HAL
open science

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

Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, Jocelyne Rouyer-Degli

To cite this version:

Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, Jocelyne Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. [Research Report] RR-2477, INRIA. 1995. inria-00074197

HAL Id: inria-00074197

https://inria.hal.science/inria-00074197v1

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

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

Zine-El-Abidine BENAISSA , Daniel BRIAUD , Pierre LESCANNE , Jocelyne ROUYER-DEGLI

N° 2477

Janvier 1995

PROGRAMME 2

Rapport de recherche

ISSN 0249-6399

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

Zine-El-Abidine BENAISSA , Daniel BRIAUD , Pierre LESCANNE ,
Jocelyne ROUYER-DEGLI

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet EURECA

Rapport de recherche n2477 — Janvier 1995 — 27 pages

Abstract: Explicit substitutions were proposed by Abadi, Cardelli, Curien, Hardin and Lévy 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. $\lambda\upsilon$ is simple and preserves strong normalisation. Apparently that important property cannot stay with another important one, namely confluence on open terms. The spirit of $\lambda\upsilon$ 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 normalization of β-reduction.

Key-words: Explicit substitutions, λ-calculus

(Résumé : tsvp)

Unité de recherche INRIA Lorraine
Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

$\lambda\upsilon$, un calcul de substitutions explicites qui preserve la forte normalisation

Résumé : Les calculs de substitutions explicites ont été proposés par Abadi, Cardelli, Curien, Hardin et Lévy dans le but d'inclure les substitutions à l'intérieur du λ-calcul et de proposer un mécanisme pour les évaluer. $\lambda\upsilon$ est une autre approche qui permet d'expliquer le processus de substitution en décomposant celle-ci en plusieures petites opérations élémentaires. Le calcul $\lambda\upsilon$ est simple et préserve la forte normalisation de la β-reduction. Néanmoins, ce calcul n'est pas confluent sur les termes ouverts. L'esprit du $\lambda\upsilon$ est lié à un autre calcul proposé par de Bruijn, appelé $C\lambda\xi\phi$. Dans ce rapport, on introduira $\lambda\upsilon$ et $C\lambda\xi\phi$ puis on comparera ces deux calculs. De plus, on prouvera plusieures propriétés sur $\lambda\upsilon$, qui sont: la correction du $\lambda\upsilon$ par rapport à la β-reduction, la confluence de $\lambda\upsilon$ sur les termes clos, i.e tous les termes du λ-calcul, et enfin la préservation de la forte normalisation de la β-réduction.

Mots-clé : Substitutions explicites, λ-calcul

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 to the variable x. In classical λ-calculus [3] the mechanism of substitution is described by a specific and external formalism. This description is part of the *epitheory* [11] which means it is not integrated into the theory. In the introduction of 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 page 6 of [11] 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 that 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 substitution answer this challenge since they contain in the same framework both a version 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 calculus of explicit substitution.

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 [13], see also [20]. Another calculus belonging to this family, which is extensively studied in this paper was proposed by one of us in [22]. Those calculi attempt to describe (perhaps naively) the principles of the implementation of λ-calculus. They do not aim to efficiency. Their main feature is that they preserve strong normalisation.

The other approach, which we propose to call the $\lambda\sigma$ family, has been proposed by Abadi, Cardelli, Curien, Hardin, Lévy and Field around 1989 [1, 2, 15, 19, 10, 24]. It follows previous research by Curien who proposed in 1983 *categorical combinators* [6, 8, 7] a combinatory logic more intuitive than the classical one. Hardin in 1987 [17, 18] 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 [9] 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 [23] has shown that those calculi do not preserve strong normalisation. More precisely, he has exhibited a simply typed term of the classical λ-calculus which starts an infinite derivation in the calculus $\lambda\sigma$ of [2], or in the calculus $\lambda\sigma_{\Uparrow}$ of [19]. This derivation goes through terms that contain compositions and cons.

