$\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation

Zine-El-Abidine BENAISSA, Daniel BRIAUD,
Pierre LESCANNE and Jocelyne ROUYER-DEGLI
Centre de Recherche en Informatique de Nancy (CNRS)
and INRIA-Lorraine
Campus Scientifique, BP 239,
F54506 Vandœuvre-lès-Nancy, France

email: Zine-El-Abidine.Benaissa@loria.fr, Daniel.Briaud@loria.fr
Pierre.Lescanne@loria.fr, Jocelyne.Rouyer@loria.fr

September 27, 1995

Abstract

Explicit substitutions were proposed by Abadi, Cardelli, Curien, Hardin and Lvy to internalise substitutions into λ-calculus and to propose a mechanism for computing on substitutions. $\lambda\upsilon$ is another view of the same concept which aims to explain the process of substitution and to decompose it in small steps. It favours simplicity and preservation of strong normalisation. This way, another important property is missed, namely confluence on open terms. In spirit, $\lambda\upsilon$ is closely related to another calculus of explicit substitutions proposed by de Bruijn and called $C\lambda\xi\phi$. In this paper, we introduce $\lambda\upsilon$, we present $C\lambda\xi\phi$ in the same framework as $\lambda\upsilon$ and we compare both calculi. Moreover, we prove properties of $\lambda\upsilon$; namely $\lambda\upsilon$ correctly implements β reduction, $\lambda\upsilon$ is confluent on closed terms, i.e., on terms of classical λ-calculus and on all terms that are derived from those terms, and finally $\lambda\upsilon$ preserves strong normalisation in the following sense: strongly β normalising terms are strongly $\lambda\upsilon$ normalising.

1 Introduction

The main mechanism of λ-calculus is β-reduction defined as $(\lambda x.a)b \to a[b/x]$, where $[b/x]$ is the substitution of the term b to the variable x. In classical λ-calculus (Barendregt, 1984) the mechanism of substitution is described by a specific and external formalism. This description is part of the *epitheory* (Curry & Feys, 1958) which means it is not integrated into the theory. In the introduction to their book Curry and Feys insist on the importance of substitution in logic in general and especially in the framework of λ-calculus. They write on page 6 of (Curry & Feys, 1958) that the synthetic theory of combinators "gives the ultimate analysis of substitutions in terms of a system of extreme simplicity. The theory of lambda-conversion is intermediate in character between synthetic theories and ordinary logic ... and it has the advantage of departing less radically from our intuition." In other words, they say that λ-calculus treats substitution better than ordinary logic, but not as well as it should and not as well as combinatory logic does, but λ-calculus is closer to our intuition of a function than combinatory logic. λ-calculi of explicit substitutions answer this challenge since they contain in the same framework both a version of the β-rule and a description of the evaluation of the substitution. Thus explicit substitutions fulfill both Curry and Feys's wishes of an internalisation of the substitution mechanism and of a system which does not depart from our intuition. There are two approaches to calculi of explicit substitutions.

De Bruijn's approach which is also ours aims to describe faithfully the mechanism of substitution with the character of "extreme simplicity" advocated by Curry and Feys for combinatory logic. Historically, the first calculus in this family was introduced by de Bruijn (de Bruijn, 1978) under the name $C\lambda\xi\phi$, see also (Kamereddine & Nederpelt, 1993; Rose & Bloo, 1995; Kamareddine & Ríos, 1995). Another calculus belonging to this family, which is extensively studied in this paper was proposed by one of us in (Lescanne, 1994). Those calculi attempt to describe (perhaps naively) the principles of the implementation of λ-calculus. They do not aim at efficiency.

The other approach, which we propose to call the $\lambda\sigma$ family, has been proposed by Abadi, Cardelli, Curien, Hardin, Lvy and Field around 1989 (Abadi *et al.*, 1990; Abadi *et al.*, 1991; Field, 1990; Hardin & Lévy, 1989; Curien *et al.*, 1992; Ríos, 1993). It follows previous research by Curien who proposed in 1983 *categorical combinators* (Curien, 1983; Curien, 1986b; Curien, 1986a) a combinatory logic more intuitive than the classical one. Hardin in 1987 (Hardin, 1987; Hardin, 1989) studied confluence on open terms for that calculus. Categorical combinators are more intuitive in the sense that they are based on λ-calculus, more precisely on λ-calculus with Cartesian products and keep its structure. An important contribution toward explicit substitutions is the $\lambda\rho$ calculus (Curien, 1991) which is a calculus for weak reduction. The calculi of the $\lambda\sigma$ family insist on confluence on open terms, i.e., on terms with variables of sort term and substitution. For that, they introduce a cons operation and a composition of substitutions which plays a central role. Contrary to expectation, Melliès (Melliès, 1995) has shown that those calculi do not preserve strong normalisation. More precisely, he has shown that the simply typed term $\lambda v.(\lambda x.(\lambda y.y)((\lambda z.z)x))((\lambda w.w)v)$ of the classical λ-calculus starts an infinite derivation in the calculus $\lambda\sigma$ of (Abadi *et al.*, 1991), or in the calculus $\lambda\sigma_{\Uparrow}$ of (Hardin & Lévy, 1989). This derivation goes through terms that contain compositions and cons.

2 The $\lambda\upsilon$-calculus

First let us remind the unfamiliar reader with De Bruijn's indices (de Bruijn, 1972). The first idea that comes in mind if one wants to avoid explicit naming of bound variables is to draw pictures. For instance, one replaces the variables by a dummy name like a box $\Box$ and one draws a line between the variable and its binders. In Fig. 1 we have represented a few terms. This is exactly the approach proposed by Bourbaki (Bourbaki, 1954). De Bruijn follows the same idea, for him variables are natural numbers, the *indices*. The index of a variable is the number of λ's one crosses before the λ that binds that variable. For instance in $\lambda x\lambda y\lambda z.x$ the index of the only occurrence of x is $\underline{3}$ and in the notation of λ-terms with indices, x will be replaced by $\underline{3}$. The indices allow us to associate directly a variable (an index) with its binder, therefore there is no need for the name of a variable next to each λ. Thus from a term a one creates an *abstraction* by adding just a λ on the front of a. For instance, $\lambda\underline{1}$ is equivalent to $\lambda x.x$ in the usual λ-calculus, $\lambda\underline{1}(\lambda\underline{1}\ \underline{2})$ is equivalent to $\lambda x.x(\lambda y.y\,x)$ and $\lambda\lambda\lambda\underline{3}$ is equivalent to $\lambda x\lambda y\lambda z.x$.

One main feature of $\lambda\upsilon$ (read *lambda-upsilon*) is that its set of operators is minimal in the sense that it contains only operators that are necessary to describe the substitution calculus. There are four operators on terms namely *abstraction*, *application*, *closure* and *variables*. The three operators on substitutions *slash*, *lift* and *shift* are introduced by need. The operator *closure* $\_[\_]$ introduces *substitutions* into the calculus. $\lambda\upsilon$ uses de Bruijn's indices (de Bruijn, 1972) and we write variables $\underline{1}$, $\underline{2}$,..., $\underline{n}$, $\underline{n+1}$, ... Notice the underlining which creates a variable (an index) out of a natural number. It is a basic operator of the theory and in particular it receives an interpretation in Fig. 3 for proving strong normalisation of υ. A term that does not contain any closure is called a *pure term* and when we want to insist that a term contains a closure we call it *impure*. The terms considered in this

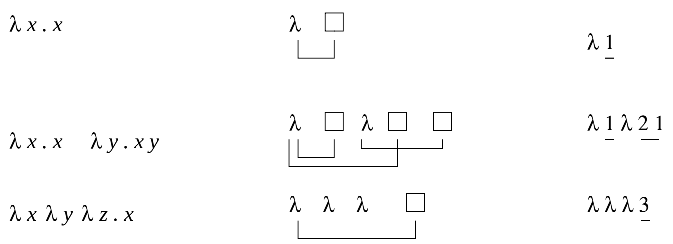

Figure 1: Bourbaki's style notations

paper are closed, which means that they do not contain free variables. In $\lambda\upsilon$, the β-rule is replaced by a more elementary rule. Unlike our predecessors who called a similar rule $(Beta)$, we call that rule (B) in order to avoid confusion with rule β. (B) is

$$(B)\ \ (\lambda a)b \to a[b/]$$

where $b/$ is the substitution with the intuitive meaning:

$$\begin{array}{llll} b/: & \underline{1} & \mapsto & b \\ & \underline{2} & \mapsto & \underline{1} \\ & & \vdots & \\ & \underline{n+1} & \mapsto & \underline{n} \\ & & \vdots & \end{array}$$

This form of (B) was introduced by Ehrhard (Ehrhrard, 1988), but we borrowed it from system τ of Rios (Ríos, 1993). Other rules are given to get rid of substitutions, these rules will form the calculus υ. $\lambda\upsilon$ is the calculus $(B)\cup\upsilon$. The first rule of υ is App, it distributes a substitution into an application (ab).

$$(App)\ \ (ab)[s] \to a[s]b[s].$$

When a substitution goes under a λ it has to be modified, namely

$$(Lambda)\ \ (\lambda a)[s] \to \lambda(a[\Uparrow(s)]).$$

$\Uparrow$ is called *Lift* and has the following intuitive meaning:

$$\begin{array}{llll} \Uparrow(s): & \underline{1} & \mapsto & \underline{1} \\ & \underline{2} & \mapsto & s(\underline{1})[\uparrow] \\ & & \vdots & \\ & \underline{n+1} & \mapsto & s(\underline{n})[\uparrow] \\ & & \vdots & \end{array}$$

$\uparrow$ is a specific substitution that just *shifts* the indices in a term.

$$
\begin{array}{lrcl}
\text{(B)} & (\lambda a)b & \to & a[b/] \\
\text{(App)} & (ab)[s] & \to & a[s]b[s] \\
\text{(Lambda)} & (\lambda a)[s] & \to & \lambda(a[\Uparrow(s)]) \\
\text{(FVar)} & \underline{1}[a/] & \to & a \\
\text{(RVar)} & \underline{n+1}[a/] & \to & \underline{n} \\
\text{(FVarLift)} & \underline{1}[\Uparrow(s)] & \to & \underline{1} \\
\text{(RVarLift)} & \underline{n+1}[\Uparrow(s)] & \to & \underline{n}[s][\uparrow] \\
\text{(VarShift)} & \underline{n}[\uparrow] & \to & \underline{n+1}
\end{array}
$$

Figure 2: The rewrite system $\lambda\upsilon$

$$
\begin{array}{rccc}
\uparrow: & \underline{1} & \mapsto & \underline{2} \\
 & \underline{2} & \mapsto & \underline{3} \\
 & & \vdots & \\
 & \underline{n} & \mapsto & \underline{n+1} \\
 & & \vdots &
\end{array}
$$

