For proving weak confluence we made use of the KB system, developed at INRIA, in its version implemented in CAML-LIGHT. The proof of termination is based on a polynomial interpretation of terms and substitutions. It is a straightforward adaptation of the interpretation for $\mathbf{\lambda env}$ given in [HaLe89]: to each term/substitution A is associated the pair $(f(A), g(A))$:

$$
\begin{array}{ll|ll|l}
f(\mathtt{n}) & = 2^{\mathtt{n}} & g(\mathtt{n}) & = 1 & \\
f(M\ M) & = f(M) + f(N) & g(M\ N) & = g(M) + g(N) + 1 & \\
f(\boldsymbol{\lambda} M) & = f(M) + 2 & g(\boldsymbol{\lambda} M) & = 2g(M) & \\
f(\boldsymbol{\mu} M) & = f(M) + 2 & g(\boldsymbol{\mu} M) & = 2g(M) & \star \\
f([L]M) & = f(L) + f(M) + 4 & g([L]M) & = 2g(L) + g(M) & \star \\
f(M \mid L) & = f(M) + f(L) & g(M \mid L) & = g(M) + g(L) + 1 & \star \\
f(M\langle s\rangle) & = f(M)f(s) & g(M\langle s\rangle) & = g(M)(g(s)+1) & \\
f(Id) & = 2 & g(Id) & = 1 & \\
f(\uparrow) & = 2 & g(\uparrow) & = 1 & \\
f(\Uparrow(s)) & = s & g(\Uparrow(s)) & = 4g(s) & \\
f(M\cdot s) & = f(M) + f(s) & g(M\cdot s) & = g(M) + g(s) + 1 & \\
f(M\ /\ s) & = f(M) + f(s) + 4 & g(M\ /\ s) & = g(M) + 2g(s) & \star \\
f(s\circ t) & = f(s)f(t) & g(s\circ t) & = g(s)(g(t)+1) &
\end{array}
$$

where new cases are marked by $\star$. Pairs are ordered under the lexicographic ordering. The set $\mathbf{\Lambda Menv}$ is proved to be well-founded with respect to this relation. The termination property follows. □

4.3 Confluence for $\boldsymbol{\lambda\mu}$env

For the purpose of the proof of confluence, the following parallel reduction $\xrightarrow{\triangleright}$, also written $\triangleright$, is introduced on $\boldsymbol{\Lambda}$**Menv**:

$$\frac{}{M \triangleright M} \qquad \frac{M \triangleright M' \quad N \triangleright N'}{(M\ N) \triangleright (M'\ N')} \qquad \frac{M \triangleright M'}{\boldsymbol{\lambda} M \triangleright \boldsymbol{\lambda} M'}$$

$$\frac{M \triangleright M' \quad L \triangleright L'}{[L]M \triangleright [L']M'} \qquad \frac{M \triangleright M' \quad L \triangleright L'}{M \mid L \triangleright M' \mid L'} \qquad \frac{M \triangleright M'}{\boldsymbol{\mu} M \triangleright \boldsymbol{\mu} M'}$$

$$\frac{}{s \triangleright s} \qquad \frac{s \triangleright s' \quad t \triangleright t'}{s \circ t \triangleright s' \circ t'} \qquad \frac{s \triangleright s'}{\Uparrow(s) \triangleright \Uparrow(s')}$$

$$\frac{M \triangleright M' \quad s \triangleright s'}{M \cdot s \triangleright M' \cdot s'} \qquad \frac{M \triangleright M' \quad s \triangleright s'}{M \ / \ s \triangleright M' \ / \ s'} \qquad \frac{M \triangleright M' \quad s \triangleright s'}{M\langle s\rangle \triangleright M'\langle s'\rangle}$$

$$\frac{N \triangleright N' \quad L \triangleright L' \quad M \triangleright M'}{[N \mid L]M \triangleright [L'](M'\ N')}$$

$$\frac{M \triangleright M' \quad N \triangleright N'}{(\boldsymbol{\lambda} M\ N) \triangleright M'\langle N' \cdot Id\rangle} \qquad \frac{M \triangleright M' \quad N \triangleright N'}{(\boldsymbol{\mu} M\ N) \triangleright \boldsymbol{\mu} M'\langle N'\langle\uparrow\rangle \ / \ Id\rangle} \qquad \frac{M \triangleright M' \quad L \triangleright L'}{[L]\boldsymbol{\mu} M \triangleright M'\langle L' \cdot Id\rangle}$$

