URL: http://www.elsevier.nl/locate/entcs/volume85.html 20 pages

Explicit Substitutions à la de Bruijn: the local and global way

Fairouz Kamareddine [1]

School of Mathematical and Computational Sciences
Heriot-Watt Univ., Riccarton
Edinburgh EH14 4AS, Scotland

Alejandro Ríos [2]

Department of Computer Science
University of Buenos Aires
Buenos Aires, Argentina

Abstract

Kamareddine and Nederpelt [9], resp. Kamareddine and Ríos [11] gave two calculi of explicit of substitutions highly influenced by de Bruijn's notation of the λ-calculus. These calculi added to the explosive pool of work on explicit substitution in the past 15 years. As far as we know, calculi of explicit substitutions: a) are unable to handle *local substitutions*, and b) have answered (positively or negatively) the question of the *termination* of the underlying calculus of substitutions. The exception to a) is the calculus of [9] where substitution is handled both locally and globally. However, the calculus of [9] does not satisfy properties like confluence and termination. The exception to b) is the λs_e-calculus [11] for which termination of the s_e-calculus, the underlying calculus of substitutions, remains unsolved. This paper has two aims:

(i) To provide a calculus à la de Bruijn which deals with local substitution and whose underlying calculus of substitutions is terminating and confluent.

(ii) To pose the problem of the termination of the substitution calculus of [11] in the hope that it can generate interest as a termination problem which at least for curiosity, needs to be settled. The answer here can go either way. On the one hand, although the $\lambda\sigma$-calculus [1] does not preserve termination, the σ-calculus itself terminates. On the other hand, could the non-preservation of termination in the λs_e-calculus imply the non-termination of the s_e-calculus?

[1] Email: fairouz@macs.hw.ac.uk
[2] Email: rios@dc.uba.ar

1 Introduction

Given $(\lambda x.xx)y$, one may not be interested in having yy as the result but rather only $(\lambda x.yx)y$. In other words, only one occurrence of x is substituted by y and the substitution can be continued later. Such *local* substitution is a major issue in functional language implementation [15]. Yet, most calculi of explicit substitutions are not able to handle this process. This paper presents an explicit substitution calculus which is able to handle local substitution.

There are two main styles of explicit substitution: the $\lambda\sigma$- and the λs_e-styles. The $\lambda\sigma$-calculus [1] reflects in its choice of operators and rules the calculus of categorical combinators [3]. The main innovation of the $\lambda\sigma$-calculus is the division of terms in two sorts: sort `term` and sort `substitution`. λs_e [11] departs from this style of explicit substitutions in two ways. First, it keeps the classical and unique sort `term` of the λ-calculus. Second, it does not use some of the categorical operators, especially those which are not present in the classical λ-calculus. The λs_e has two new operators which reflect the substitution and updating that are present in the meta-language of the λ-calculus, and so it can be said to be closer to the λ-calculus from an intuitive point of view, rather than a categorical one. The λs_e is based on the λs-calculus [10] which is a refinement of the calculus of [9] that was influenced by the Automath style and, as a result was able to handle local as well as global substitutions. The calculus of [9] however does not enjoy confluence and termination and refining it into λs (and λs_e) led to loss of local substitutions. As far as we know any explicit substitution calculus other than that of [9] is unable to handle local substitutions. For a survey of calculi of explicit substitutions, and a comparison between both $\lambda\sigma$- and λs_e-styles, see [13].

The $\lambda\sigma$- and λs_e-calculi, although in different styles, enjoy some common properties: they are both confluent, they both fail to preserve the termination of the λ-calculus, and they both simulate β-reduction. However, although the underlying substitution calculus of $\lambda\sigma$ is known to be terminating, this question remains unsettled for λs_e. This is frustrating. This question has been settled for any other calculus of explicit substitutions, so why has it proved very hard for λs_e? This paper reports on the status of this question so far.

Since the calculus of [9] and the calculus of local substitutions we will give in this paper are better described in a notation [8] highly influenced by de Bruijn's λ-calculus, we will separate the section dealing with local substitutions from that dealing with the termination of s_e.

2 The local substitution calculus

Since we are going to discuss and continue the work of [9] we shall present our calculus in *item notation* (cf. [8]). In this notation we write $a\,b = (b\,\delta)a$, $\lambda a = (\lambda)a$, $a\,\sigma^i b = (b\,\sigma^i)a$ and $\varphi^i_k a = (\varphi^i_k)a$. The σ^i-operator is the operator for explicit substitution at level i and the φ^i_k-operator stands for the explicit updating. The following nomenclature is used: $(b\,\delta)$, (λ), $(c\,\sigma^i)$, (φ^i_k) are called *items* (δ-, λ-, σ- and φ-items, respectively) and b and c the *bodies* of the respective items. A sequence of items is called a *segment.* Every term can be written as $\overline{s}\,\mathtt{n}$, where $\mathtt{n}$ is a de Bruijn index, with a convenient segment $\overline{s}$.

In order to treat local substitution [9] proposed the following rules:

$\sigma_{0\delta}$*-transition*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$((c\,\sigma^i)b\,\delta)a$	
$\sigma_{1\delta}$*-transition*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(b\,\delta)(c\,\sigma^i)a$	
σ*-destruction 1*	$(c\,\sigma^i)\mathtt{i}$	$\longrightarrow$	c	
σ*-destruction 2*	$(c\,\sigma^i)\mathtt{j}$	$\longrightarrow$	$\mathtt{j}$	if $\mathtt{j} \neq \mathtt{i}$

These rules are enough to prevent confluence. For example:
$(2\sigma^1)(1\,\delta)\mathtt{1} \to_{\sigma_{0\delta}\text{-}tr} ((2\sigma^1)1\,\delta)\mathtt{1} \to_{\sigma\text{-}dest\,1} (2\,\delta)\mathtt{1}$
$(2\sigma^1)(1\,\delta)\mathtt{1} \to_{\sigma_{1\delta}\text{-}tr} (1\,\delta)(2\sigma^1)\mathtt{1} \to_{\sigma\text{-}dest\,1} (1\,\delta)\mathtt{2}$
[9] gave a σ*-generation* rule to start β-reduction by generating a σ^1-operator:

$$\sigma\textit{-generation} \qquad (b\,\delta)(\lambda)a \longrightarrow (b\,\delta)(\lambda)((\varphi^1_0)b\,\sigma^1)a$$

Note that the starting δ-λ pair is kept after reduction. This enables the reuse of the rule to substitute another occurrence of the intended variable.

Considering only the rules introduced so far, the calculus presents another problem: terms which are strongly normalising in the classical λ-calculus, lose this property in the new calculus, and this occurs even if the application of the σ*-generation* rule is restricted to the case when the abstractor binds at least one occurrence of a de Bruijn number in a. Here is an example:
$(1\,\delta)(\lambda)(2\,\delta)\mathtt{1} \to_{\sigma-gen} (1\,\delta)(\lambda)((\varphi^1_0)1\,\sigma^1)(2\,\delta)\mathtt{1} \to_{\sigma_{0\delta}-tr}$
$(1\,\delta)(\lambda)(((\varphi^1_0)1\,\sigma^1)2\,\delta)\mathtt{1} \to_{\sigma-dest\,2} (1\,\delta)(\lambda)(2\,\delta)\mathtt{1} \to_{\sigma-gen} \cdots$
In order to solve the problem of confluence we will introduce a calculus where the rules $\sigma_{0\delta}$*-transition* and $\sigma_{1\delta}$*-transition* are modified as follows:

σ-δ*-transition 1*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i)((c\,\sigma^i)b\,\delta)a$
σ-δ*-transition 2*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i)(b\,\delta)(c\,\sigma^i)a$

Therefore, we shall be keeping the starting σ^i-item in order to reuse it. But we shall need a rule to dispose of this σ^i-item once all possible substitutions have been performed. We could try, for instance, the following:

$$\sigma\textit{-disposal} \qquad (c\,\sigma^i)a \longrightarrow a \qquad \text{if } \mathtt{i} \notin FV(a)$$

But this rule is not enough to get rid of the σ^i-item. For example:
$(1\,\sigma^1)(1\,\delta)\mathtt{2} \to_{\sigma\text{-}\delta\text{-}tr\,1} (1\,\sigma^1)((1\,\sigma^1)1\,\delta)\mathtt{2} \to_{\sigma\text{-}dest\,1} (1\,\sigma^1)(1\,\delta)\mathtt{2} \to_{\sigma\text{-}\delta\text{-}tr\,1} \cdots$
The problem is that after the substitution is performed on the index $\mathtt{1}$ we have again $\mathtt{1}$ and hence $\mathtt{1}$ will always be free in the scope of $(1\sigma^1)$.