The meaning of *Lambda* can be explained as follows. In the expression $(\lambda a)[s]$, s does not affects the $\underline{1}$'s which occur in a. Similarly, in the expression $\lambda(a[\Uparrow(s)])$, $\Uparrow(s)$ should not affect the $\underline{1}$'s which occur in a. On the other hand when $[\Uparrow(s)]$ is applied to other variables, it has to take into account that variables under λ have been renamed and to reset the name of the variables in $s(\underline{n})$ accordingly. This is done by $\uparrow$. Notice that in $\lambda\upsilon$ there is no need for a closure rule, i.e., a rule of the form $a[s][t] \to a[s \circ t]$. Indeed, in a term of the form $a[s][t]$ it is not necessary to tell how t acts on $a[s]$ since by induction one gets rid of s. Now to specify completely the behaviour of substitutions one has just to describe by rewrite rules their action on variables. Putting together all these ideas, we get the rewrite rules of Fig. 2. Notice that the system is essentially *lazy*, in the sense that the evaluation of the substitution $a[b/]$ created by $(\lambda a)b$ can be delayed. The rewrite system υ terminates or is strongly normalising. The proof is easy and can be done with elementary interpretations (functions made of polynomials and exponentials) (Lescanne, 1994; Lescanne, 1992). It is given in Fig. 3. υ is also an *orthogonal* rewrite system, which means that it is left-linear and without superposition. This property is very important both for implementation and proofs, for instance Luc Maranget (private communication) used it to prove termination (or strong normalisation) of υ by structural induction. $\lambda\upsilon$ has three sorts of objects, namely

$$
\begin{array}{lccl}
\textbf{Terms}_\upsilon & a & ::= & \underline{n} \mid aa \mid \lambda a \mid a[s] \\
\textbf{Substitutions}_\upsilon & s & ::= & a/ \mid \Uparrow(s) \mid \uparrow \\
\textbf{Naturals} & n & ::= & n+1 \mid 1.
\end{array}
$$

$\lambda\upsilon$ does not introduce composition of substitutions. This makes the system simpler than those of the σ family. Indeed for presenting a calculus of explicit substitutions, such a composition is not absolutely necessary, at least at the logical level and its introduction in other calculi seems dictated by "efficiency", laziness and code optimisation or partial evaluation, i.e., the ability to improve programs by computing under binders. If new rules dealing with composition need to be introduced, they should be first proved correct as induction theorems and then added to the system. See (Lescanne, 1994) for a discussion on the way to mechanise the introduction of composition and a comparison with other approaches. Among other systems of explicit substitutions, (Lescanne, 1994) introduces $\lambda\upsilon$ but does not prove any of its properties.

The rest of the paper is structured as follows. In Section 3, we prove that $\lambda\upsilon$ correctly implements β-reduction. In Section 4, we prove the confluence of $\lambda\upsilon$. In Section 5, we

$$\begin{array}{|rcl rcl|}
\hline
[\![\underline{n}]\!]_1 & = & 2^{[\![n]\!]_1} & [\![\underline{n}]\!]_2 & = & 2^{[\![n]\!]_2} \\
[\![n+1]\!]_1 & = & [\![n]\!]_1+1 & [\![n+1]\!]_2 & = & [\![n]\!]_2+1 \\
[\![1]\!]_1 & = & 2 & & & \\
[\![ab]\!]_1 & = & [\![a]\!]_1+[\![b]\!]_1+1 & & & \\
[\![\lambda a]\!]_1 & = & [\![a]\!]_1+1 & & & \\
[\![a[s]\ ]\!]_1 & = & [\![a]\!]_1[\![s]\!]_1 & [\![a[s]\ ]\!]_2 & = & [\![a]\!]_2[\![s]\!]_2 \\
[\![\Uparrow(s)]\!]_1 & = & [\![s]\!]_1 & [\![\Uparrow(s)]\!]_2 & = & 2[\![s]\!]_2 \\
[\![\uparrow]\!]_1 & = & 2 & [\![\uparrow]\!]_2 & = & 3 \\
[\![a/]\!]_1 & = & [\![a]\!]_1 & & & \\
\hline
\end{array}$$

Figure 3: Interpretations for proving the termination of υ

prove that $\lambda\upsilon$ preserves strong normalisation. In Section 6 we introduce de Bruijn's calculus $C\lambda\xi\phi$.

3 Soundness of the β-reduction in λυ

We write $\upsilon(a)$ for the normal form of the term a w.r.t. υ. Notice that $\upsilon(a)$ is pure, that is $\upsilon(a)$ contains no closure. β is the classical β-reduction of λ-calculus. It is the relation $a \underset{\beta}{\longrightarrow} b$ between pure terms where $a \underset{B}{\longrightarrow} b'$ and $bpsilon(b')$. This definition is correct. Indeed, let us introduce an external definition of substitution σ_0. The classical definition of β-reduction in terms of this operation (with definitions from (Hardin, 1992)), is

$$(\lambda a)b \underset{\beta}{\longrightarrow} \sigma_0(a,b)$$

where σ_0 is the instance in 0 of a function σ_n defined as follows.

$$\begin{array}{c} \sigma_n(ac,b) = \sigma_n(a,b)\sigma_n(c,b) \\ \sigma_n(\lambda a,b) = \lambda(\sigma_{n+1}(a,b)) \end{array} \qquad \sigma_n(\underline{m},b) = \begin{cases} \underline{m-1} & \text{if } m > n+1 \\ \tau_0^n(b) & \text{if } m = n+1 \\ \underline{m} & \text{if } m \leq n \end{cases}$$

where:

$$\begin{array}{c} \tau_i^n(ab) = \tau_i^n(a)\tau_i^n(b) \\ \tau_i^n(\lambda a) = \lambda(\tau_{i+1}^n(a)) \end{array} \qquad \tau_i^n(\underline{m}) = \begin{cases} \underline{m+n} & \text{if } m > i \\ \underline{m} & \text{if } m \leq i \end{cases}$$

Notice that $\tau_i^n \circ \tau_i^m = \tau_i^{n+m}$ and $\tau_i^0(a) = a$. We define a translation μ that links impure terms with σ_n and τ_n^i.

$$\begin{array}{rcl}
\mu(a[\Uparrow^n(b/)]) & = & \sigma_n(\mu(a),\mu(b)) \\
\mu(a[\Uparrow^n(\uparrow)]) & = & \tau_n^1(\mu(a)) \\
\mu(\underline{n}) & = & \underline{n} \\
\mu(ab) & = & \mu(a)\mu(b) \\
\mu(\lambda a) & = & \lambda(\mu(a))
\end{array}$$

$\Uparrow^n$ is the n^{th} iteration of n, in other words

$$\Uparrow^2(s) \ = \Uparrow(\Uparrow(s)) \qquad \text{and} \qquad \Uparrow^n(s) \ = \Uparrow(\Uparrow(\ldots(\Uparrow(s))\ldots))$$

where $\Uparrow$ is repeated n times. Notice that if a is a pure term, then $\mu(a) = a$, in particular, $\mu(\upsilon(a)) = \upsilon(a)$. The following proposition shows that both definitions coincide.

Proposition 1 *1.* $a \underset{\upsilon}{\longrightarrow} b \ \Rightarrow\ \mu(a) = \mu(b)$, *2.* $\upsilon(a) = \mu(a)$,

3. $\upsilon(a[b/]) = \sigma_0(\mu(a), \mu(b))$,

4. $a \underset{\beta}{\longrightarrow} b$ *if and only if* $a \underset{B}{\longrightarrow} b'$ *and* $b = \upsilon(b')$.

Proof: I□n order to prove the first assertion we consider only rewrites at the root of terms and for this we consider each rule of υ. The result generalises easily by structural induction to any rewrite.

- $(ab)[s] \underset{\upsilon}{\longrightarrow} a[s]b[s]$
  - case $s = \Uparrow^n(c/)$

$$\begin{aligned}\mu((ab)[s]) &= \sigma_n(\mu(ab), \mu(c)) = \sigma_n(\mu(a)\mu(b), \mu(c)) \\ &= \sigma_n(\mu(a), \mu(c))\ \sigma_n(\mu(b), \mu(c)). \\ \mu(a[s]b[s]) &= \mu(a[s])\mu(b[s]) \\ &= \sigma_n(\mu(a), \mu(c))\ \sigma_n(\mu(b), \mu(c)).\end{aligned}$$

  - case $s = \Uparrow^n(\uparrow)$
    $\mu((ab)[s]) = \tau_n^1(\mu(ab)) = \tau_n^1(\mu(a)\ \mu(b)) = \tau_n^1(\mu(a))\ \tau_n^1(\mu(b))$.
    $\mu(a[s]b[s]) = \mu(a[s])\ \mu(b[s]) = \tau_n^1(\mu(a))\ \tau_n^1(\mu(b))$.
- $(\lambda a)[s] \underset{\upsilon}{\longrightarrow} \lambda(a[\Uparrow(s)])$
  - case $s = \Uparrow^n(b/)$
    $\mu((\lambda a)[s]) = \sigma_n(\mu(\lambda a), \mu(b)) = \sigma_n(\lambda\mu(a), \mu(b))$
    $= \lambda\sigma_{n+1}(\mu(a), \mu(b))$.
    $\mu(\lambda(a[\Uparrow(s)])) = \lambda\mu(a[\Uparrow(s)]) = \lambda\sigma_{n+1}(\mu(a), \mu(b))$.
  - case $s = \Uparrow^n(\uparrow)$
    $\mu((\lambda a)[s]) = \tau_n^1(\mu(\lambda a)) = \tau_n^1(\lambda\mu(a)) = \lambda\tau_{n+1}^1(a)$.
    $\mu(\lambda(a[\Uparrow(s)])) = \lambda\mu(a[\Uparrow(s)]) = \lambda\tau_{n+1}^1(a)$.
- $\underline{1}[a/] \underset{\upsilon}{\longrightarrow} a$,
  $\mu(\underline{1}[a/]) = \sigma_0(\mu(\underline{1}), \mu(a)) = \sigma_0(\underline{1}, \mu(a)) = \tau_0^0(\mu(a)) = \mu(a)$.
- $\underline{n+1}[a/] \underset{\upsilon}{\longrightarrow} \underline{n}$
  $\mu(\underline{n+1}[a/]) = \sigma_0(\mu(\underline{n+1}), \mu(a)) = \sigma_0(\underline{n+1}, \mu(a)) = \underline{n} = \mu(\underline{n})$.
- $\underline{1}[\Uparrow(s)] \underset{\upsilon}{\longrightarrow} \underline{1}$
  - case $s = \Uparrow^n(b/)$
    $\mu(\underline{1}[\Uparrow(s)]) = \sigma_{n+1}(\mu(\underline{1}), \mu(b)) = \sigma_{n+1}(\underline{1}, \mu(b)) = \underline{1} = \mu(\underline{1})$
  - case $s = \Uparrow^n(\uparrow)$
    $\mu(\underline{1}[\Uparrow(s)]) = \tau_{n+1}^1(\mu(\underline{1})) = \tau_{n+1}^1(\underline{1}) = \underline{1} = \mu(\underline{1})$.
- $\underline{n+1}[\Uparrow(s)] \underset{\upsilon}{\longrightarrow} \underline{n}[s][\uparrow]$
  - case $s = \Uparrow^k(b/)$
    $\mu(\underline{n+1}[\Uparrow(s)]) = \sigma_{k+1}(\mu(\underline{n+1}), \mu(b)) = \sigma_{k+1}(\underline{n+1}, \mu(b))$. By case,
    - $\sigma_{k+1}(\underline{n+1}, \mu(b)) = \underline{n+1}$
      if $n \le k$,
    - $\sigma_{k+1}(\underline{n+1}, \mu(b)) = \underline{n}$
      if $n > k+1$

* and $\sigma_{k+1}(\underline{n+1}, \mu(b)) = \tau_0^{k+1}(\mu(b))$
  if $n = k+1$.

