Bridging the $\lambda\sigma$- and λs-Styles of Explicit Substitutions*

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. We present the $\lambda\omega$ and $\lambda\omega_e$ calculi, the two-sorted (`term` and `substitution`) versions of the λs (cf. [KR95a]) and λs_e (cf. [KR96a]) calculi, respectively. We establish an isomorphism between the λs-calculus and the `term` restriction of the $\lambda\omega$-calculus, which extends to an isomorphism between λs_e and the `term` restriction of $\lambda\omega_e$. Since the $\lambda\omega$ and $\lambda\omega_e$ calculi are given in the style of the $\lambda\sigma$-calculus (cf. [ACCL91]) they bridge calculi between λs and $\lambda\sigma$ and between λs_e and $\lambda\sigma$ and thus we are able to better understand one calculus in terms of the other. We improve our knowledge on the open problem of strong normalisation (SN) of the associated calculus of substitutions s_e by showing SN for two subcalculi (we use the isomorphism with $\lambda\omega_e$ for the proof of SN of one of them). Finally, we present typed versions of all the calculi and check that the above mentioned isomorphism preserves types.

As a consequence, the $\lambda\omega$-calculus is a calculus in the $\lambda\sigma$-style that simulates one step β-reduction, is confluent (on closed terms), preserves strong normalisation, its associated calculus of substitutions is SN, and its simply typed version is also SN. Moreover, the $\lambda\omega$-calculus possesses an extension $\lambda\omega_e$ that is confluent on open terms (terms with eventual metavariables of sort `term` only) and whose simply typed version is weakly normalising (on open term). As far as we know, $\lambda\omega$-calculus is the first calculus in the $\lambda\sigma$-style that has all those properties. However, the two open problems of the SN of the associated calculus of substitution of $\lambda\omega_e$ and of the preservation of strong normalisation of $\lambda\omega_e$ remain unsolved.

Introduction

There is no better way to start than by quoting Abelson and Sussman in [AS86]: *Despite the fact that substitution is a "straightforward idea", it turns out to be surprisingly complicated to give a rigorous mathematical definition of the substitution process... Indeed, there is a long history of erroneous definitions of "substitution" in the literature of logic and programming semantics.*

Most literature on the λ-calculus treats substitution as an atomic operation and leaves implicit the actual computational steps necessary to perform substitution. Substitution is usually defined with operators which do not belong to the

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

language of the λ-calculus. In any real implementation, the substitution required by β-reduction (and similar higher-order operations) must be implemented via smaller operations. Thus, there is a conceptual gap between the theory of the λ-calculus and its implementation in programming languages and proof assistants. Explicit substitution attempts to bridge this gap without abandoning the setting of the λ-calculus.

By representing substitutions in the structure of terms and by providing (first-order) reductions to propagate the substitutions, explicit substitution provides a number of benefits. A major benefit is that explicit substitution allows more flexibility in ordering work. Propagating substitutions through a particular subterm can wait until the subterm is the focus of computation. This allows a choice among the substitutions to be performed, thus improving locality of reference. Obtaining more control over the ordering of work has become an important issue in functional programming language implementation (cf. [Pey87]). The flexibility provided by explicit substitution also allows postponing unneeded work indefinitely (i.e., avoiding it completely). This can yield profits, since implicit substitution can be an inefficient, maybe even exploding, process by the many repetitions it causes. Another benefit is that explicit substitution allows formal modeling of the techniques used in real implementations, e.g., environments. Because explicit substitution is closer to real implementations, it has the potential to provide a more accurate cost model. (This possibility is particularly interesting in light of the difficulty encountered in formulating a useful cost model in terms of graph reduction [LM96, Pey87].)

Proof assistants may benefit from explicit substitution, due to the desire to perform substitutions locally and in a formal manner. Local substitutions are needed as follows. Given $xx[x{:=}y]$, one may not be interested in having yy as the result of $xx[x{:=}y]$ but rather only $yx[x{:=}y]$. In other words, one only substitutes one occurrence of x by y and continues the substitution later. Theorem provers like Nuprl [Con86] and HOL [GM93] implement substitution which allows the local replacement of some abbreviated term. This avoids a size explosion when it is necessary to replace a variable by a huge term only in specific places to prove a certain theorem.