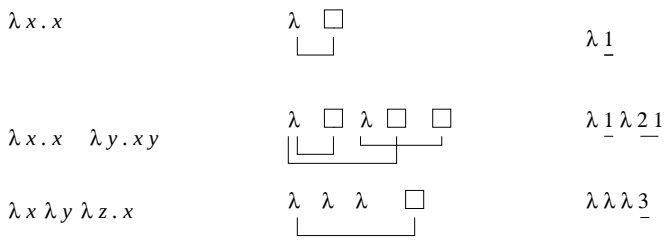

Figure 1: Bourbaki's style notations

2 The $\lambda\upsilon$-calculus

First let us remind the unfamiliar reader with de Bruijn's indices [12]. 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 Figure 1 we have represented a few terms. This is exactly the approach proposed by Bourbaki [4]. De Bruijn follows the same idea, for him variables are natural numbers, the *indices*. The index of a variable is the number of λ one crosses before the λ that binds that variable. For instance in $\lambda x \lambda y \lambda z \cdot x$ the index of the only occurrence of x is $\underline{3}$ and in the notation of λ-terms with indices, x will be replace 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 $\lambda x \cdot x$ in the usual λ-calculus, $\lambda\underline{1}(\lambda\underline{2}\ \underline{1})$ is equivalent to $\lambda x \cdot x\,(\lambda y \cdot y\ x)$ and $\lambda\lambda\lambda\underline{3}$ is equivalent to $\lambda x \lambda y \lambda z \cdot x$.

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 [12] 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 Figure 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*. 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

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

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

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

This form of (B) was introduced by Ehrhard [14], but we borrowed it from system τ of Rios [24]. 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).

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

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

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

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

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

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

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

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

(B)	$(\lambda a)b$	$\rightarrow$	$a[b/]$
(App)	$(ab)[s]$	$\rightarrow$	$a[s]b[s]$
(Lambda)	$(\lambda a)[s]$	$\rightarrow$	$\lambda(a[\Uparrow(s)])$
(FVar)	$\underline{1}[a/]$	$\rightarrow$	a
(RVar)	$\underline{n+1}[a/]$	$\rightarrow$	$\underline{n}$
(FVarLift)	$\underline{1}[\Uparrow(s)]$	$\rightarrow$	$\underline{1}$
(RVarLift)	$\underline{n+1}[\Uparrow(s)]$	$\rightarrow$	$\underline{n}[s][\uparrow]$
(VarShift)	$\underline{n}[\uparrow]$	$\rightarrow$	$\underline{n+1}$

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

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 $a[s][t] \rightarrow a[s \circ t]$. 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 behavior 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 Figure 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. The proof is easy and can be done with elementary interpretations (functions made of polynomials and exponentials) [22, 21]. It is given in Figure 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 by structural induction. $\lambda\upsilon$ has three sorts of objects, namely

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

$\lambda\upsilon$ does not introduce composition of substitutions. This makes the system simpler. 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" and/or laziness. 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 [22] for a discussion on the way to mechanize the introduction of composition and a comparison with other approaches. Among other systems of explicit substitutions, [22] introduces $\lambda\upsilon$ but does not prove any of its properties.

$[\![\underline{n}]\!]_1$	$=$	$2^{[\![n]\!]_1}$	$[\![\underline{n}]\!]_2$	$=$	$2^{[\![n]\!]_2}$
$[\![n+1]\!]_1$	$=$	$[\![n]\!]_1 + 1$	$[\![n+1]\!]_2$	$=$	$[\![n]\!]_2 + 1$
$[\![1]\!]_1$	$=$	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
$[\![a/]\!]_1$	$=$	$[\![a]\!]_1$			

Figure 3: Interpretations for proving the termination of υ

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 prove that $\lambda\upsilon$ preserves strong normalization. In Section 6 we introduce $C\lambda\xi\phi$.

3 Correction of the β-reduction in $\lambda\upsilon$

We write $\upsilon(a)$ the normal form of the term a w.r.t. υ. Notice that $\upsilon(a)$ is pure, that is $\upsilon(a) \in \mathbf{Terms}_\upsilon$. 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 $b = \upsilon(b')$. 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 [16]), 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.

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

where:

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

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.

$$\mu(a[\Uparrow^n(b/)]) \quad = \quad \sigma_n(\mu(a), \mu(b))$$

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

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

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

Proof: In order to prove the first assertion we consider each rule of υ.

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

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

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

- $\underline{n+1}[\Uparrow(s)] \underset{\upsilon}{\rightarrow} \underline{n}[s][\uparrow]$
  - 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 $n \leq k$,
    * $\sigma_{k+1}(\underline{n+1}, \mu(b)) = \underline{n}$ 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 $n \leq k$.
- $\underline{n}[\uparrow] \underset{\upsilon}{\rightarrow} \underline{n+1}$
  $\mu(\underline{n}[\uparrow]) = \tau_0^1(\mu(\underline{n})) = \tau_0^1(\underline{n}) = \underline{n+1} = \mu(\underline{n+1})$.