  $\mu(\underline{n}[s][\uparrow]) = \tau_0^1(\mu(\underline{n}[\Uparrow^k(b/)])) = \tau_0^1(\sigma_k(\mu(\underline{n}), \mu(b)))$
  $= \tau_0^1(\sigma_k(\underline{n}, \mu(b)))$. By case,

  * $\tau_0^1(\sigma_k(\mu(\underline{n}), \mu(b))) = \tau_0^1(\sigma_k(\underline{n}, \mu(b))) = \tau_0^1(\underline{n}) = \underline{n+1}$
    if $n \leq k$,
  * $\tau_0^1(\sigma_k(\mu(\underline{n}), \mu(b))) = \tau_0^1(\sigma_k(\underline{n}, \mu(b))) = \tau_0^1(\underline{n-1}) = \underline{n}$
    if $n > k+1$
  * $\tau_0^1(\sigma_k(\mu(\underline{n}), \mu(b))) = \tau_0^1(\sigma_k(\underline{n}, \mu(b)))$
    $= \tau_0^1(\tau_0^k(\mu(b))) = \tau_0^{k+1}(\mu(b))$
    if $n = k+1$, from $\tau_i^n \circ \tau_i^m = \tau_i^{n+m}$.

- case $s = \Uparrow^k(\uparrow)$
  $\mu(\underline{n+1}[\Uparrow(s)]) = \tau_{k+1}^1(\mu(\underline{n+1})) = \tau_{k+1}^1(\underline{n+1})$.
  Thus

  * $\underline{n+2}$ if $n > k$
  * $\underline{n+1}$ if $n \leq k$.

  $\mu(\underline{n}[s][\uparrow]) = \tau_0^1(\mu(\underline{n}[\Uparrow^k(\uparrow)])) = \tau_0^1\tau_k^1(\mu(\underline{n})) = \tau_0^1\tau_k^1(\underline{n})$.
  Therefore

  * $\tau_0^1(\underline{n+1}) = \underline{n+2}$ if $n > k$
  * and $\tau_0^1(\underline{n}) = \underline{n+1}$ if $n \leq k$.

- $\underline{n}[\uparrow] \underset{\upsilon}{\longrightarrow} \underline{n+1}$
  $\mu(\underline{n}[\uparrow]) = \tau_0^1(\mu(\underline{n})) = \tau_0^1(\underline{n}) = \underline{n+1} = \mu(\underline{n+1})$.

$a \overset{*}{\underset{\upsilon}{\longrightarrow}} b$ implies $\mu(a) = \mu(b)$ is proved by induction on the length of the υ derivation from a to b. This implies 2. The proof of 3 comes from $\sigma_0(\mu(a), \mu(b)) = \mu(a[b/])$, by definition of μ. $\mu(a[b/]) = \upsilon(a[b/])$ is an instance of 1. 4 is obtained from 3 by induction on the structure of a.

4 Confluence of $\lambda\upsilon$ on Terms$_\upsilon$

A key point for the confluence of β reduction in classical λ-calculus is the substitution lemma. It expresses the fact that the following β-contractions are confluent:

$$
\begin{array}{lclcl}
(\lambda x.(\lambda y.M)N)L & \underset{\beta}{\longrightarrow} & (\lambda x.M[y:=N])L & \underset{\beta}{\longrightarrow} & M[y:=N][x:=L] \\
(\lambda x.(\lambda y.M)N)L & \underset{\beta}{\longrightarrow} & (\lambda y.M[x:=L])N[x:=L] & \underset{\beta}{\longrightarrow} & M[x:=L][y:=N[x:=L]]
\end{array}
$$

Indeed, if $y \notin FreeVar(L)$, we have (Barendregt, 1984) Lemma 2.1.16:

$$M[x:=N][y:=L] \equiv M[y:=L][x:=N[y:=L]]$$

We find a similar situation in $\lambda\upsilon$. Indeed, observe that $\lambda\upsilon$ has a sole critical pair, obtained by the superposition of rule B over rule App:

$$((\lambda a)b)[s] \underset{B}{\longrightarrow} a[b/][s]$$

$$((\lambda a)b)[s] \overset{*}{\underset{\upsilon}{\longrightarrow}} (\lambda a[\Uparrow(s)])b[s] \underset{B}{\longrightarrow} a[\Uparrow(s)][b[s]/]$$

Thus, in order to get local confluence, we need to prove that $a[b/][s]$ is υ-convertible to $a[\Uparrow(s)][b[s]/]$, which we also call substitution lemma:

$$a[b/][s] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow(s)][b[s]/]$$

Lemmas 1 to 6 do the job (see a full proof of these lemmas in (Lescanne & Rouyer-Degli, 1994)). Notice that in the following we prove the substitution lemma only for pure terms, but the result remains true for impure terms since if a is impure

$$a[b/][s] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} \upsilon(a)[b/][s] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} \upsilon(a)[\Uparrow(s)][b[s]/] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow(s)][b[s]/].$$

The same lifting from pure terms to impure terms is true for each lemma in this section.

Lemma 1 *For $n \geq 1$ and $i \geq 0$, $\underline{n}[\Uparrow^{n+i}(s)] \overset{*}{\underset{\upsilon}{\longrightarrow}} \underline{n}$.*

For readability, we use the following abbreviation

$$a[\![\uparrow^i]\!] \equiv a[\uparrow] \ldots i_{\text{ times}} \ldots [\uparrow].$$

Obviously,

$$\underline{n}[\![\uparrow^i]\!] \overset{*}{\underset{\upsilon}{\longrightarrow}} \underline{n+i}.$$

Lemma 2 *For $n \geq 1$ and $i \geq 0$, $\underline{n+i}[\Uparrow^i(s)] \overset{*}{\underset{\upsilon}{\longrightarrow}} \underline{n}[s][\![\uparrow^i]\!]$.*

Corollary 1 *For $n > i \geq 0$, $\underline{n}[\Uparrow^i(\uparrow)] \overset{*}{\underset{\upsilon}{\longrightarrow}} \underline{n+1}$.*

Lemma 3 *For $i \geq 0$, $a[\Uparrow^i(\uparrow)][\Uparrow^i(b/)] \overset{*}{\underset{\upsilon}{\longrightarrow}} a$.*

Lemma 4 *For all $j \geq i \geq 0$, $a[\Uparrow^i(\uparrow)][\Uparrow^{j+1}(\uparrow)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow^j(\uparrow)][\Uparrow^i(\uparrow)]$.*

Corollary 2 *$a[\uparrow][\Uparrow^{i+1}(\uparrow)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow^i(\uparrow)][\uparrow]$, when $i = 0$ $a[\uparrow][\Uparrow(\uparrow)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\uparrow][\uparrow]$.*

Corollary 3 *For $i \geq 0$, $a[\![\uparrow^i]\!][\Uparrow^i(\uparrow)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\![\uparrow^{i+1}]\!]$.*

Lemma 5 *$a[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow^i(s)][\Uparrow^i(\uparrow)]$.*

Lemma 5 has an important corollary.

Corollary 4 *$a[\uparrow][\Uparrow(s)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[s][\uparrow]$.*

Because of its corollary, the next lemma is the key of the confluence of $\lambda\upsilon$.

Lemma 6 *$a[\Uparrow^{i+1}(s)][\Uparrow^i(b[s]/)] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow^i(b/)][\Uparrow^i(s)]$.*

Corollary 5 (Substitution Lemma) *$a[b/][s] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow(s)][b[s]/]$.*

For its use in the next lemma, the Substitution Lemma has to be iterated.

Corollary 6 *$a[b/][s_1] \ldots [s_p] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow(s_1)] \ldots [\Uparrow(s_p)][b[s_1] \ldots [s_p]/]$.*

Lemma 7 (Projection Lemma) *If $a \underset{B}{\longrightarrow} b$ then $\upsilon(a) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon(b)$. If $s \underset{B}{\longrightarrow} t$ then $\upsilon(s) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon(t)$.*

Proof: T□he second statement comes from the fact that if $s \underset{B}{\longrightarrow} t$, then $s = \Uparrow^i(a/)$ and $t = \Uparrow^i(b/)$ with $a \underset{B}{\longrightarrow} b$ and $\upsilon(s) = \Uparrow^i(\upsilon(a)/)$ and $\upsilon(t) = \Uparrow^i(\upsilon(b)/)$. Hence from the first statement $\upsilon(a) \underset{\beta}{\longrightarrow} \upsilon(b)$ and $\upsilon(s) \underset{\beta}{\longrightarrow} \upsilon(t)$. Therefore we prove the statement for a and b. The ordering based on interpretations presented in Fig. 3 is a simplification ordering, which means that it contains the subterm ordering (written $\sqsupset$ here). In the sequel we proceed by noetherian induction on this ordering. Therefore if $a \underset{\upsilon}{\longrightarrow} b$ or if b is a subterm of a i.e., $a \sqsupset b$, then b is less than a for the interpretation ordering and we can assume the induction hypothesis on b. We distinguish cases according to the structure of a.

- If $a = a_1 a_2$ is an application and if the B-redex is in a_1, since $a_1 a_2 \sqsupset a_1$ and $a_1 \xrightarrow[B]{} b_1$ by induction one gets $\upsilon(a_1) \xrightarrow[\beta]{*} \upsilon(b_1)$ and

$$\upsilon(a_1 a_2) = \upsilon(a_1)\upsilon(a_2) \xrightarrow[\beta]{*} \upsilon(b_1)\upsilon(a_2) = \upsilon(b_1 a_2).$$

  We proceed likewise if the B-redex is in a_2 or if $a = \lambda a_1$.