Lemma 4.4 *The parallel reduction $\triangleright$ is strongly confluent.*

Proof Since $\triangleright$ provides a left linear system with no critical pairs. □

Proposition 4.5 *Let A, B, C be elements of $\boldsymbol{\Lambda}$**Menv**. The following diagram holds:*

$$\begin{tikzcd} A \arrow[r, "\triangleright"] \arrow[d, "S"'] & B \arrow[d, dotted, "S^*"] \\ C \arrow[r, dotted, "S^* \triangleright S^*"] & D \end{tikzcd}$$

Proof When the two steps starting from A make no critical pair, the proposition is straightforward. So, we have to inspect the possible "critical pairs" in a sense adequate with the parallel reduction $\triangleright$. The proof is therefore by case on the derivation $A \xrightarrow{S} C$. Let us treat a single example when $A \equiv (M\ N)\langle s\rangle$ reduces to $C \equiv (M\langle s\rangle\ N\langle s\rangle)$. Then:

- $A \equiv (M\ N)\langle s\rangle \xrightarrow{\triangleright} (M'\ N')\langle s'\rangle \equiv B$: take $D \equiv (M'\langle s'\rangle\ N'\langle s'\rangle)$.
- $A \equiv (\boldsymbol{\lambda} P\ N)\langle s\rangle \xrightarrow{\triangleright} P'\langle N' \cdot Id\rangle\langle s'\rangle \equiv B$: take $D \equiv P'\langle N'\langle s'\rangle \cdot s'\rangle$.
- $A \equiv (\boldsymbol{\mu} P\ N)\langle s\rangle \xrightarrow{\triangleright} (\boldsymbol{\mu} P'\langle N'\langle\uparrow\rangle \ / \ Id\rangle)\langle s'\rangle \equiv B$: take $D \equiv \boldsymbol{\mu} P'\langle N'\langle s' \circ \uparrow\rangle \ / \ \Uparrow(s')\rangle$.

□

Theorem 4.6 *The $\boldsymbol{\lambda\mu}$**env** rewrite system is confluent.*

Proof Remark first that $\boldsymbol{\lambda\mu}$**env** $\subseteq S^* \triangleright S^* \subseteq$ $\boldsymbol{\lambda\mu}$**env**. So, the proof is mainly a diagram chasing problem. See [HaLe89, CHL92] for details and further references. □

5 Lambda-Mu Calculus as a sub-theory of $\boldsymbol{\lambda\mu}$env

The idea of this section is that our calculus codes too much terms than actually introduced in $\lambda\mu$-calculus. With the help of a formal system which gives sorts to the ground objects which will considered as well-formed from the point of view of the present study, we recover exactly as normal forms the terms of the $\lambda\mu$-calculus. Thus, we obtain simulation results analogous to the one presented in [HaLe89].

$$\frac{\mathtt{a.u} \le \mathtt{w} \quad \mathtt{a} \in \{\mathtt{l},\mathtt{m}\}}{\mathtt{w} \vdash \mathtt{a.u} : \mathtt{a}} \qquad\qquad \frac{}{Id : \mathtt{w} \to \mathtt{w}}$$

$$\frac{\mathtt{w.l} \vdash M : \mathtt{l}}{\mathtt{w} \vdash \boldsymbol{\lambda} M : \mathtt{l}} \qquad\qquad \frac{\mathtt{a} \in \{\mathtt{l},\mathtt{m}\}}{\uparrow^{\mathtt{a}} : \mathtt{w.a} \to \mathtt{w}}$$

$$\frac{\mathtt{w.m} \vdash M : \mathtt{l} \quad \mathtt{w.m} \vdash L : \mathtt{m}}{\mathtt{w} \vdash \boldsymbol{\mu}[L]M : \mathtt{l}} \qquad\qquad \frac{\mathtt{u} \vdash M : \mathtt{a} \quad \mathtt{a} \in \{\mathtt{l},\mathtt{m}\} \quad s : \mathtt{u} \to \mathtt{v}}{M \cdot s : \mathtt{u} \to \mathtt{v.a}}$$