We can try to add the classical σ-δ*-transition* rule to ensure that the σ^i-item will be disposed of at some time:

$$\sigma\text{-}\delta\textit{-transition} \qquad (c\,\sigma^i)(b\,\delta)a \longrightarrow ((c\,\sigma^i)b\,\delta)(c\,\sigma^i)a$$

But the inclusion of this rule forces us to *justify* the new calculus, since it stands for *global* substitution and is *always* needed to dispose of the σ^i-items.

In principle, we have two choices for the *σ-generation* rule: either we keep it as in [9] (see above) or we decide not to preserve the δ-λ-pairs and we state it as it is usually given in calculi of explicit substitutions (cf. [10,12]):

$$\textit{new } \sigma\textit{-generation} \qquad (b\,\delta)(\lambda)a \quad \longrightarrow \quad (b\,\sigma^1)a$$

If we admit this *new σ-generation* rule and keep our choice of operators, we are going to end up with either the λs-calculus (cf. [10]) if we decide for global updatings, or with the λt-calculus (cf. [12]) if we decide for partial updatings. But, none of these calculi permit local substitutions. Also, since these calculi do not preserve the δ-λ-pairs, their σ-destruction rules must update the free variables and hence in both calculi we have:

$$\sigma\textit{-destruction 3} \qquad (c\,\sigma^i)\mathtt{j} \quad \longrightarrow \quad \mathtt{j}-1 \qquad \text{if } \mathtt{j} > \mathtt{i}$$

But with this rule and the new σ-δ-transition rules we lose confluence:
$(1\,\sigma^1)(3\,\delta)1 \to_{\sigma\text{-}\delta\text{-}tr\,1} (1\,\sigma^1)((1\,\sigma^1)3\,\delta)1 \to_{\sigma-dest\,3} (1\,\sigma^1)(2\,\delta)1 \to_{\sigma\text{-}\delta\text{-}tr}$
$((1\,\sigma^1)2\,\delta)(1\,\sigma^1)1 \to_{\sigma-dest\,3} (1\,\delta)(1\,\sigma^1)1 \to_{\sigma-dest\,1} (1\,\delta)1$

And the following derivation is also available:
$(1\,\sigma^1)(3\,\delta)1 \to_{\sigma\text{-}\delta\text{-}tr} ((1\,\sigma^1)3\,\delta)(1\,\sigma^1)1 \to_{\sigma-dest\,3} (2\,\delta)(1\,\sigma^1)1 \to_{\sigma-dest\,1} (2\,\delta)1$

Therefore, we discard the *new σ-generation* rule in order to avoid *σ-destruction 3* and choose to keep the first version of these rules.

Finally, since the *σ-generation* rule preserves the δ-λ pair we need a rule to dispose of the pair once all the possible substitutions have been carried on, i.e. when the abstractor in the δ-λ pair does not bind any de Bruijn index. When discarding the δ-λ pair we must update the de Bruijn numbers that stand for free variables. We shall perform this updating by introducing a new family of operators: μ^i and rewriting rules for their propagation.

2.1 A first attempt

With this intuition behind our calculus, we give a formal presentation.

Definition 2.1 The terms of the calculus are given by the following grammar:

$$\Lambda\sigma\mu ::= I\!N \mid (\Lambda\sigma\mu\,\delta)\Lambda\sigma\mu \mid (\lambda)\Lambda\sigma\mu \mid (\Lambda\sigma\mu\,\sigma^i)\Lambda\sigma\mu \mid (\varphi_k)\Lambda\sigma\mu \mid (\mu^i)\Lambda\sigma\mu$$

where $i \geq 1$, $k \geq 0$. We let a, b, c, ... range over $\Lambda\sigma\mu$.

Note that the updating operators contain only one index. This is because our calculus will work with partial updatings and therefore, as for the λt-calculus [12], the lower index is enough to deal with the updating mechanism.

The notion of free variable in our calculus needs the following definition:

Definition 2.2 Let $N \subset I\!N$ and $k \geq 0$. We define

(i) $N \setminus k = \{n - k : n \in N, n > k\}$, $N + k = \{n + k : n \in N\}$

(ii) $N_{\rho k} = \{n \in N : n\rho k\}$, where $\rho \in \{<, \leq, >, \geq\}$.

We can define now the free variables of a term in $\Lambda\sigma\mu$.

σ-generation	$(b\,\delta)(\lambda)a$	$\longrightarrow$	$(b\,\delta)(\lambda)((\varphi_0)b\,\sigma^1)a$	if $\mathtt{1} \in FV(a)$
μ-generation	$(b\,\delta)(\lambda)a$	$\longrightarrow$	$(\mu^1)a$	if $\mathtt{1} \notin FV(a)$
σ-λ-transition	$(b\,\sigma^i)(\lambda)a$	$\longrightarrow$	$(\lambda)((\varphi_0)b\,\sigma^{i+1})a$	
σ-δ-transition	$(b\,\sigma^i)(a_1\delta)a_2$	$\longrightarrow$	$((b\,\sigma^i)a_1\delta)\,(b\,\sigma^i)a_2$	
σ-δ-transition 1	$(b\,\sigma^i)(a_1\delta)a_2$	$\longrightarrow$	$(b\,\sigma^i)((b\,\sigma^i)a_1\delta)a_2$	
σ-δ-transition 2	$(b\,\sigma^i)(a_1\delta)a_2$	$\longrightarrow$	$(b\,\sigma^i)(a_1\delta)\,(b\,\sigma^i)a_2$	
σ-destruction	$(b\,\sigma^i)\mathtt{n}$	$\longrightarrow$	$\begin{cases} b \text{ if } n = i \\ \mathtt{n} \text{ if } n \neq i \end{cases}$	
φ-λ-transition	$(\varphi_k)(\lambda)a$	$\longrightarrow$	$(\lambda)(\varphi_{k+1})a$	
φ-δ-transition	$(\varphi_k)(a_1\delta)a_2$	$\longrightarrow$	$((\varphi_k)a_1\delta)(\varphi_k)a_2$	
φ-destruction	$(\varphi_k)\mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+1} \text{ if } n > k \\ \mathtt{n} \quad\; \text{if } n \leq k \end{cases}$	
μ-λ-transition	$(\mu^i)(\lambda)a$	$\longrightarrow$	$(\lambda)(\mu^{i+1})a$	
μ-δ-transition	$(\mu^i)(a_1\delta)a_2$	$\longrightarrow$	$((\mu^i)a_1\delta)(\mu^i)a_2$	
μ-destruction	$(\mu^i)\mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} \text{ if } n > i \\ \mathtt{n} \quad\; \text{if } n \leq i \end{cases}$	

Fig. 1. The $\lambda\sigma\mu$-calculus

Definition 2.3 The *set of free variables* of a term in $\Lambda\sigma\mu$ is defined by:

$FV(\mathtt{n}) = \{n\}$ $\qquad FV((\varphi_k)a) = FV(a)_{\leq k} \cup (FV(a)_{>k} + 1)$

$FV((b\,\delta)a) = FV(b) \cup FV(a)$ $\qquad FV((\mu^i)a) = FV(a)_{\leq i} \cup (FV(a)_{>i} \setminus 1)$

$FV((\lambda)a) = FV(a) \setminus 1$ $\qquad FV((b\,\sigma^i)a) = FV(a)_{<i} \cup (FV(a)_{>i} \setminus 1) \cup FV(b)$

Definition 2.4 The *$\lambda\sigma\mu$-calculus* is the reduction system $(\Lambda\sigma\mu, \to)$ where $\to$ is the least compatible relation (with the operators of $\Lambda\sigma\mu$) generated by the set $\lambda\sigma\mu$ of rules in Figure 2.4. The calculus of substitutions associated with the $\lambda\sigma\mu$-calculus is the reduction system generated by the set $\lambda\sigma\mu - \{$*σ-generation*, *μ-generation*$\}$ and we call it the $\sigma\mu$-calculus.