- If the B-redex is $a = (\lambda a_1)a_2$ then $b = a_1[a_2/]$ and $\upsilon(a) = (\lambda\upsilon(a_1))\upsilon(a_2)$. By definition of β, one has

$$\upsilon(a) \xrightarrow[\beta]{} \upsilon(\upsilon(a_1)[\upsilon(a_2)/]) = \upsilon(b).$$

- If a is a closure then $a = a'[s_1]\dots[s_p]$ and $b = b'[t_1]\dots[t_p]$.
  - $a = (a_1\ a_2)[s_1]\dots[s_p]$ and $b = (b_1\ a_2)[s_1]\dots[s_p]$. The B redex occurs inside a_1 with $a_1 \xrightarrow[B]{} b_1$ then $a_1[s_1]\dots[s_p] \xrightarrow[B]{} b_1[s_1]\dots[s_p]$, and as

$$(a_1\ a_2)[s_1]\dots[s_p] \xrightarrow[\upsilon]{} a_1[s_1]\dots[s_p]\ a_2[s_1]\dots[s_p] \sqsupset a_1[s_1]\dots[s_p],$$

    by induction

$$\upsilon(a_1[s_1]\dots[s_p]) \xrightarrow[\beta]{*} \upsilon(b_1[s_1]\dots[s_p])$$

    and

$$\begin{aligned}\upsilon((a_1\, a_2)[s_1]\dots[s_p]) &= \upsilon(a_1[s_1]\dots[s_p])\ \upsilon(a_2[s_1]\dots[s_p])\\ &\xrightarrow[\beta]{*} \upsilon(b_1[s_1]\dots[s_p])\ \upsilon(a_2[s_1]\dots[s_p]) = \upsilon((b_1 a_2)[s_1]\dots[s_p]),\end{aligned}$$

    and the same if the B rewrite takes place inside a_2 or inside a s_i.
  - $a = ((\lambda a_3)a_2)[s_1]\dots[s_p]$ and $b = a_3[a_2/][s_1]\dots[s_p]$.

$$\begin{aligned}\upsilon(a) &= \upsilon(\lambda(\upsilon(a_3[\Uparrow(s_1)]\dots[\Uparrow(s_p)]))\ \upsilon(a_2[s_1]\dots[s_p]))\\ &\xrightarrow[\beta]{} \upsilon(\upsilon(a_3[\Uparrow(s_1)]\dots[\Uparrow(s_p)])[\upsilon(a_2[s_1]\dots[s_p])/])\\ &= \upsilon(a_3[\Uparrow(s_1)]\dots[\Uparrow(s_p)][a_2[s_1]\dots[s_p]/])\end{aligned}$$

    and by corollary 6,

$$= \upsilon(a_3[a_2/][s_1]\dots[s_p]) = \upsilon(b).$$

  - $a = (\lambda a_1)[s_1]\dots[s_p]$. If $a_1 \xrightarrow[B]{} b_1$ or $s_i \xrightarrow[B]{} t_i$,

$$\lambda(a_1[\Uparrow(s_1)]\dots[\Uparrow(s_p)]) \xrightarrow[B]{} \lambda(b_1[\Uparrow(t_1)]\dots[\Uparrow(t_p)])$$

    and we can apply the induction hypothesis.
  - $a = \underline{n}[s_1]\dots[s_p]$. The B redex is inside a s_i with $s_i = \Uparrow^{j_i}(c_i/)$. If $i > 1$, then $\underline{n}[s_1] \xrightarrow[\upsilon]{+} a_1$ where a_1 is not a closure and

$$a_1[s_2]\dots[s_p] \xrightarrow[B]{} a_1[t_2]\dots[t_p]$$

    where all the t_j are equal to s_j except t_i which is $\Uparrow^{j_i}(d_i/)$ with $c_i \xrightarrow[B]{} d_i$. The result comes by induction. If the B redex is inside s_1,

$$s_1 = \Uparrow^{j_1}(c_1/) \xrightarrow[B]{} t_1 = \Uparrow^{j_1}(d_1/).$$

    By case, one gets:

* $n=1$ and $j_1=0$,

$$\underline{1}[c_1/][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} c_1[s_2]\dots[s_p] \underset{B}{\longrightarrow} d_1[s_2]\dots[s_p]$$

$$\underline{1}[d_1/][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} d_1[s_2]\dots[s_p]$$

and the result comes by induction.

* $n=k+1$ and $j_1=0$,

$$\underline{k+1}[c_1/][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{k}[s_2]\dots[s_p]$$

$$\underline{k+1}[d_1/][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{k}[s_2]\dots[s_p]$$

and the result is immediate.

* $n=1$ and $j_1=j+1$,

$$\underline{1}[\Uparrow^{j+1}(c_1/)][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{1}[s_2]\dots[s_p]$$

$$\underline{1}[\Uparrow^{j+1}(d_1/)][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{1}[s_2]\dots[s_p]$$

and like above the result is immediate.

* $n=k+1$ and $j_1=j+1$.

$$\underline{k+1}[\Uparrow^{j+1}(c_1/)][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{k}[\Uparrow^{j}(c_1/)][\uparrow][s_2]\dots[s_p]$$

$$\underline{k+1}[\Uparrow^{j+1}(d_1/)][s_2]\dots[s_p] \underset{\upsilon}{\longrightarrow} \underline{k}[\Uparrow^{j}(d_1/)][\uparrow][s_2]\dots[s_p]$$

and the result comes by induction.

Theorem 1 (Confluence Theorem) $\lambda\upsilon$ *is confluent on* $\mathbf{Terms}_\upsilon$.

Proof: T□he proof of the theorem resembles the proof of a similar theorem by Abadi et al. in (Abadi *et al.*, 1991) which in turn was based on Hardin's interpretation method (Hardin, 1989) with modifications due to the change of substitution calculus from σ to υ. It relies on the Projection Lemma.

5 $\lambda\upsilon$ preserves strong β normalisation

The essential difference between $\underset{\beta}{\longrightarrow}$ on one hand and, $\underset{\lambda\upsilon}{\longrightarrow}$ and $\underset{\lambda\sigma}{\longrightarrow}$ on the other hand is that β rewrites with B and then normalises with υ or σ in order to remove all the closures, whereas $\lambda\upsilon$ or $\lambda\sigma$ rewrite also with B but perform or postpone reductions of closures created by B. This raises the following question. Are strongly β normalising λ terms strongly $\lambda\upsilon$ normalising or strongly $\lambda\sigma$ normalising? The answer is "yes" for $\lambda\upsilon$ whereas Melliès gave a negative answer for $\lambda\sigma$. There are strongly β normalising λ terms, even simply typed λ terms, which are not strongly $\lambda\sigma$ normalising. The difficulty is that it could happen that $a \underset{B}{\longrightarrow} b$ and $\upsilon(a)=\upsilon(b)$. In that case, the reduced B-redex of a lies in the substitution part of a subterm which is a closure. That closure is eliminated by rule *Rvar* or rule $FVarLift$ which are the only rules of υ that can delete a B-redex[1]. Thus in the projection lemma, it could be the case that we perform a B-reduction that does not correspond to a β-reduction on the υ normal form, we could therefore make more (but not infinitely many more) B reductions than β reductions. The key of the proof of preservation of strong normalisation is the fact that, in $\lambda\upsilon$, closures can only be created by B unlike $\lambda\sigma$ where closures are also created by Map

$$(a\cdot s)\circ t \;\rightarrow\; a[t]\cdot(s\circ t).$$

Therefore, given a closure the B rewrite that creates it can always be traced back. This will be expressed more formally through lemmas 8 and 9. First let us recall the reader what we call a *position* in a term. Although it has been understood in what precedes, it plays a main role in the following proofs and has to be made precise.

[1] *App* also deletes B-redexes, but *Lambda* enables them immediately.

5.1 Tracing the creation of closures

Definition 1 (Position) *A position in a* λ *term* t *is a sequence of numbers* 1 *or* 2*, such that*

- $t_{|\varepsilon} = t$
- *If* $t_{|p} = a[s]$*, then* $t_{|p1} = a$ *and* $t_{|p2} = s$*.*
- *If* $t_{|p} = \lambda(a)$*, then* $t_{|p1} = a$*.*
- *If* $t_{|p} = a_1 a_2$*, then* $t_{|p1} = a_1$ *and* $t_{|p2} = a_2$*.*
- *If* $t_{|p} = b/$*, then* $t_{|p1} = b$*.*
- *If* $t_{|p} = \Uparrow(s)$*, then* $t_{|p1} = s$*.*

$t_{|p}$ is called the subterm of t at position p or the occurrence at position p (see (Dershowitz & Jouannaud, 1990) p. 250). Positions are compared by the prefix order. p is a prefix of q, if there exists p' such that $p\,p' = q$.

Definition 2 (Replacement) *The term* $t\{u\}_p$ *obtained by replacing the subterm of* t *at position* p *by* u *is the term written* $t\{u\}_p$ *and defined by*

- $(t\{u\}_p)_{|pp'} = u_{|p'}$*,*
- $(t\{u\}_p)_{|p'} = t_{|p'}\{u\}_{p''}$ *if* $p = p'p''$*.*
- $(t\{u\}_p)_{|q} = t_{|q}$ *if* p *and* q *are disjoint, i.e.,* q *is none of the above cases.*

We use the non classic notation $t\{u\}_p$ for the classic notation $t[u/p]$ to avoid confusion with $t[u/]$. Rewriting the term t at the position p by the rule B into the term t' means that there exists a substitution (in the usual sense) f such that $t_{|p} = \mathsf{f}((\lambda a)b)$ and $t' = t\{\mathsf{f}(a[b/])\}_p$ which we write $t \underset{B,p}{\longrightarrow} t'$. One can similarly define rewrites at p for other rules of $\lambda\upsilon$.

Lemma 8 *Let* $a, b \in \mathbf{Terms}_\upsilon$ *such that* $a \underset{\lambda\upsilon}{\longrightarrow} b = t\{d[\Uparrow^i(e/)]\}_p$*. Then,*

1. *either* $a = t'\{d'[\Uparrow^j(e'/)]\}_{p'}$ *and* ($e' \underset{\lambda\upsilon}{\longrightarrow} e$ *or* $e' = e$*),*
2. *or* $a = t\{(\lambda d)e\}_p$*.*

Proof: A□s a rewrites to b, $a = u\{l\}_q$ and $b = u\{r\}_q$, with l, a $\lambda\upsilon$-redex, and r the corresponding $\lambda\upsilon$-reduct. We proceed by a case analysis based on the relative positions of $d[\Uparrow^i(e/)]$ and of the reduct r. Both are subterms of b, namely $b_{|p} = d[\Uparrow^i(e/)]$, $b_{|q} = r$.

