From $\lambda\sigma$ to $\lambda\upsilon$
a journey through calculi of explicit substitutions

Pierre LESCANNE
Centre de Recherche en Informatique de Nancy (CNRS)
and INRIA-Lorraine
Campus Scientifique, BP 239,
F54506 Vandœuvre-lès-Nancy, France

email: Pierre.Lescanne@loria.fr

Abstract

This paper gives a systematic description of several calculi of explicit substitutions. These systems are orthogonal and have easy proofs of termination of their substitution calculus. The last system, called $\lambda\upsilon$, entails a very simple environment machine for strong normalization of λ-terms.

1 Introduction

The main mechanism of λ-calculus is β-conversion which is usually defined as $(\lambda x.a)b \to a\{b/x\}$, where $\{b/x\}$ is the substitution of the term b by the variable x. In classical λ-calculus [Bar84] the mechanism of substitution is usually described at a meta-level by a specific and external formalism, unlike λ-calculi of explicit substitutions which contain in the same framework both the β-rule and a description of the evaluation of the substitution. λ-calculi of explicit substitutions are first order term rewrite systems. Such calculi allow nice and uniform descriptions of implementations of λ-calculus. Three main λ-calculi with explicit substitutions have already been proposed: one called the $\lambda\sigma$-calculus by Abadi, Cardelli, Curien and Lévy [ACCL91] is confluent on pure λ-terms. The others called the $\lambda\sigma_{\Uparrow}$-calculus by Hardin and Lévy [HL89, CHL92] and $\lambda\tau$ by Ríos [Río93], are confluent on open λ-terms, i.e., λ-terms containing variables. All these calculi use De Bruijn notations. In this paper we consider the problem essentially from the point of view of first order rewrite systems and as a consequence we systematically build new orthogonal calculi with explicit substitutions where orthogonal means linear with no superposition [HL91]. The approach is systematic in the sense that we try first to introduce as few forms of substitution as possible and second to design the rewrite systems by examining the left-hand sides. Our goal is to reduce every substitution which is amenable to a simpler form and eventually to eliminate them all. These calculi are good candidates for efficient implementations in functional programming languages or as abstract machines. As a result of this quest we get a system we call $\lambda\upsilon$ which is extremely simple. It has a rule *(Beta)* plus seven other structurally simple rules and a somewhat naive proof of termination. We also discuss how confluent calculi, i.e., alternatives to $\lambda\sigma_{\Uparrow}$, can be built and we describe the U-machine which is an environment machine derived from $\lambda\upsilon$. Although done quite independently, this study is parallel to that of Asperti [Asp92] who adopts a categorical point of view. Connections between categorical machines, categorical combinators and our rewrite systems should be deepened.

Copyright 1994 ACM. Appeared in the Proceedings of the 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1994, pp. 60-69.

(Beta)	$(\lambda a)b$	$\to$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\to$	$(a[s])(b[s])$
(Abs)	$(\lambda a)[s]$	$\to$	$\lambda(a[V(1)\cdot(s \circ \uparrow)])$
(Clos)	$a[s][t]$	$\to$	$a[s \circ t]$
(VarId)	$V(1)[id]$	$\to$	$V(1)$
(VarCons)	$V(1)[a \cdot s]$	$\to$	a
(IdL)	$id \circ s$	$\to$	s
(ShiftId)	$\uparrow \circ\, id$	$\to$	$\uparrow$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\to$	s
(AssEnv)	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$
(MapEnv)	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$

Figure 1: The rewrite system $\lambda\sigma$ (Abadi et al.)

2 The problem

In [ACCL91], Abadi, Cardelli, Curien and Lévy propose $\lambda\sigma$ a calculus with explicit substitutions which is confluent on pure λ-terms, i.e., terms without variables of type λ-term. The main idea of $\lambda\sigma$ is a mechanism to denote and to evaluate substitutions. $\lambda\sigma$ uses De Bruijn notations for λ-terms. The reader is supposed to be familiar with them (see [Cur93] for an introduction). For taking substitutions into account the syntax of De Bruijn λ-terms (or λ-terms for short) is slightly modified. A substitution part is added and a calculus with three sorts **Terms**, **Substitutions** and **Naturals** is created. λ-terms have the following grammar:

$$\begin{array}{lcll} \textbf{Terms} & a & ::= & V(\mathtt{n}) \mid ab \mid \lambda a \mid a[s] \\ \textbf{Substitutions} & s & ::= & id \mid \uparrow \mid a \cdot s \mid s \circ t \\ \textbf{Naturals} & \mathtt{n} & ::= & \mathtt{1} \end{array}$$

$a[s]$ is called a *closure* and is the result of applying the substitution s to the term a. Thus every variable is represented by $V(\mathtt{n})$[1] where $\mathtt{n}$ is a positive natural number. In their $\lambda\sigma$-calculus, Abadi et al. propose two constant substitutions, *identity* denoted by id and *shift* denoted by $\uparrow$. $\uparrow$ assigns to the variable $V(\mathtt{i})$ the term $V(\mathrm{S}(\mathtt{i}))$, that is

$$\uparrow:\ V(\mathtt{1}) \mapsto V(2), V(2) \mapsto V(3), \ldots, V(\mathtt{i}) \mapsto V(\mathrm{S}(\mathtt{i})), \ldots$$

Notice that since $V(\mathtt{i})[\uparrow] = V(\mathrm{S}(\mathtt{i}))$ there is no need for natural number notations, in particular S is not necessary. The $\lambda\sigma$-calculus also has a *cons* operation that modifies a substitution as follows. If a is a term and s a substitution, then $a \cdot s$ is the substitution

$$\begin{array}{rl} a \cdot s: & V(\mathtt{1}) \mapsto a \\ & V(2) \mapsto s(V(\mathtt{1})) \\ & \vdots \\ & V(\mathrm{S}(\mathtt{i})) \mapsto s(V(\mathtt{i})) \\ & \vdots \end{array}$$

Substitutions need also to be composed, and so the operator $\circ$ is proposed for this. The rules of $\lambda\sigma$ are given in Figure 1. The system σ, i.e., $\lambda\sigma \setminus \{Beta\}$, is confluent (with 10 convergent critical pairs) and strongly normalizing. However $\lambda\sigma$ is not confluent on open λ-terms, but only on pure closed λ-terms.

A confluent calculus called $\lambda\sigma_{\Uparrow}$ was proposed by Hardin and Lévy [HL89, CHL92] (Figure 2). A new operation on substitutions called *Lift* and denoted by $\Uparrow$ is introduced as

$$\begin{array}{rl} \Uparrow(s): & V(\mathtt{1}) \mapsto V(\mathtt{1}) \\ & V(2) \mapsto s(V(\mathtt{1}))[\uparrow] \\ & \vdots \\ & V(\mathrm{S}(\mathtt{i})) \mapsto s(V(\mathtt{i}))[\uparrow] \\ & \vdots \end{array}$$

$\Uparrow(s)$ can be seen as an abbreviation of $V(\mathtt{1}) \cdot (s \circ \uparrow)$. It makes the rule *Abs* structurally simpler. The grammar of $\lambda\sigma_{\Uparrow}$ is

$$\begin{array}{lcll} \textbf{Terms} & a & ::= & V(\mathtt{n}) \mid ab \mid \lambda a \mid a[s] \\ \textbf{Substitutions} & s & ::= & id \mid \uparrow \mid \Uparrow(s) \mid a \cdot s \mid s \circ t \\ \textbf{Naturals} & \mathtt{n} & ::= & \mathrm{S}(\mathtt{n}) \mid \mathtt{1} \end{array}$$

The sub-system $\sigma_{\Uparrow}$, i.e., $\lambda\sigma_{\Uparrow} \setminus \{Beta\}$ is strongly normalizing. Its proof of termination is an easy game for ORME [Les92] which uses its implementation of elementary interpretations (see below the discussion on the proof of termination of ϕ). Here we use a lexicographical composition of two elementary interpretations (Figure 3). Using this, the proof of confluence of $\sigma_{\Uparrow}$ can be fully mechanized in the completion procedure implemented in ORME [Les90]. 59 critical pairs are computed.

[1] We introduce this V explicitly for at least two reasons. First for proving termination we give an interpretation of this operator: it is therefore useful to write it explicitly. Second, as we will see later, natural numbers will be used elsewhere in the calculus.