Note that the problem of loss of strong normalisation for terms which are strongly normalising in the classical λ-calculus still persists. For example:

$(\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)\mathtt{1} \to_{\sigma\text{-}\delta\text{-}tr\,1} (\mathtt{1}\,\sigma^1)((\mathtt{1}\,\sigma^1)\mathtt{2}\,\delta)\mathtt{1} \to_{\sigma-dest} (\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)\mathtt{1} \to \cdots$

Note also that this calculus is not confluent. E.g., let $a = ((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)\mathtt{1}$:
$a \to_{\sigma\text{-}\delta\text{-}tr\,2} ((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)\mathtt{1} \to_{\sigma-dest} ((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)(\mathtt{1}\,\delta)\mathtt{1} \to_{\sigma\text{-}\delta\text{-}tr\,2}$
$((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)(\mathtt{2}\,\delta)((\mathtt{1}\,\delta)\mathtt{1}\,\sigma^1)(\mathtt{1}\,\delta)\mathtt{1} \twoheadrightarrow_{\sigma\text{-}\delta\text{-}tr,\,\sigma-dest} (\mathtt{2}\,\delta)((\mathtt{1}\,\delta)\mathtt{1}\,\delta)(\mathtt{1}\,\delta)\mathtt{1}$

But also $a \rightarrow_{\sigma\text{-}\delta\text{-}tr} (((1\,\delta)1\,\sigma^1)2\,\delta)((1\,\delta 1)\sigma^1)1 \twoheadrightarrow_{\sigma-dest} (2\,\delta)(1\,\delta)1$

Finally, $\lambda\sigma\mu$ not only does not solve the problem of confluence and does not preserve strong normalisation, but it also is not a first order rewriting system in the classical sense since both generation rules are conditional and extra and maybe costly calculations must be performed to evaluate the conditions.

The only properties that we have proved, concern a subsystem of $\lambda\sigma\mu$, we call it $\sigma\mu^-$, and is obtained by deleting *σ-δ-tr 1* and *σ-δ-tr 2* from $\sigma\mu$.

Lemma 2.5 *$\sigma\mu^-$ is SN and CR and the set of $\sigma\mu^-$-normal forms is exactly the set of pure terms.*

Proof. Analogous to the proof of this property for λt [12]. □

2.2 A better attempt

In the previous section we have given several counterexamples, the majority of which are based on the fact that rules like *σ-δ-tr 1* and *σ-δ-tr 2* can be used several times to perform the *same* substitution. Therefore, these rules are not adequate to formalise the notion of local substitution.

In order to prevent a rule like *σ-δ-tr 1* to evaluate the same substitution several times we are going to introduce a unary operator L to mark the term where the substitution has been locally performed and we will not allow the substitution to be evaluated again on marked terms. Let us try the following:

preliminary σ-δ-local 1 $\qquad (c\,\sigma^i)(b\,\delta)a \longrightarrow (c\,\sigma^i)((L)(c\,\sigma^i)b\,\delta)a$

Now this rule poses still the problem of normalisation:

$(c\,\sigma^i)(b\,\delta)a \rightarrow (c\,\sigma^i)((L)(c\,\sigma^i)b\,\delta)a \rightarrow (c\,\sigma^i)((L)(c\,\sigma^i)(L)(c\,\sigma^i)b\,\delta)a \rightarrow \cdots$

To prevent this we propose to introduce another family of σ-operators that we denote σ^i_{Loc} and we modify the rule as follows:

σ-δ-local 1 $\qquad (c\,\sigma^i)(b\,\delta)a \longrightarrow (c\,\sigma^i_{Loc})((L)(c\,\sigma^i)b\,\delta)a$

And to dispose of these new operators we add:

σ_{Loc}-disposal 1 $\qquad (c\,\sigma^i_{Loc})((L)b\,\delta)a \longrightarrow (b\,\delta)(c\,\sigma^i)a$

We must also add the following, in order to be able to perform local substitution in the other branch of the application:

σ-δ-local 2 $\qquad (c\,\sigma^i)(b\,\delta)a \longrightarrow (c\,\sigma^i_{Loc})(b\,\delta)(L)(c\,\sigma^i)a$

σ_{Loc}-disposal 2 $\qquad (c\,\sigma^i_{Loc})(b\,\delta)(L)a \longrightarrow ((c\,\sigma^i)b\,\delta)a$

We are approaching the right solution but the confluence problem persist:

$(c\,\sigma^i_{Loc})((L)b\,\delta)(L)a \rightarrow_{\sigma_{Loc}-disp\,1} (b\,\delta)(c\,\sigma^i)(L)a$

And on the other hand $(c\,\sigma^i_{Loc})((L)b\,\delta)(L)a \rightarrow_{\sigma_{Loc}-disp\,2} ((c\,\sigma^i)(L)b\,\delta)a$

But this problem has an easy solution: split the family of operators σ^i_{Loc} into one family that stands for the local substitution performed in the left branch of the application and another family for the right branch. We denote these families σ^i_L and σ^i_R, respectively. Hence, we propose:

σ_R-*generation*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i_R)((L)(c\,\sigma^i)b\,\delta)a$
σ_R-*destruction*	$(c\,\sigma^i_R)((L)b\,\delta)a$	$\longrightarrow$	$(b\,\delta)(c\,\sigma^i)a$
σ_L-*generation*	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i_L)(b\,\delta)(L)(c\,\sigma^i)a$
σ_L-*destruction*	$(c\,\sigma^i_L)(b\,\delta)(L)a$	$\longrightarrow$	$((c\,\sigma^i)b\,\delta)a$

With this formulation we solve several problems at the same time: the distinction between σ^i_R and σ^i_L allow us to obtain confluence for the calculus of substitution and the distinction between σ^i-operators on one hand and σ^i_R and σ^i_L on the other is a good sign for the preservation of strong normalisation. Moreover, with this formulation we are not forced to preserve the δ-λ pairs and hence we do not need to introduce conditions on free or bound variables and furthermore we do not need the introduction of the μ-operator and all the rules that it generates. Now, we present formally the calculus:

Definition 2.6 The terms of the calculus are given by the following grammar:

$$\Lambda s_L ::= I\!N \mid (\Lambda s_L\,\delta)\Lambda s_L \mid (L)\Lambda s_L \mid (\lambda)\Lambda s_L \mid (\Lambda s_L\,\sigma^i)\Lambda s_L \mid (\varphi^i_k)\Lambda s_L$$
$$\mid (\Lambda s_L\,\sigma^i_L)\Lambda s_L \mid (\Lambda s_L\,\sigma^i_R)\Lambda s_L \qquad \text{where } i \geq 1,\ k \geq 0$$

We let a, b, c, ... range over Λs_L. Note that we come back to the updating operators of λs. In fact, the calculus we will define is λs where σ-δ-*transition* is replaced by the four rules above. Note also that $\Lambda \subset \Lambda s \subset \Lambda s_L$.

Definition 2.7 The λs_L-*calculus* is the reduction system $(\Lambda s_L, \rightarrow_{\lambda s_L})$, where $\rightarrow_{\lambda s_L}$ is the least compatible reduction on Λs_L generated by the rules in Figure 2. We use λs_L to denote this set of rules. The calculus of substitutions associated with the λs_L-calculus is the reduction system generated by the set $\lambda s_L - \{\sigma\text{-}generation\}$ and we call it the σ_L-calculus.

Lemma 2.8 shows that the λs-calculus can be simulated in the λs_L-calculus:

Lemma 2.8 *Let a, $b \in \Lambda s$, if $a \rightarrow_{\lambda s} b$ then $a \rightarrow_{\lambda s_L} b$.*

Proof. It is enough to show that the σ-δ-*transition* rule can be simulated in the λs_L-calculus. This may be done by consecutive application either of σ_R-*generation* and σ_R-*destruction* or σ_L-*generation* and σ_L-*destruction.* □

We conclude now that the λs_L-calculus simulates classical β-reduction:

Corollary 2.9 *Let a, $b \in \Lambda$, if $a \rightarrow_\beta b$ then $a \rightarrow_{\lambda s_L} b$.*

Proof. Using the previous lemma and the simulation of β in λs (cf. [10]). □

We are going to prove now confluence and strong normalisation of the σ_L-calculus, in order to have existence and uniqueness of σ_L-normal forms.