1. p, q are disjoint positions. By definition of rewriting $a_{|p'} = b_{|p'}$ for each position p' disjoint of q. Therefore: $a_{|p} = d[\Uparrow^i(e/)]$.
2. p, q are not disjoint. $d[\Uparrow^i(e/)]$ is a subterm of r or vice-versa.
   (a) r is a strict subterm of $d[\Uparrow^i(e/)]$. As $\lambda\upsilon$ only rewrites terms of sort $\mathbf{Terms}_\upsilon$, the reduct is either in d or in e. Thus $a = t\{d'[\Uparrow^i(e'/)]\}_p$ with ($d' \underset{\lambda\upsilon}{\longrightarrow} d$ and $e' = e$) or ($e' \underset{\lambda\upsilon}{\longrightarrow} e$ and $d' = d$).
   (b) $d[\Uparrow^i(e/)]$ is a subterm of r. In that case, the $\lambda\upsilon$-rewrite produces a reduct which contains $d[\Uparrow^i(e/)]$. Hence, a subterm g of the right hand side of the $\lambda\upsilon$-rule matches $d[\Uparrow^i(e/)]$ itself or matches a term of the form $w\{d[\Uparrow^i(e/)]\}$ which contains $d[\Uparrow^i(e/)]$. If g is a variable, then g occurs in the left hand side and the result follows. Else, g has to be a closure $g = f[s]$ and matches $d[\Uparrow^i(e/)]$. One of the following rules has been used:

- (App). This means

$$b = u\{d'[\Uparrow^i(e/)]d[\Uparrow^i(e/)]\}_q$$

or

$$b = u\{d[\Uparrow^i(e/)]d'[\Uparrow^i(e/)]\}_q.$$

In the first case, this implies $a = u\{(d'd)[\Uparrow^i(e/)]\}_q$. The other case is similar.
- (Lambda). This means : $b = u\{\lambda(d[\Uparrow^{j+1}(e/)])\}_q$ with $i = j+1$. This implies : $a = u\{(\lambda d)[\Uparrow^j(e/)]\}_q$.
- (B). This means : $b = t\{d[e/]\}_p$ with $i = 0$. Then $a = t\{(\lambda d)e\}_p$.
- (RVarLift). This means : $b = u\{\underline{n}[\Uparrow^i(e/)][\uparrow]\}_q$ and implies : $a = u\{\underline{n+1}[\Uparrow^{i+1}(e/)]\}_q$.

Lemma 9 *Let $a_1,\ldots,a_n \in$ **Terms**$_\upsilon$ such that $a_i \underset{\lambda\upsilon}{\longrightarrow} a_{i+1}$, $1 \le i \le n-1$, and $a_n = t\{d[\Uparrow^i(e/)]\}_p$. Then,*

1. *either there is an i such that $a_i = t'\{(\lambda d')e'\}_{p'}$ and $e' \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} e$,*
2. *or $a_1 = t'\{d'[\Uparrow^j(e'/)]\}_{p'}$ and $e' \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} e$.*

Proof: B□y induction on n. The basic case $n = 1$ is immediate. Suppose :

$$a_1 \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} a_n \underset{\lambda\upsilon}{\longrightarrow} a_{n+1} = t\{d[\Uparrow^i(e/)]\}_p$$

By previous lemma, either $a_n = t\{(\lambda d)e\}_p$ and $i = n$, or $a_n = t'\{d'[\Uparrow^j(e'/)]\}_{p'}$ with $e' \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} e$ and we apply the induction hypothesis.

5.2 Commutation of external positions

Definition 3 (External position) *The set $Ext(a)$ of* external positions *of a term a is the set defined as:*

$$\begin{array}{rcl} Ext(ab) & = & 1Ext(a) \cup 2Ext(b) \cup \{\varepsilon\} \\ Ext(\lambda a) & = & 1Ext(a) \cup \{\varepsilon\} \\ Ext(a[s]) & = & 1Ext(a) \cup \{\varepsilon\} \\ Ext(\underline{n}) & = & \{\varepsilon\} \end{array}$$

Intuitively external positions are those under no brackets, i.e., in no substitution part of any closure. A rewrite which takes place at an external position is said *external*, otherwise it is said *internal*. If one wants to make precise that a rewrite $\underset{\lambda\upsilon}{\longrightarrow}$ is external (resp. internal) one writes $\underset{\lambda\upsilon}{\longrightarrow}^{ext}$ (resp. $\underset{\lambda\upsilon}{\longrightarrow}^{int}$).

Lemma 10 *If $p \in Ext(a)$ and if $a \underset{B,p}{\longrightarrow} b$, then $\upsilon(a) \underset{\beta}{\longrightarrow} \upsilon(b)$. In particular, if $\upsilon(a)$ is strongly β normalising, $\upsilon(a) \neq \upsilon(b)$.*

The proof is similar to the proof of the projection lemma. There is exactly one β rewrite since υ may not duplicate or eliminate a subterm at an external position.

We also use the contraposition: if $\upsilon(a)$ is strongly β normalising, $\upsilon(a) = \upsilon(b)$, and $a \underset{B,p}{\longrightarrow} b$, then p is internal.

In the following lemma, $\overset{1,2}{\longrightarrow}$ means one or two rewrites and $\overset{0,1,2}{\longrightarrow}$ means zero, one or two rewrites.

Lemma 11 (Commutation Lemma) *If $\upsilon(a)$ is strongly β normalising, $\upsilon(a) = \upsilon(b)$ and $a \underset{\lambda\upsilon,p}{\longrightarrow}{}^{int} \cdot \underset{\upsilon,q}{\longrightarrow}{}^{ext} b$ then*
$a \overset{1,2}{\underset{\upsilon}{\longrightarrow}}{}^{ext} \cdot \overset{0,1,2}{\underset{\lambda\upsilon}{\longrightarrow}}{}^{int} b$.

Proof: T□he proof is by case analysis on the first rewrite position p relatively to the second one q.

- p and q are disjoint, then it is clear that we can permute the two rewrites, thus $a \underset{\upsilon}{\longrightarrow}{}^{ext} \cdot \underset{\lambda\upsilon}{\longrightarrow}{}^{int} b$.
- p is a strict prefix of q, this case is impossible, indeed if p is an internal position in b (a rewrite at an internal position remains internal) and $b_{|q}$ is a subterm of $b_{|p}$ then q is not an external position.
- q is a prefix of p, let us analyse each υ-rewrite rule at q.
  - (App) is applied, $b_{|q} = c_1[s]\ c_2[s]$, then there are only three possible cases to rewrite $a_{|q}$.
    - If $s' \underset{\lambda\upsilon}{\longrightarrow} s$ and $(c_1\ c_2)[s'] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} (c_1\ c_2)[s] \underset{\upsilon}{\longrightarrow}{}^{ext} c_1[s]\ c_2[s]$, then
    $$(c_1\ c_2)[s'] \underset{\upsilon}{\longrightarrow}{}^{ext} c_1[s']\ c_2[s'] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c_1[s]\ c_2[s'] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c_1[s]\ c_2[s].$$
    - If $c_1' \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c_1$ and $(c_1'\ c_2)[s] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} (c_1\ c_2)[s] \underset{\upsilon}{\longrightarrow}{}^{ext} c_1[s]\ c_2[s]$, then
    $$(c_1'\ c_2)[s] \underset{\upsilon}{\longrightarrow}{}^{ext} c_1'[s]\ c_2[s] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c_1[s]\ c_2[s].$$
    - Similarly if $c_2 \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c_2'$.

    Notice that, the term $A\{(c_1\ c_2)[s]\}_q$ cannot be produced by an internal rewrite on a at p since $a_{|p}$ is a subterm of $a_{|q}$ and q is an external position.
  - (Lambda) is applied, $b_{|q} = \lambda c[\Uparrow(s)]$, then there are only two possible cases,
    - If $s' \underset{\lambda\upsilon}{\longrightarrow} s$ and $(\lambda c)[s'] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} (\lambda c)[s] \underset{\upsilon}{\longrightarrow}{}^{ext} (\lambda c)[\Uparrow(s)]$, then
    $$(\lambda c)[s'] \underset{\upsilon}{\longrightarrow}{}^{ext} \lambda c[\Uparrow(s')] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} \lambda c[\Uparrow(s)].$$
    - If $c' \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c$ and $(\lambda c')[s] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} (\lambda c)[s] \underset{\upsilon}{\longrightarrow}{}^{ext} \lambda c[\Uparrow(s)]$, then
    $$(\lambda c')[s] \underset{\upsilon}{\longrightarrow}{}^{ext} \lambda c'[\Uparrow(s)] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} \lambda c[\Uparrow(s)].$$
  - (FVar) is applied, then $b_{|q} = c$ such that $\underline{1}[c'/] \underset{\lambda\upsilon}{\longrightarrow}{}^{int} \underline{1}[c/] \underset{\upsilon}{\longrightarrow}{}^{ext} c$, with $c' \underset{\lambda\upsilon}{\longrightarrow} c$. Three possible cases which depend on the nature (internal or external) of the rewrite of c'.
    - $c' \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c$, then
    $$\underline{1}[c'/] \underset{\upsilon}{\longrightarrow}{}^{ext} c' \underset{\lambda\upsilon}{\longrightarrow}{}^{int} c.$$
    - $c' \underset{\upsilon}{\longrightarrow}{}^{ext} c$, then
    $$\underline{1}[c'/] \underset{\upsilon}{\longrightarrow}{}^{ext} c' \underset{\upsilon}{\longrightarrow}{}^{ext} c.$$
    - $c' \underset{B}{\longrightarrow}{}^{ext} c$, this case is impossible under the hypothesis $\upsilon(a) = \upsilon(b)$ and $\upsilon(a)$ is strongly β normalising, indeed we have also
    $$a = A\{1[c'/]\}_q \underset{\upsilon}{\longrightarrow} A\{c'\}_q \underset{B}{\longrightarrow}{}^{ext} A\{c\}_q = b$$

and $\upsilon(a) = \upsilon(A\{c'\}_q)$ is strongly β normalising then by Lemma 10

$$\upsilon(A\{c'\}_q) \neq \upsilon(A\{c\}_q),$$

and

$$\begin{array}{rcccl} \upsilon(a) & = & \upsilon(A\{\underline{1}[c'/]\}_q) & = & \upsilon(A\{c'\}_q), \\ \upsilon(b) & = & \upsilon(A\{c\}_q). & & \end{array}$$

then $\upsilon(a) \neq \upsilon(b)$.