Formalisation helps in studying the termination and confluence properties of systems. Without formalisation, important properties such as the correctness of substitutions often remain unestablished, causing mistrust in the implementation. In fact, it is known that the first implementation of substitution in Automath [NGdV94] was incorrect, and that most of the bugs in the implementation of LCF came from clashes of bound variables in strange situations [Pau90]. As the implementation of substitution in many theorem provers is not based on a formal system, it is not clear what properties their underlying substitution has, nor can their implementations be compared. Thus, it helps to have a choice of explicit substitution systems whose properties have already been established. This is witnessed by the recent theorem prover ALF, which is formally based on Martin-Löf's type theory with explicit substitution [Mag95]. Another justification for explicit substitution in theorem proving is that some researchers believe "tactics" can be replaced by the notion of incomplete proofs, which are believed to need explicit substitutions [MH95, Mag95].

The last fifteen years have seen an increasing interest in formalising substitution explicitly; various calculi including new operators to denote substitution have been proposed. Amongst these calculi we mention $C\lambda\xi\phi$ [dB78]; the calculi of categorical combinators [Cur86]; $\lambda\sigma$ [ACCL91], $\lambda\sigma_{\Uparrow}$ [CHL92], $\lambda\sigma_{SP}$ [Río93], referred to as the $\lambda\sigma$-family; $\lambda\upsilon$ [BBLRD95], a descendant of the $\lambda\sigma$-family; $\varphi\sigma BLT$ [KN93], $\lambda\mathtt{exp}$ [Blo95], λs [KR95a], λs_e [KR96a], and $\lambda\zeta$ [MH95]. All these calculi (except $\lambda\mathtt{exp}$) are described in a de Bruijn setting where natural numbers play the role of variables and the set of terms Λ on which substitution will be made explicit is defined by: $\Lambda ::= \mathbb{N} \mid (\Lambda\Lambda) \mid (\lambda\Lambda)$. But, why so many varieties of systems of explicit substitutions and the search still continues? The following section attempts to explain:

The $\lambda\sigma$-calculus (cf. [ACCL91]) reflects in its choice of operators and rules the calculus of categorical combinators (cf. [Cur86]). The main innovation of the $\lambda\sigma$-calculus is the division of terms in two sorts: sort `term` and sort `substitution`. We depart from this style of explicit substitutions in two ways. First, we keep the classical and unique sort `term` of the λ-calculus. Second, we do not use some of the categorical operators, especially those which are not present in the classical λ-calculus. We introduce new operators which reflect the substitution and updating that are only present in the meta-language of the λ-calculus. By doing so, we believe that our calculi are closer to the λ-calculus from an intuitive point of view, rather than a categorical one.

But, what properties does one look for in calculi that are to be the basis for programming languages? We attempt to list some of those desired properties for calculi of explicit substitutions:

1. **Termination or Strong Normalisation (SN):** For a calculus of explicit substitutions $\lambda\mathtt{subst}$, does the underlined calculus of substitutions `subst` terminate? This question is of course important. One does not want to include non-terminating rules to the λ-calculus.
2. **Confluence (CR):** Is the substitution calculus $\lambda\mathtt{subst}$ confluent on:
   (a) Ground terms? (I.e. terms of Λ above with explicit substitutions)
   (b) Open terms? It is possible to consider, besides the classical variables (now numbers), real variables (which correspond to meta-variables in the classical setting). The terms obtained with this extended syntax are called *open terms* and they can be considered as *contexts*, the new variables corresponding to *place-holders*. The interest in studying the calculi on open terms is that they alow, for instance, the representation of incomplete proofs where the place-holder stands for the still unknown part of the proof. Calculi on open terms have also provided the tools to prune the search space in unification algorithms (cf. [DHK95]).
3. **Simulation of β-reduction:** If a evaluates in the λ-calculus (using only β-reduction) to b, does a evaluate to b in the λsubst-calculus (using the β-rule and the substitutions rules)?