The proof of 2 comes from $\sigma_0(\mu(a), \mu(b)) = \mu(a[b/])$, by definition of μ. $\mu(a[b/]) = \upsilon(a[b/])$ comes from Part 1. □

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:

$$(\lambda x.(\lambda y.M)N)L \underset{\beta}{\rightarrow} (\lambda x.M[y := N])L \underset{\beta}{\rightarrow} M[y := N][x := L]$$

$$(\lambda x.(\lambda y.M)N)L \underset{\beta}{\rightarrow} (\lambda y.M[x := L])N[x := L] \underset{\beta}{\rightarrow} M[x := L][y := N[x := L]]$$

Indeed, if $y \notin FreeVar(L)$, we have [3] 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:

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

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

Thus, in order to get local confluence, we need to prove this υ-conversion, which we also call substitution lemma :

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

Lemmas 1 to 6 do the job. Notice that in the following we prove the subsitution lemma only for pure terms, but the result remains true for impure terms since if a is impure

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

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

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

For readability, we use the following abbreviation

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

Obviously,

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

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

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

Lemma 3 *For $i \geq 1$, $a[\Uparrow^i(\uparrow)][\Uparrow^i(b/)] \underset{\upsilon}{\overset{*}{\rightarrow}} a$.*

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

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

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

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

Lemma 5 has an important corollary.

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

By the corollary it entails, the next lemma which 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, Substitution Lemma has to be iterated.

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

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

Proof: The second statement comes from the fact that if $s \underset{B}{\rightarrow} t$, then $s = \Uparrow^i(a/)$ and $t = \Uparrow^i(b/)$ with $a \underset{B}{\rightarrow} b$ and $\upsilon(s) = \Uparrow^i(\upsilon(a)/)$ and $\upsilon(t) = \Uparrow^i(\upsilon(b)/)$. Hence from the first statement $\upsilon(a) \underset{\beta}{\rightarrow} \upsilon(b)$ and $\upsilon(s) \underset{\beta}{\rightarrow} \upsilon(t)$. Therefore we prove the statement for a and b. We proceed by noetherian induction on the lexicographic product of the two well-founded relations $(\underset{\upsilon}{\rightarrow}, \sqsupset)$, where $\sqsupset$ is the subterm relation. 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 $a_1 \underset{B}{\rightarrow} b_1$ by induction one gets $\upsilon(a_1) \overset{*}{\underset{\beta}{\rightarrow}} \upsilon(b_1)$ and

$$\upsilon(a_1 a_2) = \upsilon(a_1)\upsilon(a_2) \overset{*}{\underset{\beta}{\rightarrow}} \upsilon(b_1)\upsilon(a_2) = \upsilon(b_1 a_2).$$

  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

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

- 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 $a_1 \underset{B}{\rightarrow} b_1$ then $a_1[s_1]\dots[s_p] \underset{B}{\rightarrow} b_1[s_1]\dots[s_p]$, and as

$$(a_1 a_2)[s_1]\dots[s_p] \underset{\upsilon}{\rightarrow} a_1[s_1]\dots[s_p]a - 2[s_1]\dots[s_p] \sqsupset a_1[s_1]\dots[s_p],$$

    by induction

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

    and

$$\upsilon((a_1 a_2)[s_1]\dots[s_p]) = \upsilon(a_1[s_1]\dots[s_p])\upsilon(a_2[s_1]\dots[s_p])$$
$$\overset{*}{\underset{\beta}{\rightarrow}} \upsilon(b_1[s_1]\dots[s_p])\upsilon(a_2[s_1]\dots[s_p]) = \upsilon((b_1 a_2)[s_1]\dots[s_p]),$$

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

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

  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 $a_1 \underset{B}{\rightarrow} b_1$ or $s_i \underset{B}{\rightarrow} t_i$,

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

  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 $\underline{n}[s_1] \underset{\upsilon}{\overset{+}{\rightarrow}} a_1$ where a_1 is not a closure and

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

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

$$s_1 = \Uparrow^{j_1}(c_1/) \underset{B}{\rightarrow} t_1 = \Uparrow^{j_1}(d_1/).$$

  By case, one gets:
  - $n = 1$ and $j_1 = 0$,

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

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

    and the result comes by induction.
  - $n = k + 1$ and $j_1 = 0$,

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

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

    and the result is immediate.
  - $n = 1$ and $j_1 = j + 1$,

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

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

    and like above the result is immediate.