$$\frac{\mathtt{w} \vdash M : \mathtt{l} \quad \mathtt{w} \vdash L : \mathtt{m}}{\mathtt{w} \vdash M \mid L : \mathtt{m}} \qquad\qquad \frac{s : \mathtt{u} \to \mathtt{v} \quad \mathtt{a} \in \{\mathtt{l},\mathtt{m}\}}{\Uparrow^{\mathtt{a}}(s) : \mathtt{u.a} \to \mathtt{v.a}}$$

$$\frac{\mathtt{w} \vdash M : \mathtt{l} \quad s : \mathtt{v} \to \mathtt{w}}{\mathtt{v} \vdash M\langle s\rangle : \mathtt{l}} \qquad\qquad \frac{\mathtt{u} \vdash M : \mathtt{l} \quad s : \mathtt{u} \to \mathtt{v.m}}{M \ / \ s : \mathtt{u} \to \mathtt{v.m}}$$

$$\frac{s : \mathtt{u} \to \mathtt{v} \quad t : \mathtt{v} \to \mathtt{w}}{s \circ t : \mathtt{u} \to \mathtt{w}}$$

Table 3: Sorted system for $\boldsymbol{\lambda\mu}\mathbf{env}^{\circ}$

5.1 A sorted system

An alternative approach to de Bruijn coding is well fitted to our metamathematical study of the simulation. Considering that integers also code sequences with an one-letter alphabet, we simply introduce a coding consisting in words form with a two letters alphabet, say $\{l, m\}$. de Bruijn original idea consisted in relating each occurrence of a variable with its distance to its binder. But here, we can cross two sorts of binders. Therefore, the coding will consist in using letter l (resp. m) when λ-binder (resp. μ-binder) is crossed. We choose arbitrarily the binder-to-variable way for the word code; in that case, the first letter gives the sort of the variable. In every case, words coding variable positions are non empty. This is the heart of our sorted system.

So let us introduce the set $\mathbb{W}$ as $\{\mathtt{l},\mathtt{m}\}\star$, and $\mathbb{W}^{+} \equiv \{\mathtt{l},\mathtt{m}\}+$. The empty word is ϵ, and construction of words is allowed through the operation $\mathtt{a.w}$, where $\mathtt{a} \in \{\mathtt{l},\mathtt{m}\}$ and $\mathtt{w} \in \mathbb{W}$. We allow for $\mathtt{.}$ as an operation of concatenation as well. The set $\mathbb{W}$ is partially ordered by the relation $\mathtt{u} \le \mathtt{v}$ if there exists $\mathtt{x} \in \mathbb{W}$ such that $\mathtt{v} = \mathtt{w.u}$; this operation is undefined otherwise.

We define $\mathbf{\Lambda Menv}^{\circ}$ as the set of *well-formed* terms, sequences and substitutions inductively defined through the sorted system presented in table 3. Terms of sort $\mathtt{l}$ (resp. $\mathtt{m}$) are **terms**, resp. **sequences**, and terms of sort $\mathtt{u} \to \mathtt{v}$ are **substitutions**. The set of well formed terms is denoted $\mathbf{\Lambda Mterms}^{\circ}$.

In this new setting, our previous example

$$M \equiv \boldsymbol{\lambda} x \boldsymbol{\lambda} f(\boldsymbol{\mu}\alpha[\alpha](f\ \boldsymbol{\mu}\delta[\alpha](f\ x))\ N)$$

is coded as

$$M = \boldsymbol{\lambda\lambda}(\boldsymbol{\mu}[\mathtt{m}](\mathtt{lm}\ \boldsymbol{\mu}[\mathtt{mm}](\mathtt{lmm}\ \mathtt{llmm})\ N)$$

Since the calculus is embedded in $\boldsymbol{\lambda\mu}\mathbf{env}$, the relevant rewrite rules should not have to be precised. Moreover, the following proposition shows $\mathbf{\Lambda Menv}^{\circ}$ is actually closed under the reductions rules introduced in section 4, table 4.