(Beta)	$(\lambda a)b$	$\to$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$
(Lambda)	$(\lambda a)[s]$	$\to$	$\lambda(a[\Uparrow(s)])$
(Clos)	$a[s][t]$	$\to$	$a[s \circ t]$
(VarShift1)	$V(\mathtt{n})[\uparrow]$	$\to$	$V(\mathrm{S}(\mathtt{n}))$
(VarShift2)	$V(\mathtt{n})[\uparrow \circ s]$	$\to$	$V(\mathrm{S}(\mathtt{n}))[s]$
(FVarCons)	$V(\mathtt{1})[a \cdot s]$	$\to$	a
(FVarLift1)	$V(\mathtt{1})[\Uparrow(s)]$	$\to$	$V(\mathtt{1})$
(FVarLift2)	$V(\mathtt{1})[\Uparrow(s) \circ t]$	$\to$	$V(\mathtt{1})[t]$
(RVarCons)	$V(\mathrm{S}(\mathtt{n}))[a \cdot s]$	$\to$	$V(\mathtt{n})[s]$
(RVarLift1)	$V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)]$	$\to$	$V(\mathtt{n})[s \circ \uparrow]$
(RVarLift2)	$V(\mathrm{S}(\mathtt{n}))[\Uparrow(s) \circ t]$	$\to$	$V(\mathtt{n})[s \circ (\uparrow \circ t)]$
(AssEnv)	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$
(MapEnv)	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\to$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\to$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\to$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\to$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\to$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\to$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\to$	s
(IdR)	$s \circ id$	$\to$	s
(LiftId)	$\Uparrow(id)$	$\to$	id
(Id)	$a[id]$	$\to$	a

Figure 2: The rewrite system $\lambda\sigma_{\Uparrow}$ (Hardin and Lévy)

3 The $\lambda\phi$-calculus

In this section we start constructing systematically a calculus called ϕ for substitutions. To guide a systematic construction we apply two principles. First, since the goal of a calculus is to build normal forms that do not contain any substitution, ϕ provides a reduction for every term that contains a substitution. This may require introducing new operators, but — this is the second principle — we do this only when necessary, in other words when already introduced operators do not allow us to describe a reduced form. We will see that actually both σ and $\sigma_{\Uparrow}$ contain superfluous operators, namely $\circ$ and $\uparrow$. A systematic successful construction should provide an orthogonal rewrite system, i.e., a linear rewrite system without superposition. This is the case for ϕ. Since a non reduced form is a term that contains a substitution part, we have to consider for reduction only terms of the form $\underline{a}[\underline{s}]$ and provide a reducing rule for each pattern. At first we have four operators for terms, namely *application*, *abstraction*, *closure* and *variable naming*, and two operators for substitutions introduced by the *Beta* rule, namely $\cdot$ and id. We proceed by case. We design left-hand sides by pattern refinement and right-hand sides according to the underlying semantics as described in Section 2 or that given by the system σ.

- $\underline{a}$ is an application. This means that the term has the form $(ab)[s]$, and naturally we get the rule:

$$(\mathrm{App}) \quad (ab)[s] \quad \to \quad (a[s])(b[s]).$$

- $\underline{a}$ is an abstraction. This means that the term has the form $(\lambda a)[s]$. To reduce it one needs to introduce a

(Beta)	$(\lambda a)b$	$\to$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$
(Lambda)	$(\lambda a)[s]$	$\to$	$\lambda(a[\Uparrow(s)])$
(VarId)	$V(\mathtt{n})[id]$	$\to$	$V(\mathtt{n})$
(FVarCons)	$V(\mathtt{1})[a \cdot s]$	$\to$	a
(RVarCons)	$V(\mathrm{S}(\mathtt{n}))[a \cdot s]$	$\to$	$V(\mathtt{n})[s]$
(FVarLift)	$V(\mathtt{1})[\Uparrow(s)]$	$\to$	$V(\mathtt{1})$
(RVarLift)	$V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)]$	$\to$	$V(\mathtt{n})[\Phi(s,\mathtt{1})]$
(VarPhiId)	$V(\mathtt{n})[\Phi(id,\mathtt{p})]$	$\to$	$V(\mathtt{n}+\mathtt{p})$
(VarPhiPhi)	$V(\mathtt{n})[\Phi(\Phi(s,\mathtt{q}),\mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s,\mathtt{q}+\mathtt{p})]$
(FVarPhiCons)	$V(\mathtt{1})[\Phi(a \cdot s,\mathtt{p})]$	$\to$	$a[\Phi(id,\mathtt{p})]$
(RVarPhiCons)	$V(\mathrm{S}(\mathtt{n}))[\Phi(a \cdot s,\mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s,\mathtt{p})]$
(FVarPhiLift)	$V(\mathtt{1})[\Phi(\Uparrow(s),\mathtt{p})]$	$\to$	$V(\mathrm{S}(\mathtt{p}))$
(RVarPhiLift)	$V(\mathrm{S}(\mathtt{n}))[\Phi(\Uparrow(s),\mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s,\mathrm{S}(\mathtt{p}))]$
(Plus1)	$\mathtt{n}+\mathtt{1}$	$\to$	$\mathrm{S}(\mathtt{n})$
(PlusS)	$\mathtt{n}+\mathrm{S}(\mathtt{m})$	$\to$	$\mathrm{S}(\mathtt{n}+\mathtt{m})$

Figure 4: The rewrite system $\lambda\phi$

$[\![V(\mathtt{n})]\!]_1$	$=$	$[\![\mathtt{n}]\!]_1$	$[\![V(\mathtt{n})]\!]_2$	$=$	$[\![\mathtt{n}]\!]_2$
$[\![\mathrm{S}(\mathtt{n})]\!]_1$	$=$	$[\![\mathtt{n}]\!]_1$	$[\![\mathrm{S}(\mathtt{n})]\!]_2$	$=$	$[\![\mathtt{n}]\!]_2+1$
$[\![\mathtt{1}]\!]_1$	$=$	2	$[\![\mathtt{1}]\!]_2$	$=$	2
$[\![ab]\!]_1$	$=$	$[\![a]\!]_1+[\![b]\!]_1$	$[\![ab]\!]_2$	$=$	$[\![a]\!]_2+[\![b]\!]_2+1$
$[\![\lambda a]\!]_1$	$=$	$[\![a]\!]_1+1$	$[\![\lambda a]\!]_2$	$=$	$[\![a]\!]_2$
$[\![a[s]\,]\!]_1$	$=$	$[\![a]\!]_1[\![s]\!]_1$	$[\![a[s]\,]\!]_2$	$=$	$[\![a]\!]_2([\![s]\!]_2+1)$
$[\![a \cdot s]\!]_1$	$=$	$[\![a]\!]_1+[\![s]\!]_1$	$[\![a \cdot s]\!]_2$	$=$	$[\![a]\!]_2+[\![s]\!]_2$
$[\![id]\!]_1$	$=$	2	$[\![id]\!]_2$	$=$	2
$[\![\Uparrow(s)]\!]_1$	$=$	$[\![s]\!]_1$	$[\![\Uparrow(s)]\!]_2$	$=$	$4[\![s]\!]_2$
$[\![\Phi(s,\mathtt{p})]\!]_1$	$=$	$[\![s]\!]_1([\![\mathtt{p}]\!]_1-1)$	$[\![\Phi(s,\mathtt{p})]\!]_2$	$=$	$[\![s]\!]_2([\![\mathtt{p}]\!]_2+1)$
$[\![\mathtt{p}+\mathtt{q}]\!]_1$	$=$	$[\![p]\!]_1+[\![q]\!]_1-2$	$[\![\mathtt{p}+\mathtt{q}]\!]_2$	$=$	$[\![p]\!]_2+2[\![q]\!]_2$

Figure 5: Interpretations for proving the termination of ϕ

(App)	0	$s_2>0$
(Lambda)	$s_1-1>0$	
(VarId)	$n_1>0$	
(FVarCons)	$a_1+2s_1>0$	
(RVarCons)	$n_1a_1>0$	
(FVarLift)	$2s_1-2>0$	
(RVarLift)	0	$n_2s_2+4s_2+1>0$
(VarPhiId)	$2n_1p_1-3n_1-p_1+2>0$	
(VarPhiPhi)	$n_1s_1(p_1-2)(q_1-2)\geq 0$	$n_2s_2p_2(q_2-1)>0$
(FVarPhiCons)	$2s_1(p_1-1)>0$	
(RVarPhiCons)	$n_1a_1(p_1-1)>0$	
(FVarPhiLift)	$2s_1p_1-2s_1-p_1>0$	
(RVarPhiLift)	0	$3n_2s_2p_2+2n_2s_2$ $+4s_2p_2+4s_2+1>0$
(Plus1)	0	$3>0$
(PlusS)	0	$1>0$