Lemma 2.10 *The σ_L-calculus is locally confluent.*

σ-generation	$(b\,\delta)(\lambda)a$	$\longrightarrow$	$(b\,\sigma^1)a$
σ-λ-transition	$(b\,\sigma^j)(\lambda)a$	$\longrightarrow$	$(\lambda)(b\,\sigma^{j+1})a$
σ_R-generation	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i_R)((L)(c\,\sigma^i)b\,\delta)a$
σ_R-destruction	$(c\,\sigma^i_R)((L)b\,\delta)a$	$\longrightarrow$	$(b\,\delta)(c\,\sigma^i)a$
σ_L-generation	$(c\,\sigma^i)(b\,\delta)a$	$\longrightarrow$	$(c\,\sigma^i_L)(b\,\delta)(L)(c\,\sigma^i)a$
σ_L-destruction	$(c\,\sigma^i_L)(b\,\delta)(L)a$	$\longrightarrow$	$((c\,\sigma^i)b\,\delta)a$
σ-destruction	$(b\,\sigma^j)\mathtt{n}$	$\longrightarrow$	$\begin{cases}\mathtt{n-1} & \text{if } n > j\\ (\varphi^j_0)b & \text{if } n = j\\ \mathtt{n} & \text{if } n < j\end{cases}$
φ-λ-transition	$(\varphi^i_k)(\lambda)a$	$\longrightarrow$	$(\lambda)(\varphi^i_{k+1})a$
φ-δ-transition	$(\varphi^i_k)(a_1\delta)a_2$	$\longrightarrow$	$((\varphi^i_k)a_1\delta)(\varphi^i_k)a_2$
φ-destruction	$(\varphi^i_k)\mathtt{n}$	$\longrightarrow$	$\begin{cases}\mathtt{n+i-1} & \text{if } n > k\\ \mathtt{n} & \text{if } n \le k\end{cases}$

Fig. 2. The λs_L-calculus

Proof. By Knuth-Bendix Theorem it is enough to study the critical pairs. There is only one, namely the one generated by the *σ_R-generation* and *σ_L-generation* rules. It can be closed using *σ_L-destruction* and *σ_R-destruction.* □

The proof of SN is not immediate. We envisage a proof by structural induction split in the following lemmas. We note SN the set of terms in Λs_L which are σ_L-strongly normalising. Our aim is to prove that $SN = \Lambda s_L$.

Lemma 2.11 *Let a, $b \in \Lambda s_L$, the following hold:*

(i) *$(b\,\delta)a \in SN$ iff $a \in SN$ and $b \in SN$.*

(ii) *$(\lambda)a \in SN$ iff $a \in SN$.*

(iii) *$(L)a \in SN$ iff $a \in SN$.*

Proof. No σ_L-rule has an application, an abstraction or a mark at the root.□

In the following lemmas we use the notation: $\lg(a)$ stands for the length of term a and is defined as usual, $\mathrm{dp}(a)$ stands for the depth of a, i.e. the length of the longest derivation to its σ_L-normal form. We use $\mathrm{dp}(a)$ for $a \in SN$.

Lemma 2.12 *For $i \geq 1$ and $k \geq 0$, if $a \in SN$ then $(\varphi^i_k)a \in SN$.*

Proof. By induction on the ordinal $(\mathrm{dp}(a), \lg(a))$.

If $(\mathrm{dp}(a), \lg(a)) = (0, 1)$ then $a = \mathtt{n}$; obvious. If $\varphi^i_k a$ is a normal form, then obvious. Therefore we study all possible reducts of $(\varphi^i_k)a$ and prove them SN.

If $(\varphi^i_k)a \to (\varphi^i_k)b$, with $a \to b$, we conclude by IH since $\mathrm{dp}(a) > \mathrm{dp}(b)$.

If the reduction is at the root we must analyse the three possible rules. We just study *φ-δ-transition*: We have $(\varphi^i_k)((b\,\delta)c) \to ((\varphi^i_k)b\,\delta)(\varphi^i_k)c$. Now $\mathrm{dp}((b\,\delta)c) \geq \mathrm{dp}(b)$, $\mathrm{dp}((b\,\delta)c) \geq \mathrm{dp}(c)$, $\mathrm{lg}((b\,\delta)c) > \mathrm{lg}(b)$ and $\mathrm{lg}((b\,\delta)c) > \mathrm{lg}(c)$. Hence by IH, $(\varphi^i_k)b \in SN$ and $(\varphi^i_k)c \in SN$, and we use Lemma 2.11.1. □

Lemma 2.13 *For $i \geq 1$, if $a, b \in SN$ then $(b\,\sigma^i)a \in SN$.*

Proof. By induction on the ordinal $(\mathrm{dp}(a), \mathrm{lg}(a), \mathrm{dp}(b))$.

If $(\mathrm{dp}(a), \mathrm{lg}(a), \mathrm{dp}(b)) = (0, 1, 0)$, then $a = \mathtt{n}$ and b is in normal form. The result is obvious if $n \neq i$, whereas if $n = i$ we use the previous lemma.

The proof follows now the lines of the previous lemma, but an interesting case arises when considering the reduction at the root by the *σ_R-generation* or the *σ_L-generation* rule. Let us study for instance the latter.

Therefore we have $a = (d\,\delta)c$ and $(b\,\sigma^i)(d\,\delta)c \to (b\,\sigma^i_L)(d\,\delta)(L)(b\,\sigma^i)c$. Let us assume that $(b\,\sigma^i_L)(d\,\delta)(L)(b\,\sigma^i)c \notin SN$.

Since $\mathrm{dp}((d\,\delta)c) \geq \mathrm{dp}(c)$ and $\mathrm{lg}((d\,\delta)c) > \mathrm{lg}(c)$, by IH we have $(b\,\sigma^i)c \in SN$ and by Lemma 2.11.3, $L((b\,\sigma^i)c) \in SN$. Now, since $a \in SN$, we have $d \in SN$, and by Lemma 2.11.1, we conclude $(d\,\delta)(L)(b\,\sigma^i)c \in SN$.

Therefore, since $b \in SN$, there must be an infinite derivation beginning at $(b\,\sigma^i_L)(d\,\delta)(L)(b\,\sigma^i)c$ which reduces at the root. Furthermore, since there are no rules which reduce applications or marks, there exist d', c', b' such that $d \twoheadrightarrow d'$, $(b\,\sigma^i)c \twoheadrightarrow c'$, $b \twoheadrightarrow b'$ and $(b\,\sigma^i_L)(d\,\delta)(L)(b\,\sigma^i)c \twoheadrightarrow (b'\,\sigma^i_L)(d'\,\delta)(L)c' \to ((b'\,\sigma^i)d'\,\delta)c' \to \cdots$ But the fact that this derivation is infinite is a contradiction because by IH, we have $(b\,\sigma^i)c \in SN$, and hence $c' \in SN$, and also by IH we have $(b\,\sigma^i)d \in SN$, and hence $(b'\,\sigma^i)d' \in SN$. Therefore, by Lemma 2.11.1, $((b'\,\sigma^i)d'\,\delta)c' \in SN$. We conclude that $(b\,\sigma^i_L)(d\,\delta)(L)(b\,\sigma^i)c$ must be SN. □

Lemma 2.14 *For $i \geq 1$, if $a, b \in SN$ then $(b\,\sigma^i_L)a \in SN$ and $(b\,\sigma^i_R)a \in SN$.*

Proof. By induction on the ordinal $(\mathrm{dp}(a), \mathrm{dp}(b))$. Use the previous lemma when considering the reduction at the root. □

Theorem 2.15 *The σ_L-calculus is strongly normalising.*

Proof. By induction on a we prove that every $a \in \Lambda s_L$ is SN.

If $a = \mathtt{n}$, it is obviously SN.

If $a = (c\,\delta)b$ or $a = (\lambda)b$ or $a = (L)b$, use Lemma 2.11.

If $a = (\varphi^i_k)b$ use Lemma 2.12.

If $a = (c\,\sigma^i)b$ use Lemma 2.13.

