# A $\lambda$-calculus à la de Bruijn with explicit substitutions 7th international conference on Programming Languages: Implementations, Logics and Programs, PLILP95, LNCS 982, pages 45-62

Fairouz Kamareddine and Alejandro Ríos

Department of Computing Science, 17 Lilybank Gardens, University of Glasgow, Glasgow G12 8QQ, Scotland, fax: +44 41 330 4913, *email*: fairouz@dcs.gla.ac.uk and rios@dcs.gla.ac.uk

**Abstract.** The aim of this paper is to present the $\lambda s$-calculus which is a very simple $\lambda$-calculus with explicit substitutions and to prove its confluence on closed terms and the preservation of strong normalisation of $\lambda$-terms. We shall prove strong normalisation of the corresponding calculus of substitution by translating it into the $\lambda\sigma$-calculus [ACCL91], and therefore the relation between both calculi will be made explicit. The confluence of the $\lambda s$-calculus is obtained by the "interpretation method" ([Har89], [CHL92]). The proof of the preservation of normalisation follows the lines of an analogous result for the $\lambda\upsilon$-calculus (cf. [BBLRD95]). The relation between $\lambda s$ and $\lambda\upsilon$ is also studied.

## 1 Introduction

Most literature on the $\lambda$-calculus considers substitution as an implicit operation. It means that the computations to perform substitution are usually described with operators which do not belong to the language of the $\lambda$-calculus. There has however been an interest in formalising substitution explicitly; various calculi including new operators to denote substitution and new rules to handle these operators have been proposed. Amongst these calculi we mention $C\lambda\xi\phi$ (cf. [dB78b]); the calculi of categorical combinators (cf. [Cur86]); $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\sigma_{SP}$ (cf. [ACCL91], [CHL92], [Río93]) referred to as the $\lambda\sigma$-family; $\lambda\upsilon$ (cf. [BBLRD95]), a descendant of the $\lambda\sigma$-family and $\varphi\sigma BLT$ (cf. [KN93]). The basic features of these systems of substitution depart quite extensively from the classical $\lambda$-calculus while in this paper we propose a system which remains as close as possible to it.

Furthermore, for the above systems either strong normalisation (SN) has not been studied (as for $C\lambda\xi\phi$ and $\varphi\sigma BLT$) or negative results (cf. [Mel95]) have been established concerning the preservation of SN (for the $\lambda\sigma$-family). In particular, these negative results imply that the simplest typed versions of these calculi are not SN. One positive and recent result concerning the preservation of SN is that for $\lambda\upsilon$ (cf. [BBLRD95]) for which, as far as we know, there is still work in progress.

As stated in [ACCL91], the $\lambda\sigma$-calculi and the calculi of combinators give full formal accounts of the process of computation and they make it easy to derive machines for the $\lambda$-calculus and to show the correctness of these machines. Hence, the $\lambda\sigma$-calculus is proposed as a step in closing the gap between the classical $\lambda$-calculus and concrete implementations. We believe that the $\lambda s$-calculus presented in this paper offers another possibility for closing this gap and, being closer to the $\lambda$-calculus, it preserves strong normalisation. Furthermore, we think that in the presence of the negative results of [Mel95] calculi like $\lambda s$ are worth studying.

The main interest in introducing the $\lambda s$-calculus is to provide a calculus of explicit substitutions which would have both the property of preserving strong normalisation and a confluent extension on open terms. As far as we know no such calculus has yet been proposed. There are calculi of explicit substitutions which are confluent on open terms: the $\lambda\sigma_{\Uparrow}$- calculus (cf. [HL89] and [CHL92]), but, as mentioned above, the non-preservation of strong normalisation for $\lambda\sigma_{\Uparrow}$ has recently been proved. There are also calculi which satisfy the preservation property: the $\lambda\upsilon$-calculus (cf. [BBLRD95]), but this calculus is not confluent on open terms. Moreover, in order to get a confluent extension, the introduction of a composition operator for substitutions seems unavoidable, but precisely this operator is the cause of the non-preservation of strong normalisation as shown in [Mel95]. We believe that the $\lambda s$-calculus, while preserving strong normalisation, could admit a confluent extension on open terms thanks to the fact that composition of substitutions (in the sense of the $\lambda\sigma$-calculi) could be handleld indirectly and in a very subtle way via a new family of rules mimicking the substitution lemma for the classical $\lambda$-calculus (see lemma 4 below).

Mention to a very close calculus to the $\lambda s$-calculus can be already found in [Cur86], exercise 1.2.7.2, where reference to previous unpublished notes of Y. Lafont is given. The $\varphi\sigma BLT$-calculus is also of this kind but the essential difference is that the redex is preserved when the $\beta$-rule is applied. The calculus we are going to study, we call it $\lambda s$, is obtained in a very natural way from the classical $\lambda$-calculus in de Bruijn notation: we just orientate the equalities defining the meta-operators of substitution and include them as new operators of the language.

We prove in this paper the confluence (CR) of the $\lambda s$-calculus on closed terms (these terms contain all terms of the classical $\lambda$-calculus) and the preservation of strong normalisation (terms which are strongly normalising in the $\lambda$-calculus are also strongly normalising in $\lambda s$). We also compare the $\lambda s$-calculus to $\lambda\sigma$ and $\lambda\upsilon$ via translation functions.

## 2 Preliminaries

We begin by giving a quick presentation of the $\lambda$-calculus à la de Bruijn and the $\lambda\sigma$-calculus.

### 2.1 The classical $\lambda$-calculus in de Bruijn notation

We shall assume the reader familiar with de Bruijn indices (see [dB72] and [dB78a]) which can be explained via the following two examples: $\lambda x\lambda y.xy$ is written using de Bruijn indices as $\lambda\lambda(\mathtt{2}\,\mathtt{1})$ and $\lambda x\lambda y.(x(\lambda z.zx))y$ is written as $\lambda\lambda(\mathtt{2}(\lambda(\mathtt{1}\,\mathtt{3}))\mathtt{1})$.

Remark here that variables are removed and are replaced by natural numbers. These numbers are informative as to the $\lambda$ which binds the occurrence of the variable.

Hence in the second example, the same $x$ was translated into 2 and 3 according to the different positions, whereas $z$ and $y$ become the same de Bruijn index, 1.

The interest in introducing de Bruijn indices is that they avoid clashes of variable names and therefore neither $\alpha$-conversion nor Barendregt's convention are needed. Here is the $\lambda$-calculus à la de Bruijn.

**Definition 1** *We define $\Lambda$, the* set of terms with de Bruijn indices*, as follows:*

$$\Lambda ::= \mathbb{N} \mid (\Lambda\Lambda) \mid (\lambda\Lambda)$$

*We use $a, b, \ldots$ to range over $\Lambda$ and $m, n, \ldots$ to range over $\mathbb{N}$ (positive natural numbers). Furthermore, we assume the usual conventions about parentheses and avoid them when no confusion occurs. Throughout the whole article, $a = b$ is used to mean that $a$ and $b$ are syntactically identical.*

When rewriting a term $a$ with variable names into its de Bruijn version, we consider $a$ to be a subterm of $\lambda x_1 \ldots x_k.a$ where $x_1 \ldots x_k$ are all the free variables of $a$. For instance: $\lambda x.xyz$ becomes $\lambda\mathtt{123}$ (or $\lambda\mathtt{132}$) and $(\lambda x.xy)y$ becomes $(\lambda\mathtt{12})\mathtt{1}$. In order for this to work independently of the order in which the free variables appear, we assume that the set of variable names is ordered and call this ordered set *the free variable list*. For example, if the list was $\cdots, z, y, x$ then the term to be translated should be prefixed with $\cdots, \lambda z, \lambda y, \lambda x$ before its translation. Thus, $\lambda x.yz$ translates as $\lambda\mathtt{34}$ whereas $\lambda x.zy$ translates as $\lambda\mathtt{43}$. Now check that $(\lambda x\lambda y.zxy)(\lambda x.yx)$ translates as $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31})$ and that $\lambda u.z(\lambda x.yx)u$ translates to $\lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$.

In order to define $\beta$-reduction à la de Bruijn, we must define the substitution of a variable by a term $b$ in a term $a$. Therefore, we must identify amongst the numbers of a term $a$ those that correspond to the variable that is being substituted for and we need to update the term to be substituted in order to preserve the correct bindings of its variables.

For example, translating $(\lambda x\lambda y.zxy)(\lambda x.yx) \to_\beta \lambda u.z(\lambda x.yx)u$ to de Bruijn notation we get $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \to_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$. But if we simply replace $\mathtt{2}$ in $\lambda\mathtt{521}$ by $\lambda\mathtt{31}$ we get $\lambda\mathtt{5}(\lambda\mathtt{31})\mathtt{1}$, which is not correct. We needed to decrease $\mathtt{5}$ as one $\lambda$ disappeared and to increment the free variables of $\lambda\mathtt{31}$ as they occur within the scope of one more $\lambda$.

For incrementing the free variables we need a family of updating functions:

**Definition 2** *The* updating functions $U_k^i : \Lambda \to \Lambda$ *for $k \geq 0$ and $i \geq 1$ are defined inductively as follows:*

$$U_k^i(ab) = U_k^i(a)\, U_k^i(b) \qquad U_k^i(\mathtt{n}) = \begin{cases} \mathtt{n} + \mathtt{i} - 1 & \textit{if } \; n > k \\ \mathtt{n} & \textit{if } \; n \leq k\,. \end{cases}$$
$$U_k^i(\lambda a) = \lambda(U_{k+1}^i(a))$$

The intuition behind $U_k^i$ is the following: $k$ tests for free variables and $i - 1$ is the value by which a variable, if free, must be incremented.