Proposition 5.1 *Let $A \in \mathbf{\Lambda Menv}^{\circ}$. If $A \to B$ by application of a rewrite rule, then $B \in \mathbf{\Lambda Menv}^{\circ}$.*

Proof It is sufficient to observe that the rewrite rules preserve the sort given to $A \in \mathbf{\Lambda Menv}^{\circ}$, and to check the case where (A, B) is a rule. $\square$

When introducing this sorted system, the intention was to keep from $\mathbf{\Lambda Menv}$ only those ground terms which correspond to *true* $\lambda\mu$-terms. But substitutions may still exist in these ground terms. So we must concentrate on terms which are in normal form with respect to the subset (Subst) of rewrite rules related to the substitution process. The Subst-normal form of an element $A \in \mathbf{\Lambda M}$ (or $\mathbf{\Lambda Menv}^{\circ}$) is noted *snf (A)*. This notation extends naturally to subsets. The Subst-derivation will be written $\xrightarrow{S}$ as well. We get

Proposition 5.2 *Subs-normal forms in $\boldsymbol{\Lambda Menv}^{\circ}$ are as follows:*

$$
\begin{array}{llcl}
\textbf{\textit{Terms}} & M & ::= & \mathtt{1}.w \mid \boldsymbol{\lambda} M \mid \boldsymbol{\mu}[\mathtt{m}.w]M \mid (M\ M) \\
\textbf{\textit{Sequences}} & L & ::= & \mathtt{m}.w \mid (M \mid L) \\
\textbf{\textit{Substitutions}} & s & ::= & M \cdot s \mid L \cdot s \mid t \mid M\ /\ u \\
\textit{where} & t & ::= & \Uparrow^{w}(s) \circ \uparrow^{w} \\
\textit{and} & u & ::= & t \mid M\ /\ u
\end{array}
$$

where $w \in \mathcal{W}$. We allow ourselves the convention that Id is erased in compositions. The following abbreviations are used: $\Uparrow^{a}(\Uparrow^{w}(s)) \equiv \Uparrow^{w.a}(s)$ and $\uparrow^{a} \circ \uparrow^{w} \equiv \uparrow^{a.w}$.

Proof The proof is by structural induction. For a term, we have to show that it can not contain any substitution: a sub-term with the shape $M\langle s\rangle$. For a substitution it is sufficient to study the case $s \circ t$, and to discuss on the structure of s. The part played by the sorts for this result is central. Without them, it would have been impossible to avoid normal forms like $\Uparrow(s) \circ M\ /\ t$, although such a substitution cannot appear through a regular substitution process. With respect to terms, this importance has been already emphasized. □

As a corollary, notice that ground terms in Subst-normal form are exactly $\lambda\mu$-terms, as expected. from now on, we identify the sets $\mathbf{\Lambda M}$ of $\lambda\mu$-terms and *snf ($\boldsymbol{\Lambda Mterms}^{\circ}$)*.

5.2 Substitutions in de Bruijn setting

Before the simulation results, we define the substitution mechanisms for the subset of well-formed terms, sequences and substitutions. In agreement with our more precise notation, they are translated as $M\{\mathtt{w} \leftarrow N\}$ and $M\{\mathtt{w} \Leftarrow N\}$ respectively. Let us make precise the definitions:

$$
\begin{array}{rcl}
\mathtt{u}\{\mathtt{w} \leftarrow P\} & = & \left\{ \begin{array}{ll} (\mathtt{u} - \mathtt{1}.\mathtt{w}).\mathtt{w} & \text{if } \mathtt{u} \geq \mathtt{1}.\mathtt{w} \\ t^{\mathtt{w}}_{\epsilon}(P) & \text{if } \mathtt{u} = \mathtt{w} \\ \mathtt{u} & \text{if } \mathtt{u} < varw \end{array} \right. \\
(M\ N)\{\mathtt{w} \leftarrow P\} & = & (M\{\mathtt{w} \leftarrow P\}\ N\{\mathtt{w} \leftarrow P\}) \\
(\boldsymbol{\lambda} M)\{\mathtt{w} \leftarrow P\} & = & \boldsymbol{\lambda} M\{\mathtt{w}.\mathtt{1} \leftarrow P\} \\
(\boldsymbol{\mu}[\mathtt{u}]M)\{\mathtt{w} \leftarrow P\} & = & \boldsymbol{\mu}[\mathtt{u}]M\{\mathtt{w}.\mathtt{m} \leftarrow P\}
\end{array}
$$