If $a = (c\,\sigma^i_L)b$ or $a = (c\,\sigma^i_R)b$ use Lemma 2.14. □

Theorem 2.16 *The σ_L-calculus is confluent.*

Proof. By Newman's Lemma, the previous lemma and Lemma 2.10. □

3 The status of the open question of termination of s_e

The λs_e-calculus, like the $\lambda\sigma$-calculus, simulates β-reduction, is confluent (on open terms[3]) [11] and does not preserve strong normalisation (we say does not have PSN) [6]. However, although strong normalisation (SN) of the σ-calculus (the substitution calculus associated with the $\lambda\sigma$-calculus) has been established, it is still unknown whether strong normalisation of the s_e-calculus (the substitution calculus associated with the λs_e-calculus) holds. Only weak normalisation of the s_e-calculus is known so far [11].

The s_e-calculus (see Definition 3.8) has the *σ-σ-transition* rule which seems to be responsible for the difficulties in establishing SN of s_e. However, Zantema showed that the *σ-σ-transition* scheme on its own is SN [11].

This section is a discussion of the status of strong normalisation of the s_e-calculus. We show that the set of rules s_e is the union of two disjoint sets of rules *σ-σ-tr.+φ-σ-tr.* and the rest of the rules where each of these two sets gives a calculus which is SN. However, commutation does not hold and hence modularity cannot be used to obtain SN of s_e. In addition, the distribution elimination [17] and recursive path ordering methods are not applicable and we remain unsure whether s_e is actually SN or not.

3.1 The classical λ-calculus in de Bruijn notation

We assume the reader familiar with de Bruijn notation [5]. We define Λ, the *set of terms with de Bruijn indices*, by: $\Lambda ::= I\!\!N \mid (\Lambda\Lambda) \mid (\lambda\Lambda)$.
We use $a, b, \ldots$ to range over Λ and $m, n, \ldots$ to range over $I\!\!N$ (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. We write $\to^+$ and $\twoheadrightarrow$ to denote the transitive and the reflexive transitive closures of a reduction notion $\to$. We say that a reduction $\to$ is *compatible on* Λ when for all a, b, $c \in \Lambda$, we have $a \to b$ implies $a\,c \to b\,c$, $c\,a \to c\,b$ and $\lambda a \to \lambda b$.

As β-reduction à la de Bruijn involves the substitution of a variable $\mathtt{n}$ for a term b in a term a, we need to update the terms:

Definition 3.1 Let the *updating functions* $U^i_k : \Lambda \to \Lambda$ for $k \geq 0$ be $i \geq 1$ be:

$$U^i_k(ab) = U^i_k(a)\,U^i_k(b) \qquad U^i_k(\mathtt{n}) = \begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k\,. \end{cases}$$

$$U^i_k(\lambda a) = \lambda(U^i_{k+1}(a))$$

Definition 3.2 The *meta-substitutions at level j*, for $j \geq 1$, of a term $b \in \Lambda$ in a term $a \in \Lambda$, denoted $a\{\!\{\mathtt{j} \leftarrow b\}\!\}$, is defined inductively on a as follows:

[3] The λs_e-calculus is confluent on the whole set of open terms whereas $\lambda\sigma$ is confluent on the open terms without metavariables of sort `substitution` as is shown in [16].

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

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

Definition 3.3 *β-reduction* is the least compatible reduction on Λ generated by: (β-rule) $(\lambda a)\, b \to_\beta a\{\!\{\mathtt{1} \leftarrow b\}\!\}$
The *λ-calculus à la de Bruijn*, is the reduction system with rewriting rule β.

3.2 The λs- and λs_e-calculi

λs [10] handles explicitly the meta-operators of definitions 3.1 and 3.2. Hence, the syntax of the λs-calculus is obtained by adding two families of operators :

- $\{\, \sigma^j \}_{j \geq 1}$, which denotes the explicit substitution operators. The term $a\, \sigma^j b$ stands for term a where all free occurrences of the variable corresponding to de Bruijn index j are to be substituted by term b.
- $\{\varphi_k^i\}_{k \geq 0\ i \geq 1}$, which denotes the updating functions necessary when working with de Bruijn numbers to fix the variables of the term to be substituted.

Definition 3.4 The *set* Λs *of terms of the λs-calculus* is given as follows:

$$\Lambda s ::= I\!N \mid \Lambda s \Lambda s \mid \lambda \Lambda s \mid \Lambda s\, \sigma^j \Lambda s \mid \varphi_k^i \Lambda s \qquad \textit{where} \quad j,\, i \geq 1\,, \quad k \geq 0\,.$$

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

A reduction $\to$ on Λs is *compatible* if for all a, b, $c \in \Lambda s$, if $a \to b$ then $a\, c \to b\, c$, $c\, a \to c\, b$, $\lambda a \to \lambda b$, $a\, \sigma^j c \to b\, \sigma^j c$, $c\, \sigma^j a \to c\, \sigma^j b$ and $\varphi_k^i a \to \varphi_k^i b$.

To *σ-generation* which mimicks the β-rule, we add a set of rules which are the equations in definitions 3.1 and 3.2 oriented from left to right.

Definition 3.5 The *λs-calculus* is the reduction system $(\Lambda s, \to_{\lambda s})$, where $\to_{\lambda s}$ is the least compatible reduction on Λs generated by the set λs of the rules of Figure 3. The *s-calculus*, the *calculus of substitutions associated with the λs-calculus*, is the reduction system generated by the set $s = \lambda s - \{\sigma\textit{-generation}\}$.

Lemma 3.6 (cf. [10]) *The following holds:*

(i) *(SN and CR of s) The s-calculus is strongly normalising and confluent on Λs. Hence, every term a has a unique s-normal form denoted $s(a)$.*

(ii) *The set of s-normal forms is exactly Λ.*

(iii) *For all a, $b \in \Lambda s$ we have:* $s(a\, b) = s(a) s(b)$, $s(\lambda a) = \lambda(s(a))$, $s(\varphi_k^i a) = U_k^i(s(a))$, $s(a\, \sigma^j b) = s(a)\{\!\{\mathtt{j} \leftarrow s(b)\}\!\}$.

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

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

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

(vii) *(CR of λs) The λs-calculus is confluent on Λs.*

σ-generation	$(\lambda a)\, b$	$\longrightarrow$	$a\, \sigma^1\, b$
σ-λ-transition	$(\lambda a)\, \sigma^j b$	$\longrightarrow$	$\lambda(a \sigma^{j+1} b)$
σ-app-transition	$(a_1\, a_2)\, \sigma^j b$	$\longrightarrow$	$(a_1\, \sigma^j b)\, (a_2\, \sigma^j b)$
σ-destruction	$\mathtt{n}\, \sigma^j b$	$\longrightarrow$	$\begin{cases} \mathtt{n-1} & \text{if } n > j \\ \varphi_0^j\, b & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases}$
φ-λ-transition	$\varphi_k^i(\lambda a)$	$\longrightarrow$	$\lambda(\varphi_{k+1}^i\, a)$
φ-app-transition	$\varphi_k^i(a_1\, a_2)$	$\longrightarrow$	$(\varphi_k^i\, a_1)\, (\varphi_k^i\, a_2)$
φ-destruction	$\varphi_k^i\, \mathtt{n}$	$\longrightarrow$	$\begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \leq k \end{cases}$

Fig. 3. The λs-calculus

(viii) *(Preservation of SN) Pure terms which are strongly normalising in the λ-calculus are also strongly normalising in the λs-calculus.*

Open terms were introduced in the λs-calculus as follows (see [11]):

Definition 3.7 The set of *open terms*, noted Λs_{op} is given as follows:

$$\Lambda s_{op} ::= \mathbf{V} \mid I\!N \mid \Lambda s_{op} \Lambda s_{op} \mid \lambda \Lambda s_{op} \mid \Lambda s_{op}\, \sigma^j \Lambda s_{op} \mid \varphi_k^i \Lambda s_{op}$$

where $j,\, i \geq 1\,,\;\; k \geq 0$ and where $\mathbf{V}$ stands for a set of variables, over which X, Y, ... range. We take a, b, c to range over Λs_{op}. Furthermore, *closures*, *pure terms* and *compatibility* are defined as for Λs.

Working with open terms one loses confluence as shown by the example:

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

and $(X\sigma^1 Y)\sigma^1\mathtt{1}$ and $((\lambda X)\sigma^1\mathtt{1})(Y\sigma^1\mathtt{1})$ have no common reduct. This example also shows that even local confluence is lost. In order to solve this problem, [11] added to the λs-calculus a set of rules that guarantees confluence.