Now we define the family of meta-substitution functions:

**Definition 3** *The* meta-substitutions at level $i$*, for $i \geq 1$, of a term $b \in \Lambda$ in a term $a \in \Lambda$, denoted $a\{\!\{\mathtt{i} \leftarrow b\}\!\}$, is defined inductively on $a$ as follows:*

$$(a_1 a_2)\{\!\{\mathtt{i} \leftarrow b\}\!\} \;=\; (a_1\{\!\{\mathtt{i} \leftarrow b\}\!\})\,(a_2\{\!\{\mathtt{i} \leftarrow b\}\!\}) \qquad \mathtt{n}\{\!\{\mathtt{i} \leftarrow b\}\!\} \;=\; \begin{cases} \mathtt{n}-\mathtt{1} & \textit{if} \;\; n > i \\ U_0^i(b) & \textit{if} \;\; n = i \\ \mathtt{n} & \textit{if} \;\; n < i\,. \end{cases}$$

$$(\lambda a)\{\!\{\mathtt{i} \leftarrow b\}\!\} \;=\; \lambda(a\{\!\{\mathtt{i}+\mathtt{1} \leftarrow b\}\!\})$$

Ultimately, the intention is to define $(\lambda a)b \to_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$ (see definition 4 below). The first two equalities propagate the substitution through applications and abstractions and the last one carries out the substitution of the intended variable (when $n = i$) by the updated term. If the variable is not the intended one it must be decreased by 1 if it is free (case $n > i$) beacuse one $\lambda$ has disappeared, whereas if it is bound (case $n < i$) it must remain unaltered.

It is easy to check that $(\lambda\mathtt{521})\{\!\{\mathtt{1} \leftarrow (\lambda\mathtt{31})\}\!\} = \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$. This will mean $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \to_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$.

The following lemmas establish the properties of the meta-substitutions and updating functions. The Meta-substitution and Distribution lemmas are crucial to prove the confluence of $\lambda s$. The proofs of lemmas 1 - 6 are obtained by induction on $a$. Furthermore, the proof of lemma 3 requires lemma 2 with $p = 0$; the proof of lemma 4 uses lemmas 1 and 3 both with $k = 0$; finally, lemma 5 with $p = 0$ is needed to prove lemma 6.

**Lemma 1** *For* $k < n \le k + i$ *we have:* $U_k^i(a) = U_k^{i+1}(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$.

**Lemma 2** *For* $p \le k < j + p$ *we have:* $U_k^i(U_p^j(a)) = U_p^{j+i-1}(a)$.

**Lemma 3** *For* $i \le n - k$ *we have:* $U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n}-\mathtt{i}+\mathtt{1} \leftarrow b\}\!\})$.

**Lemma 4 (Meta-substitution lemma)** *For* $1 \le i \le n$ *we have:*

$$a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n}+\mathtt{1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n}-\mathtt{i}+\mathtt{1} \leftarrow c\}\!\}\}\!\}$$

**Lemma 5** *For* $m \le k + 1$ *we have:* $U_{k+p}^i(U_p^m(a)) = U_p^m(U_{k+p+1-m}^i(a))$.

**Lemma 6 (Distribution lemma)** *For* $n \le k + 1$ *we have:*

$$U_k^i(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U_{k+1}^i(a)\{\!\{\mathtt{n} \leftarrow U_{k-n+1}^i(b)\}\!\}.$$

**Definition 4** $\beta$-reduction *is the least compatible relation on $\Lambda$ generated by:*

*($\beta$-rule)* $\qquad (\lambda a)\,b \to_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$

*The* $\lambda$-calculus à la de Bruijn, *abbreviated* $\lambda$-calculus *is the reduction system whose only rewriting rule is $\beta$.*

**Theorem 1** *The $\lambda$-calculus à la de Bruijn is confluent.*

**Proof:** The $\lambda$-calculus with de Bruijn indices and the classical $\lambda$-calculus with variable names are isomorphic (cf. [Mau85]). The confluence of the latter (cf. [Bar84] thm. 3.2.8) is hence transportable to the $\lambda$-calculus à la de Bruijn.

A proof which does not use the mentioned isomorphism is given in [Río93] (corol. 3.6) as a corollary of a more general result concerning the $\lambda\sigma$-calculus. $\square$

Finally, the following lemma ensures the good passage of the $\beta$-rule through the meta-substitutions and the $U_k^i$. It is crucial for the proof of the confluence of $\lambda s$.

**Lemma 7** *Let $a$, $b$, $c$, $d \in \Lambda$.*

1. *If* $c \to_\beta d$ *then* $U^i_k(c) \to_\beta U^i_k(d)$ .
2. *If* $c \to_\beta d$ *then* $a\{\!\{\mathtt{i} \leftarrow c\}\!\} \twoheadrightarrow_\beta a\{\!\{\mathtt{i} \leftarrow d\}\!\}$ .
3. *If* $a \to_\beta b$ *then* $a\{\!\{\mathtt{i} \leftarrow c\}\!\} \to_\beta b\{\!\{\mathtt{i} \leftarrow c\}\!\}$ .

**Proof:**

1. Induction on $c$. We just check the interesting case which arises when $c = c_1 c_2$ and the reduction takes place at the root, i.e. $c_1 = (\lambda a)$, $c_2 = b$ and $d = a\{\!\{\mathtt{1} \leftarrow b\}\!\}$:
$$U^i_k((\lambda a)b) = (\lambda(U^i_{k+1}(a)))U^i_k(b) \to_\beta U^i_{k+1}(a)\{\!\{\mathtt{1} \leftarrow U^i_k(b)\}\!\} \overset{L\,6}{=} U^i_k(a\{\!\{\mathtt{1} \leftarrow b\}\!\})$$
2. Induction on $a$ using 1 above.
3. Induction on $a$. The interesting case is again $a = (\lambda d)e$ and $b = d\{\!\{\mathtt{1} \leftarrow e\}\!\}$:
$$((\lambda d)e)\{\!\{\mathtt{i} \leftarrow c\}\!\} = (\lambda(d\{\!\{\mathtt{i+1} \leftarrow c\}\!\}))(e\{\!\{\mathtt{i} \leftarrow c\}\!\}) \to_\beta$$
$$(d\{\!\{\mathtt{i+1} \leftarrow c\}\!\})\{\!\{\mathtt{1} \leftarrow e\{\!\{\mathtt{i} \leftarrow c\}\!\}\}\!\} \overset{L\,4}{=} (d\{\!\{\mathtt{1} \leftarrow e\}\!\})\{\!\{\mathtt{i} \leftarrow c\}\!\} \qquad \square$$

## 2.2 The $\lambda\sigma$-calculus

The $\lambda\sigma$-calculus ([ACCL91]) is a formalism which enables explicit substitution. Its syntax is two-sorted: the sort `term` of *terms* and the sort `substitution` of *explicit substitutions*. These can be interpreted as a sequence of terms and the result of executing a substitution in a term can be interpreted as the term obtained by replacing the occurrences of the $n$-th index of de Bruijn in the term by the $n$-th term of the sequence. This intuitive interpretation is developped and illustrated with many examples in [ACCL91].

Here are the syntax and the rules of the calculus:

**Definition 5** *The syntax of the* $\lambda\sigma$-calculus *is given by:*

$$\begin{array}{ll} \textbf{Terms} & \Lambda\sigma^t ::= \mathtt{1} \mid \Lambda\sigma^t\Lambda\sigma^t \mid \lambda\Lambda\sigma^t \mid \Lambda\sigma^t[\Lambda\sigma^s] \\ \textbf{Substitutions} & \Lambda\sigma^s ::= id \mid \uparrow \mid \Lambda\sigma^t \cdot \Lambda\sigma^s \mid \Lambda\sigma^s \circ \Lambda\sigma^s \end{array}$$

*The set, denoted* $\lambda\sigma$, *of rules of the* $\lambda\sigma$-calculus *is the following:*

| | |
|---|---|
| *(Beta)* | $(\lambda a)\, b \longrightarrow a\,[b \cdot id]$ |
| *(VarId)* | $\mathtt{1}\,[id] \longrightarrow \mathtt{1}$ |
| *(VarCons)* | $\mathtt{1}\,[a \cdot s] \longrightarrow a$ |
| *(App)* | $(a\, b)[s] \longrightarrow (a\,[s])\,(b\,[s])$ |
| *(Abs)* | $(\lambda a)[s] \longrightarrow \lambda(a\,[\mathtt{1} \cdot (s \circ \uparrow)])$ |
| *(Clos)* | $(a\,[s])[t] \longrightarrow a\,[s \circ t]$ |
| *(IdL)* | $id \circ s \longrightarrow s$ |
| *(ShiftId)* | $\uparrow \circ\, id \longrightarrow \uparrow$ |
| *(ShiftCons)* | $\uparrow \circ\, (a \cdot s) \longrightarrow s$ |
| *(Map)* | $(a \cdot s) \circ t \longrightarrow a\,[t] \cdot (s \circ t)$ |
| *(Ass)* | $(s_1 \circ s_2) \circ s_3 \longrightarrow s_1 \circ (s_2 \circ s_3)$ |

*The set of rules of the* $\sigma$-calculus *is* $\lambda\sigma - \{(Beta)\}$. *We use* $a, b, c, \ldots$ *to range over* $\Lambda\sigma^t$ *and* $s, t, \ldots$ *to range over* $\Lambda\sigma^s$.