4. **Preservation of Termination (PSN):** If a terminates in the λ-calculus, does it terminate in the λsubst-calculus?

The first calculus of explicit substitutions $\lambda\sigma$ (after the categorical combinators), did enjoy 1, 2a, and 3. It did not possess 2b however. Therefore, $\lambda\sigma_{\Uparrow}$ [HL89, CHL92] was proposed. $\lambda\sigma_{\Uparrow}$ is a variant of $\lambda\sigma$ that is confluent on open terms. Nevertheless, 4 remained unknown for $\lambda\sigma$ or $\lambda\sigma_{\Uparrow}$ until Melliès proved that $\lambda\sigma_{\Uparrow}$ (as well as both the rest of the $\lambda\sigma$-family and the categorical combinators) does not preserve SN [Mel95]. Since then till now, it has remained an open problem whether there is indeed a calculus of substitutions that satisfies all the properties $1 \ldots 4$.

$\lambda\sigma_{\Uparrow}$ satisfies 1, 2, and 3, whereas $\lambda\upsilon$ achieves 1, 2a, 3 and 4 by removing the composition operator. However, [FKP96] provides two calculi of explicit substitutions that have the composition operator and still have PSN. As we said, a calculus enjoying 1 to 4 remains to be found (remark that $\lambda\upsilon$ and the calculi of [FKP96] do not enjoy 2b).

Recently, the $\lambda\zeta$-calculus (cf. [MH95]) has been proposed as a calculus which preserves strong normalisation and is itself confluent on open terms. In other words, $\lambda\zeta$ satisfies 1, 2, and 4. The $\lambda\zeta$-calculus works with two new applications that allow the passage of substitutions within classical applications only if these applications have a head variable. This is done to cut the branch of the critical pair which is responsible for the non-confluence of $\lambda\upsilon$ on open terms. Hence, $\lambda\zeta$ preserves strong normalisation and is itself confluent on open terms. Unfortunately, $\lambda\zeta$ is not able to simulate one step β-reduction as shown in [MH95]. Instead, it simulates only a "big step" β-reduction.

Another line of expliciting substitutions has been made in [KN93, KR95a, KR96a, KR96, KRW97]. In [KN93], the λ-calculus was rewritten using a notation influenced strongly by de Bruijn's notation for Automath [NGdV94]. In that notation [KN95], every λ-term is simply a sequence of *items* followed by a variable. This *item notation*, allowed also the introduction of so called *substitution items* and the inclusion of rules that explicit the passage of substitution. Alas however, the calculus of [KN93] does not satisfy 1 nor 2 nor 4. For this reason, [KR95a] set out to find the part of the calculus of [KN93] that satisfies as much of 1 to 4 as possible. The solution was to extend the λ-calculus with explicit substitutions by turning de Bruijn's meta-operators into object-operators. (Mention to a very close calculus to the λs-calculus can be already found in [Cur86], exercise 1.2.7.2, where reference to previous unpublished notes of Y. Lafont is given.) The resulting calculus λs remains intuitively as close to the λ-calculus as possible for a calculus of explicit substitution. λs (like $\lambda\upsilon$) satisfies all of 1, 2a, 3 and 4. Moreover, λs has an extension λs_e (cf. [KR96a]) that is confluent on open terms (hence λs_e satisfies 2a and 2b). Also, λse satisfies 3. It is still an open problem whether λs_e satisfies 1 or 4. And, it remains to find a calculus that satisfies all of 1 to 4.

The presence of such varieties of calculi of explicit substitutions, and the puzzling open problems, make it desirable to find a common framework between both styles so that maybe their complementary properties can be combined and hence a calculus satisfying $1 \ldots 4$ may be found.

All the above discussion was concerned with the type-free λ-calculus extended with explicit substitutions. However, type theory is at the heart of the theory and implementation of programming languages and theorem provers. For this reason, no calculus can really bridge the gap between theory and implementation and be a useful one for programming languages and theorem proving if there was no way to accommodate types. So, is there also a story for typed explicit substitutions?