Definition 3.8 The set of rules λs_e is λs together with the rules of Figure 4. The *λs_e-calculus* is the reduction system $(\Lambda s_{op}, \to_{\lambda s_e})$ where $\to_{\lambda s_e}$ is the least compatible reduction on Λs_{op} generated by the set of rules λs_e. The *s_e-calculus*, the *calculus of substitutions associated with the λs_e-calculus*, is the rewriting system generated by the set of rules $s_e = \lambda s_e - \{\sigma\text{-}generation\}$.

Lemma 3.9 (cf. [11]) *The following holds:*

(i) *(WN and CR of s_e) The s_e-calculus is weakly normalising and confluent.*

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

(iii) *(CR of λs_e) The λs_e-calculus is confluent on open terms.*

σ-σ-transition	$(a\,\sigma^i b)\,\sigma^j\,c$	$\longrightarrow$	$(a\,\sigma^{j+1}\,c)\,\sigma^i\,(b\,\sigma^{j-i+1}\,c)$	if	$i \leq j$
σ-φ-transition 1	$(\varphi^i_k\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi^{i-1}_k\,a$	if	$k < j < k+i$
σ-φ-transition 2	$(\varphi^i_k\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi^i_k(a\,\sigma^{j-i+1}\,b)$	if	$k+i \leq j$
φ-σ-transition	$\varphi^i_k(a\,\sigma^j\,b)$	$\longrightarrow$	$(\varphi^i_{k+1}\,a)\,\sigma^j\,(\varphi^i_{k+1-j}\,b)$	if	$j \leq k+1$
φ-φ-transition 1	$\varphi^i_k\,(\varphi^j_l\,a)$	$\longrightarrow$	$\varphi^j_l\,(\varphi^i_{k+1-j}\,a)$	if	$l+j \leq k$
φ-φ-transition 2	$\varphi^i_k\,(\varphi^j_l\,a)$	$\longrightarrow$	$\varphi^{j+i-1}_l\,a$	if	$l \leq k < l+j$

Fig. 4. The extra rules of the λs_e-calculus

(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)	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow$	$s_1 \circ (s_2 \circ s_3)$

Fig. 5. The $\lambda\sigma$-calculus

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

3.3 The $\lambda\sigma$-calculus and the termination of the σ-calculus

Definition 3.10 The syntax of the *$\lambda\sigma$-calculus* [1] is given by:

$$\textbf{Terms} \qquad \Lambda\sigma^t ::= \mathtt{1} \mid \Lambda\sigma^t\Lambda\sigma^t \mid \lambda\Lambda\sigma^t \mid \Lambda\sigma^t[\Lambda\sigma^s]$$

$$\textbf{Substitutions}\ \Lambda\sigma^s ::= id \mid \uparrow \mid \Lambda\sigma^t \cdot \Lambda\sigma^s \mid \Lambda\sigma^s \circ \Lambda\sigma^s$$

The set, denoted $\lambda\sigma$, of rules of the *$\lambda\sigma$-calculus* is given in Figure 5.

The set of rules of the *σ-calculus* is $\lambda\sigma - \{(Beta)\}$. We use $a, b, c, \ldots$ to range over $\Lambda\sigma^t$ and $s, t, \ldots$ to range over $\Lambda\sigma^s$. 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 $\mathtt{n}$ as $\mathtt{1}[\uparrow^{n-1}]$. So, $\Lambda \subset \Lambda\sigma^t$.

Theorem 3.11 *The σ-calculus is strongly normalising (SN).*

There are various proofs of this theorem in the literature:

(i) The first strong normalisation proof of σ is based on the strong normalisation of *SUBST* [7], which is, within *CCL*, the set of rewriting rules that compute the substitutions. See [7].

(ii) The proof in [4] shows the termination of σ via a strict translation from σ to another calculus σ_0 (an economic variant of σ) and the termination of σ_0. The calculus σ_0 is one sorted and treats both $\circ$ and $[\,]$ as $\circ$, observing that $\circ$ and $[\,]$ behave in the same way.

(iii) Zantema gives two proofs in [17,18]. The first is based on a suitable generalisation of polynomial orders to show the termination of the calculus σ_0 given below (and hence the termination of σ). The second uses semantic labelling to show the termination of σ.

We will explain why these techniques for showing SN of σ do not apply to s_e.

Definition 3.12 [The σ_0-calculus] The set of terms $\Lambda\sigma_0$ of the σ_0-calculus has the abstract syntax $\quad s,t ::= 1 \mid id \mid\uparrow\mid \lambda s \mid s\circ t \mid s\cdot t.$
The set, denoted σ_0, of rules of the calculus is the following:

(VrId)	$1 \circ id \to 1$	(ShId)	$\uparrow \circ id \to \uparrow$
(VrCons)	$1 \circ (s\cdot t) \to s$	(Abs)	$(\lambda s)\circ t \to \lambda(s \circ (1 \cdot (t\circ \uparrow)))$
(ShCons)	$\uparrow \circ (s\cdot t) \to t$	(Map)	$(s\cdot t)\circ u \to (s\circ u)\cdot(t\circ u)$
(IdL)	$id \circ s \to s$	(Ass)	$(s\circ t)\circ u \to s\circ(t\circ u)$

Remark 3.13 σ_0 is a particular case of the system *Subst* of *CCL*. Rules $(VrId)$ and $(ShId)$ are particular cases of the right identity rule. Hence, the techniques of (i) and (ii) above for showing SN for *SUBST* and σ_0 will have similar status with respect to s_e.

The methods of techniques (i) .. (iii) above do not apply to s_e:

- **Problem 1: Unable to use recursive path ordering** By taking a look at the s_e-rules (Definition 3.8), it becomes obvious that the unfriendly rules, with respect to SN, are *σ-σ-transition* and to a lesser extent *φ-σ-transition.* These rules prevent us from finding an order on the set of operators in order to solve the normalisation problem with a recursive path ordering (rpo).
- **Problem 2: Unable to use Zantema's distribution elimination lemma.** The s_e-rules "look like" associative rules but unfortunately they are not; e.g. in *σ-σ-transition* one could think the σ^j-operator distributes over the σ^i-operator, but it is not a "true" distribution: σ^j changes to σ^{j+1} when acting on the first term and to σ^{j-i+1} when acting on the second. This prevents use of Zantema's distribution elimination method [17] to show SN.

Another technique to show SN is modularity where SN is proved for certain subcalculi os s_e which are shown to satisfy a commutation property. We show in the next section that indeed s_e can be divided into two subcalculi which are SN, but that unfortunately, the needed commutation results do not hold.

3.4 Dividing s_e in two disjoint sets $s + *\varphi$ and $*\sigma$

Definition 3.14 We define the following sets of rules:
$*\varphi = \{\sigma\text{-}\varphi\text{-tr.1},\ \sigma\text{-}\varphi\text{-tr.2},\ \varphi\text{-}\varphi\text{-tr.1},\ \varphi\text{-}\varphi\text{-tr.2}\}$,
$*\sigma = \{\sigma\text{-}\sigma\text{-tr.},\ \varphi\text{-}\sigma\text{-tr.}\}$,
$*\varphi^{-} = \{\sigma\text{-}\varphi\text{-tr.1},\ \varphi\text{-}\varphi\text{-tr.2}\}$, $*\varphi^{--} = \{\sigma\text{-}\varphi\text{-tr.2},\ \varphi\text{-}\varphi\text{-tr.1}\}$.

Note that $s_e = (s+*\varphi)+*\sigma$. We shall prove in this section that both calculi generated by the set of rules $s+*\varphi$ (Theorem 3.17) and $*\sigma$ (Theorem 3.28) are SN. Unfortunately, these calculi do not possess the property of commutation needed to ensure that their union s_e is SN (see Example 3.31).