**Notation 1** *For a given set of rules* $\mathcal{R}$ *we take* $\rightarrow_{\mathcal{R}}$ *to be the reduction relation of the* $\mathcal{R}$*-calculus (i.e. the least compatible relation containing the rules of* $\mathcal{R}$*).*

*We take* $\twoheadrightarrow_{\mathcal{R}}$ *to be the derivation relation of the* $\mathcal{R}$*-calculus (i.e. the least reflexive and transitive relation containing* $\rightarrow_{\mathcal{R}}$*) and we denote by* $\rightarrow^{+}_{\mathcal{R}}$ *the transitive closure of* $\rightarrow_{\mathcal{R}}$ *(i.e. the least transitive relation containing* $\rightarrow_{\mathcal{R}}$*).*

*For any two relations* $\rightarrow$ *and* $\rightarrow'$*, by* $a \rightarrow . \rightarrow' b$ *we mean* $(\exists c)(a \rightarrow c \rightarrow' b)$.

*Finally, we write* $a \twoheadrightarrow^{n}_{\mathcal{R}} b$ *to mean that the derivation from* $a$ *to* $b$ *consists of* $n$ *steps of* $\mathcal{R}$*-reduction.*

*When it will be clear from the context, we may omit the subscript* $\mathcal{R}$.

*We recall that* $a$ *is a* $\mathcal{R}$-normal form *if there exists no* $b$ *such that* $a \rightarrow_{\mathcal{R}} b$*. We say that* $c$ *is a* $\mathcal{R}$-normal form of $d$ *if* $c$ *is a* $\mathcal{R}$*-normal form and* $d \twoheadrightarrow_{\mathcal{R}} c$.

**Notation 2** *For every substitution* $s$ *we define the* iteration of the composition of $s$ *inductively as* $s^1 = s$ *and* $s^{n+1} = s \circ s^n$. *We use the convention* $s^0 = id$.

Note that the only de Bruijn index used is $\mathtt{1}$, but we can code $\mathtt{n}$ by the term $\mathtt{1}[\uparrow^{n-1}]$. By so doing, we have $\Lambda \subset \Lambda\sigma^t$.

$\beta$-reduction of the $\lambda$-calculus is interpreted in the $\lambda\sigma$-calculus in two steps. The first, obtained by the application of *(Beta)*, consists in generating the substitution. The second step executes the propagation of this substitution, using the set of the $\sigma$-rules, until the concerned variables are reached. The reader is invited to check that $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \twoheadrightarrow_{\lambda\sigma} \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$.

We summarize now the properties of the $\sigma$- and $\lambda\sigma$-calculi:

**Theorem 2** *The* $\sigma$*-calculus is strongly normalising (SN) and confluent (CR).*

**Proof:** We know three proofs of strong normalisation: [HL86], [CHR92] and [Zan94]. Local confluence (WCR) is ensured by analysis of critical pairs (cf. [Río93], annex B), and the Knuth-Bendix theorem ([KB70] or [Hue80]). Now Newman's lemma, which states that SN+WCR yields CR ([Bar84], prop. 3.1.25), guarantees confluence. □

**Theorem 3** *The* $\lambda\sigma$*-calculus is confluent.*

**Proof:** See [ACCL91], theorem 3.2. This proof is based on the confluence of $\sigma$, that of the $\lambda$-calculus and the technique of interpretation. □

## 3 The $\lambda s$-calculus and its confluence

The idea is to handle explicitly the meta-operators defined in definitions 2 and 3. Therefore, the syntax of the $\lambda s$-calculus is obtained by adding to the syntax of the $\lambda$-calculus à la de Bruijn two families of operators :

- $\{\sigma^i\}_{i\geq 1}$ This family is meant to denote the explicit substitution operators. Each $\sigma^i$ is an infix operator of arity 2 and $a\,\sigma^i b$ has as intuitive meaning the term $a$ where all free occurrences of the variable corresponding to the de Bruijn number $i$ are to be substituted by the term $b$.

– $\{\varphi^i_k\}_{k\geq 0 \ i\geq 1}$ This family is meant to denote the updating functions necessary when working with de Bruijn numbers to fix the variables of the term to be substituted.

**Definition 6** *The set of* terms of the $\lambda s$-calculus*, noted $\Lambda s$ is given as follows:*

$$\Lambda s ::= \mathbb{N} \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s\, \sigma^i \Lambda s \mid \varphi^i_k \Lambda s \qquad \textit{where} \quad i \geq 1\,, \quad k \geq 0\,.$$

*We take $a$, $b$, $c$ to range over $\Lambda s$. A term of the form $a\,\sigma^i b$ is called a* closure*. Furthermore, a term containing neither $\sigma$'s nor $\varphi$'s is called a* pure term*.*

The $\lambda s$-calculus should carry out, besides $\beta$-reduction, the computations of updating and substitution explicitly. For that reason we include, besides the rule mimicking the $\beta$-rule (*$\sigma$-generation*), a set of rules which are the equations in definitions 2 and 3 oriented from left to right.

**Definition 7** *The* $\lambda s$-calculus *is given by the following rewriting rules:*

| | |
|---|---|
| *$\sigma$-generation* | $(\lambda a)\, b \longrightarrow a\,\sigma^1\, b$ |
| *$\sigma$-$\lambda$-transition* | $(\lambda a)\,\sigma^i b \longrightarrow \lambda(a\,\sigma^{i+1}\, b)$ |
| *$\sigma$-app-transition* | $(a_1\, a_2)\,\sigma^i b \longrightarrow (a_1\,\sigma^i b)\,(a_2\,\sigma^i b)$ |
| *$\sigma$-destruction* | $\mathtt{n}\,\sigma^i b \longrightarrow \begin{cases} \mathtt{n}-1 & \textit{if} \quad n > i \\ \varphi^i_0\, b & \textit{if} \quad n = i \\ \mathtt{n} & \textit{if} \quad n < i \end{cases}$ |
| *$\varphi$-$\lambda$-transition* | $\varphi^i_k(\lambda a) \longrightarrow \lambda(\varphi^i_{k+1}\, a)$ |
| *$\varphi$-app-transition* | $\varphi^i_k(a_1\, a_2) \longrightarrow (\varphi^i_k\, a_1)\,(\varphi^i_k\, a_2)$ |
| *$\varphi$-destruction* | $\varphi^i_k\,\mathtt{n} \longrightarrow \begin{cases} \mathtt{n}+\mathtt{i}-1 & \textit{if} \quad n > k \\ \mathtt{n} & \textit{if} \quad n \leq k \end{cases}$ |

*We use $\lambda s$ to denote this set of rules. The calculus of substitutions associated with the $\lambda s$-calculus is the rewriting system whose rules are $\lambda s - \{\sigma\text{-generation}\}$ and we call it* $s$-calculus.

In order to give the translation into the $\lambda\sigma$-calculus we give the following two definitions.

**Definition 8** *For $k \geq 0$ and $i \geq 1$ we define $s_{k\,i} = \mathtt{1} \cdot \mathtt{2} \cdot \ldots \cdot \mathtt{k} \cdot \uparrow^{k+i-1}$ (we use the convention $s_{0\,i} = \uparrow^{i-1}$ and hence $s_{0\,1} = id$).*

**Definition 9** *Let $b \in \Lambda\sigma^t$, we define a family of substitutions $(b_k)_{k\geq 1}$ as follows:*

$$b_1 = b[id] \cdot id \qquad b_2 = \mathtt{1} \cdot b[\uparrow] \cdot \uparrow \qquad \ldots \qquad b_{i+1} = \mathtt{1} \cdot \mathtt{2} \cdot \ldots \cdot \mathtt{i} \cdot b[\uparrow^i] \cdot \uparrow^i \qquad \ldots$$

Using the rules *(Map)*, *(Clos)*, *(Ass)* and *(IdL)* it is easy to verify that:

**Remark 1** $\mathtt{1} \cdot (b_i \circ \uparrow) \twoheadrightarrow_\sigma b_{i+1}$ *and* $\mathtt{1} \cdot (s_{k\,i} \circ \uparrow) \twoheadrightarrow_\sigma s_{k+1\,i}$.