Figure 6: Inequalities occurring in the proof of termination of ϕ

$$
\begin{array}{lcl}
[\![V(\mathtt{n})]\!]_1 & = & 2^{[\![\mathtt{n}]\!]_1} \\
[\![\mathrm{S}(\mathtt{n})]\!]_1 & = & [\![\mathtt{n}]\!]_1 + 1 \\
[\![\mathtt{1}]\!]_1 & = & 2 \\
[\![ab]\!]_1 & = & [\![a]\!]_1 + [\![b]\!]_1 \\
[\![\lambda a]\!]_1 & = & [\![a]\!]_1 + 2 \\
[\![a[s]\,]\!]_1 & = & [\![a]\!]_1 [\![s]\!]_1 \\
[\![a \cdot s]\!]_1 & = & [\![a]\!]_1 + [\![s]\!]_1 \\
[\![\uparrow]\!]_1 & = & 2 \\
[\![id]\!]_1 & = & 2 \\
[\![s \circ\ t]\!]_1 & = & [\![s]\!]_1 [\![t]\!]_1 \\
[\![\Uparrow(s)]\!]_1 & = & [\![s]\!]_1 \\
 & & \\
[\![V(\mathtt{n})]\!]_2 & = & [\![\mathtt{n}]\!]_2 \\
[\![\mathrm{S}(\mathtt{n})]\!]_2 & = & [\![\mathtt{n}]\!]_2 \\
[\![\mathtt{1}]\!]_2 & = & 2 \\
[\![ab]\!]_2 & = & [\![a]\!]_2 + [\![b]\!]_2 + 1 \\
[\![\lambda a]\!]_2 & = & 2[\![a]\!]_2 \\
[\![a[s]\,]\!]_2 & = & ([\![a]\!]_2 [\![s]\!]_2) + [\![a]\!]_2 \\
[\![a \cdot s]\!]_2 & = & [\![a]\!]_2 + [\![s]\!]_2 + 1 \\
[\![\uparrow]\!]_2 & = & 2 \\
[\![id]\!]_2 & = & 2 \\
[\![s \circ\ t]\!]_2 & = & ([\![s]\!]_2 [\![t]\!]_2) + [\![s]\!]_2 \\
[\![\Uparrow(s)]\!]_2 & = & 4[\![s]\!]_2
\end{array}
$$

Figure 3: Interpretations for the termination of $\sigma_{\Uparrow}$

new operator that transforms s and produces a new substitution $\Uparrow(s)$ to be put under λ. After Hardin et al., we denote this operator $\Uparrow$. We get their rule:

$$\text{(Lambda)} \quad (\lambda a)[s] \quad \to \quad \lambda(a[\Uparrow(s)]).$$

- $\underline{a}$ is a closure. This means that the term has the form $a[s][t]$. In this case *there is no need for a rule*. Indeed we introduce the induction hypothesis that the system reduces any term with a substitution part, so by induction it must reduce $a[s]$.
- $\underline{a}$ is a variable. This will now constitute the rest of this paragraph.

We consider terms of the form $V(\mathtt{n})[\underline{s}]$ and work by case on patterns for substitutions. Two cases may arise: for a pattern $\underline{s}$ the effect of $\underline{s}$ on $V(\mathtt{n})$ can be described by a unique rule with left-hand side $V(\mathtt{n})[\underline{s}]$ (prefixed by *Var* in $\lambda\sigma_{\Uparrow}$ terminology) or this description requires two rules with left-hand sides $V(\mathtt{1})[\underline{s}]$ (prefixed by *FVar* in $\lambda\sigma_{\Uparrow}$ terminology) and $V(\mathrm{S}(\mathtt{n}))[\underline{s}]$ (prefixed by *RVar* in $\lambda\sigma_{\Uparrow}$ terminology) .

∘ $\underline{s}$ is just id. One rule is enough, which is simply:

$$\text{(VarId)} \quad V(\mathtt{n})[id] \quad \to \quad V(\mathtt{n}).$$

∘ $\underline{s}$ has the form $a \cdot s$. Two rules are necessary namely:

$$
\begin{array}{lrcl}
\text{(FVarCons)} & V(\mathtt{1})[a \cdot s] & \to & a \\
\text{(RVarCons)} & V(\mathrm{S}(\mathtt{n}))[a \cdot s] & \to & V(\mathtt{n})[s].
\end{array}
$$

∘ $\underline{s}$ has the form $\Uparrow(s)$. Two rules are necessary, the first rule is trivially:

$$\text{(FVarLift)} \quad V(\mathtt{1})[\Uparrow(s)] \quad \to \quad V(\mathtt{1}).$$

The second rule cannot be expressed directly, as its right-hand side would be $s \circ \uparrow$ in σ. This requires the introduction of a new operator temporarily called φ. Then we get the rule

$$\text{(RVarLift')} \quad V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)] \quad \to \quad V(\mathtt{n})[\varphi(s)].$$

Let us further consider patterns of the form

$$V(\mathtt{n})[\varphi(\underline{s})]$$

and more specifically a pattern of the form

$$V(\mathtt{n})[\varphi(\varphi(s))]$$

which would represent $V(\mathtt{n})[(s \circ \uparrow) \circ \uparrow]$. Since we have no way to describe the reduction of such a pattern, we decide (eureka!) to represent $s \circ \uparrow^p$ by a unique operator $\Phi(s, \mathtt{p})$ and we forget φ. Asperti [Asp92] denote this $\uparrow^p (s)$ and calls it a shift combinator. We get the rule:

$$\text{(RVarLift)} \quad V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)] \quad \to \quad V(\mathtt{n})[\Phi(s, \mathtt{1})]$$

Now we have to reduce patterns containing Φ.

⋄ $\underline{s}$ has the form $\Phi(\Phi(s,\mathtt{p}),\mathtt{q})$. We get the rule:

(VarPhiPhi)
$$V(\mathtt{n})[\Phi(\Phi(s,\mathtt{p}),\mathtt{q})] \;\to\; V(\mathtt{n})[\Phi(s,\mathtt{p}+\mathtt{q})].$$

This will later requires us to introduce rules for $+$.

⋄ $\underline{s}$ has the form $\Phi(id,\mathtt{p})$. We get the rule:

$$\text{(VarPhiId)} \quad V(\mathtt{n})[\Phi(id,\mathtt{p})] \quad \to \quad V(\mathtt{n}+\mathtt{p}).$$

It is interesting to notice that we exploit the fact that Φ and V use the same naturals, hence the need for a special notation V for variables.

⋄ $\underline{s}$ has the form $\Phi(a \cdot s, \mathtt{p})$. We get the two rules:

(FVarPhiCons)
$$V(\mathtt{1})[\Phi(a \cdot s, \mathtt{p})] \to a[\Phi(id,\mathtt{p})]$$

(RVarPhiCons)
$$V(\mathrm{S}(\mathtt{n}))[\Phi(a \cdot s, \mathtt{p})] \to V(\mathtt{n})[\Phi(s,\mathtt{p})].$$

⋄ $\underline{s}$ has the form $\Phi(\Uparrow(s),\mathtt{p})$. We get the two rules:

(FVarPhiLift)
$$V(\mathtt{1})[\Phi(\Uparrow(s),\mathtt{p})] \to V(\mathrm{S}(\mathtt{p}))$$

(RVarPhiLift)
$$V(\mathrm{S}(\mathtt{n}))[\Phi(\Uparrow(s),\mathtt{p})] \to V(\mathtt{n})[\Phi(s,\mathrm{S}(\mathtt{p}))].$$

All the patterns have been exhausted, but we now need two rules for $+$ in positive natural numbers:

$$
\begin{array}{lrcl}
\text{(Plus1)} & \mathtt{n}+\mathtt{1} & \to & \mathrm{S}(\mathtt{n}) \\
\text{(PlusS)} & \mathtt{n}+S(\mathtt{m}) & \to & S(\mathtt{n}+\mathtt{m}).
\end{array}
$$