3.5 SN of $s + *\varphi$

We prove that $s + *\varphi$ is SN by giving a weight that decreases by reduction. We begin by defining two weight functions needed for the final weight:

Definition 3.15 Let $P : \Lambda s_{op} \to I\!N$ and $W : \Lambda s_{op} \to I\!N$ be defined by:

$$\begin{array}{ll}
P(X) = P(\mathtt{n}) = 2 & W(X) = W(\mathtt{n}) = 1 \\
P(a\,b) = P(a) + P(b) & W(a\,b) = W(a) + W(b) + 1 \\
P(\lambda a) = P(a) & W(\lambda a) = W(a) + 1 \\
P(a\,\sigma^j b) = j * P(a) * P(b) & W(a\,\sigma^j b) = 2 * W(a) * (W(b) + 1) \\
P(\varphi^i_k a) = (k+1) * (P(a) + 1) & W(\varphi^i_k a) = 2 * W(a)
\end{array}$$

Lemma 3.16 *For a, $b \in \Lambda s_{op}$ the following hold:*

(i) *If $a \to_{s+*\varphi} b$ then $W(a) \geq W(b)$.*
(ii) *If $a \to_{s+*\varphi^-} b$ then $W(a) > W(b)$.*
(iii) *If $a \to_{*\varphi^{--}} b$ then $P(a) > P(b)$.*

Proof. By induction on a: if the reduction is internal, the induction hypothesis applies; otherwise, the theorem must be checked for each rule. □

Theorem 3.17 *The $s + *\varphi$-calculus is SN.*

Proof. The previous lemma ensures that the ordinal $(W(a), P(a))$ decreases with the lexicographical order for each $s + *\varphi$-reduction. □

3.6 The $\lambda\omega$- and $\lambda\omega_e$-calculi

Recall that the $*\sigma$-calculus consists of the two painful rules σ-σ-tr. and φ-σ-tr. which are at the heart of our inability to use the rpo method or the methods of Zantema. In order to establish SN of $*\sigma$, we will use an isomorphism established in [13] between λs_e and $\lambda\omega_e$, a calculus written in the $\lambda\sigma$-style.

In order to express λs-terms in the $\lambda\sigma$-style, [13] split the closure operator of $\lambda\sigma$ (denoted in a semi-infix notation as $-[-]$) in a family of closures operators that were denoted also with a semi-infix notation as $-[-]_i$, where i ranges on the set of natural numbers. [13] also admitted as basic operators the iterations of $\uparrow$ and therefore had a countable set of basic substitutions $\uparrow^n$, where n ranges on the set of natural numbers. By doing so, the updating operators of λs become available as $-[\uparrow^n]_i$. Finally, [13] introduced a *slash* operator of sort `term` $\to$ `substitution` which transforms a term a into a substitution $a/$. This operator may be considered as *consing with id* (in the $\lambda\sigma$-jargon) and was first introduced and exploited in the $\lambda\upsilon$-calculus (cf. [2]). Here is the formalisation of this syntax and the rewriting rules of $\lambda\omega$:

Definition 3.18 The set $\Lambda\omega$ of terms of the $\lambda\omega$-calculus, is defined as $\Lambda\omega^t \cup \Lambda\omega^s$, where $\Lambda\omega^t$ and $\Lambda\omega^s$ are mutually defined as follows ($j \geq 1$ and $i \geq 0$):

$$\begin{array}{ll} \textbf{Terms} & \Lambda\omega^t ::= I\!N \mid \Lambda\omega^t\Lambda\omega^t \mid \lambda\Lambda\omega^t \mid \Lambda\omega^t[\Lambda\omega^s]_j \\ \textbf{Substitutions} & \Lambda\omega^s ::= \uparrow^i \mid \Lambda\omega^t/ \end{array}$$

The set, denoted $\lambda\omega$, of rules of the *$\lambda\omega$-calculus* is given as follows:

$$\boxed{\begin{array}{lrcl} \sigma\textit{-generation} & (\lambda a)\, b & \longrightarrow & a\,[b/]_1 \\ \sigma\textit{-app-transition} & (a\, b)[s]_j & \longrightarrow & (a\,[s]_j)\,(b\,[s]_j) \\ \sigma\textit{-}\lambda\textit{-transition} & (\lambda a)[s]_j & \longrightarrow & \lambda(a[s]_{j+1}) \\ \sigma\textit{-/-destruction} & \mathtt{n}[a/]_j & \longrightarrow & \begin{cases} \mathtt{n-1} & \text{if } n > j \\ a[\uparrow^{j-1}]_1 & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases} \\ \sigma\textit{-}\uparrow\textit{-destruction} & \mathtt{n}[\uparrow^i]_j & \longrightarrow & \begin{cases} \mathtt{n+i} & \text{if } n \geq j \\ \mathtt{n} & \text{if } n < j \end{cases} \end{array}}$$

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

Definition 3.19 Let $\mathbf{V}$ stand for a set of variables, over which X, Y, ... range. The set $\Lambda\omega_{op}$ of *open terms*, is defined as $\Lambda\omega^t_{op} \cup \Lambda\omega^s_{op}$, where $\Lambda\omega^t_{op}$ and $\Lambda\omega^s_{op}$ are mutually defined as follows ($j \geq 1$ and $i \geq 0$):

$$\begin{array}{ll} \textbf{Open Terms} & \Lambda\omega^t_{op} ::= \mathbf{V} \mid I\!N \mid \Lambda\omega^t_{op}\Lambda\omega^t_{op} \mid \lambda\Lambda\omega^t_{op} \mid \Lambda\omega^t_{op}[\Lambda\omega^s_{op}]_j \\ \textbf{Substitutions} & \Lambda\omega^s_{op} ::= \uparrow^i \mid \Lambda\omega^t_{op}/ \end{array}$$

We take a, b, c to range over $\Lambda\omega^t_{op}$ and s, t, $\ldots$ over $\Lambda\omega^s_{op}$. *Closures*, *pure terms* and *compatibility* are defined as expected. The set $\lambda\omega_e$ of rules of the

$\lambda\omega_e$*-calculus* is obtained by adding to $\lambda\omega$ the new following rules:

σ-/-*transition*	$a\,[b/]_k[s]_j$	$\longrightarrow$	$a\,[s]_{j+1}[b[s]_{j-k+1}/]_k$ if $k \leq j$
/-$\uparrow$-*transition*	$a\,[\uparrow^i]_k[b/]_j$	$\longrightarrow$	$\begin{cases} a[b/]_{j-i}[\uparrow^i]_k & \text{if } k+i \leq j \\ a[\uparrow^{i-1}]_k & \text{if } k \leq j < k+i \end{cases}$
$\uparrow$-$\uparrow$-*transition*	$a\,[\uparrow^i]_k[\uparrow^l]_j$	$\longrightarrow$	$\begin{cases} a[\uparrow^l]_{j-i}[\uparrow^i]_k & \text{if } k+i < j \\ a[\uparrow^{i+l}]_k & \text{if } k \leq j \leq k+i \end{cases}$

The set of rules of the ω_e-*calculus* is $\lambda\omega_e - \{\sigma\text{−}generation\}$.

Remark 3.20 Note that the rule schemes /-$\uparrow$ and $\uparrow$-$\uparrow$ can be merged into the single scheme $a\,[\uparrow^i]_k[s]_j \to a[s]_{j-i}[\uparrow^i]_k$ for $k+i<j$ but they must be kept distinct when $k+i=j$ if SN is to hold. The $\uparrow$-$\uparrow$-scheme, if admitted when $k+i=j$, may generate an infinite loop (e.g., if $i=k=l=1$ and $j=2$).

[13] established an isomorphism between λs_e and $\lambda\omega_e$ and also between λs and $\lambda\omega$. These isomorphisms translate properties of λs and λs_e to $\lambda\omega$ and $\lambda\omega_e$, respectively. Hence, all the results mentioned above concerning λs and λs_e translate into corresponding results for the sort `term` to $\lambda\omega$ and $\lambda\omega_e$.