**Definition 10** *The* translation function $T : \Lambda s \to \Lambda\sigma^t$ *is defined by:*

$$\begin{array}{lll} T(\mathtt{n}) = \mathtt{n} & T(a\,b) = T(a)T(b) & T(a\,\sigma^i b) = T(a)[T(b)_i] \\ & T(\lambda a) = \lambda(T(a)) & T(\varphi^i_k a) = T(a)[s_{k\,i}] \end{array}$$

**Theorem 4** *If $a \to_s b$ then $T(a) \overset{+}{\to}_\sigma T(b)$.*

**Proof:** Induction on $a$. We just check, as an example, the case $a = \mathtt{n}\,\sigma^i c$ when the reduction takes place at the root:

$$T(\mathtt{n}\,\sigma^i c) = \mathtt{n}[T(c)_i] \overset{+}{\to}_\sigma \begin{cases} \mathtt{n-1} = T(\mathtt{n-1}) & \text{if } \; n > i \\ T(c)[\uparrow^{i-1}] = T(\varphi^i_0 c) & \text{if } \; n = i \\ \mathtt{n} = T(\mathtt{n}) & \text{if } \; n < i \end{cases}$$ □

**Corollary 1** *The reduction $\to_s$ is strongly normalising.*

**Proof:** Use Theorem 2. □

**Remark 2** *The reduction $\to_s$ is locally confluent.*

**Proof:** There are no critical pairs and the theorem of Knuth-Bendix applies trivially. □

**Corollary 2** *The reduction $\to_s$ is confluent.*

**Proof:** Newman's Lemma (see proof of thm. 2) yields CR. □

These corollaries guarantee the existence and unicity of $s$-normal forms ($s$-nf), which we shall use to interpret the $\lambda s$-calculus in the $\lambda$-calculus. We shall denote the $s$-nf of a term $a$ by $s(a)$. The following lemma characterizes $s$-normal forms.

**Lemma 8** *The set of $s$-normal forms is exactly $\Lambda$.*

**Proof:** Check first by induction on $a$ that $a\,\sigma^i b$ and $\varphi^i_k a$ are not normal forms. Then check by induction on $a$ that if $a$ is an $s$-nf then $a \in \Lambda$. Conclude by observing that every term in $\Lambda$ is in $s$-nf. □

As there are no $s$-rules whose left-hand side is an application or an abstraction, the following properties of $s$-normal forms (which will be used throughout without explicit mention) are immediate.

**Lemma 9** *For all $a$, $b \in \Lambda s$: $s(a\,b) = s(a)s(b)$ and $s(\lambda a) = \lambda(s(a))$ .*

We establish now the relation between the operators $\sigma^i$ and $\varphi^i_k$ and the meta-operators of the classical de Bruijn setting: $\_\,\{\!\{\mathtt{i} \leftarrow \_\}\!\}$ and $U^i_k$.

**Lemma 10** *For all $a$, $b \in \Lambda s$ we have:*

$$s(\varphi^i_k a) = U^i_k(s(a)) \;\; \textit{and} \;\; s(a\,\sigma^i b) = s(a)\{\!\{\mathtt{i} \leftarrow s(b)\}\!\} \, .$$

**Proof:** Prove the first equality for terms in $s$-nf, i.e. use an inductive argument on $c \in \Lambda$ to show $s(\varphi^i_k c) = U^i_k(s(c))$. Let now $a \in \Lambda s$, $s(\varphi^i_k a) = s(\varphi^i_k s(a)) = U^i_k(s(s(a))) = U^i_k(s(a))$.
Prove the second claim similarly using the first claim. $\Box$

We give now the key result that allows us to use the Interpretation Method in order to get confluence: the good passage of the $\sigma$-generation rule to the $s$-normal forms.

**Proposition 1** *Let* $a, b \in \Lambda s$ *, if* $a \to_{\sigma-gen} b$ *then* $s(a) \twoheadrightarrow_\beta s(b)$ *.*

**Proof:** Induction on $a$. We just study the interesting cases.

$a = c\,d$ **:** If the reduction takes place within $c$ or $d$ just use the inductive hypothesis (IH). The interesting case is when $c = \lambda e$ and hence $b = e\,\sigma^1 d$:

$$s((\lambda e)d) = (\lambda s(e))(s(d)) \to_\beta s(e)\{\!\{\mathtt{1} \leftarrow s(d)\}\!\} \overset{L\,10}{=} s(e\,\sigma^1 d)$$

$a = c\,\sigma^i d$ **:** If the reduction takes place within c, i.e. $c \to_{\sigma-gen} e$ and $b = e\,\sigma^i d$, then

$$s(c\,\sigma^i d) \overset{L\,10}{=} s(c)\{\!\{\mathtt{i} \leftarrow s(d)\}\!\} \overset{IH\,\&\,L\,7.3}{\twoheadrightarrow_\beta} s(e)\{\!\{\mathtt{i} \leftarrow s(d)\}\!\} \overset{L\,10}{=} s(e\,\sigma^i d)$$

If the reduction takes place within $d$, lemma 7.2 applies.

$a = \varphi^i_k c$ **:** The reduction must take place within $c$. Use lemma 10 and lemma 7.1. $\Box$

Now, the following corollaries are immediate.

**Corollary 3** *Let* $a, b \in \Lambda s$ *, if* $a \twoheadrightarrow_{\lambda s} b$ *then* $s(a) \twoheadrightarrow_\beta s(b)$ *.*

**Corollary 4 (Soundness)** *Let* $a, b \in \Lambda$ *, if* $a \twoheadrightarrow_{\lambda s} b$ *then* $a \twoheadrightarrow_\beta b$ *.*

Finally, before proving confluence, we verify that the $\lambda s$-calculus is powerful enough to simulate $\beta$-reduction.

**Lemma 11 (Simulation of $\beta$-reduction)** *Let* $a, b \in \Lambda$*, if* $a \to_\beta b$ *then* $a \twoheadrightarrow_{\lambda s} b$ *.*

**Proof:** Induction on $a$. As usual the interesting case is when $a = (\lambda c)d$ and $b = c\{\!\{\mathtt{1} \leftarrow d\}\!\}$:

$$(\lambda c)d \to_{\sigma-gen} c\sigma^1 d \twoheadrightarrow_s s(c\sigma^1 d) \overset{L10}{=} s(c)\{\!\{\mathtt{1} \leftarrow s(d)\}\!\} \overset{c,d\in\Lambda}{=} c\{\!\{\mathtt{1} \leftarrow d\}\!\} \qquad \Box$$

**Theorem 5** *The* $\lambda s$*-calculus is confluent.*

**Proof:** We interpret the $\lambda s$-calculus into the $\lambda$-calculus via $s$-normalisation. We have:

$$\begin{tikzcd}
 & b \arrow[rr, two heads, "s"] & & s(b) \arrow[dr, "\beta"] & \\
a \arrow[ur, "\lambda s"] \arrow[rr, two heads, "s"] \arrow[dr, "\lambda s"'] & & s(a) \arrow[ur, "\beta"] \arrow[dr, "\beta"'] & \text{Thm. 1} & d \\
 & c \arrow[rr, two heads, "s"] & & s(c) \arrow[ur, "\beta"'] &
\end{tikzcd}$$

The existence of the arrows $s(a) \twoheadrightarrow_\beta s(b)$ and $s(a) \twoheadrightarrow_\beta s(c)$ is guaranteed by corollary 3. We can close the diagram thanks to the confluence of $\lambda$-calculus and finally lemma 11 ensures $s(b) \twoheadrightarrow_{\lambda s} d$ and $s(b) \twoheadrightarrow_{\lambda s} d$ proving thus CR for the $\lambda s$-calculus. □

## 4 The $\lambda s$-calculus preserves strong normalisation

In this section we shall prove that every term $a$ which is strongly normalising (all its derivations are finite) in the $\lambda$-calculus (denoted $a \in \beta$-SN) is also strongly normalising in the $\lambda s$-calculus (denoted $a \in \lambda s$-SN). In particular, pure simply typed terms will be strongly normalising in $\lambda s$.

This result is not valid for the $\lambda\sigma$-calculus, neither for its confluent version $\lambda\sigma_{\Uparrow}$, neither for the calculus of categorical combinators, as was recently proved by the counterexamples of Melliès (see [Mel95]). But there is work in progress to prove it for the $\lambda\upsilon$-calculus (cf. [BBLRD95]).

The natural translation of $\lambda s$ into $\lambda\upsilon$ which we shall give in this section is good enough to ensure the preservation of strong normalisation for $\lambda s$ as soon as the result will be obtained for $\lambda\upsilon$. However, the general idea in [BBLRD95] can be adapted for the preservation of strong normalisation of $\lambda s$.

We begin by presenting the $\lambda\upsilon$-calculus and the translation.

**Definition 11** *The* terms of the $\lambda\upsilon$-calculus *are given by the following syntax:*

$$\begin{array}{ll}
\textbf{Terms} & \Lambda\upsilon^t ::= \mathbb{N} \mid \Lambda\upsilon^t\Lambda\upsilon^t \mid \lambda\Lambda\upsilon^t \mid \Lambda\upsilon^t[\Lambda\upsilon^s] \\
\textbf{Substitutions} & \Lambda\upsilon^s ::= \uparrow \mid \Uparrow(\Lambda\upsilon^s) \mid \Lambda\upsilon^t/
\end{array}$$

*The set, denoted $\lambda\upsilon$, of rules of the* $\lambda\upsilon$-calculus *is the following:*

| | |
|---|---|
| *(Beta)* | $(\lambda a)\, b \longrightarrow a\,[b/]$ |
| *(App)* | $(a\, b)[s] \longrightarrow (a\,[s])\,(b\,[s])$ |
| *(Abs)* | $(\lambda a)[s] \longrightarrow \lambda(a\,[\Uparrow(s)])$ |
| *(FVar)* | $\mathtt{1}\,[a/] \longrightarrow a$ |
| *(RVar)* | $\mathtt{n+1}\,[a/] \longrightarrow \mathtt{n}$ |
| *(FVarLift)* | $\mathtt{1}\,[\Uparrow(s)] \longrightarrow \mathtt{1}$ |
| *(RVarLift)* | $\mathtt{n+1}\,[\Uparrow(s)] \longrightarrow \mathtt{n}[s][\uparrow]$ |
| *(VarShift)* | $\mathtt{n}[\uparrow] \longrightarrow \mathtt{n+1}$ |

*We use $a, b, c, \ldots$ to range over $\Lambda\upsilon^t$ and $s, t, \ldots$ to range over $\Lambda\upsilon^s$.*

This choice of operators and rules is based on the idea of expressing the *(Beta)*-rule as economically as possible. In the $\lambda\sigma$-calculus it reads $(\lambda a)\, b \to a[b \cdot id]$ and requires the introduction of the operators $\cdot$ and $id$. Just one unary operator can do the job, this operator is denoted by $/$ in the $\lambda\upsilon$-syntax. Hence $a/$ plays the role of the $\lambda\sigma$-term $a \cdot id$. Now the *(Abs)*-rule, which in $\lambda\sigma$ reads $(\lambda a)[s] \to \lambda(a\,[\mathtt{1} \cdot (s \circ \uparrow)])$, must be modified to avoid the use of $\cdot$ which is no longer available. Hence the introduction of $\Uparrow$ and the intuitive interpretation of $\Uparrow(s)$ as the $\lambda\sigma$-term $\mathtt{1} \cdot (s \circ \uparrow)$.