Then we get the orthogonal system of fifteen rules for ϕ given in Figure 4. $\lambda\phi$ is the system $\phi \cup \{Beta\}$. The grammar of $\lambda\phi$ is:

$$
\begin{array}{llcl}
\textbf{Terms} & a & ::= & V(\mathtt{n}) \mid ab \mid \lambda a \mid a[s] \\
\textbf{Substitutions} & s & ::= & id \mid a \cdot s \mid\ \Uparrow(s) \mid \Phi(s,\mathtt{n}) \\
\textbf{Naturals} & \mathtt{n} & ::= & \mathrm{S}(\mathtt{n}) \mid \mathtt{1}
\end{array}
$$

Termination of ϕ and confluence of $\lambda\phi$

Before speaking about proofs of termination, let say a few words about the interpretation method. It relies on the naive idea that for proving termination of rewrite systems it is natural to associate a natural number $[\![t]\!]$ with each ground term t and to prove that rewriting always decreases this number. But since the rewrite relation is usually a relation between open terms (terms with variables) the best we can do is to associate with a term $t(x_1,\ldots,x_n)$ a function over the naturals

$$[\![t(x_1,\ldots,x_n)]\!](X_1,\ldots,X_n)$$

that we call an interpretation. Interpretation are extended to terms from interpretations given for basic operators: λ, Φ, id, etc. in our case. Proving that a rewrite system $(l_i \to r_i)$ terminates boils down proving that the function $[\![l_i]\!]$ bounds the function $[\![r_i]\!]$, i.e., for all its values. If we allow any kind of interpretation, this may be a hard problem, but most of the time one restricts the interpretations to be polynomials or polynomials and exponentials (elementary functions). The problem remains undecidable, but heuristics, implemented in ORME, cover a large scope and most of the known examples of proof of termination based on polynomial or elementary interpretations fall in that scope [Les92]. This is the case for $\sigma_{\Uparrow}$. Note that an extension of the interpretation method has been devised by Hans Zantema [Zan93] to provide a proof of termination of σ (a hard problem).

The proof of the termination of ϕ is simple and can be made using the polynomial interpretations of Figure 5 suggested by Paul Zimmermann. We see that for each term t and for $i = 1, 2$, $[\![t]\!]_i > 2$. The array of Figure 6 gives for each rule the sign of the difference between the interpretation of the left-hand side and that of the right-hand side with the conventions $[\![\mathrm{n}]\!]_1 = n_1$, $[\![s]\!]_1 = s_1$, $[\![p]\!]_1 = p_1 \ldots$, $[\![\mathrm{n}]\!]_2 = n_2$, $[\![s]\!]_2 = s_2$, etc.

The proof of the confluence of $\lambda\phi$ on closed terms is exactly like that of $\lambda\sigma$ proposed by Abadi et al. [ACCL91] except that in their Proposition 3.1 one should write $\sigma(a[a_1 \cdot a_2 \cdot \ldots \cdot a_m \cdot \Phi(id, \mathrm{p})])$ instead of $\sigma(a[a_1 \cdot a_2 \cdot \ldots \cdot a_m \cdot \uparrow^{\mathrm{p}})])$.

The $\lambda\bar{\phi}$-calculus

With $\cdot$ and the operator Φ at hand it is possible to get rid of $\Uparrow$. This can be done in the calculus that we call $\bar{\phi}$. In this calculus we rename the rule *Lambda* as *LambdaPhi* and we state it as:

$$(\mathrm{LambdaPhi}) \quad (\lambda a)[s] \quad \to \quad \lambda(a[1 \cdot \Phi(s, 1)])$$

Then rules *App*, *VarId*, *FVarCons*, *RVarCons*, *VarPhiId*, *VarPhiPhi*, *FVarPhiCons*, *RVarPhiCons* are the same and we get the rewrite system $\bar{\phi}$ with only eleven rules which is also orthogonal (Figure 7). The grammar of $\lambda\bar{\phi}$ is:

Terms	a	$::=$	$V(\mathrm{n}) \mid ab \mid \lambda a \mid a[s]$
Substitutions	s	$::=$	$id \mid a \cdot s \mid \Phi(s, \mathrm{n})$
Naturals	n	$::=$	$\mathrm{S}(\mathrm{n}) \mid 1$

Its proof of termination relies on a similar interpretation as ϕ, which is given in Figure 8.

4 The $\lambda\psi$-calculus

Our principle of introducing operators by need was not applied to rule (*Beta*). However Ríos [Río93] has done that in his calculus (Figure 14). Indeed he introduces a new operator he calls / and the rule becomes:

$$(\mathrm{Beta}_\tau) \quad (\lambda a)b \quad \to \quad a[b/]$$

and a systematic construction can again be done. As previously the rule *Lambda* requires the introduction of an operator $\Uparrow$ eliminated by the rules *FVarLift* and *RVarLift* which in turn introduces the operator Φ. Two rules *FVar* and *RVar* eliminate /. The rule *FVarPhiSl* with left-hand side $V(1)[\Phi(a/, \mathrm{p})]$ requires introducing an operator Ψ ($\Psi(\mathrm{p})$ means $\uparrow^p$), which is eliminated by one rule namely:

$$(\mathrm{VarPsi}) \quad V(\mathrm{n})[\Psi(\mathrm{p})] \quad \to \quad V(\mathrm{n}+\mathrm{p}).$$

We still have the rule *FVarPhiLift*, *RVarPhiLift*, *VarPhiPhi* and we need a new rule we call *VarPhiPsi* for eliminating Ψ inside Φ. The grammar of $\lambda\psi$ is:

Terms	a	$::=$	$V(\mathrm{n}) \mid ab \mid \lambda a \mid a[s]$
Substitutions	s	$::=$	$a/ \mid\ \Uparrow(s) \mid \Phi(s, \mathrm{n}) \mid \Psi(n)$
Naturals	n	$::=$	$\mathrm{S}(\mathrm{n}) \mid 1$

For proving the termination we take an interpretation very similar to that of $\lambda\phi$ (Figure 10). We give the same array as for $\lambda\phi$ with only the new rules (Figure 11).

5 The $\lambda\upsilon$-calculus

In $\lambda\psi$, Φ was introduced to eliminate $\uparrow$ in

$$V(\mathrm{S}(\mathrm{n}))[\Uparrow(s)].$$

Six rules were necessary to eliminate Φ, but these rules introduced Ψ and $+$ for which respectively one and two rules were necessary. In this section we propose a very simple calculus that uses a trick in describing the right-hand side of $V(\mathrm{S}(\mathrm{n}))[\Uparrow(s)]$ (rule *RVarLift″*). As before we keep the rules *Beta*$_\tau$, *(App)*, *(Lambda)*, *(FVar)*, *(RVar)* and *(RVarLift)*. For the rule *(RVarLift)* we remember φ and we reintroduce the rule

$$(\mathrm{RVarLift'}) \quad V(\mathrm{S}(\mathrm{n}))[\Uparrow(s)] \quad \to \quad V(\mathrm{n})[\varphi(s)].$$

This leads to the rule

$$(\mathrm{FVarphiSl}) \quad V(1)[\varphi(a/)] \quad \to \quad a[\uparrow]$$

where the newly introduced operator is naturally written $\uparrow$ and called *shift*. *Shift* is naturally associated with the rule:

$$(\mathrm{VarShift}) \quad V(\mathrm{n})[\uparrow] \quad \to \quad V(\mathrm{S}(\mathrm{n})).$$

Here now is the trick. Since $\uparrow$ would be introduced anyway, let us try to minimize the number of operators by replacing *(RVarLift')* by:

$$(\mathrm{RVarLift''}) \quad V(\mathrm{S}(\mathrm{n}))[\Uparrow(s)] \quad \to \quad V(\mathrm{n})[s][\uparrow].$$

We then get the very simple system given in Figure 12 that we call $\lambda\upsilon$ (read lambda-upsilon). It has only 7 + 1 rules, 3 substitution operators and its left-hand sides are elegantly simple. Its proof of termination checked by ORME is given by the elementary interpretations of Figure 13. / can take any interpretation and the second interpretations are only for orienting *(RVarLift")* and *(VarShift)* and therefore need to be known on $\_[\_]$, $\Uparrow$, $\uparrow$, V and S.