and

$$
\begin{array}{rcl}
\mathtt{u}\{\mathtt{w} \Leftarrow P\} & = & \mathtt{u} \\
(M\ N)\{\mathtt{w} \Leftarrow P\} & = & (M\{\mathtt{w} \Leftarrow P\}\ N\{\mathtt{w} \Leftarrow P\}) \\
(\boldsymbol{\lambda} M)\{\mathtt{w} \Leftarrow P\} & = & \boldsymbol{\lambda} M\{\mathtt{w}.\mathtt{1} \Leftarrow P\} \\
(\boldsymbol{\mu}[\mathtt{u}]M)\{\mathtt{w} \Leftarrow P\} & = & \left\{ \begin{array}{ll} \boldsymbol{\mu}[\mathtt{p}](M\{\mathtt{w}.\mathtt{m} \Leftarrow P\}\ t^{\mathtt{w}.\mathtt{m}}_{\epsilon}(P)) & \text{if } \mathtt{u} = \mathtt{w} \\ \boldsymbol{\mu}[\mathtt{u}]M\{\mathtt{w}.\mathtt{m} \Leftarrow P\} & \text{if } \mathtt{u} \neq \mathtt{w} \end{array} \right.
\end{array}
$$

For the lifting operation, we get:

$$
\begin{array}{rcl}
t^{\mathtt{w}}_{\mathtt{v}}(\mathtt{u}) & = & \left\{ \begin{array}{ll} \mathtt{u}.\mathtt{w} & \text{if } \mathtt{u} > \mathtt{v} \\ \mathtt{u} & \text{otherwise} \end{array} \right. \\
t^{\mathtt{w}}_{\mathtt{v}}(M\ N) & = & (t^{\mathtt{w}}_{\mathtt{v}}(M)\ t^{\mathtt{w}}_{\mathtt{v}}(N)) \\
t^{\mathtt{w}}_{\mathtt{v}}(\boldsymbol{\lambda} M) & = & \boldsymbol{\lambda} t^{\mathtt{w}}_{\mathtt{v}.\mathtt{1}}(M) \\
t^{\mathtt{w}}_{\mathtt{v}}(\boldsymbol{\mu}[\mathtt{u}]M) & = & \boldsymbol{\mu}[t^{\mathtt{w}}_{\mathtt{v}.\mathtt{m}}(\mathtt{u})]t^{\mathtt{w}}_{\mathtt{v}.\mathtt{m}}(M)
\end{array}
$$

5.3 Simulation results

The following definition gives the simulated reduction for $\lambda\mu$-terms within $\boldsymbol{\lambda\mu}\mathbf{env}$.

Definition 5.3 *Let M, $N \in \mathbf{\Lambda M}$. The simulated reductions are given by:*

$$\begin{array}{ll} \boldsymbol{\beta} & M \stackrel{Sim\beta}{\longrightarrow} N \textit{ iff } M \stackrel{(Beta)}{\longrightarrow} P \textit{ and } N = \text{snf}\,(P) \\ \boldsymbol{\mu} & M \stackrel{Sim\mu}{\longrightarrow} N \textit{ iff } M \stackrel{(Mu)}{\longrightarrow} P \textit{ and } N = \text{snf}\,(P) \\ \boldsymbol{\rho} & M \stackrel{Sim\rho}{\longrightarrow} N \textit{ iff } M \stackrel{(Rho)}{\longrightarrow} P \textit{ and } N = \text{snf}\,(P) \end{array}$$

We start by the following key lemma.

Lemma 5.4 *Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \mathbb{W}$, $s, M \in \mathbf{\Lambda M}$. We get*

- *If $\boldsymbol{u} \geq \boldsymbol{v}$ and $\boldsymbol{w} \geq \boldsymbol{v}$, then $\boldsymbol{u}\langle \Uparrow^{\boldsymbol{w}}(s)\rangle \xrightarrow{S^*} \boldsymbol{u}\text{-}\boldsymbol{v}\langle \Uparrow^{\boldsymbol{w}\text{-}\boldsymbol{v}}(s) \circ \uparrow^{\boldsymbol{v}}\rangle$.*
- *$t^{\boldsymbol{w}}_{\boldsymbol{v}}(M) = \text{snf}\,(M\langle \Uparrow^{\mathtt{v}}(\uparrow^{\mathtt{w}})\rangle)$.*

We prove that these definitions actually give simulations for the usual reductions.