**Notation 3** *For $a \in \Lambda\upsilon^t$ and $s \in \Lambda\upsilon^s$ we denote:*
- $\Uparrow^i(s) = \Uparrow(\Uparrow(\ldots \Uparrow(s)\ldots))$ *(i times). By $\Uparrow^0(s)$ we mean $s$.*
- $a[s]^i = a[s][s]\ldots[s]$ *(i times). By $a[s]^0$ we mean $a$.*

**Definition 12** *The "natural" translation $S : \Lambda s \to \Lambda\upsilon^t$ is given by:*

$$\begin{array}{lll} S(\mathtt{n}) = \mathtt{n} & S(a\,b) = S(a)S(b) & S(a\,\sigma^i b) = S(a)[\Uparrow^{i-1}(S(b)/)] \\ & S(\lambda a) = \lambda(S(a)) & S(\varphi^i_k a) = S(a)[\Uparrow^k(\uparrow)]^{i-1} \end{array}$$

It is easy to check by induction on $a$ that $a \to_{\sigma-gen} b$ implies $S(a) \xrightarrow{+}_{\lambda\upsilon} S(b)$ and that $a \to_s b$ implies $S(a) \twoheadrightarrow_{\lambda\upsilon} S(b)$. Therefore, preservation of SN for $\lambda\upsilon$ yields preservation of SN for $\lambda s$.

**Notation 4** *We write $a \underset{p}{\to} b$ in order to denote that $p$ is the occurrence of the redex which is contracted. Therefore $a \underset{\epsilon}{\to} b$ means that the reduction takes place at the root. If no specification is made the reduction must be understood as a $\lambda s$-reduction.*

*Furthermore, we denote by $\prec$ the prefix order between occurrences of a term. Therefore if $p$, $q$ are occurrences of the term $a$ such that $p \prec q$, and we write $a_p$ (resp. $a_q$) for the subterm of $a$ at occurrence $p$ (resp. $q$), then $a_q$ is a subterm of $a_p$.*

For example, if $a = \mathtt{2}\sigma^3((\lambda\mathtt{1})\mathtt{4})$, we have $a_1 = \mathtt{2}$, $a_2 = (\lambda\mathtt{1})\mathtt{4}$, $a_{21} = \lambda\mathtt{1}$, $a_{211} = \mathtt{1}$, $a_{22} = \mathtt{4}$. Since, for instance, $2 \prec 21$, $a_{21}$ is a subterm of $a_2$.

The aim of the three following lemmas is to assert that all the $\sigma$'s in the last term of a derivation beginning with a $\lambda$-term must have been created at some previous step by a $\sigma$-generation and to trace the history of these closures. The first of them explains this situation for a one-step derivation where the redex is at the root:

**Lemma 12** *If $a \xrightarrow[\epsilon]{} C[d\sigma^i e]$ then one of the following must hold:*

1. *$a = (\lambda d)e$, $C = \Box$ (a hole) and $i = 1$.*
2. *$a = C'[d'\sigma^j e]$ for some context $C'$, some term $d'$ and some natural $j$.*

**Proof:** We must check for every rule $a \to b$ in $\lambda s$ that if $d\sigma^i e$ occurs in $b$ then $a = (\lambda d)e$ or $d'\sigma^j e$ occurs in $a$. We just check the interesting rules:

**($\sigma$-gen) :** If $b = d\sigma^i e$ then $i = 1$ and $a = (\lambda d)e$. Otherwise $b = b_1\sigma^1 b_2$ and $d\sigma^i e$ occurs either in $b_1$ or in $b_2$, both cases are immediate since now $a = (\lambda b_1)b_2$.

**($\sigma$-$\lambda$-trans) :** If $b = \lambda(d\sigma^i e)$ then $i > 1$ and $a = (\lambda d)\sigma^{i-1}e$; take $d' = \lambda d$ and $j = i - 1$. If the occurrence of $d\sigma^i e$ is in a deeper position (i.e. if $d\sigma^i e$ is a proper subterm of $b$), proceed as in the previous case.

**($\sigma$-app-trans) :** If, for instance, $b = (c\,\sigma^i e)(d\sigma^i e)$ then $a = (c\,d)\,\sigma^i e$; take $d' = c\,d$. For deeper positions the result is straightforward. □

The second lemma generalizes the previous one.

**Lemma 13** *If $a \to C[d\sigma^i e]$ then one of the following must hold:*

1. *$a = C[(\lambda d)e]$ and $i = 1$.*
2. *$a = C'[d'\sigma^j e']$ where $e' = e$ or $e' \to e$.*

**Proof:** Induction on $a$, using lemma 12 for the reductions at the root. □

Finally, the third lemma gives the result for arbitrary derivations.

**Lemma 14** *Let $a_1 \to \ldots \to a_n \to a_{n+1} = C[d\sigma^i e]$ then $a_1 = C'[d'\sigma^j e']$ or there exists $k \leq n$ such that $a_k = C'[(\lambda d')e']$ and $a_{k+1} = C'[d'\sigma^1 e']$. In both cases $e' \twoheadrightarrow e$.*

**Proof:** Induction on $n$ and use the previous lemma. □

We shall define now the notions of internal and external reductions. The intuitive meaning of an internal reduction is a reduction that takes place somewhere at the right of a $\sigma^i$ operator. An external reduction is a reduction that is not internal.

We give a definition by induction. Another possibility is to define first the notion of internal and external position (occurrence) as is done in [BBLRD95].

**Definition 13** *The reduction $\xrightarrow{\texttt{int}}_{\lambda s}$ is defined by the following rules:*

$$\frac{a \longrightarrow_{\lambda s} b}{c\,\sigma^i a \xrightarrow{\texttt{int}}_{\lambda s} c\,\sigma^i b} \qquad \frac{a \xrightarrow{\texttt{int}}_{\lambda s} b}{a\,c \xrightarrow{\texttt{int}}_{\lambda s} b\,c} \qquad \frac{a \xrightarrow{\texttt{int}}_{\lambda s} b}{c\,a \xrightarrow{\texttt{int}}_{\lambda s} c\,b}$$

$$\frac{a \xrightarrow{\texttt{int}}_{\lambda s} b}{\lambda a \xrightarrow{\texttt{int}}_{\lambda s} \lambda b} \qquad \frac{a \xrightarrow{\texttt{int}}_{\lambda s} b}{a\,\sigma^i c \xrightarrow{\texttt{int}}_{\lambda s} b\,\sigma^i c} \qquad \frac{a \xrightarrow{\texttt{int}}_{\lambda s} b}{\varphi^i_k a \xrightarrow{\texttt{int}}_{\lambda s} \varphi^i_k b}$$

*Therefore, $\xrightarrow{\texttt{int}}_{\lambda s}$ is the least compatible relation closed under* $\dfrac{a \longrightarrow_{\lambda s} b}{c\,\sigma^i a \xrightarrow{\texttt{int}}_{\lambda s} c\,\sigma^i b}$ .

**Definition 14** *The reduction* $\xrightarrow{\texttt{ext}}_s$ *is defined by induction. The axioms are the rules of the s-calculus and the inference rules are the following:*

$$\frac{a \xrightarrow{\texttt{ext}}_s b}{a\,c \xrightarrow{\texttt{ext}}_s b\,c} \quad \frac{a \xrightarrow{\texttt{ext}}_s b}{c\,a \xrightarrow{\texttt{ext}}_s c\,b} \quad \frac{a \xrightarrow{\texttt{ext}}_s b}{\lambda a \xrightarrow{\texttt{ext}}_s \lambda b} \quad \frac{a \xrightarrow{\texttt{ext}}_s b}{a\,\sigma^i c \xrightarrow{\texttt{ext}}_s b\,\sigma^i c} \quad \frac{a \xrightarrow{\texttt{ext}}_s b}{\varphi^i_k a \xrightarrow{\texttt{ext}}_s \varphi^i_k b}$$

*Analogously, an external $\sigma$-generation is defined by the axiom* $(\lambda a)b \xrightarrow{\texttt{ext}}_{\sigma-gen} a\sigma^1 b$ *and the five inference rules stated above where* $\xrightarrow{\texttt{ext}}_s$ *is replaced by* $\xrightarrow{\texttt{ext}}_{\sigma-gen}$ .