Theorem 3.21 (cf. [13]) *The following hold:*

(i) *The ω-calculus is SN and confluent on $\Lambda\omega^t$.*

(ii) *Let $a, b \in \Lambda$. If $a \twoheadrightarrow_{\lambda\omega} b$ then $a \twoheadrightarrow_\beta b$. If $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda\omega} b$.*

(iii) *The $\lambda\omega$-calculus is confluent on $\Lambda\omega^t$.*

(iv) *Pure terms which are SN in the λ-calculus are also SN in the $\lambda\omega$-calculus.*

(v) *The ω_e-calculus is weakly normalising and confluent.*

(vi) *The $\lambda\omega_e$-calculus is confluent on open terms.*

(vii) *Let $a, b \in \Lambda$. If $a \twoheadrightarrow_{\lambda\omega_e} b$ then $a \twoheadrightarrow_\beta b$. If $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda\omega_e} b$.*

3.7 SN of $*\sigma$

To prove SN for $*\sigma$ we will use the isomorphism presented in Section 3.6 and the technique that Zantema used to prove SN for the calculus whose only rule is σ-σ-*transition* (cf. [11]). Following this isomorphism, the schemes σ-σ-*tr.* and φ-σ-*tr.* of λs_e both translate into the same scheme of $\lambda\omega_e$, namely σ-/-*transition* of Definition 3.19. Hence, to show that $*\sigma$ is SN, it is enough to show that the calculus whose only rule is σ-/-*transition*, let us call it σ-/-calculus, is SN. To do so, we use the following Lemma of Zantema (cf. [14]):

Lemma 3.22 *Any reduction relation $\to$ on a set T satisfying the three properties below is strongly normalising:*

(i) *$\to$ is weakly normalising.*

(ii) $\to$ *is locally confluent.*

(iii) $\to$ *is increasing, i.e.,* $\exists$ *function* $f : T \mapsto I\!N$ *where* $a \to b \Rightarrow f(a) < f(b)$.

For weak normalisation of the σ-$/$ $-$ *calculus* we use the technique of [11]:

Definition 3.23 We say that $c \in \Lambda\omega^t$ is an *external normal form* if $c = a[s_1]_{i_1} \cdots [s_n]_{i_n}$ where $a \neq e[d/]_k$ and if $s_k = b_k/$ then $i_k > i_{k+1}$. We denote the set of external normal forms ENF.

Lemma 3.24 *Let* $c = a[s_1]_{i_1} \cdots [s_n]_{i_n} \in ENF$ *and let* $i_n \leq i_{n+1}$ *and* $s_n = b_n/$ *then there exists a* σ-$/$*-derivation* $c \to^+ a[t_1]_{j_1} \cdots [t_{n+1}]_{j_{n+1}} \in ENF$ *such that* $j_{n+1} = i_n$ *and for every* r *with* $1 \leq r \leq n+1$ *we have either* $t_r = s_k$ *for some* $k \leq n+1$ *or* $t_r = (a_p[s_{n+1}])/$ *for some* $s_p = a_p/$ *with* $1 \leq p \leq n$.

Proof. By induction on n. □

Lemma 3.25 *Let* $c = a[s_1]_{i_1} \cdots [s_n]_{i_n}$ *such that* $a \neq e[d/]_k$. *There exists a* σ-$/$*-derivation* $c \twoheadrightarrow a[t_1]_{j_1} \cdots [t_n]_{j_n} \in ENF$ *such that for every* r *with* $1 \leq r \leq n+1$ *we have either* $t_r = s_k$ *for some* $k \leq n$ *or* $t_r = (a_{p_{r\,1}}[s_{p_{r\,2}}]_{k_2} \cdots [s_{p_{r\,n}}]_{k_n})/$ *with* $1 \leq p_{r\,1} \leq \cdots \leq p_{r\,n} \leq n$ *and with some* $s_p = a_{p_{r\,1}}/$ *(*$1 \leq p \leq n$*).*

Proof. By induction on n, using the previous lemma. □

Lemma 3.26 *The* σ-$/$*-calculus is weakly normalising.*

Proof. Assume a term c not having a normal form for which every term smaller (in size) than c admits a normal form. Let $c = a[s_1]_{i_1} \cdots [s_n]_{i_n}$ such that $a \neq e[d/]_k$. By Lemma 3.25, $c \twoheadrightarrow a[t_1]_{j_1} \cdots [t_n]_{j_n} \in ENF$. As a, t_1, $\cdots$ t_n are all smaller than c, they admit a normal form. Replacing each of them by its normal form in $a[t_1]_{j_1} \cdots [t_n]_{j_n}$ gives a normal form for c. Absurd. □

Theorem 3.27 *The* σ-$/$*-calculus is strongly normalising on* $\Lambda\omega^t$.

Proof. Use Lemma 3.22. (i) was shown in Lemma 3.26. (ii) follows from a critical pair analysis and (iii) is shown by choosing $f(a)$ to be the size of a.□

Since both rule schemes in $*\sigma$ translate into the single σ-$/$ rule scheme, the isomorphism gives:

Theorem 3.28 *The* $*\sigma$*-calculus is strongly normalising.*

3.8 Modularity fails

Now that $s + *\varphi$ and $*\sigma$ are SN the question arises whether the whole system can be shown SN using a modularity result. The answer is negative for the classical modularity theorem of Bachmair-Dershowitz, which we recall here.

Definition 3.29 A rewrite relation R commutes over S if whenever $a \to_S b \to_R c$, there is an alternative derivation $a \to_R d \to_{R \cup S} c$.

Theorem 3.30 (Bachmair-Dershowitz-85) *Let* R *commute over* S. *The combined system* $R \cup S$ *is SN iff* R *and* S *both are SN.*

Example 3.31 shows that no commutation exists between $s + *\varphi$ and $*\sigma$ and so the Bachmair-Dershowitz's Theorem cannot be used to get SN of s_e.

Example 3.31 To show that $*\sigma$ does not commute over $s + *\varphi$, let $k + i \leq j$, $h \leq j - i + 1$ and $h > k + 1$. Now take the following derivation:
$(\varphi^i_k(a\,\sigma^h b))\,\sigma^j c \to_{*\varphi} \varphi^i_k((a\,\sigma^h b)\,\sigma^{j-i+1} c) \to_{\sigma-\sigma-tr} \varphi^i_k((a\,\sigma^{j-i+2} c)\,\sigma^h (b\,\sigma^{j-i-h+2} c))$
It is easy to see that $(\varphi^i_k(a\,\sigma^h b))\,\sigma^j c$ does not contain any $*\sigma$-redex.

On the other hand, $s + *\varphi$ does not commute over $*\sigma$ either:
Let $i \leq j$ and let us consider the following derivation:
$((\lambda a)\,\sigma^i b)\,\sigma^j c) \to_{\sigma-\sigma-tr} ((\lambda a)\,\sigma^{j+1} c)\,\sigma^i (b\,\sigma^{j-i+1} c) \to_s (\lambda(a\,\sigma^{j+2} c))\,\sigma^i (b\,\sigma^{j-i+1} c)$
But reducing the only s-redex in $((\lambda a)\,\sigma^i b)\,\sigma^j c)$ we get $(\lambda(a\,\sigma^{i+1} b))\,\sigma^j c$ which also has a unique s-redex. Reducing it we get $\lambda((a\,\sigma^{i+1} b)\,\sigma^{j+1} c)$ and now there is only the *σ-σ-transition* redex which gives us $\lambda((a\,\sigma^{j+2} c)\sigma^{i+1}(b\,\sigma^{j-i+1} c))$ which has no further redexes. Therefore, $(\lambda(a\,\sigma^{j+2} c))\,\sigma^i (b\,\sigma^{j-i+1} c)$ cannot be reached from $((\lambda a)\,\sigma^i b)\,\sigma^i c)$ with an s_e-derivation beginning with an s-step.

4 Conclusion

This paper attempted two goals:

(i) It gave a calculus of explicit substitutions which allows local as well as global substitutions. We showed that this calculus simulates beta reduction and that the underlying calculus of substitutions is strongly normalising and confluent. A calculus of local explicit substitutions was given in [9], however that calculus did not enjoy good theoretical properties.

(ii) It explained the problems faced in showing that the s_e-calculus is strongly normalising. We are not sure whether the answer is positive or negative at this stage. We leave this problem as a challenge to the community.

References

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

[2] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Functional Programming*, 6(5), 1996.

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

[4] P-L Curien, T. Hardin, and A. Ríos. Strong normalisation of substitutions. *Logic and Computation*, 6:799–817, 1996.

[5] N. G. 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.

[6] B. Guillaume. *Un calcul des substitutions avec etiquettes.* PhD thesis, Université de Savoie, Chambéry, France, 1999.

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

[8] F. Kamareddine and R. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.

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

[10] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *LNCS*, 982:45–62, 1995.

[11] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, 1997.

[12] F. Kamareddine and A. Ríos. Bridging de Bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the IGPL*, 6(6):843–874, 1998.

[13] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-styles of explicit substitutions. *Logic and Computation*, 10(3):349–380, 2000.

[14] J.-W. Klop. Term rewriting systems. *Handbook of Logic in Computer Science*, II, 1992.

[15] S.L. Peyton-Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

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

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

[18] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.