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

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

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

and the result comes by induction.

□

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

Proof: The proof of the theorem resembles the proof of a similar theorem by Abadi et al. in [2] which in turn was based on Hardin's interpretation method [18] 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 normalization

The essential difference between $\underset{\beta}{\rightarrow}$ on one hand and, $\underset{\lambda\upsilon}{\rightarrow}$ and $\underset{\lambda\sigma}{\rightarrow}$ 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 normalisable λ terms strongly $\lambda\upsilon$ normalisable or strongly $\lambda\sigma$ normalisable? The answer is "yes" for $\lambda\upsilon$ whereas Melliès gave a negative answer for $\lambda\sigma$. There are strongly β normalisable λ terms, even simply typed λ terms, which are not strongly $\lambda\sigma$ normalisable. The difficulty is that it could happen that $a \underset{B}{\rightarrow} b$ 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 tracked back. This will be expressed more formally through lemmas 8 and 9. First let us remind 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.

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

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

- $t_{|\epsilon} = t$
- *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* $t_{|p1} = s$.

Positions are compared by the prefix order. $\underline{p \text{ is a prefix of } q}$, 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.*

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 $t \underset{B,p}{\longrightarrow} t'$.

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: As 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 $d[\Uparrow^i(e/)]$ or $E\{d[\Uparrow^i(e/)]\}$. 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 $a_i \xrightarrow[\lambda\upsilon]{} a_{i+1}$, $1 \leq i \leq n-1$, 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 $e' \xrightarrow[\lambda\upsilon]{*} e$,*
2. *or $a_1 = t'\{d'[\Uparrow^j(e'/)]\}_{p'}$ and $e' \xrightarrow[\lambda\upsilon]{*} e$.*

Proof: By induction on n. The basic case $n = 1$ is immediate. Suppose :

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

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

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

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

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 $\underset{\lambda v}{\longrightarrow}$ is external (resp. internal) one writes $\underset{\lambda v}{\longrightarrow}^{ext}$ (resp. $\underset{\lambda v}{\longrightarrow}^{int}$).

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

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

We also use the contraposition: if $v(a)$ is strongly β normalisable, $v(a) = v(b)$, 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 $v(a)$ is strongly β normalisable, $v(a) = v(b)$ and $a \underset{\lambda v,p}{\longrightarrow}^{int} \cdot \underset{v,q}{\longrightarrow}^{ext} b$ then*
$a \overset{1,2}{\underset{v}{\longrightarrow}}^{ext} \cdot \overset{0,1,2}{\underset{\lambda v}{\longrightarrow}}^{int} b$.

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

- p and q are disjoints, then it is clear that we can permute the two rewrites, thus $a \underset{v}{\longrightarrow}^{ext} \cdot \underset{\lambda v}{\longrightarrow}^{int} b$.
- 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 v-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 $s' \underset{\lambda v}{\longrightarrow} s$ and $(c_1\ c_2)[s'] \underset{\lambda v}{\longrightarrow}^{int} (c_1\ c_2)[s] \underset{v}{\longrightarrow}^{ext} c_1[s]\ c_2[s]$, then
      $$(c_1\ c_2)[s'] \underset{v}{\longrightarrow}^{ext} c_1[s']\ c_2[s'] \underset{\lambda v}{\longrightarrow}^{int} c_1[s]\ c_2[s'] \underset{\lambda v}{\longrightarrow}^{int} c_1[s]\ c_2[s].$$
    - If $c_1' \underset{\lambda v}{\longrightarrow}^{int} c_1$ and $(c_1'\ c_2)[s] \underset{\lambda v}{\longrightarrow}^{int} (c_1\ c_2)[s] \underset{v}{\longrightarrow}^{ext} c_1[s]\ c_2[s]$, then
      $$(c_1'\ c_2)[s] \underset{v}{\longrightarrow}^{ext} c_1'[s]\ c_2[s] \underset{\lambda v}{\longrightarrow}^{int} c_1[s]\ c_2[s].$$
    - Similarly if $c_2 \underset{\lambda v}{\longrightarrow}^{int} c_2'$.