Yes there is. And the story is as fascinating as that of the type-free λ-calculus with explicit substitution. It goes as follows:

$\lambda\sigma$ does not preserve strong normalisation and the counterexample given in [Mel95] to prove it happens to be a very decent typable term. Therefore, typed $\lambda\sigma$ is not SN. On the other hand, $\lambda\upsilon$ preserves strong normalisation and its simply typed version is strongly normalising. The same applies to λs which, (like $\lambda\upsilon$) preserves strong normalisation and its simply typed version is strongly normalising. Extending second order λ-calculus with explicit substitutions remains an active subject of research.

Recently, $\lambda\sigma$ (with open terms) has been shown to be weakly normalising [GL97]. [KR97] applied a revised version of the technique of [GL97] to show that the simply typed version of λs_e (with open terms) is also weakly normalising. It remains to be seen whether the simply typed λs_e is strongly normalising (the simply typed $\lambda\sigma$ with open terms is not strongly normalising).

Hence, there are three open problems for the λs_e-calculus: 1. is the s_e-calculus SN, 2. does the λs_e-calculus preserve SN and 3. is the simply typed λs_e calculus SN. 1 has been positively answered for the calculus of substitutions associated with $\lambda\sigma$ (and its confluent extension $\lambda\sigma_{\Uparrow}$). 2 and 3 have been negatively answered for $\lambda\sigma$ and $\lambda\sigma_{\Uparrow}$.

Based on this, we believe that a formulation of λs and λs_e in the $\lambda\sigma$-style could be useful to tackle these problems or at least to better understand one calculus in terms of the other. Therefore we provide the $\lambda\omega$ and $\lambda\omega_e$ calculi, which are two-sorted: sort `term` and sort `substitution`, and hence closer to $\lambda\sigma$. When restricting these calculi to the sort `term` we obtain calculi which are isomorphic to λs and λs_e, respectively.

In Section 1 we review the calculi of substitutions $\lambda\sigma$, λs and λs_e and their properties. The new calculi $\lambda\omega$ and $\lambda\omega_e$ are introduced in Section 2. In Section 3 we give the isomorphisms between $\lambda\omega$ and λs and between $\lambda\omega_e$ and λs_e. In Section 4 we prove some results concerning strong normalisation of subcalculi of s_e. One of this results is obtained by using the isomorphism with ω_e and the strong normalisation of the corresponding subcalculus of ω_e, which is easier to obtain. The other result of SN is obtained using an appropriate lexicographical weight. In Section 5 we recall the typed versions of λs, λs_e and $\lambda\sigma$ and introduce the typed $\lambda\omega$ and $\lambda\omega_e$ calculi. We conclude by proving that the isomorphism introduced in Section 3 preserves types.

1 The old calculi

1.1 The classical λ-calculus in de Bruijn notation

We assume the reader familiar with de Bruijn notation (cf. [dB72, KR96b]). We define Λ, 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 Λ 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, and write $\rightarrow^+$ and $\twoheadrightarrow$ to denote the transitive and the reflexive transitive closures of a reduction notion $\rightarrow$. We say that a reduction $\rightarrow$ is *compatible on* Λ when for all a, b, $c \in \Lambda$, we have $a \rightarrow b$ implies $a\,c \rightarrow b\,c$, $c\,a \rightarrow c\,b$ and $\lambda a \rightarrow \lambda b$.

In order to define β-reduction à la de Bruijn, we must define the substitution of a variable $\mathtt{n}$ for a term b in a term a. Therefore, we need to update the term b:

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

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

$$U_k^i(\lambda a) = \lambda(U_{k+1}^i(a))$$

Now we define the family of meta-substitution functions:

Definition 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:*

$$(a_1a_2)\{\!\{\mathtt{j} \leftarrow b\}\!\} \;=\; (a_1\{\!\{\mathtt{j} \leftarrow b\}\!\})\,(a_2\{\!\{\mathtt{j} \leftarrow b\}\!\}) \qquad \mathtt{n}\{\!\{\mathtt{j} \leftarrow b\}\!\} \;=\; \begin{cases} \mathtt{n-1} & \textit{if } \ n > j \\ U_0^j(b) & \textit{if } \ n = j \\ \mathtt{n} & \textit{if } \ n < j\,. \end{cases}$$