Theorem 5.5 *For any terms M, $N \in \boldsymbol{\Lambda Menv}^{\circ}$ and $\boldsymbol{r} \in \{\boldsymbol{\beta}, \boldsymbol{\mu}, \boldsymbol{\rho}\}$, $M \xrightarrow{\boldsymbol{r}} N$ iff $M \stackrel{Sim\boldsymbol{r}}{\longrightarrow} P$.*

Proof The proof follows [HaLe89, CHL92]. We fix arbitrarily $\boldsymbol{r} \equiv \mu$ in the discussion. The key point is the following. Let $M \equiv \ldots(\boldsymbol{\mu}A\ B)\ldots$. Assume $M \xrightarrow{\boldsymbol{\mu}} N$ and $M \stackrel{\text{Sim}\boldsymbol{\mu}}{\longrightarrow} Q$. Then $N \equiv \ldots\boldsymbol{\mu}(A\{\mathtt{m} \Leftarrow B\})\ldots$ and $Q \equiv$ *snf* $(\ldots\boldsymbol{\mu}A\langle B\langle\uparrow^{\mathtt{m}}\rangle \circ Id\rangle\ldots)$. But $M \in \mathbf{\Lambda Mterms}^{\circ}$, and so (Subst)-redexes can only be created in the subtree where the μ-redex is contracted. So $Q \equiv \ldots$ *snf* $(\boldsymbol{\mu}A\langle B\langle\uparrow^{\mathtt{m}}\rangle \circ Id\rangle)\ldots$. Therefore, given $\mathtt{u}, \mathtt{v}, \mathtt{w} \in \mathbb{W}$ and $M, N \in \mathbf{\Lambda Menv}^{\circ}$, we are left to prove $M\{\mathtt{w} \Leftarrow N\} =$ *snf* $(M\Uparrow^{\boldsymbol{w}}(N\langle\uparrow^{\mathtt{m}}\rangle\ /\ Id))$. At this point, the preceding lemma is used.

□

Conversely, we want now to relate rewrite rules corresponding to reductions rules with (simulated) reductions on $\lambda\mu$-terms.

Theorem 5.6 *For any A, $B \in \boldsymbol{\Lambda Menv}^{\circ}$ the following diagram holds:*

$$\begin{tikzcd} A \arrow[r, "\boldsymbol{\lambda\mu env}^{\star}"] \arrow[d, "S^{\star}"'] & B \arrow[d, "S^{\star}"] \\ \text{snf}\,(A) \arrow[r, dotted, "\boldsymbol{\beta\mu\rho}^{\star}"] & \text{snf}\,(B) \end{tikzcd}$$

Proof It suffices to show, for each (Red) $\in \{(\text{Beta}), (\text{Mu}), (\text{Rho})\}$, that the corresponding simulated relation $\stackrel{\text{Sim}red}{\longrightarrow}$ satisfies the following diagram:

$$\begin{tikzcd} A \arrow[r, "\text{(Red)}"] \arrow[d, "S^{\star}"'] & B \arrow[d, "S^{\star}"] \\ snf\ (A) \arrow[r, dotted, "\mathbf{Simred}^{\star}"] & snf\ (B) \end{tikzcd}$$

The proof is by induction on the pairs $(\mathbf{size}(M), \mathbf{length}(M))$ ordered lexically, where $\mathbf{size}(M)$ is the size of the abstract syntax tree coding M and $\mathbf{length}(M)$ is the maximal length of a Subst*-reduction starting from M. We proceed by case on the structure of A. The proof is tedious, and brings no new difficulty, compared to [CHL92]. □

6 Conclusion and further developments

In [HaLe89], a confluent rewrite system has been proposed as a theoretical basis for the modelisation of implementations of the β-reductions. This work relies on the (pure) λ-calculus as a paradigm of (pure) functional programming languages. $\boldsymbol{\lambda}$**env** provides a good theoretical framework to study the abstracts properties of implementations of these languages.