  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 β strongly β normalisable, 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 normalisable then by lemma 10 $\upsilon(A\{c'\}_q) \neq \upsilon(A\{c\}_q)$, and

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

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

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

□

Lemma 12 *Let a_1 be a strongly β normalisable 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 $\upsilon(a''_i) \underset{\beta}{\longrightarrow} \upsilon(a'_i) = \upsilon(a''_{i+1})$, hence

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

Since a_1 is strongly β normalisable, 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 application of Commutation Lemma, one can create by shifting $\mathsf{n}_{a_P} + 1$ external υ rewrites starting from a_p (see Figure 4), which is not possible.□

Definition 4 (Minimal $\lambda\upsilon$ derivation) *An infinite $\lambda\upsilon$ derivation $a_1 \underset{\lambda\upsilon,p_1}{\longrightarrow} a_2 \underset{\lambda\upsilon,p_2}{\longrightarrow} \cdots a_n \underset{\lambda\upsilon,p_n}{\longrightarrow} a_{n+1} \cdots$ starting from the pure term a_1 is* minimal *if for $a_1 \underset{\lambda\upsilon,p_1}{\longrightarrow} a_2 \underset{\lambda\upsilon,p_2}{\longrightarrow} \cdots a_n \underset{\lambda\upsilon,q_n}{\longrightarrow} b_{n+1} \cdots$ any other infinite derivation, for all p', $q_n \neq p_n p'$ (see Figure 5).*

That means that either p_n and q_n are disjoint positions or q_n is above p_n (i.e., $p_n = q_n q'$). In other words, one rewrites always the lowest possible redex to keep non termination. This does not preclude a possible $\lambda\upsilon$ redex in a_n at a position lower than p_n, but minimality says that rewriting at such a position leads to a finite derivation.

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

Definition 5 (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 normalisable, then a_1 is strongly $\lambda\upsilon$ normalisable.*

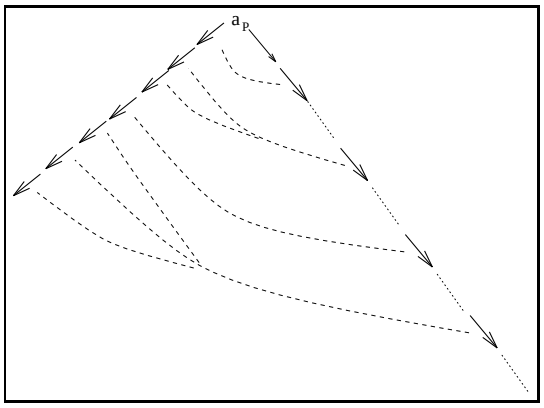

Figure 4: Shifting external υ rewrites. υ rewrites are emphasized

Proof: The proof is by contradiction. Suppose a_1 is a pure term strongly β normalisable, but non strongly $\lambda\upsilon$ normalisable. Let us consider a *minimal* infinite $\lambda\upsilon$ derivation $\mathcal{D}$ starting with the term a_1. By Lemma 12 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 $\mathcal{D}' = (a_1, \ldots, a_N, a'_{N+1}, \ldots, a'_i, \ldots)$ starting with the same N first terms and such that all the internal $\lambda\upsilon$ rewrites take place inside the closure $c[\Uparrow^i(b_N/)]$. $\mathcal{D}'$ is a minimal infinite derivation. In $\mathcal{D}'$ we have

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

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 in which there are necessarily B rewrites. 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 $J \leq N$ and a position p_J such that:

$$a_J = t'\{(\lambda c')b\}_{p_J} \underset{B,p_J}{\longrightarrow} t'\{c'[b/]\}_{p_J} = a'_J$$