(Beta)	$(\lambda a)b$	$\to$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$
(LambdaPhi)	$(\lambda a)[s]$	$\to$	$\lambda(a[V(\mathtt{1}) \cdot \Phi(s, \mathtt{1})])$
(VarId)	$V(\mathtt{n})[id]$	$\to$	$V(\mathtt{n})$
(FVarCons)	$V(\mathtt{1})[a \cdot s]$	$\to$	a
(RVarCons)	$V(\mathrm{S}(\mathtt{n}))[a \cdot s]$	$\to$	$V(\mathtt{n})[s]$
(VarPhiId)	$V(\mathtt{n})[\Phi(id, \mathtt{p})]$	$\to$	$V(\mathtt{n}+\mathtt{p})$
(VarPhiPhi)	$V(\mathtt{n})[\Phi(\Phi(s, \mathtt{q}), \mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s, \mathtt{q}+\mathtt{p})]$
(FVarPhiCons)	$V(\mathtt{1})[\Phi(a \cdot s, \mathtt{p})]$	$\to$	$a[\Phi(id, \mathtt{p})]$
(RVarPhiCons)	$V(\mathrm{S}(\mathtt{n}))[\Phi(a \cdot s, \mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s, \mathtt{p})]$
(Plus1)	$\mathtt{n}+\mathtt{1}$	$\to$	$\mathrm{S}(\mathtt{n})$
(PlusS)	$\mathtt{n}+S(\mathtt{m})$	$\to$	$S(\mathtt{n}+\mathtt{m})$

Figure 7: The rewrite system $\lambda\bar{\phi}$

$[\![V(\mathtt{n})]\!]_1$	$=$	$[\![\mathtt{n}]\!]_1$	$[\![V(\mathtt{n})]\!]_2$	$=$	$[\![\mathtt{n}]\!]_2$
$[\![\mathrm{S}(\mathtt{n})]\!]_1$	$=$	$[\![\mathtt{n}]\!]_1$	$[\![\mathrm{S}(\mathtt{n})]\!]_2$	$=$	$[\![\mathtt{n}]\!]_2+1$
$[\![\mathtt{1}]\!]_1$	$=$	2	$[\![\mathtt{1}]\!]_2$	$=$	2
$[\![ab]\!]_1$	$=$	$[\![a]\!]_1+[\![b]\!]_1$	$[\![ab]\!]_2$	$=$	$[\![a]\!]_2+[\![b]\!]_2+1$
$[\![\lambda a]\!]_1$	$=$	$[\![a]\!]_1+1$	$[\![\lambda a]\!]_2$	$=$	$[\![a]\!]_2$
$[\![a[s]\ ]\!]_1$	$=$	$[\![a]\!]_1[\![s]\!]_1$	$[\![a[s]\ ]\!]_2$	$=$	$[\![a]\!]_2[\![s]\!]_2+1$
$[\![a \cdot s]\!]_1$	$=$	$[\![a]\!]_1+[\![s]\!]_1-2$	$[\![a \cdot s]\!]_2$	$=$	$[\![a]\!]_2+[\![s]\!]_2$
$[\![id]\!]_1$	$=$	2	$[\![id]\!]_2$	$=$	2
$[\![\Uparrow(s)]\!]_1$	$=$	$[\![s]\!]_1$	$[\![\Uparrow(s)]\!]_2$	$=$	$4[\![s]\!]_2$
$[\![\Phi(s,\mathtt{p})]\!]_1$	$=$	$[\![s]\!]_1([\![\mathtt{p}]\!]_1-1)$	$[\![\Phi(s,\mathtt{p})]\!]_2$	$=$	$[\![s]\!]_2([\![\mathtt{p}]\!]_2+1)$
$[\![\mathtt{p}+\mathtt{q}]\!]_1$	$=$	$[\![p]\!]_1+[\![q]\!]_1-2$	$[\![\mathtt{p}+\mathtt{q}]\!]_2$	$=$	$[\![p]\!]_2+2[\![q]\!]_2$

Figure 8: Interpretations for proving the termination of $\bar{\phi}$

(Beta$_\tau$)	$(\lambda a)b$	$\to$	$a[b/]$
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$
(Lambda)	$(\lambda a)[s]$	$\to$	$\lambda(a[\Uparrow(s)])$
(FVar)	$V(\mathtt{1})[a/]$	$\to$	a
(RVar)	$V(\mathrm{S}(\mathtt{n}))[a/]$	$\to$	$V(\mathtt{n})$
(FVarLift)	$V(\mathtt{1})[\Uparrow(s)]$	$\to$	$V(\mathtt{1})$
(RVarLift)	$V(\mathrm{S}(\mathtt{n}))[\Uparrow(s)]$	$\to$	$V(\mathtt{n})[\Phi(s, \mathtt{1})]$
(FVarPhiSl)	$V(\mathtt{1})[\Phi(a/, \mathtt{p})]$	$\to$	$a[\Psi(\mathtt{p})]$
(RVarPhiSl)	$V(\mathrm{S}(\mathtt{n}))[\Phi(a/, \mathtt{p})]$	$\to$	$V(\mathrm{S}(\mathtt{n}+\mathtt{p}))$
(FVarPhiLift)	$V(\mathtt{1})[\Phi(\Uparrow(s), \mathtt{p})]$	$\to$	$V(\mathrm{S}(\mathtt{p}))$
(RVarPhiLift)	$V(\mathrm{S}(\mathtt{n}))[\Phi(\Uparrow(s), \mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s, \mathrm{S}(\mathtt{p}))]$
(VarPhiPhi)	$V(\mathtt{n})[\Phi(\Phi(s, \mathtt{q}), \mathtt{p})]$	$\to$	$V(\mathtt{n})[\Phi(s, \mathtt{q}+\mathtt{p})]$
(VarPhiPsi)	$V(\mathtt{n})[\Phi(\Psi(\mathtt{q}), \mathtt{p})]$	$\to$	$V((\mathtt{n}+\mathtt{q})+\mathtt{p})$
(VarPsi)	$V(\mathtt{n})[\Psi(\mathtt{p})]$	$\to$	$V(\mathtt{n}+\mathtt{p})$
(Plus1)	$\mathtt{n}+\mathtt{1}$	$\to$	$\mathrm{S}(\mathtt{n})$
(PlusS)	$\mathtt{n}+S(\mathtt{m})$	$\to$	$S(\mathtt{n}+\mathtt{m})$

Figure 9: The rewrite system $\lambda\psi$

$$
\begin{array}{lcl}
[\![V(\mathrm{n})]\!]_1 & = & [\![\mathrm{n}]\!]_1 \\
[\![\mathrm{S}(\mathrm{n})]\!]_1 & = & [\![\mathrm{n}]\!]_1 \\
[\![1]\!]_1 & = & 2 \\
[\![ab]\!]_1 & = & [\![a]\!]_1 + [\![b]\!]_1 \\
[\![\lambda a]\!]_1 & = & [\![a]\!]_1 + 1 \\
[\![a[s]\,]\!]_1 & = & [\![a]\!]_1 [\![s]\!]_1 \\
[\![\Uparrow(s)]\!]_1 & = & [\![s]\!]_1 \\
[\![\Phi(s,\mathrm{p})]\!]_1 & = & [\![s]\!]_1([\![\mathrm{p}]\!]_1 - 1) \\
[\![\mathrm{p}+\mathrm{q}]\!]_1 & = & [\![p]\!]_1 + [\![q]\!]_1 - 2 \\
[\![\Psi(\mathrm{p})]\!]_1 & = & [\![\mathrm{p}]\!]_1 \\
[\![a/]\!]_1 & = & [\![a]\!]_1 + 1 \\
\\
[\![V(\mathrm{n})]\!]_2 & = & [\![\mathrm{n}]\!]_2 \\
[\![\mathrm{S}(\mathrm{n})]\!]_2 & = & [\![\mathrm{n}]\!]_2 + 1 \\
[\![1]\!]_2 & = & 2 \\
[\![ab]\!]_2 & = & [\![a]\!]_2 + [\![b]\!]_2 + 1 \\
[\![\lambda a]\!]_2 & = & [\![a]\!]_2 \\
[\![a[s]\,]\!]_2 & = & [\![a]\!]_2([\![s]\!]_2 + 1) \\
[\![\Uparrow(s)]\!]_2 & = & 4[\![s]\!]_2 \\
[\![\Phi(s,\mathrm{p})]\!]_2 & = & [\![s]\!]_2([\![\mathrm{p}]\!]_2 + 1) \\
[\![\mathrm{p}+\mathrm{q}]\!]_2 & = & [\![p]\!]_2 + 2[\![q]\!]_2 \\
[\![\Psi(\mathrm{p})]\!]_2 & = & [\![\mathrm{p}]\!]_2 \\
[\![a/]\!]_2 & = & [\![a]\!]_2
\end{array}
$$