Note that the inference rules $\dfrac{a \xrightarrow{\texttt{ext}}_s b}{c\,\sigma^i a \xrightarrow{\texttt{ext}}_s c\,\sigma^i b}$ and $\dfrac{a \xrightarrow{\texttt{ext}}_{\sigma-gen} b}{c\,\sigma^i a \xrightarrow{\texttt{ext}}_{\sigma-gen} c\,\sigma^i b}$ are excluded from the definitions of external $s$-reduction and external $\sigma$-generation, respectively. Thus, as we expected, external reductions will not occur at the right of a $\sigma^i$ operator. This will permit us to write $\xrightarrow{+}_\beta$ instead of $\twoheadrightarrow_\beta$ in proposition 2.

**Remark 3** *By inspecting the inference rules one checks immediately that:*

1. *If* $a \xrightarrow{\texttt{int}}_{\lambda s} \lambda b$ *then* $a = \lambda c$ *and* $c \xrightarrow{\texttt{int}}_{\lambda s} b$ .
2. *If* $a \xrightarrow{\texttt{int}}_{\lambda s} b\,c$ *then* $a = d\,e$ *and* $((d \xrightarrow{\texttt{int}}_{\lambda s} b$ *and* $e = c)$ *or* $(e \xrightarrow{\texttt{int}}_{\lambda s} c$ *and* $d = b))$.
3. $a \xrightarrow{\texttt{int}}_{\lambda s} \mathtt{n}$ *is impossible.*

The following lemma is a slight but essential variation of proposition 1. A step of *external* $\sigma$-generation is studied now and the lemma ensures that we have *at least one step of $\beta$-reduction* between the corresponding $s$-normal forms.

**Proposition 2** *Let* $a, b \in \Lambda s$. *If* $a \xrightarrow{\texttt{ext}}_{\sigma-gen} b$ *then* $s(a) \xrightarrow{+}_\beta s(b)$ .

**Proof:** Induction on $a$. The lines of this proof follow the proof of proposition 1. Now, the point is that in the case $a = c\,\sigma^i d$, the reduction cannot take place within $d$ because it is external, and this is the only case that forced us to consider the reflexive-transitive closure because of lemma 7.2. □

The following lemma plays a fundamental rôle in lemma 16 and hence in the Preservation theorem.

**Lemma 15 (Commutation lemma)** *Let* $a, b \in \Lambda s$ *such that* $s(a) \in \beta$*-SN and* $s(a) = s(b)$. *If* $a \xrightarrow{\texttt{int}}_{\lambda s} . \xrightarrow{\texttt{ext}}_s b$ *then* $a \xrightarrow{\texttt{ext}}{}^{+}_{s} . \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} b$ .

**Proof:** By a careful induction on $a$ while analysing the positions of the redexes. The detailed proof is given in the appendix. □

**Lemma 16** *Let $a$ be a strongly normalising term of the $\lambda$-calculus. For every infinite $\lambda s$-derivation* $a \to_{\lambda s} b_1 \to_{\lambda s} \cdots \to_{\lambda s} b_n \to_{\lambda s} \cdots$, *there exists $N$ such that for $i \geq N$ all the reductions* $b_i \to_{\lambda s} b_{i+1}$ *are internal.*

**Proof:** An infinite $\lambda s$-derivation must contain infinite $\sigma$-generations, since the $s$-calculus is SN. The first rule must also be a $\sigma$-generation beacuse $a$ is a pure term. We can thus write the derivation as follows:

$$a = a_1 \rightarrow_{\sigma-gen} a'_1 \twoheadrightarrow_s a_2 \rightarrow_{\sigma-gen} a'_2 \twoheadrightarrow_s \cdots \twoheadrightarrow_s a_n \rightarrow_{\sigma-gen} a'_n \twoheadrightarrow_s \cdots$$

By proposition 2, there must be only a finite number of external $\sigma$-generations (otherwise we can construct an infinite $\beta$-derivation contradicting the hypothesis $a \in \beta$-SN). Therefore there exists $P$ such that for $i \geq P$ we have $a_i \xrightarrow{\texttt{int}}_{\sigma-gen} a'_i$. Furthermore, by proposition 1, $s(a_i) \twoheadrightarrow_\beta s(a'_i)$ for all $i$, and therefore

$$a = s(a) = s(a_1) \twoheadrightarrow_\beta s(a'_1) = s(a_2) \twoheadrightarrow_\beta s(a'_2) = \cdots = s(a_n) \twoheadrightarrow_\beta s(a'_n) = \cdots$$

Since $a$ is $\beta$-SN, we conclude that $s(a_i)$ is $\beta$-SN for all $i$ and that therefore there exists $M \geq P$ such that for $i \geq M$ we have $s(a_i) = s(a'_i)$. We claim that there exists $N \geq M$ such that for $i \geq N$ all the $s$-rewrites are also internal. Otherwise, there would be an infinity of external $s$-rewrites and at least one copy of each of these external rewrites can be brought, by the Commutation Lemma, in front of $a_M$, and so generate an infinite $s$-derivation beginning at $a_M$, which is a contradiction. This intuitive idea can be formally stated as:

**Fact:** *If there exists an infinite derivation* $a_M \xrightarrow{\texttt{ext}}\!\!\!\!\twoheadrightarrow^n_s b \longrightarrow\!\!\!\!\twoheadrightarrow^m c \xrightarrow{\texttt{ext}}_s d \longrightarrow \cdots$ *where all the rewrites in* $b \longrightarrow\!\!\!\!\twoheadrightarrow c$ *are either* $\xrightarrow{\texttt{ext}}_s$ *or* $\xrightarrow{\texttt{int}}_{\lambda s}$ *, then there exists an infinite derivation* $a_M \xrightarrow{\texttt{ext}}\!\!\!\!\twoheadrightarrow^{n+1}_s b' \longrightarrow\!\!\!\!\twoheadrightarrow d \longrightarrow \cdots$*.*

**Proof of Fact:** The fact is easily proved by induction on $m$, using the Commutation lemma. We remark that $M$ has been so chosen in order to satisfy the hypothesis of this lemma. □

In order to prove the Preservation Theorem we need two definitions.

**Definition 15** *An infinite $\lambda s$-derivation $a_1 \rightarrow \cdots \rightarrow a_n \rightarrow \cdots$ is* minimal *if for every step of reduction $a_i \underset{p}{\rightarrow}_{\lambda s} a_{i+1}$, every other derivation beginning with $a_i \underset{q}{\rightarrow}_{\lambda s} a'_{i+1}$ where $p \prec q$, is finite.*

The intuitive idea of a minimal derivation is that if one rewrites at least one of its steps within a subterm of the actual redex, then an infinite derivation is impossible.

**Definition 16** Skeletons *are defined by the following syntax:*

$$\textbf{Skeletons } K ::= \mathbb{N} \mid K\,K \mid \lambda K \mid K\,\sigma^i[.] \mid \varphi^i_k K$$

*The* skeleton of a term *a is defined by induction as follows:*

$$\begin{array}{lll} Sk(\texttt{n}) = \texttt{n} & Sk(a\,b) = Sk(a)Sk(b) & Sk(a\,\sigma^i b) = Sk(a)\,\sigma^i[.] \\ & Sk(\lambda a) = \lambda Sk(a) & Sk(\varphi^i_k a) = \varphi^i_k Sk(a) \end{array}$$

**Remark 4 (Properties of the skeleton)**

1. *Each occurrence of* [.] *in the skeleton of a corresponds to an external closure of the term a (by external closure, we mean a closure that is not at the right of any other closure), and this correspondence is a bijection.*

2. *Internal closures (those which are at the right of another closure) vanish in the skeleton.*
3. *If* $a \xrightarrow{\mathtt{int}}_{\lambda s} b$ *then* $Sk(a) = Sk(b)$.

**Theorem 6 (Preservation of strong normalisation)** *If a pure term is strongly normalising in the* $\lambda$*-calculus, then it is strongly normalising in the* $\lambda s$*-calculus.*

**Proof:** Suppose $a$ is a strongly normalising term in the $\lambda$-calculus, but not $\lambda s$-SN. Let us consider a minimal infinite $\lambda s$-derivation $\mathcal{D} \,:\, a \to a_1 \to \cdots \to a_n \to \cdots$. By lemma 16, there exists $N$, such that for $i \geq N$, $a_i \to a_{i+1}$ is internal. Therefore, by the previous remark, $Sk(a_i) = Sk(a_{i+1})$ for $i \geq N$. As there are only a finite number of closures in $Sk(a_N)$ and as the reductions within these closures are independent, an infinite subderivation of $\mathcal{D}$ must take place within the same and unique closure in $Sk(a_N)$ and , evidently, this subderivation is also minimal. Let us call it $\mathcal{D}'$ and let $C$ be the context such that $a_N = C[c\,\sigma^i d]$ and $c\,\sigma^i d$ is the closure where $\mathcal{D}'$ takes place. Therefore we have:

$$\mathcal{D}' \;:\; a_N = C[c\,\sigma^i d] \xrightarrow{\mathtt{int}}_{\lambda s} C[c\,\sigma^i d_1] \xrightarrow{\mathtt{int}}_{\lambda s} \cdots \xrightarrow{\mathtt{int}}_{\lambda s} C[c\,\sigma^i d_n] \xrightarrow{\mathtt{int}}_{\lambda s} \cdots$$

Since $a$ is a pure term, lemma 14 ensures the existence of $I \leq N$ such that

$$a_I = C'[(\lambda c')d'] \to a_{I+1} = C'[c'\sigma^1 d'] \;\text{ and }\; d' \twoheadrightarrow d.$$

But let us consider the following derivation:

$$\mathcal{D}'' \;:\; a \twoheadrightarrow a_I = C'[(\lambda c')d'] \twoheadrightarrow C'[(\lambda c')d] \to C'[(\lambda c')d_1] \to \cdots \to C'[(\lambda c')d_n] \to \cdots$$

In this infinite derivation the redex in $a_I$ is within $d'$ which is a proper subterm of $(\lambda c')d'$, whereas in $\mathcal{D}$ the redex in $a_I$ is $(\lambda c')d'$ and this contradicts the minimality of $\mathcal{D}$. □

## 5 Conclusion

There are two unsolved problems concerning $\lambda s$ which we want to discuss briefly.