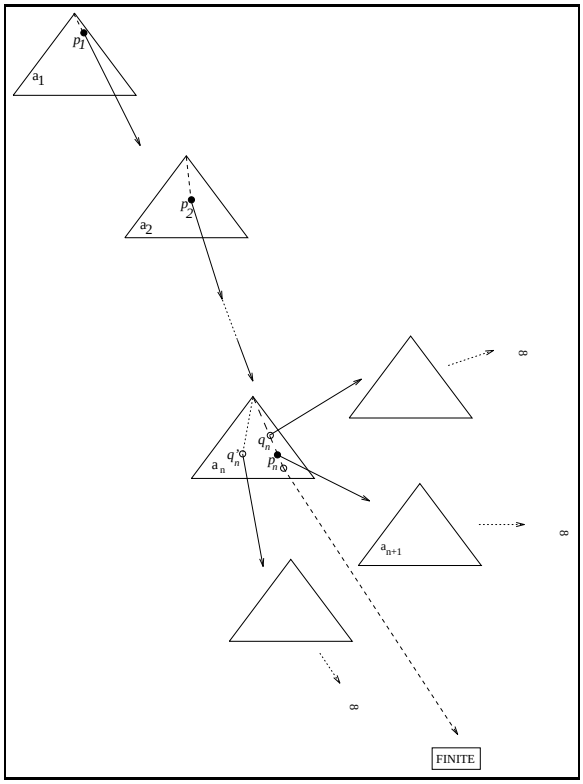

Figure 5: A minimal $\lambda\upsilon$ derivation

where t' is a context. Moreover $b \xrightarrow[\lambda\upsilon]{*} b_N$ and $t'\{c'[b/]\}_{p_J} \xrightarrow[\lambda\upsilon]{*} t\{c[\Uparrow^i(b_N/)]\}_p$. 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 $\mathcal{D}''$ all the rewrites from a_J are below p_J, especially the rewrite from $t'\{(\lambda c')b_N\}_{p_J}$ to $t'\{(\lambda c')b_{N+1}\}_{p_J}$. That contradicts the minimality of the derivation $\mathcal{D}'$. $\square$

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

6 The system $C\lambda\xi\phi$

In [13], 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 $\underline{\theta}$'s correspond to substitutions of the form $\Uparrow^i(\uparrow)$ and $\Uparrow^i(\updownarrow)$, where $\updownarrow$ is the subtitution 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:

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

To include this substitution in our notations, we propose to write $\underline{\theta_2}$ as $\updownarrow$ and to call it a *transposition*. The behavior 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 generalization 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 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 these 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 generalization 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 $\lambda\xi\phi$ (Figure 6). Let us call $\mathbf{Terms}_{\xi\phi}$ the set of terms described by the grammar:

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

$[\![\ ]\!]$ denotes substitutions that rename variables, those that are written $\underline{\theta}$ in de Bruijn's notations and $[\ /]$ 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 normalizing by using the lexicographic

(B)	$(\lambda a)b$	$\to$	$a[b/]$
(App$_1$)	$(a\ b)[c/]$	$\to$	$a[c/]\ b[c/]$
(Lambda$_1$)	$(\lambda a)[b/]$	$\to$	$\lambda(a[\![\updownarrow]\!][b[\![\uparrow]\!]/])$
(FVar)	$\underline{1}[a/]$	$\to$	a
(RVar)	$\underline{n+1}[a/]$	$\to$	$\underline{n}$
(App$_2$)	$(a\ b)[\![s]\!]$	$\to$	$a[\![s]\!]\ b[\![s]\!]$
(Lambda$_2$)	$(\lambda a)[\![s]\!]$	$\to$	$\lambda(a[\![\Uparrow(s)]\!])$
(FVarLift)	$\underline{1}[\![\Uparrow(s)]\!]$	$\to$	$\underline{1}$
(RVarLift)	$\underline{n+1}[\![\Uparrow(s)]\!]$	$\to$	$\underline{n}[\![s]\!][\![\uparrow]\!]$
(VarShift)	$\underline{n}[\![\uparrow]\!]$	$\to$	$\underline{n+1}$
(Transp$_1$)	$\underline{1}[\![\updownarrow]\!]$	$\to$	$\underline{2}$
(Transp$_2$)	$\underline{2}[\![\updownarrow]\!]$	$\to$	$\underline{1}$
(Transp$_3$)	$\underline{n+2}[\![\updownarrow]\!]$	$\to$	$\underline{n+2}$

Figure 6: The rewrite system $\lambda\xi\phi$

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:

$$\begin{array}{rcl} \iota(\underline{n}) & = & \underline{1} \\ \iota(a\ b) & = & \iota(a)\ \iota(b) \\ \iota(\lambda a) & = & \lambda(\iota(a)) \\ \iota(a[\![s]\!]) & = & \iota(a) \\ \iota(a[b/]) & = & \iota(a)[\iota(b)/] \end{array}$$

$a <_\iota b$ 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.