– (RVar) is applied, then $b_{|q} = \underline{n}$, such that,

$$\underline{n+1}[c'/] \underset{\lambda\upsilon}{\longrightarrow}^{int} \underline{n+1}[c/] \underset{\upsilon}{\longrightarrow}^{ext} \underline{n}, \text{ then } \underline{n+1}[c'/] \underset{\upsilon}{\longrightarrow}^{ext} \underline{n}.$$

– (FVarLift) is applied, then $b_{|q} = \underline{1}$, such that,

$$\underline{1}[\Uparrow(s')] \underset{\lambda\upsilon}{\longrightarrow}^{int} \underline{1}[\Uparrow(s)] \underset{\upsilon}{\longrightarrow}^{ext} \underline{1}, \text{ then } \underline{1}[\Uparrow(s')] \underset{\upsilon}{\longrightarrow}^{ext} \underline{1}.$$

– (RVarLift) is applied, then $b_{|q} = \underline{n}[s][\uparrow]$, such that,
$\underline{n+1}[\Uparrow(s')] \underset{\lambda\upsilon}{\longrightarrow}^{int} \underline{n+1}[\Uparrow(s)] \underset{\upsilon}{\longrightarrow}^{ext} \underline{n}[s][\uparrow]$, then

$$\underline{n+1}[\Uparrow(s')] \underset{\upsilon}{\longrightarrow}^{ext} \underline{n}[s'][\uparrow] \underset{\lambda\upsilon}{\longrightarrow}^{int} \underline{n}[s][\uparrow].$$

Before iterating the previous lemma, notice that

$$\overset{1,2}{\underset{\upsilon}{\longrightarrow}}{}^{ext} \subseteq \overset{+}{\underset{\upsilon}{\longrightarrow}}{}^{ext}$$

and

$$\overset{0,1,2}{\underset{\lambda\upsilon}{\longrightarrow}}{}^{int} \subseteq \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}}{}^{int}$$

and that we may weaken the condition of the Commutation Lemma as

$$\underset{\lambda\upsilon}{\longrightarrow}^{int} \underset{\upsilon}{\longrightarrow}^{ext} \cdot \subseteq \overset{+}{\underset{\upsilon}{\longrightarrow}}{}^{ext} \cdot \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}}{}^{int}.$$

Therefore the hypotheses of the commutation lemmas of the appendix apply.

Lemma 12 (Iterative Commutation Lemma) *Let $a_0 \ldots a_n$ be $n+1$ terms such that $\upsilon(a_0)$ is strongly β normalising, $\upsilon(a_i) = \upsilon(a_0)$ and $a_{i-1} \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}}{}^{int} \cdot \underset{\upsilon}{\longrightarrow}^{ext} a_i$ for $1 \leq i \leq n$. Then $a_0 \overset{n}{\underset{\upsilon}{\longrightarrow}}{}^{ext} \cdot (\underset{\lambda\upsilon}{\longrightarrow}^{int} \cup \underset{\upsilon}{\longrightarrow}^{ext})^* \ a_n$.*

Proof: O□ne applies Lemma 15 of the appendix with $S = \underset{\upsilon}{\longrightarrow}^{ext}$ and $R = \underset{\lambda\upsilon}{\longrightarrow}^{int} \cap E_\upsilon$, where $a \; E_\upsilon \; b$ means $\upsilon(a) = \upsilon(b)$.

Lemma 13 *Let a_1 be a strongly β normalising pure term. In each infinite $\lambda\upsilon$ derivation of terms starting with a_1 there exists an N such that for $i \geq N$ all the $\lambda\upsilon$ rewrites are internal.*

Proof: A□ $\lambda\upsilon$ derivation $a_1, a_2, \ldots, a_n, \ldots$ starting from a_1 can be written:

$$a_1 \underset{B}{\longrightarrow}^{ext} a'_1 \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} a''_2 \underset{B}{\longrightarrow}^{ext} a'_2 \cdots a''_i \underset{B}{\longrightarrow}^{ext} a'_i \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}} a''_{i+1} \underset{B}{\longrightarrow}^{ext} a'_{i+1} \cdots$$

where the rewrites from a'_i to a''_{i+1} are either υ rewrites or internal B rewrites. By Lemma 10, we have $\upsilon(a''_i) \underset{\beta}{\longrightarrow} \upsilon(a'_i) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon(a''_{i+1})$, hence

$$a_1 = \upsilon(a_1) \overset{+}{\underset{\beta}{\longrightarrow}} \upsilon(a''_2) \cdots \upsilon(a''_i) \overset{+}{\underset{\beta}{\longrightarrow}} \upsilon(a''_{i+1}) \cdots$$

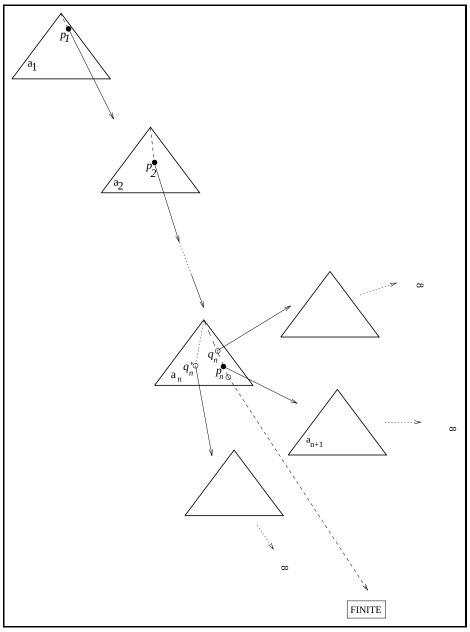

Figure 4: A minimal $\lambda\upsilon$ derivation

Since a_1 is strongly β normalising, there are only finitely many β rewrites. Therefore the number of external B rewrites in the $\lambda\upsilon$ derivation $a_1, a_2, a_3, \ldots$ is finite. Thus there exists a P such that for $i \geq P$ we have only internal B rewrites:

$$a_P \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}}^{int} \cdot \underset{\upsilon}{\longrightarrow}^{ext} \cdot \overset{*}{\underset{\lambda\upsilon}{\longrightarrow}}^{int} \cdots$$

We can also claim that there exists an $N \geq P$ such that for $i \geq N$, the υ rewrites are internal. Indeed since υ is strongly normalising there exists a natural number n_{a_P} such that no $\lambda\upsilon$ derivation starting at a_P can begin with more than n_{a_P} υ rewrites. If one supposes there are infinitely many external υ rewrites in an infinite $\lambda\upsilon$-derivation starting from a_P, there are at least $\mathsf{n}_{a_P} + 1$ of them. By Iterative Commutation Lemma, one can create $\mathsf{n}_{a_P} + 1$ external υ rewrites starting from a_P which is not possible.

5.3 Minimal derivations

Definition 4 (Derivation ordering) *Let a_1 be a term and $\mathcal{D}$ and $\mathcal{D}'$ two $\lambda\upsilon$ derivations starting from a_1.*

$$\mathcal{D} = a_1 \underset{\lambda\upsilon, p_1}{\longrightarrow} a_2 \underset{\lambda\upsilon, p_2}{\longrightarrow} \cdots a_n \underset{\lambda\upsilon, p_n}{\longrightarrow} a_{n+1} \cdots,$$

is said to be smaller *than*

$$\mathcal{D}' = a_1 \underset{\lambda\upsilon, q_1}{\longrightarrow} a_2 \underset{\lambda\upsilon, q_2}{\longrightarrow} \cdots a_n \underset{\lambda\upsilon, q_n}{\longrightarrow} a'_{n+1} \cdots$$

if $p_i = q_i$ for $i < n$ and q_n is a strict prefix of p_n.

A derivation starting from a_1 can be characterised by the sequence $(p_1, p_2, \ldots)$ of its positions, therefore the derivation ordering is nothing but the lexicographic ordering on those sequences.

Definition 5 (Minimal $\lambda\upsilon$ derivation) *An infinite $\lambda\upsilon$ derivation $\mathcal{D}$ which starts from a pure term a_1 is* minimal *if there is no* infinite *derivation starting from a_1 which is smaller than $\mathcal{D}$.*

Let us insist on two facts. First the *minimal derivation* is not minimal among all the derivations (finite or infinite), but only among the infinite derivations (see Fig. 4). Second, such a minimal derivation always exists, whenever an infinite derivation exists.

5.4 Main Theorem

We need also another definition which we call *frontier* and which represents the set of closures at external positions.

Definition 6 (Frontier) *The frontier of a term a, denoted $Fr(a)$, is the set of external positions p such that $a_{|p}$ is a closure, i.e. is of the form $\_[\_]$.*

Theorem 2 (Preservation of normalisation) *If a pure term a_1 is strongly β normalising, then a_1 is strongly $\lambda\upsilon$ normalising.*

Proof: T□he proof is by contradiction. Suppose a_1 is a pure strongly β normalising term, but non strongly $\lambda\upsilon$ normalising. Let us consider a *minimal infinite* $\lambda\upsilon$ derivation $\mathcal{D}$ starting with the term a_1. By Lemma 13 there exists N after which only internal $\lambda\upsilon$ rewrites take place. We have $Fr(a_N) = Fr(a_{N+1})$, since a closure at the frontier can be created only by an external rewrite. The cardinal of $Fr(a_N)$ is finite, we can therefore choose a position p in $Fr(a_N)$ such that $a_{N|_p} = c[\Uparrow^i(b_N/)]$ and such that the minimal $\lambda\upsilon$ derivation contains infinitely many rewrites below p. The rewrites below each p in $Fr(a_N)$ are independent, we can therefore extract from the derivation $\mathcal{D}$ an infinite $\lambda\upsilon$ derivation $\mathcal{D}' = (a_1, \ldots, a_N, a'_{N+1}, \ldots, a'_j, \ldots)$ starting with the same N first terms and such that all the internal $\lambda\upsilon$ rewrites after the N^{th} take place inside the closure $c[\Uparrow^i(b_N/)]$. In $\mathcal{D}'$ we have

$$a_N = t\{c[\Uparrow^i(b_N/)]\}_p \xrightarrow[\lambda\upsilon]{int} a'_{N+1} = t\{c[\Uparrow^i(b_{N+1}/)]\}_p \xrightarrow[\lambda\upsilon]{int} \cdots a'_j = t\{c[\Uparrow^i(b_j/)]\}_p \cdots$$

where t is a context, $p \in Fr(a_N)$ and the sequence $(b_N, b_{N+1}, \ldots, b_j, \ldots)$ is an infinite derivation. From Lemma 9 we know that the closure $c[\Uparrow^i(b_N/)]$ has been created sometime before N, by a B rewrite. Lemma 9 says also that there exists $J < N$ and a position p_J such that:

$$a_J = t'\{(\lambda c')b\}_{p_J} \xrightarrow[B,p_J]{} t'\{c'[b/]\}_{p_J} = a_{J+1}$$

where t' is a context. Moreover $b \xrightarrow[\lambda\upsilon]{*} b_N$. Let us consider the following infinite $\lambda\upsilon$ derivation $\mathcal{D}''$ defined as

$$\begin{array}{lcl}
a_1 & \xrightarrow[\lambda\upsilon]{} & a_2 \\
 & \vdots & \\
a_J & = & t'\{(\lambda c')b\}_{p_J} \\
 & \xrightarrow[\lambda\upsilon]{*} & t'\{(\lambda c')b_N\}_{p_J} \\
 & \xrightarrow[\lambda\upsilon]{} & t'\{(\lambda c')b_{N+1}\}_{p_J} \\
 & \vdots & \\
 & \xrightarrow[\lambda\upsilon]{} & t'\{(\lambda c')b_l\}_{p_J} \\
 & \vdots &
\end{array}$$