The first of them is the confluence of $\lambda s$ on open terms. We remind that by open terms we mean terms which admit variables of sort `term`, namely open terms are given by the following syntax:

$$\Lambda s_{op} ::= \mathbf{V} \mid \mathbb{N} \mid \Lambda s_{op}\Lambda s_{op} \mid \lambda\Lambda s_{op} \mid \Lambda s_{op}\,\sigma^i \Lambda s_{op} \mid \varphi^i_k \Lambda s_{op} \qquad \textit{where} \quad i \geq 1\,, \quad k \geq 0$$

and where $\mathbf{V}$ stands for a set of variables, over which $X$, $Y$, ... range.

Working with open terms one loses confluence as shown by:

$$((\lambda X)Y)\sigma^1 \mathtt{1} \to (X\sigma^1 Y)\sigma^1 \mathtt{1} \qquad ((\lambda X)Y)\sigma^1 \mathtt{1} \to ((\lambda X)\sigma^1 \mathtt{1})(Y\sigma^1 \mathtt{1})$$

Moreover, local confluence is lost. Since $((\lambda X)\sigma^1 \mathtt{1})(Y\sigma^1 \mathtt{1}) \twoheadrightarrow (X\sigma^2 \mathtt{1})\sigma^1(Y\sigma^1 \mathtt{1})$, the solution to the problem seems easy: add to $\lambda s$ rules obtained by orienting the equalities given by the lemmas 1 - 6. For instance, the rule corresponding to the Meta-substitution lemma (lemma 4) would solve the critical pair in our counterexample.

We believe that by adding these rules to $\lambda s$ we could get local confluence. But the problem of confluence of the extended system seems a difficult one. As a matter of fact it was one of the open questions in Lafont's notes.

We note here that the same problem for $\lambda\sigma$ leads to the introduction of $\lambda\sigma_{SP}$, the latter being locally confluent on open terms but confluent only on semi-closed terms (terms with variables of sort `term` but without variables of sort `substitution`)(cf. [Río93]).

The second problem we wish to present is the strong normalisation of the typed version of $\lambda s$. We believe, as suggested by P.-L. Curien, that a translation, say $\mathcal{T}$, similar to the one presented in [CR91] would allow us to use the preservation of SN obtained in this paper to get the desired result. But this translation should have the additional property $\mathcal{T}(a) \twoheadrightarrow_{\lambda s} a$, and this is not the case, but possibly could be if the system considered is the extension of $\lambda s$ suggested by the first problem. This would permit to get SN for this extended calculus, and get as a corollary SN for the simply typed $\lambda s$.

**Acknowledgements.** We thank Pierre-Louis Curien for his useful discussions and comments. We are also grateful for the interactions we had with Pierre Lescanne, Roel Bloo and Antonio Bucciarelli. Last but not least, we would like to thank Jeroen Krabbendam and the anonymous referees for their useful remarks.

This work was carried out under EPSRC grant GR/K25014.

## References

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

[Bar84] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics (revised edition)*. North Holland, 1984.

[BBLRD95] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Personal communication*, 1995.

[CHL92] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report RR 1617, INRIA, Rocquencourt, 1992. To appear in the JACM.

[CHR92] P.-L. Curien, T. Hardin, and A. Ríos. Strong Normalization of Substitutions in Proceedings of MFCS'92. In I.M. Havel and V. Koubek, editors, *Lecture Notes in Computer Science 629*, pages 209–217, Prague, 1992. Springer-Verlag.

[CR91] P.-L. Curien and A. Ríos. Un résultat de Complétude pour les substitutions explicites. *Comptes Rendus de l'Académie des Sciences*, 312, I:471–476, 1991.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986. Revised edition : Birkhäuser (1993).

[dB72] N. de Bruijn. Lambda-Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[dB78a] N. de Bruijn. Lambda-Calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, 40:348–356, 1978.

[dB78b] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical Report TH-Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978.

[Har89] T. Hardin. Confluence Results for the Pure Strong Categorical Logic CCL : $\lambda$-calculi as Subsystems of CCL. *Theoretical Computer Science*, 65(2):291–342, 1989.

[HL86] T. Hardin and A. Laville. Proof of Termination of the Rewriting System SUBST on CCL. *Theoretical Computer Science*, 46:305–312, 1986.

[HL89] T. Hardin and J.-J. Lévy. A Confluent Calculus of Substitutions. *France-Japan Artificial Intelligence and Computer Science Symposium*, December 1989.

[Hue80] G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the Association for Computing Machinery*, 27:797–821, October 1980.

[KB70] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[KN93] F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.

[Mau85] M. Mauny. *Compilation des langages fonctionnels dans les combinateurs catégoriques. Application au langage ML.* PhD thesis, Université Paris VII, Paris, France, 1985.

[Mel95] P.-A. Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate in Proceedings of TLCA'95. *Lecture Notes in Computer Science*, 902, 1995.

[Río93] A. Ríos. *Contribution à l'étude des $\lambda$-calculs avec substitutions explicites.* PhD thesis, Université de Paris 7, 1993.

[Zan94] H. Zantema. Termination of term rewriting: interpretation and type elimination. *J. Symbolic Computation*, 17(1):23–50, 1994.

## Appendix

**Proof of lemma 15 :** By induction on $a$. The basic case which is $a = \mathtt{n}$ is trivial.

$a = a_1 a_2$ **:** Since we are dealing with an internal reduction there are only two possibilities: $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'$ or $a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_2'$. Let us study, for instance, the first one. Since the external reduction cannot take place at the root (because no $s$-rule contracts an application), there are only two cases left:

- $a = a_1 a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' a_2 \xrightarrow{\mathtt{ext}}_{s} a_1'' a_2$ and $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' \xrightarrow{\mathtt{ext}}_{s} a_1''$. Let us verify that the hypotheses are valid for $a_1$ and $a_1''$ in order to use the IH.
  1. If $s(a_1 a_2) = s(a_1)s(a_2)$ is $\beta$-SN, so is $s(a_1)$.
  2. If $s(a_1 a_2) = s(a_1'' a_2)$ then $s(a_1)s(a_2) = s(a_1'')s(a_2)$ and so $s(a_1) = s(a_1'')$.

  Therefore, by IH $a_1 \xrightarrow{\mathtt{ext}}{}_{s}^{+} . \overset{\mathtt{int}}{\twoheadrightarrow}_{\lambda s} a_1''$ , and then $a_1 a_2 \xrightarrow{\mathtt{ext}}{}_{s}^{+} . \overset{\mathtt{int}}{\twoheadrightarrow}_{\lambda s} a_1'' a_2$ .
- $a = a_1 a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' a_2 \xrightarrow{\mathtt{ext}}_{s} a_1' a_2'$ with $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'$ and $a_2 \xrightarrow{\mathtt{ext}}_{s} a_2'$. We can simply commute the reductions: $a = a_1 a_2 \xrightarrow{\mathtt{ext}}_{s} a_1 a_2' \xrightarrow{\mathtt{int}}_{\lambda s} a_1' a_2'$.

$a = \lambda a_1$ **:** The reduction must take place within $a_1$. There is no difficulty in checking the hypotheses and then using the IH.

$a = a_1 \sigma^i a_2$ **:** Again, as we are analysing an internal reduction, two cases arise:

$a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'$ **:** The external reduction can only take place within $a_1$ or at the root:

- $a = a_1 \sigma^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' \sigma^i a_2 \xrightarrow{\mathtt{ext}}_{s} a_1'' \sigma^i a_2$ and $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' \xrightarrow{\mathtt{ext}}_{s} a_1''$. Let us verify that the hypotheses are valid for $a_1$ and $a_1''$ in order to use the IH.
  1. We know that $s(a_1 \sigma^i a_2)$ is $\beta$-SN. If we suppose that $s(a_1)$ is not $\beta$-SN, there exists an infinite derivation $s(a_1) \to_\beta c_1 \to_\beta \ldots \to_\beta c_n \to_\beta \ldots$ in $\Lambda$, i.e. with $c_k = s(c_k)$ for all $k$. Now, by lemma 7.3 we obtain an infinite derivation:

  $$s(a_1 \sigma^i a_2) = s(a_1)\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\} \to_\beta s(c_1)\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\} \to_\beta \ldots$$

  which is a contradiction. Therefore $s(a_1)$ is $\beta$-SN.