$\kappa_1(\underline{n})$	$=$	$2^{\kappa_1(n)}$	$\kappa_2(\underline{n})$	$=$	$2^{\kappa_2(n)}$
$\kappa_1(n+1)$	$=$	$\kappa_1(n)+1$	$\kappa_2(n+1)$	$=$	$\kappa_2(n)+1$
$\kappa_1(1)$	$=$	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			
$\kappa_1(a/)$	$=$	any			

Figure 7: Interpretations for proving the termination of υ

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 by classical methods that the rewriting relation $\overset{*}{\underset{\lambda\xi\phi}{\longrightarrow}}$ defined on $\mathbf{Terms}_{\xi\phi}$ and generated by $\lambda\xi\phi$ is confluent.

Actually the system $C\lambda\xi\phi$ can be presented without substitutions $[b/]$. The rule B is replaced by four rules $\{B_{App}, B_{Lambda}, B_F, B_R\}$ and the new system $\lambda\xi\phi_{Red}$ is given in Figure 8. This calculus has properties somewhat different form the other calculi of explicit substitutions and these properties should be carefully studied.

The systems $\lambda\upsilon$, $\lambda\xi\phi$ and $\lambda\xi\phi_{Red}$ share the same goal. All three 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 extreme simplicity suggested by Curry, but this is debatable.

7 Conclusion

$\lambda\upsilon$ has had several extensions, namely to handle more complex structures of environment or to include η-rules [5]. Theorem 2 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

(B_{App})	$(\lambda(a\ b))\ c$	$\to$	$(\lambda a)c\ (\lambda b)c$
(B_{Lambda})	$(\lambda(\lambda a))\ b$	$\to$	$\lambda((\lambda a[\![\updownarrow]\!])\ b[\![\uparrow]\!])$
(B_F)	$(\lambda\,\underline{1})\ a$	$\to$	a
(B_R)	$(\lambda\,\underline{n+1})\ a$	$\to$	$\underline{n}$
(App_2)	$(a\ b)[\![s]\!]$	$\to$	$a[\![s]\!]\ b[\![s]\!]$
(Lambda_2)	$(\lambda a)[\![s]\!]$	$\to$	$\lambda(a[\![\Uparrow(s)]\!])$
(FVarLift)	$\underline{1}[\![\Uparrow(s)]\!]$	$\to$	$\underline{1}$
(RVarLift)	$\underline{n+1}[\![\Uparrow(s)]\!]$	$\to$	$\underline{n}[\![s]\!][\![\uparrow]\!]$
(VarShift)	$\underline{n}[\![\uparrow]\!]$	$\to$	$\underline{n+1}$
(Transp_1)	$\underline{1}[\![\updownarrow]\!]$	$\to$	$\underline{2}$
(Transp_2)	$\underline{2}[\![\updownarrow]\!]$	$\to$	$\underline{1}$
(Transp_3)	$\underline{n+2}[\![\updownarrow]\!]$	$\to$	$\underline{n+2}$

Figure 8: The rewrite system $\lambda\xi\phi_{Red}$

discussion and for mentioning us his papers, 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 encouragements.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. Technical Report 54, Digital Systems Research Center, February 1990. Preliminary version in *Proc. of the 17th POPL conference, Orlando (Fla., USA).*

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

[3] 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.

[4] N. Bourbaki. *Éléments de mathématiques: Théories des ensembles*, volume 1. Hermann & Cie, 1954.

[5] D. Briaud. An explicit *Eta* rewrite rule. In M. Dezani, editor, *Int. Conf. on Typed Lambda Calculus and Applications*, 1995.

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

[7] P.-L. Curien. Categorical combinators. *Information and Control*, 69:188–254, 1986.

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

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

[10] 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.

[11] H. B. Curry and Feys. *Combinatory Logic*, volume 1. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1958.

[12] N. G. de Bruijn. 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, 1972.

[13] N. G. de Bruijn. 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, 1978.

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

[15] J. Field. On laziness and optimality in lambda interpreters: Tools for specification and analysis. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 1–15, San Fransisco, 1990. ACM.

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

[17] Th. Hardin. *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, 1987.

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

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

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

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

[22] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In Hans Boehm, editor, *Proceedings of the 21st Annual ACM Symposium on Principles Of Programming Languages, Portland (Or., USA)*, pages 60–69. ACM, 1994.

[23] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In M. Dezani, editor, *Int. Conf. on Typed Lambda Calculus and Applications*, 1995.

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

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399