In $\mathcal{D}''$, one has either

$$a_J \xrightarrow[\lambda\upsilon]{} t'\{(\lambda c')\, b_0\}_{p_J} \text{ and } b \xrightarrow[\lambda\upsilon]{} b_0 \xrightarrow[\lambda\upsilon]{*} b_N$$

or

$$a_J \xrightarrow[\lambda\upsilon]{} t'\{(\lambda c')\ b_{N+1}\}_{p_J} \text{ and } b = b_N$$

In both cases, a_J rewrites below p_J. Therefore, $\mathcal{D}''$ is smaller than $\mathcal{D}$. $\mathcal{D}''$ is infinite. That contradicts the minimality of the derivation $\mathcal{D}$.

Corollary 7 *Typed pure terms in* $\mathbf{Term}_\upsilon$ *are strongly* $\lambda\upsilon$ *normalising.*

6 De Bruijn's system $C\lambda\xi\phi$ and our presentation $\lambda\xi\phi$

In (de Bruijn, 1978), N. G. de Bruijn presents the first calculus of explicit substitutions which he calls $C\lambda\xi\phi$. As his notations are somewhat difficult to read and different of these we are used to, we propose to describe his rules in notations similar to those used in the previous section.

Starting from rule (B), de Bruijn distinguishes two kinds of substitutions: substitutions that rename variables and substitutions that assign terms to variables. The substitutions of the first kind are associated with functions $\theta\ : \mathbb{N} \to \mathbb{N}$. In our notations θ's correspond to substitutions of the form $\Uparrow^i(\uparrow)$ and $\Uparrow^i(\updownarrow)$, where $\updownarrow$ is the substitution defined below. The calculus of explicit substitution proposes a notation for representing those functions, and distinguishes a function from its associated explicit substitution. The explicit substitution associated with function θ will be written $\underline{\theta}$. Actually de Bruijn uses $\xi(n)$ for our $\underline{n}$ and $\phi(\theta)$ for our $\underline{\theta}$, hence the name $C\lambda\xi\phi$. Among those functions de Bruijn considers a function which he names θ_2 and which corresponds to:

$$\begin{array}{llcl}
\theta_2 : & \underline{1} & \mapsto & \underline{2} \\
& \underline{2} & \mapsto & \underline{1} \\
& \underline{3} & \mapsto & \underline{3} \\
& & \vdots & \\
& \underline{n+2} & \mapsto & \underline{n+2} \\
& & \vdots &
\end{array}$$

To include this substitution in our notations, we propose to write $\underline{\theta_2}$ as $\updownarrow$ and to call it a *transposition*. The behaviour of $\updownarrow$ can be described by its effect on indices as follows:

$$\begin{array}{lrcl}
(Transp_1) & \underline{1}[\updownarrow] & \to & \underline{2} \\
(Transp_2) & \underline{2}[\updownarrow] & \to & \underline{1} \\
(Transp_3) & \underline{n+2}[\updownarrow] & \to & \underline{n+2}
\end{array}$$

The effect of a function $\theta : \mathbb{N} \to \mathbb{N}$ on pure terms is described by de Bruijn with the following rules. In them, de Bruijn distinguishes constant functions, e.g., c of arity 0, f of arity 1, and g of arity 2.

$$\begin{array}{lrcl}
(A_1) & c[\underline{\theta}] & \to & c \\
(A_2) & \underline{n}[\underline{\theta}] & \to & \underline{\theta(n)} \\
(A_4) & (f\ a)[\underline{\theta}] & \to & f(a[\underline{\theta}]) \\
(A_6) & a[\underline{\theta}][\underline{\theta'}] & \to & a[\underline{\theta'\cdot\theta}] \\
(A_7) & (\lambda a)[\underline{\theta}] & \to & \lambda(a[\underline{L(\theta)}]) \\
(A_8) & (g\ a\ b)[\underline{\theta}] & \to & g\ (a[\underline{\theta}]\ b[\underline{\theta}])
\end{array}$$

where $L(\theta)(1) = 1$ and $L(\theta)(n+1) = \theta(n)+1$, and $\theta'\cdot\theta(n) = \theta'(\theta(n))$. (A_9) is a rule scheme which is just a generalisation of (A_1), (A_4), and (A_8) to functions of arity $n = 3, \ldots$ Rules (A_3) and (A_5) are omitted purposely since they are not relevant here. Actually in (A_6) and (A_7), $\underline{\theta'\cdot\theta}$ and $\underline{L(\theta)}$ are defined directly on the underlying functions. L is just the *Lift* operation that is written $\Uparrow$ in our notations and $\cdot$ is the composition written $\circ$ in

$$
\begin{array}{lrcl}
\text{(B)} & (\lambda a)b & \to & a[b/] \\
\\
(\text{App}_1) & (a\ b)[c/] & \to & a[c/]\ b[c/] \\
(\text{Lambda}_1) & (\lambda a)[b/] & \to & \lambda(a\llbracket\updownarrow\rrbracket[b\llbracket\uparrow\rrbracket/]) \\
(\text{FVar}) & \underline{1}[a/] & \to & a \\
(\text{RVar}) & \underline{n+1}[a/] & \to & \underline{n} \\
\\
(\text{App}_2) & (a\ b)\llbracket s\rrbracket & \to & a\llbracket s\rrbracket\ b\llbracket s\rrbracket \\
(\text{Lambda}_2) & (\lambda a)\llbracket s\rrbracket & \to & \lambda(a\llbracket\Uparrow(s)\rrbracket) \\
(\text{FVarLift}) & \underline{1}\llbracket\Uparrow(s)\rrbracket & \to & \underline{1} \\
(\text{RVarLift}) & \underline{n+1}\llbracket\Uparrow(s)\rrbracket & \to & \underline{n}\llbracket s\rrbracket\llbracket\uparrow\rrbracket \\
(\text{VarShift}) & \underline{n}\llbracket\uparrow\rrbracket & \to & \underline{n+1} \\
(\text{Transp}_1) & \underline{1}\llbracket\updownarrow\rrbracket & \to & \underline{2} \\
(\text{Transp}_2) & \underline{2}\llbracket\updownarrow\rrbracket & \to & \underline{1} \\
(\text{Transp}_3) & \underline{n+2}\llbracket\updownarrow\rrbracket & \to & \underline{n+2}
\end{array}
$$

Figure 5: The rewrite system $\lambda\xi\phi$

contemporary notations. Notice that the composition introduced in rule (A_6) is not used elsewhere and is not necessary for a complete definition.

The second kind of substitutions are those of the form $t/$.

$$
\begin{array}{lrcl}
(B_1) & c[t/] & \to & c \\
(B_2) & \underline{n+1}[t/] & \to & \underline{n} \\
(B_3) & \underline{1}[t/] & \to & t \\
(B_4) & a[\uparrow][t/] & \to & a \\
(B_6) & (f\ a)[t/] & \to & f(a[t/]) \\
(B_7) & (\lambda a)[t/] & \to & \lambda(a[\updownarrow][t[\uparrow]/]) \\
(B_{10}) & g\ (a\ b)[t/] & \to & g(a[t/]\ b[t/])
\end{array}
$$

As above, (B_{11}) is a rule scheme which is just a generalisation of (B_1), (B_6) and (B_{10}). Likewise, rules (B_5) and (B_8) are omitted purposely since they are not relevant here.

This system inspires us a calculus of explicit substitutions which we call $\lambda\xi\phi$ (Fig. 5). Let us call $\mathbf{Terms}_{\xi\phi}$ the set of terms described by the grammar:

$$
\begin{array}{lcll}
\mathbf{Terms}_{\xi\phi} & a & ::= & \underline{n} \mid ab \mid \lambda a \mid a\llbracket s\rrbracket \ \mid a[t/] \\
\mathbf{Substitutions}_{\xi\phi} & s & ::= & \Uparrow(s) \mid \uparrow \mid \updownarrow \\
\mathbf{Naturals} & n & ::= & n+1 \mid 1.
\end{array}
$$

$\llbracket\ \rrbracket$ denotes substitutions that rename variables, they are written $\underline{\theta}$ in de Bruijn's notations. $[\ /]$ denotes substitutions that assign a term to the index $\underline{1}$. We call $\xi\phi$ the system $\lambda\xi\phi\setminus(B)$, $\xi\phi$ can be shown to be strongly normalising by using the lexicographic products $(<_\iota, <_{\kappa_1}, <_{\kappa_2})$. $<_\iota$ is defined by the interpretation $\iota : \mathbf{Terms}_{\xi\phi} \to \mathbf{Terms}_\xi$ where $\mathbf{Terms}_\xi$ is described by the grammar:

$$
\begin{array}{lcll}
\mathbf{Terms}_{\xi} & a & ::= & \underline{n} \mid ab \mid \lambda a \mid a[t/] \\
\mathbf{Naturals} & n & ::= & n+1 \mid 1.
\end{array}
$$

and ι is described as follows:

$$
\begin{array}{rcl}
\iota(\underline{n}) & = & \underline{1} \\
\iota(a\ b) & = & \iota(a)\ \iota(b) \\
\iota(\lambda a) & = & \lambda(\iota(a))
\end{array}
$$

$$
\begin{array}{|rcl rcl|}
\hline
\kappa_1(\underline{n}) & = & 2^{\kappa_1(n)} & \kappa_2(\underline{n}) & = & 2^{\kappa_2(n)} \\
\kappa_1(n+1) & = & \kappa_1(n)+1 & \kappa_2(n+1) & = & \kappa_2(n)+1 \\
\kappa_1(1) & = & 2 & & & \\
\kappa_1(ab) & = & \kappa_1(a)+\kappa_1(b)+1 & & & \\
\kappa_1(\lambda a) & = & \kappa_1(a)+1 & & & \\
\kappa_1(a[s]\,) & = & \kappa_1(a)\kappa_1(s) & \kappa_2(a[s]) & = & \kappa_2(a)\kappa_2(s) \\
\kappa_1(\Uparrow(s)) & = & \kappa_1(s) & \kappa_2(\Uparrow(s)) & = & 2\kappa_2(s) \\
\kappa_1(\uparrow) & = & 2 & \kappa_2(\uparrow) & = & 3 \\
\kappa_1(\updownarrow) & = & 2 & & & \\
\kappa_1(a/) & = & \mathit{any} & & & \\
\hline
\end{array}
$$

Figure 6: Interpretations for proving the termination of υ

$$
\begin{array}{rcl}
\iota(a[\![s]\!]) & = & \iota(a) \\
\iota(a[b/]) & = & \iota(a)[\iota(b)/]
\end{array}
$$

$a <_\iota b$ if and only if $\iota(a) <_\xi \iota(b)$ where $<_\xi$ is a lexicographic path ordering described in $\mathbf{Terms}_\xi$ by the precedence that says that an abstraction is less than a closure and less than an application which could be pictured by the following inequalities $\lambda < \_[\_/]$ and $\lambda < \_\,\_$.