Our starting point has been to extend this result to Parigot's $\lambda\mu$-calculus. This choice is strongly motivated. On the one hand, this calculus has been given a solid logical justification, and it captures the computational contents from proofs for a natural deduction system with multiple conclusions, which allows to develop proofs in classical logic. And the $\lambda\mu$-calculus shares the same properties as the λ-calculus: confluence, and strong normalization when it is a relevant question. On the other hand, the link between classical proofs and the use of control structures in functional languages has been strongly established. Thus, we have considered the $\lambda\mu$-calculus as a good candidate as a paradigm for the functional programming languages extended with control structures: sub parts of Caml and Scheme for example. From that point of view, the extension of Hardin-Lévy work to this calculus is a first step.

In this paper, we proposed a confluent rewrite system ($\boldsymbol{\lambda\mu}$**env**), containing $\boldsymbol{\lambda}$**env** and as close as possible to the presentations of the $\lambda\mu$-calculus given in [Par91, Par93]. Our results extend those given for $\boldsymbol{\lambda}$**env**, and allow similarly to consider the $\lambda\mu$-calculus as a sub-theory of $\boldsymbol{\lambda\mu}$**env**. However, the introduction of a sorted system has shown necessary in order to avoid the confusion between λ-variables and μ-variables. In the meantime, this study has brought to light some aspects of the $\lambda\mu$-calculus. The $\boldsymbol{\rho}$ rule, called *renaming* rule, should be actually considered as a full reduction rule. And the syntax for $\lambda\mu$-terms enriched in a natural way, with the introduction the *sequence* structure is closer to the actual specific substitution introduced by Parigot. Sequences allows the substitution process to be explicited. Moreover, it can be observed that our syntactical treatment meets the technical tools developed in [Par93] for the purpose of proving strong normalization property. Therefore, this work seems to offer a better view on the initial $\lambda\mu$-calculus. In [Au94], we develop these remarks. Also, according to our initial motivation, we propose different environment machines for the implementation of the functional programming languages extended with control structures.

Acknowledgments

Thanks to Christine Paulin and Eduardo Gimenez for their attentive reading. Special thanks to Elena Zucca for her numerous comments and suggestions on a previous version of this paper.

References

[ACCL90] M. Abadi, L. Cardelli, P.-L. Curien, J.-J. Lévy, *Explicit Substitutions*. ACM conference on Principles of Programming Languages, San Francisco, 1990.

[Au94] P. Audebaud, *Environment machines for functional languages extended with control structures*. ENS-Lyon Research Report in preparation.

[CHL92] P.-L. Curien, T. Hardin, J.-J. Lévy, *Confluence properties of weak and strong calculi of explicit substitutions*. INRIA Research Report **1617**.

[Gri90] T. Griffin, *Y formulae-as-types notion of control*, Proceedings of POPL, 1990.

[Gro94] Ph. de Groote, *On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control*. In F. Pfenning ed, Proceedings of 5th International Conference, LPAR'94, Kiev (Ukraine). LNAI **822**.

[HaLe89] T. Hardin, J.-J. Lévy, *A confluent calculus of substitutions*, France-Japan Artificial Intelligence and Computer Science Symposium, Izu, 1989.

[Par91] M. Parigot, *$\lambda\mu$-calculus: an algorithmic interpretation of Classical Natural Deduction*. Proc. International Conference on Logic Programming and Automated Reasoning, St Petersburg (Russia) 1991. Springer-Verlag LNCS **624**, pp 190-201.

[Par93] M. Parigot, *Strong normalization for the second order classical natural deduction*. In Proceedings of the eight annual IEEE symposium on Logic in Computer Science, LICS'93.

[Par93] M. Parigot, *Classical Proofs as Programs*. In G. Gotlod, A. Leitsch and D. Mundici eds., Proceedings of the third Kurt Gödel Colloquium KGC'93. LNCS **733**.