Figure 10: Interpretations for the termination of ψ

(FVar)	$a_1 + 2 > 0$
(RVar)	$n_1 a_1 > 0$
(FVarPhiSl)	$a_1 p_1 + 2p_1 - 2a_1 - 2 > 0$
(RVarPhiSl)	$n_1 a_1 p_1 - n_1 a_1 + n_1 p_1 - 2n_1 - p_1 + 2 > 0$
(VarPhiPsi)	$n_1 q_1 p_1 - n_1 q_1 - n_1 - q_1 - p_1 + 4 > 0$
(VarPsi)	$n_1 p_1 - n_1 - p_1 + 2 > 0$

Figure 11: Inequalities occurring in the proof of termination of ψ

$$
\begin{array}{lrcl}
(\mathrm{Beta}_\tau) & (\lambda a)b & \rightarrow & a[b/] \\
\\
(\mathrm{App}) & (ab)[s] & \rightarrow & a[s]b[s] \\
(\mathrm{Lambda}) & (\lambda a)[s] & \rightarrow & \lambda(a[\Uparrow(s)]) \\
\\
(\mathrm{FVar}) & V(1)[a/] & \rightarrow & a \\
(\mathrm{RVar}) & V(\mathrm{S}(\mathrm{n}))[a/] & \rightarrow & V(\mathrm{n}) \\
\\
(\mathrm{FVarLift}) & V(1)[\Uparrow(s)] & \rightarrow & V(1) \\
(\mathrm{RVarLift}") & V(\mathrm{S}(\mathrm{n}))[\Uparrow(s)] & \rightarrow & V(\mathrm{n})[s][\uparrow] \\
\\
(\mathrm{VarShift}) & V(\mathrm{n})[\uparrow] & \rightarrow & V(\mathrm{S}(\mathrm{n}))
\end{array}
$$

Figure 12: The rewrite system $\lambda\upsilon$

$$
\begin{array}{rcl@{\qquad}rcl}
[\![V(\mathrm{n})]\!]_1 & = & 2^{[\![\mathrm{n}]\!]_1} & [\![V(\mathrm{n})]\!]_2 & = & 2^{[\![\mathrm{n}]\!]_2} \\
[\![\mathrm{S}(\mathrm{n})]\!]_1 & = & [\![\mathrm{n}]\!]_1 + 1 & [\![\mathrm{S}(\mathrm{n})]\!]_2 & = & [\![\mathrm{n}]\!]_2 + 1 \\
[\![1]\!]_1 & = & 2 & & & \\
[\![ab]\!]_1 & = & [\![a]\!]_1 + [\![b]\!]_1 + 1 & & & \\
[\![\lambda a]\!]_1 & = & [\![a]\!]_1 + 1 & & & \\
[\![a[s]\,]\!]_1 & = & [\![a]\!]_1[\![s]\!]_1 & [\![a[s]\,]\!]_2 & = & [\![a]\!]_2[\![s]\!]_2 \\
[\![\Uparrow(s)]\!]_1 & = & [\![s]\!]_1 & [\![\Uparrow(s)]\!]_2 & = & [\![s]\!]_2 + 1 \\
[\![\uparrow]\!]_1 & = & 2 & [\![\uparrow]\!]_2 & = & 2 \\
[\![a/]\!]_1 & = & \mathit{any} & & &
\end{array}
$$

Figure 13: Interpretations for the termination of υ

$$
\begin{array}{lrcl}
(\mathrm{Beta}_\tau) & (\lambda a)b & \rightarrow & a[b/] \\
\\
(\mathrm{App}) & (ab)[s] & \rightarrow & a[s]b[s] \\
(\mathrm{Lambda}) & (\lambda a)[s] & \rightarrow & \lambda(a[\Uparrow(s)]) \\
\\
(\mathrm{Clos}) & a[s][t] & \rightarrow & a[s\circ t] \\
(\mathrm{AssEnv}) & (s\circ t)\circ u & \rightarrow & s\circ(t\circ u) \\
(\mathrm{MapSl}) & a/\circ s & \rightarrow & \Uparrow(s)\circ a[s]/ \\
\\
(\mathrm{FVar}) & V(1)[a/] & \rightarrow & a \\
(\mathrm{FVarLift1}) & V(1)[\Uparrow(s)] & \rightarrow & V(1) \\
(\mathrm{FVarLift2}) & V(1)[\Uparrow(s)\circ t] & \rightarrow & V(1)[t] \\
\\
(\mathrm{Shift}) & \uparrow\circ a/ & \rightarrow & \mathit{id} \\
\\
(\mathrm{ShiftLift1}) & \uparrow\circ\Uparrow(s) & \rightarrow & s\circ\uparrow \\
(\mathrm{ShiftLift2}) & \uparrow\circ(\Uparrow(s)\circ t) & \rightarrow & s\circ(\uparrow\circ t) \\
(\mathrm{Lift1}) & \Uparrow(s)\circ\Uparrow(t) & \rightarrow & \Uparrow(s\circ t) \\
(\mathrm{Lift2}\ ) & \Uparrow(s)\circ(\Uparrow(t)\circ u) & \rightarrow & \Uparrow(s\circ t)\circ u \\
\\
(\mathrm{IdL}) & \mathit{id}\circ s & \rightarrow & s \\
(\mathrm{IdR}) & s\circ\mathit{id} & \rightarrow & s \\
(\mathrm{LiftId}) & \Uparrow(\mathit{id}) & \rightarrow & \mathit{id} \\
(\mathrm{Id}) & a[\mathit{id}] & \rightarrow & a
\end{array}
$$

Figure 14: The rewrite system $\lambda\tau$

The grammar of $\lambda\upsilon$ is:

$$
\begin{array}{llcl}
\textbf{Terms} & a & ::= & V(\mathrm{n}) \mid ab \mid \lambda a \mid a[s] \\
\textbf{Substitutions} & s & ::= & a/ \mid \Uparrow(s) \mid \uparrow \\
\textbf{Naturals} & \mathrm{n} & ::= & \mathrm{S}(\mathrm{n}) \mid 1
\end{array}
$$

Section 7 gives a machine for strong reduction of λ-calculus derived from $\lambda\upsilon$.

6 A systematic construction of confluent calculi

As sketched in [ACCL91] it is also possible to use systematic methods for computing confluent calculi. The main idea is to introduce the rule *(Beta)* and to complete the system in order to make it at most locally confluent. Without interaction with the user, the completion process usually diverges, but based on the intended semantics, there are ways to avoid such a divergence (see [Her88]). Let us give some principles used in the case of λ-calculus.

- Generalize any rule of the form $a[s] \rightarrow a[t]$ or $\Uparrow(s) \rightarrow \Uparrow(t)$ to a rule $s \rightarrow t$, indeed if a substitution has the