κ_1 and κ_2 are interpretations from $\mathbf{Terms}_{\xi\phi}$ to the set of elementary functions over $\mathbb{N}$. We conclude that $\xi\phi$ is strongly normalising. $\xi\phi$ also is orthogonal, i.e., left-linear and without superposition, $\xi\phi$ is then confluent.

There are two critical pairs between B and App_1 on one side and between B and App_2 on another side. The critical pairs are:

$$
\begin{array}{rcl}
a[b/][c/] & = & a[\![\updownarrow]\!][c[\![\updownarrow]\!]/][b[c/]/] \\
a[b/][\![s]\!] & = & a[\![\Uparrow(s)]\!][b[\![s]\!]/]
\end{array}
$$

Those critical pairs can be proved as inductive lemmas in $\mathbf{Terms}_{\xi\phi}/\overset{*}{\underset{\lambda\xi\phi}{\longleftrightarrow}}$, i.e., modulo the equality generated by $\lambda\xi\phi$ on $\mathbf{Terms}_{\xi\phi}$. Then it can be proved that the rewriting relation $\overset{*}{\underset{\lambda\xi\phi}{\longrightarrow}}$ defined on $\mathbf{Terms}_{\xi\phi}$ and generated by $\lambda\xi\phi$ is confluent.

The systems $\lambda\upsilon$ and $\lambda\xi\phi$ share the same goal. Both introduce operators by necessity. In $\lambda\upsilon$, substitutions of both kinds are lifted when put under λ, whereas in $\lambda\xi\phi$ only *renaming substitutions* are because there is a way to avoid lifting of substitutions of type $a/$. The calculi are different in the form, but are similar in spirit. We feel that $\lambda\upsilon$ is slightly closer to the aim of extreme simplicity suggested by Curry, but this is debatable.

7 Conclusion

$\lambda\upsilon$ has had extensions, namely to include η-rules (Briaud, 1995). Preservation of strong normalisation of $\lambda\upsilon$ together with confluence of $\lambda\sigma_\Uparrow$ on open terms raises an interesting challenge, namely, finding a calculus of explicit substitutions which is confluent on open terms and preserves strong normalisation.

Acknowledgement. We would like to thank Nicolaas G. de Bruijn for an interesting discussion and for mentioning his papers to us, Alejandro Ríos for a very careful reading and discussions, Georges Gonthier and Luc Maranget for suggesting improvements in the proof of Theorem 2, Roberto Amadio, Philippe de Groote and Paul Zimmermann for interesting discussions, and Pierre-Louis Curien for encouragement.

References

Abadi, M., Cardelli, L., Curien, P.-L., & Lévy, J.-J. 1990 (Feb.). *Explicit Substitutions*. Tech. rept. 54. Digital Systems Research Center. Preliminary version in *Proc. of the 17th POPL conference, Orlando (Fla., USA)*.

Abadi, M., Cardelli, L., Curien, P.-L., & Lévy, J.-J. 1991. Explicit Substitutions. *Journal of Functional Programming*, **1**(4), 375–416.

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

Bourbaki, N. 1954. *Éléments de mathématiques: Théories des ensembles*. Vol. 1. Hermann & C^{ie}.

Briaud, D. 1995. An explicit Eta rewrite rule. *In:* Dezani, M. (ed), *Int. Conf. on Typed Lambda Calculus and Applications*.

Curien, P.-L. 1983. *Combinateurs catégoriques, algorithmes séquentiels et programmation applicative*. Thèse de Doctorat d'Etat, Université Paris 7.

Curien, P.-L. 1986a. Categorical Combinators. *Information and Control*, **69**, 188–254.

Curien, P.-L. 1986b. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman.

Curien, P.-L. 1991. An abstract framework for environment machines. *Theoretical Computer Science*, **82**, 389–402.

Curien, P.-L., Hardin, Th., & Lévy, J.-J. 1992 (Feb.). *Confluence properties of weak and strong calculi of explicit substitutions*. RR 1617. INRIA, Rocquencourt.

Curry, H. B., & Feys. 1958. *Combinatory Logic*. Vol. 1. Amsterdam: Elsevier Science Publishers B. V. (North-Holland).

de Bruijn, N. G. 1972. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, **75**(5), 381–392.

de Bruijn, N. G. 1978. *A namefree lambda calculus with facilities for internal definition of expressions and segments*. TH-Report 78-WSK-03. Technological University Eindhoven, Netherlands, Department of Mathematics.

Dershowitz, N., & Jouannaud, J.-P. 1990. Rewrite Systems. *Chap. 6, pages 244–320 of:* van Leeuwen, J. (ed), *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V. (North-Holland).

Ehrhrard, T. 1988. *Une Sémantique Catégorique des Types Dépendants. Application au Calcul des Constructions.* Thèse de Doctorat d'Université, Universite Paris VII.

Field, J. 1990. On laziness and optimality in Lambda Interpreters: Tools for Specification and Analysis. *Pages 1–15 of: Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*. ACM, San Fransisco.

Hardin, T. 1992. Eta-conversion for the languages of explicit substitutions. *Pages 306–321 of:* Kirchner, H., & Levi, G. (eds), *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*. Lecture Notes in Computer Science, vol. 632. Springer-Verlag.

Hardin, Th. 1987. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs*. Thèse de Doctorat d'Etat, Université Paris 7.

Hardin, Th. 1989. Confluence results for the pure strong categorical combinatory logic CCL: λ-calculi as subsystems of CCL. *Theoretical Computer Science*, **65**, 291–342.

Hardin, Th., & Lévy, J.-J. 1989. A confluent calculus of substitutions. *In: France-Japan Artificial Intelligence and Computer Science Symposium*.

Kamareddine, F., & Ríos, A. 1995. A λ-calculus à la de Bruijn with explicit substitutions. *In: PLILP'95*. Lecture Notes in Computer Science. Springer-Verlag.

Kamereddine, F., & Nederpelt, R.P. 1993. On stepwise explicit substitutions. *International Journal of Foundations of Computer Science*, **4**(3), 197–240.

Lescanne, P. 1992. Termination of rewrite systems by elementary interpretations. *Pages 21–36 of:* Kirchner, Hélène, & Levi, G. (eds), *Proceedings 3rd International Conference on Algebraic and Logic Programming, Volterra (Italy)*. Lecture Notes in Computer Science, vol. 632. Springer-Verlag.

Lescanne, P. 1994. From λσ to λυ, a journey through calculi of explicit substitutions. *Pages 60–69 of:* Boehm, Hans (ed), *Proceedings of the 21st Annual ACM Symposium on Principles Of Programming Languages, Portland (Or., USA)*. ACM.

Lescanne, P., & Rouyer-Degli, J. 1994 (Jan.). *The Calculus of explicit substitutions λυ*. Tech. rept. RR-2222. INRIA-Lorraine.

Melliès, P.-A. 1995. Typed λ-calculi with explicit substitutions may not terminate. *In:* Dezani, M. (ed), *Int. Conf. on Typed Lambda Calculus and Applications*.

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

Rose, K., & Bloo, R. 1995 (Apr.). *Deriving Requirements for Preservation of Strong Normalisation in Lambda Calculi with Explicit Substitution*. Available as `ftp://ftp.diku.dk/diku/users/kris/Explicit-PSN.ps`.

A Two commutation results

In this appendix, we prove two commutation results on abstract relations, which are folklore.

Lemma 14 *If* $R\cdot S \subseteq S^+\cdot R^*$ *then* $(R\cup S)^*\cdot S \subseteq S\cdot (R\cup S)^*$.

Proof: A□ctually one proves $(\forall n\in \mathbb{N})\ \ (R\cup S)^n\cdot S \subseteq S\cdot(R\cup S)^*$ by induction on n. If $n=0$ it is obvious. Otherwise

$$\begin{array}{rcll}
(R\cup S)^{n+1}\cdot S & = & (R\cup S)\cdot(R\cup S)^n\cdot S & \\
& \subseteq & (R\cup S)\cdot S\cdot(R\cup S)^* & \text{by induction}\\
& = & R\cdot S\cdot(R\cup S)^* \,\cup\, S\cdot S\cdot(R\cup S)^* & \\
& \subseteq & S^+\cdot R^*\cdot(R\cup S)^* \,\cup\, S\cdot S\cdot(R\cup S)^* & \text{by hypothesis}\\
& = & S\cdot(S^*\cdot R^*\cdot(R\cup S)^* \,\cup\, S\cdot(R\cup S)^*) & \\
& = & S\cdot(R\cup S)^* &
\end{array}$$

Lemma 15 *If* $R\cdot S \subseteq S^+\cdot R^*$ *then* $(\forall n\in\mathbb{N})\ \ ((R\cup S)^*\cdot S)^n \subseteq S^n\cdot(R\cup S)^*$.

Proof: B□y induction on n. If $n = 0$ it is obvious. Otherwise

$$\begin{array}{rcll} ((R \cup S)^* \cdot S)^{n+1} & = & (R \cup S)^* \cdot S \cdot ((R \cup S)^* \cdot S)^n & \\ & \subseteq & S \cdot (R \cup S)^* \cdot ((R \cup S)^* \cdot S)^n & \text{by Lemma 14} \\ & = & S \cdot ((R \cup S)^* \cdot S)^n & \\ & \subseteq & S \cdot S^n \cdot (R \cup S)^* & \text{by induction} \\ & = & S^{n+1} \cdot (R \cup S)^* & \end{array}$$