Beta	$(\boldsymbol{\lambda} M\ N)$	$\to$	$M\langle N \cdot Id\rangle$
Mu	$(\boldsymbol{\mu} M\ N)$	$\to$	$\boldsymbol{\mu} M\langle N\langle\uparrow\rangle \ / \ Id\rangle$
Rho	$[L](\boldsymbol{\mu} M)$	$\to$	$M\langle L \cdot Id\rangle$
SeqApp	$[N \mid L]M$	$\to$	$[L](M\ N)$
LamTerm	$(\boldsymbol{\lambda} M)\langle s\rangle$	$\to$	$\boldsymbol{\lambda} M\langle\Uparrow(s)\rangle$
MuTerm	$(\boldsymbol{\mu} M)\langle s\rangle$	$\to$	$\boldsymbol{\mu} M\langle\Uparrow(s)\rangle$
AppTerm	$(M\ N)\langle s\rangle$	$\to$	$(M\langle s\rangle\ N\langle s\rangle)$
NamTerm	$([L]M)\langle s\rangle$	$\to$	$[L\langle s\rangle]M\langle s\rangle$
SeqTerm	$(M \mid L)\langle s\rangle$	$\to$	$M\langle s\rangle \mid L\langle s\rangle$
Closure	$M\langle s\rangle\langle t\rangle$	$\to$	$M\langle s \circ t\rangle$
IdEnv	$M\langle Id\rangle$	$\to$	M
RefShift1	$\mathtt{n}\langle\uparrow\rangle$	$\to$	$\mathtt{n+1}$
RefShift2	$\mathtt{n}\langle\uparrow \circ s\rangle$	$\to$	$\mathtt{n+1}\langle s\rangle$
FRefLift1	$\mathtt{1}\langle\Uparrow(s)\rangle$	$\to$	$\mathtt{1}$
FRefLift2	$\mathtt{1}\langle\Uparrow(s) \circ t\rangle$	$\to$	$\mathtt{1}\langle t\rangle$
RRefLift1	$\mathtt{n+1}\langle\Uparrow(s)\rangle$	$\to$	$\mathtt{n}\langle s \circ \uparrow\rangle$
RRefLift2	$\mathtt{n+1}\langle\Uparrow(s) \circ t\rangle$	$\to$	$\mathtt{n}\langle s \circ (\uparrow \circ t)\rangle$
FRefMap	$\mathtt{1}\langle M \cdot s\rangle$	$\to$	M
RRefMap	$\mathtt{n+1}\langle M \cdot s\rangle$	$\to$	$\mathtt{n}\langle s\rangle$
FRefArg	$\mathtt{1}\langle M \ / \ s\rangle$	$\to$	$M \mid \mathtt{1}\langle s\rangle$
RRefArg	$\mathtt{n+1}\langle M \ / \ s\rangle$	$\to$	$\mathtt{n+1}\langle s\rangle$
LiftId	$\Uparrow(Id)$	$\to$	Id
MapEnv	$M \cdot s \circ t$	$\to$	$M\langle t\rangle \cdot (s \circ t)$
ArgEnv	$M \ / \ s \circ t$	$\to$	$M\langle t\rangle \ / \ (s \circ t)$
ArgMap	$M \ / \ (L \cdot s)$	$\to$	$(M \mid L) \cdot s$
LiftLift1	$\Uparrow(s) \circ \Uparrow(t)$	$\to$	$\Uparrow(s \circ t)$
LiftLift2	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\to$	$\Uparrow(s \circ t) \circ u$
LiftMap	$\Uparrow(s) \circ M \cdot t$	$\to$	$M\langle t\rangle \cdot (s \circ t)$
LiftArg	$\Uparrow(s) \circ M \ / \ t$	$\to$	$M \ / \ (\Uparrow(s) \circ t)$
ShiftMap	$\uparrow \circ M \cdot s$	$\to$	s
ShiftArg	$\uparrow \circ M \ / \ s$	$\to$	$\uparrow \circ s$
ShiftLift1	$\uparrow \circ \Uparrow(s)$	$\to$	$s \circ \uparrow$
ShiftLift2	$\uparrow \circ (\Uparrow(s) \circ t)$	$\to$	$s \circ (\uparrow \circ t)$
IdL	$Id \circ s$	$\to$	s
IdR	$s \circ Id$	$\to$	s
AssEnv	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$

Table 4: The full $\boldsymbol{\lambda\mu}$**env** rewrite system