(Beta)	$(\lambda a)b$	$\to$	$a[b \cdot id]$
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$
(Lambda)	$(\lambda a)[s]$	$\to$	$\lambda(a[\Uparrow(s)])$
(Clos)	$a[s][t]$	$\to$	$a[s \circ t]$
(AssEnv)	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$
(MapEnv)	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$
(FVarCons)	$V(1)[a \cdot s]$	$\to$	a
(ShiftCons)	$\uparrow \circ (a \cdot s)$	$\to$	s
(FVarLift)	$V(1)[\Uparrow(s)]$	$\to$	$V(1)$
(FVarLift')	$V(1)[\Uparrow(s) \circ t]$	$\to$	$V(1)[t]$
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\to$	$s \circ \uparrow$
(ShiftLift2)	$\uparrow \circ (\Uparrow(s) \circ t)$	$\to$	$s \circ (\uparrow \circ t)$
(VarPhiId)	$V(\mathtt{n})[\Phi(id, \mathtt{p})]$	$\to$	$V(\mathtt{n}+\mathtt{p})$
(VarPhiId')	$V(\mathtt{n})[\Phi(id, \mathtt{p}) \circ t]$	$\to$	$V(\mathtt{n}+\mathtt{p})[t]$
(PhiPhi)	$\Phi(\Phi(s, \mathtt{q}), \mathtt{p})$	$\to$	$\Phi(s, \mathtt{q}+\mathtt{p})$
(FVarPhiCons)	$V(1)[\Phi(a \cdot s, \mathtt{p})]$	$\to$	$a[\Phi(id, \mathtt{p})]$
(FVarPhiCons')	$V(1)[\Phi(a \cdot s, \mathtt{p}) \circ t]$	$\to$	$a[\Phi(id, \mathtt{p}) \circ t]$
(VarPhiCons)	$\uparrow \circ \Phi(a \cdot s, \mathtt{p})$	$\to$	$\Phi(s, \mathtt{p})$
(VarPhiCons')	$\uparrow \circ (\Phi(a \cdot s, \mathtt{p}) \circ t)$	$\to$	$\Phi(s, \mathtt{p}) \circ t$
(FVarPhiLift)	$V(1)[\Phi(\Uparrow(s), \mathtt{p})]$	$\to$	$V(\mathrm{S}(\mathtt{p}))$
(FVarPhiLift')	$V(1)[\Phi(\Uparrow(s), \mathtt{p}) \circ t]$	$\to$	$V(\mathrm{S}(\mathtt{p}))[t]$
(VarPhiLift)	$\uparrow \circ \Phi(\Uparrow(s), \mathtt{p})$	$\to$	$\Phi(s, \mathrm{S}(\mathtt{p}))$
(VarPhiLift')	$\uparrow \circ (\Phi(\Uparrow(s), \mathtt{p}) \circ t)$	$\to$	$\Phi(s, \mathrm{S}(\mathtt{p})) \circ t$
(ShiftPhi)	$\uparrow \circ \Phi(id, \mathtt{p})$	$\to$	$\Phi(id, \mathrm{S}(\mathtt{p}))$
(ShiftPhi')	$\uparrow \circ (\Phi(id, \mathtt{p}) \circ t)$	$\to$	$\Phi(id, \mathrm{S}(\mathtt{p})) \circ t$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\to$	$\Uparrow(s \circ t)$
(Lift2)	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\to$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\to$	$a \cdot (s \circ t)$
(IdL)	$id \circ s$	$\to$	s
(IdR)	$s \circ id$	$\to$	s
(LiftId)	$\Uparrow(id)$	$\to$	id
(Id)	$a[id]$	$\to$	a
(Plus1)	$\mathtt{n}+1$	$\to$	$\mathrm{S}(\mathtt{n})$
(PlusS)	$\mathtt{n}+\mathrm{S}(\mathtt{m})$	$\to$	$\mathrm{S}(\mathtt{n}+\mathtt{m})$
(PlusA)	$\mathtt{n}+(\mathtt{m}+\mathtt{p})$	$\to$	$(\mathtt{n}+\mathtt{m})+\mathtt{p}$

Figure 15: The rewrite system $\lambda\phi_{\Uparrow}$

same effect on any terms we can declare that they are equal and that $\Uparrow$ is one-to-one. Actually due to rules *(App)* and *(Lambda)* we may also generalize rules of the form $V(\mathtt{n})[s] \to V(\mathtt{n})[t]$ to $s \to t$. Similarly, rules of the form $\lambda a \to \lambda b$ are generalized to $a \to b$.

- Divergences are often generated by terms of the form $a[s][t] = b[s'][t']$ when $s \circ t$ or $s' \circ t'$ or both can be reduced by rules *(AssEnv)* and other rules that include $\circ$. Therefore in an attempt to get a confluent system by completion introduce $\circ$ and rules *(Clos)* and *(AssEnv)*.
- Replace variables $V(\mathrm{S}(\mathtt{n}))$ by $V(\mathtt{n})[\uparrow]$.

If we apply this method to λv we get $\lambda\tau$ of Ríos (Figure 14) and if we apply this method to $\lambda\phi$, we get the system $\lambda\phi_{\Uparrow}$ (Figure 15).

7 The U-machine

This journey will be a success if it ends with the construction of an abstract machine for normalizing λ-terms. Actually in λ-calculus there are two kinds of normalizations: weak and strong. Weak normal forms are terms with no β-redex at the root. *Weak normalization* is a process to get weak normal forms, it is typically the reduction in functional programming languages like ML or HASKELL. In addition to the above mentioned β-redexes *strong normalization* allows also reduction of β-redexes occurring under λ, i.e., in the subterm a of a term of the form λa or of β-redexes occurring in the parameter part of a variable function, i.e., in a subterm a_i of a term $V(n)a_1 \ldots a_p$. In a strong normal form there is no β-redex at all. Strong normalization is an interesting tool for manipulating functional programs in particular for simplifying them. It is also used in higher order theorem proving. Strong normalization is usually harder to describe. In our case, it requires recursively creating new instances of the machine. These machines are called to reduce a specific term, say t, in a specific environment, say e. Such calls are written $\mathtt{nf}(a, e)$ in the sequel. The called machines produce results used by the calling machine. To perform a somewhat faithful strong normalization, it is essential to design a machine that does a reduction only when necessary.

For designing our machine that we call the U-machine, we proceeded from the following natural idea. If λv is simple then it should entail a conceptually simple machine for (weak and strong reduction) normalization of λ-calculus. We did not start form scratch since the U-machine has similarities with a former machine due to Krivine and described by P.L. Curien in his book [Cur93] (see also [Cré90]). The U-machine is easy to explain since it sticks closely to λv and relies on normalization in this calculus. Since its basic transitions are strongly related to the rules of λv, we keep the same name for them.[2]

The U-machine is an *environment machine*. It has *states* made of three components: a term, an environment and a stack. Each non-final state is matched by one instruction or transition (the machine is deterministic) and the state is modified according to the right-hand side of the corresponding transition. *Environments* in the U-machine are lists of *actions* to perform on variables. These actions are pairs of the form (c, i), where the *index* i is the number of *Lift*'s to do before performing more elementary actions, and c is either a shift $\uparrow$ or a *closure* that is a pair $\langle a, e\rangle$ of a term and an environment. Closure $\langle a, e\rangle$ corresponds more or less to the substitution $a[s_e]/$ where s_e is associated with the environment e (see the function ξ below). The action $(\langle a, e\rangle, 0)$ on the top of the current environment should be understood either as "Evaluate the term t in the environment e and return it as value" if the term in the state is the variable $V(1)$ or as "Skip this action and decrease the number of this variable" if the term in the state is a variable which is not $V(1)$. This corresponds to rules *(FVar)* and *(RVar)* in λv. In transition *(APP)* applied to term ab the evaluation of the term b in the current environment is delayed. A closure is created and put on a *stack* for further evaluation if necessary. A state of the U-machine has three components and therefore its structure is:

[2] except for the transition *(LBA-BET)* which is a combination of *(Lambda)* and *(Beta$_\tau$)*.

$$
\begin{array}{rcll}
(ab, e, p) & \xrightarrow[U]{} & (a, e, \langle b, e\rangle :: p) & (APP) \\
(\lambda a, e, \langle b, e'\rangle :: p) & \xrightarrow[U]{} & (a, \mathsf{Lift\_env}(e) \oplus [(\langle b, e'\rangle, 0], p) & (LBA - BET) \\
(V(1), (c, i+1) :: e, p) & \xrightarrow[U]{} & (V(1), e, p) & (FVARLIFT) \\
(V(S(n)), (c, i+1) :: e, p) & \xrightarrow[U]{} & (V(n), (c, i) :: (\uparrow, 0) :: e, p) & (RVARLIFT'') \\
(V(1), (\langle a, e\rangle, 0) :: e', p) & \xrightarrow[U]{} & (a, e \oplus e', p) & (FVAR) \\
(V(S(n)), (\langle a, e\rangle, 0) :: e', p) & \xrightarrow[U]{} & (V(n), e', p) & (RVAR) \\
(V(n), (\uparrow, 0) :: e, p) & \xrightarrow[U]{} & (V(S(n)), e, p) & (VARSHIFT)
\end{array}
$$

Figure 16: The U-machine

Name	reference	confluent	orth.	subst. ops.	size
$\lambda\sigma$	[ACCL91]			$id \cdot \uparrow \circ$	10
$\lambda\sigma_{\Uparrow}$	[HL89]	x		$id \cdot \Uparrow \uparrow \circ$	23
$\lambda\phi$	*this paper*		x	$id \cdot \Uparrow \Phi$	15
$\lambda\bar{\phi}$	*this paper*		x	$id \cdot \Phi$	11
$\lambda\psi$	*this paper*		x	$/ \Uparrow \Phi \Psi$	15
$\lambda\upsilon$	*this paper*		x	$/ \Uparrow \uparrow$	7
$\lambda\tau$	[Río93]	?		$/ \; id \Uparrow \uparrow \circ$	17
$\lambda\phi_{\Uparrow}$	*this paper*	?		$id \cdot \Uparrow \uparrow \Phi \circ$	34