2. We know that $s(a_1\,\sigma^i a_2) = s(a_1''\,\sigma^i a_2)$ and, by corollary 3, $s(a_1) \twoheadrightarrow_\beta s(a_1'')$. If $s(a_1) \overset{+}{\twoheadrightarrow}_\beta s(a_1'')$, then by lemma 7.3 we have

$$s(a_1)\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\} \overset{+}{\twoheadrightarrow}_\beta s(a_1'')\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\}\ .$$

But this is a contradiction since

$$s(a_1)\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\} = s(a_1\,\sigma^i a_2) = s(a_1''\,\sigma^i a_2) = s(a_1'')\{\!\{\mathtt{i} \leftarrow s(a_2)\}\!\}$$

and we are assuming that $s(a_1\,\sigma^i a_2)$ is $\beta$-SN. Therefore, $s(a_1) = s(a_1'')$.

We can now apply the IH to $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1' \xrightarrow{\mathtt{ext}}_s a_1''$ to obtain $a_1 \xrightarrow{\mathtt{ext}}{}^{+}_{s} \,.\, \overset{\mathtt{int}}{\twoheadrightarrow}_{\lambda s} a_1''$, and hence $a_1\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}{}^{+}_{s} \,.\, \overset{\mathtt{int}}{\twoheadrightarrow}_{\lambda s} a_1''\,\sigma^i a_2$.

– $a = a_1\,\sigma^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s b$, $a_1 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'$, and the external reduction takes place at the root. We study the three possible rules:
  - *($\sigma$-$\lambda$-trans)* : We have $a_1' = \lambda c'$ and $b = \lambda(c'\sigma^{i+1}a_2)$. Remark 3.1 ensures that $a_1 = \lambda c$ and $c \xrightarrow{\mathtt{int}}_{\lambda s} c'$. We can then commute:

$$a = a_1\,\sigma^i a_2 = (\lambda c)\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \lambda(c\sigma^{i+1}a_2) \xrightarrow{\mathtt{int}}_{\lambda s} \lambda(c'\sigma^{i+1}a_2) = b$$

  - *($\sigma$-app-trans)* : We have $a_1' = c'd'$ and $b = (c'\,\sigma^i a_2)(d'\,\sigma^i a_2)$. Remark 3.2 ensures that $a_1 = c\,d$ and, either $c \xrightarrow{\mathtt{int}}_{\lambda s} c'$ and $d = d'$, or $d \xrightarrow{\mathtt{int}}_{\lambda s} d'$ and $c = c'$. In both cases we can commute as in the previous case.
  - *($\sigma$-dest)* : We have $a_1' = \mathtt{n}$ and this is impossible by remark 3.3.

$a_2 \to_{\lambda s} a_2'$ **:** As in the previous case, the external reduction can take place within $a_1$ or at the root:

– $a = a_1\,\sigma^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1\,\sigma^i a_2' \xrightarrow{\mathtt{ext}}_s a_1'\,\sigma^i a_2'$, $a_2 \to_{\lambda s} a_2'$ and $a_1 \xrightarrow{\mathtt{ext}}_s a_1'$. We can commute to obtain: $a = a_1\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s a_1'\,\sigma^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1'\,\sigma^i a_2'$.

– $a = a_1\,\sigma^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} a_1\,\sigma^i a_2' \xrightarrow{\mathtt{ext}}_s b$, $a_2 \to_{\lambda s} a_2'$ and the external reduction takes place at the root. We study the three possible rules:
  - *($\sigma$-$\lambda$-trans)* : We have $a_1 = \lambda c$ and $b = \lambda(c\sigma^{i+1}a_2')$. We can commute:

$$a = a_1\,\sigma^i a_2 = (\lambda c)\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \lambda(c\sigma^{i+1}a_2) \xrightarrow{\mathtt{int}}_{\lambda s} \lambda(c\sigma^{i+1}a_2') = b$$

  - *($\sigma$-app-trans)* : We have $a_1 = c\,d$ and $b = (c\,\sigma^i a_2')(d\,\sigma^i a_2')$. We can commute generating two internal steps:

$$a = a_1\,\sigma^i a_2 = (c\,d)\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s (c\,\sigma^i a_2)(d\,\sigma^i a_2) \xrightarrow{\mathtt{int}}_{\lambda s}$$

$$(c\,\sigma^i a_2')(d\,\sigma^i a_2) \xrightarrow{\mathtt{int}}_{\lambda s} (c\,\sigma^i a_2')(d\,\sigma^i a_2') = b$$

  - *($\sigma$-dest)* : We have $a_1 = \mathtt{n}$.
    If $n > i$ then $b = \mathtt{n} - \mathtt{1}$. But $\mathtt{n}\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \mathtt{n} - \mathtt{1}$.
    If $n < i$ then $b = \mathtt{n}$. But $\mathtt{n}\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \mathtt{n}$.
    If $n = i$ then $b = \varphi_0^i a_2'$. We must now consider wether $a_2 \to_{\lambda s} a_2'$ is external or internal. If it is internal we can commute to obtain:

$$a = a_1\,\sigma^i a_2 = \mathtt{n}\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \varphi_0^i a_2 \xrightarrow{\mathtt{int}}_{\lambda s} \varphi_0^i a_2' = b\ .$$

    But if it is external we must check that it is in fact an $s$-reduction to conclude:

$$a = a_1\,\sigma^i a_2 = \mathtt{n}\,\sigma^i a_2 \xrightarrow{\mathtt{ext}}_s \varphi_0^i a_2 \xrightarrow{\mathtt{ext}}_s \varphi_0^i a_2' = b$$

giving us $a \xrightarrow{\texttt{ext}}{}^{+}_{s} b \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} b$ . So we must prove that $a_2 \xrightarrow{\texttt{ext}}_{\sigma-gen} a'_2$ is impossible. It is for the treatment of this case that our additional hypotheses are necessary.

Suppose that $a_2 \xrightarrow{\texttt{ext}}_{\sigma-gen} a'_2$, then $\varphi^i_0 a_2 \xrightarrow{\texttt{ext}}_{\sigma-gen} \varphi^i_0 a'_2$. By proposition 2, $s(\varphi^i_0 a_2) \overset{+}{\to}_\beta s(\varphi^i_0 a'_2)$. But by hypothesis, $s(\mathtt{n}\,\sigma^i a_2) = s(a) = s(b) = s(\varphi^i_0 a'_2)$ and, since $n = i$, $s(\mathtt{n}\,\sigma^i a_2) = s(\varphi^i_0 a_2)$. This contradicts the hypothesis $s(a)$ is $\beta$-SN.

Therefore $a_2 \xrightarrow{\texttt{ext}}_s a'_2$ and we are done.

$a = \varphi^i_k a_1$ **:** Two possibilities according to the position of the external reduction.

– $\varphi^i_k a_1 \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} \varphi^i_k a'_1 \xrightarrow{\texttt{ext}}_s \varphi^i_k a''_1$ and $a_1 \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} a'_1 \xrightarrow{\texttt{ext}}_s a''_1$ . Let us check the hypotheses:

1. We know that $s(\varphi^i_k a_1) = U^i_k(s(a_1))$ is $\beta$-SN. To prove that $s(a_1)$ is $\beta$-SN, suppose it is not, and use lemma 7.1 to find an infinite derivation for $s(\varphi^i_k a_1)$.
2. We know that $s(\varphi^i_k a_1) = s(\varphi^i_k a''_1)$, hence $U^i_k(s(a_1)) = U^i_k(s(a''_1))$. We also know, by corollary 3, that $s(a_1) \twoheadrightarrow_\beta s(a''_1)$. If $s(a_1) \overset{+}{\to}_\beta s(a''_1)$, then by lemma 7.1 we have $U^i_k(s(a_1)) \overset{+}{\to}_\beta U^i_k(s(a''_1))$. But this is a contradiction since $U^i_k(s(a_1)) = s(\varphi^i_k a_1) = s(\varphi^i_k a''_1) = U^i_k(s(a''_1))$ and we are assuming that $s(\varphi^i_k a_1)$ is $\beta$-SN. Therefore, $s(a_1) = s(a''_1)$.

We can then apply IH and conclude easily.

– $\varphi^i_k a_1 \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} \varphi^i_k a'_1 \xrightarrow{\texttt{ext}}_s b$ , $a_1 \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} a'_1$ and the external reduction takes place at the root. Three rules are possible:

- *($\varphi$-$\lambda$-trans)* : We have $a'_1 = \lambda c'$ and $b = \lambda(\varphi^i_{k+1} c')$. Remark 3.1 ensures that $a_1 = \lambda c$ and $c \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} c'$. We can then commute:

$$a = \varphi^i_k a_1 = \varphi^i_k(\lambda c) \xrightarrow{\texttt{ext}}_s \lambda(\varphi^i_{k+1} c) \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} \lambda(\varphi^i_{k+1} c') = b\,.$$

- *($\varphi$-app-trans)* : We have $a'_1 = c'd'$ and $b = (\varphi^i_k c')(\varphi^i_k d')$. Remark 3.2 ensures that $a_1 = c\,d$ and, either $c \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} c'$ and $d = d'$, or $d \overset{\texttt{int}}{\twoheadrightarrow}_{\lambda s} d'$ and $c = c'$. In both cases we can commute as in the previous case.
- *($\varphi$-dest)* : We have $a'_1 = \mathtt{n}$ and this is impossible by remark 3.3. $\square$

This article was processed using the LaTeX macro package with LLNCS style