Figure 17: Summary of calculi of explicit substitutions

$$
\begin{array}{lcl}
\mathsf{state} & = & \mathsf{term} \times \mathsf{env} \times \mathsf{stack} \\
\mathsf{env} & = & ((\uparrow \cup \mathsf{closure}) \times \mathsf{IN})\ \mathsf{list} \\
\mathsf{closure} & = & \mathsf{term} \times \mathsf{env} \\
\mathsf{stack} & = & \mathsf{closure\ list}
\end{array}
$$

term's are De Bruijn's pure lambda-terms, i.e., have no substitution part and use De Bruijn's notations. Elements of stack are denoted by p (for the French word "pile"). An operation Lift_env transforms environments by adding 1 to every index. It is used in the transition *(LBA-BET)* where one needs to lift a whole environment. It is defined by

$$
\begin{array}{rcl}
\mathsf{Lift\_env}([\,]) & \to & [\,] \\
\mathsf{Lift\_env}((c, i) :: e) & \to & (c, i+1) :: \mathsf{Lift\_env}(e)
\end{array}
$$

The operator $\_ \oplus \_$ appends one environment to another. It is used in transitions *(LBA-BET)* and *(FVAR)*. In a good implementation, both Lift_env and $\_ \oplus \_$ are called by need, that is they are evaluated on just the part of the environment that is necessary for enabling a further transition. The U-machine has seven transitions (see Figure 16). This way, it performs weak normalization. In particular, if $(a, [\,], [\,]) \xrightarrow[U]{*} (b, [\,], [\,])$ and no more transition can apply, then b is the weak normal form of a. Two kinds of state are not reducible by any transition, namely states of the form $(\lambda a, e[\,])$ and states of the form $(V(n), [\,], p)$ Strong normalization reduces those states. For that we also introduce two inference rules that correspond to recursive calls to the machine.

$$
\frac{(a, e, [\,]) \xrightarrow[U]{*} (\lambda b, e', [\,])}{\mathsf{nf}(a, e) \xrightarrow[nf]{} \lambda\, \mathsf{nf}(b, \mathsf{Lift\_env}(e'))} \quad (L)
$$

$$
\frac{(a, e, [\,]) \xrightarrow[U]{*} (V(n), [\,], [\langle b_1, e_1\rangle; \ldots; \langle b_q, e_q\rangle])}{\mathsf{nf}(a, e) \xrightarrow[nf]{} V(n)\ \mathsf{nf}(b_1, e_1) \ldots \mathsf{nf}(b_q, e_q)} \quad (V)
$$

The strong normal form of a term a is the value computed by $\mathsf{nf}(a, [\,])$, i.e., the term we get when all the nf's are eliminated from $\mathsf{nf}(a, [\,])$. The interpretation ξ of a state is

$$\xi(a, e, [\langle b_1, e_1\rangle, \ldots, \langle b_q, e_q\rangle]) \equiv a[e] b_1[e_1] \ldots b_q[e_q]$$

where $a[e] \equiv a[s_1; \ldots; s_q]$ means $a[s_1] \ldots [s_q]$, $a[\langle b, e\rangle, i]$ means $a[\Uparrow^i (b[e]/)]$ and $a[\uparrow, i]$ means $a[\Uparrow^i (\uparrow)]$. The same holds for the b_i's, therefore each state describes a λ-term with substitution of the $\lambda\upsilon$-calculus. A proof of correctness of the U-machine is based on $\lambda\upsilon$ and comes from the facts that

$$
\begin{array}{c}
\xi(a, e, [\langle b_1, e_1\rangle, \ldots, \langle b_q, e_q\rangle]) \\
\xrightarrow[\lambda\upsilon]{*} \\
\xi(a', e', [\langle b'_1, e'_1\rangle, \ldots, \langle b'_r, e'_r\rangle]) \\
\text{iff} \\
(a, e, [\langle b_1, e_1\rangle, \ldots, \langle b_q, e_q\rangle]) \\
\xrightarrow[U]{*} \\
(a', e', [\langle b'_1, e'_1\rangle, \ldots, \langle b'_r, e'_r\rangle])
\end{array}
$$

and

$$\mathsf{nf}(a, [\,]) \xrightarrow[nf]{!} b \quad \text{iff} \quad a \xrightarrow[\lambda\upsilon]{!} b \quad \text{iff} \quad a \xrightarrow[\beta]{!} b$$

8 Conclusion

Although systematic, this paper rests on some intuitions. Basically we can say that the creation of the adequate operators, the design of left-hand sides and their reduction is systematic whereas the design of the right-hand sides (rule $(RVarLift'')$ for instance), the proof of termination and the generalization of operators (from ϕ to Φ for instance) require invention. This paper contains many tables, but we feel that a final one (Figure 17) would be useful to summarize the results obtained. It gives for each system its name, a reference to a paper where it is presented, its confluence on open terms (x means proved, ? means conjectured), its orthogonality, the operators it uses to describe substitutions and its size, i.e., the number of its rules not including *(Beta)*.

Currently we are examining the design of concrete machines for efficient evaluation of λ-calculus, derived from the U-machine, and we are comparing our approach with categorical machines.

Acknowledgment: I would like to thank people who helped me during this research: Paul Zimmermann, Jocelyne Rouyer, Alejandro Ríos, Roberto Amadio, Wayne Snyder, Pierre-Louis Curien, Miki Hermann, Jean-Luc Rémy, Christiane Guyot, the research group EURÉCA in Nancy and the GDR Programmation.

References

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. of Functional Programming*, 1(4):375–416, 1991.

[Asp92] A. Asperti. A categorical understanding of environment machines. *J. of Functional Programming*, 2(1):23–59, January 1992.

[Bar84] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[CHL92] P.-L. Curien, Th. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. RR 1617, INRIA, Rocquencourt, February 1992.

[Cré90] P. Crégut. An abstract machine for the normalization of λ-calculus. In *Proc. Conf. on Lisp and Functional Progamming*, 1990.

[Cur93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Birkhäuser, 1993. 2nd edition.

[Har92] T. Hardin. Eta-conversion for the languages of explicit substitutions. In H. Kirchner and G. Levi, editors, *Proc. 3rd Int. Conf. on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 306–321. Springer-Verlag, September 1992.

[Her88] M. Hermann. Vademecum of divergent term rewriting systems. Research report 88–R–082, Centre de Recherche en Informatique de Nancy, 1988.

[HL89] Th. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, Izu, 1989.

[HL91] G. Huet and J.-J. Lévy. Computations in orthogonal rewriting systems, I. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic*, chapter 11. The MIT Press, 1991.

[Les90] P. Lescanne. Implementation of completion by transition rules + control: ORME. In H. Kirchner and W. Wechler, editors, *Proc. 2nd Int. Conf. on Algebraic and Logic Programming, Nancy (France)*, volume 463 of *Lecture Notes in Computer Science*, pages 262–269. Springer-Verlag, 1990.

[Les92] P. Lescanne. Termination of rewrite systems by elementary interpretations. In H. Kirchner and G. Levi, editors, *Proc. 3rd Int. Conf. on Algebraic and Logic Programming, Volterra (Italy)*, volume 632 of *Lecture Notes in Computer Science*, pages 21–36. Springer-Verlag, September 1992.

[Río93] A. Ríos. *Contributions à l'étude des λ-calculs avec des substitutions explicites.* Thèse de Doctorat d'Université, U. Paris VII, 1993.

[Rit92] E. Ritter. *Categorical Abstract Machines for Higher-Order Typed Lambda Calculi.* PhD thesis, Cambridge U., Trinity College, September 1992.

[Zan93] H. Zantema. Termination of term rewriting by interpretation. In M. Rusinowitch and J.L. Rémy, editors, *Conditional Term Rewriting Systems, proceedings third international workshop CTRS-92*, volume 656 of *Lecture Notes in Computer Science*, pages 155–167. Springer, 1993. Full version appeared as report RUU-CS-92-14, Utrecht University.