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

The following gives the properties of meta-substitution and updating (cf. [KR95a]):

Lemma 1 *Let* a, b, $c \in \Lambda$. *We have:*

1. *for* $k < n < k + i : U_k^{i-1}(a) = U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\}$.
2. *for* $l \leq k < l + j : U_k^i(U_l^j(a)) = U_l^{j+i-1}(a)$.
3. *for* $k + i \leq n : U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n-i+1} \leftarrow b\}\!\})$.
4. *for* $i \leq n : a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n+1} \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n-i+1} \leftarrow c\}\!\}\}\!\}$.
5. *for* $l + j \leq k + 1 : U_k^i(U_l^j(a)) = U_l^j(U_{k+1-j}^i(a))$.
6. *for* $n \leq k + 1 : U_k^i(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U_{k+1}^i(a)\{\!\{\mathtt{n} \leftarrow U_{k-n+1}^i(b)\}\!\}$.

Definition 3 β-reduction *is the least compatible reduction on* Λ *generated by:*

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

The λ-calculus (à la de Bruijn)*, is the reduction system whose only rewriting rule is* β*.*

Theorem 1 *The λ-calculus à la de Bruijn is confluent.*

1.2 The λs-calculus

The idea of λs is to handle explicitly the meta-operators given in definitions 1 and 2. Therefore, 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. Each σ^j is an infix operator of arity 2 and $a\,\sigma^j b$ has as intuitive meaning the term a where all free occurrences of the variable corresponding to the de Bruijn index j are to be substituted by the term b.
- $\{\varphi^i_k\}_{k\geq 0\ i\geq 1}$, which denotes the updating functions necessary when working with de Bruijn numbers to fix the variables of the term to be substituted.

Definition 4 *The* set of terms, *noted* Λs , of the λs-calculus *is given as follows:*

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

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. *Λ denotes the set of pure terms.*

A compatible reduction *on Λs is a reduction $\to$ such that 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 $\varphi^i_k a \to \varphi^i_k b$.*

We include, besides the rule mimicking the β-rule (*σ-generation*), a set of rules which are the equations in definitions 1 and 2 oriented from left to right.

Definition 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 following rules:*

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	$\mathtt{n}\,\sigma^j b \longrightarrow \begin{cases} \mathtt{n-1} & \textit{if} \quad n > j \\ \varphi^j_0\, b & \textit{if} \quad n = j \\ \mathtt{n} & \textit{if} \quad n < j \end{cases}$
φ-λ-transition	$\varphi^i_k(\lambda a) \longrightarrow \lambda(\varphi^i_{k+1}\, a)$
φ-app-transition	$\varphi^i_k(a_1\, a_2) \longrightarrow (\varphi^i_k\, a_1)\,(\varphi^i_k\, a_2)$
φ-destruction	$\varphi^i_k\,\mathtt{n} \longrightarrow \begin{cases} \mathtt{n+i-1} & \textit{if} \quad n > k \\ \mathtt{n} & \textit{if} \quad n \leq k \end{cases}$

We use λs to denote this set of rules. The s-calculus, the calculus of substitutions associated with the λs-calculus, *is the reduction system generated by the set of rules $s = \lambda s - \{\sigma\text{-generation}\}$.*

The σ-*generation* rule starts β-reduction by generating a substitution operator at the first level (σ^1). The σ-*app* and σ-λ rules allow this operator to travel throughout the term until its arrival to the variables. If a variable should be affected by the substitution, the σ-*destruction* rules (case $j = n$) carry out the substitution of the variable by the updated term, thus introducing the updating operators. Finally, the other rules compute the updating.

We state now the main properties of the λs-calculus (cf. [KR95a, KR95b]).

Theorem 2 (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 $s(a)$.*

Lemma 2 *The following holds:*

1. *The set of s-normal forms is exactly Λ.*
2. *For all $a, b \in \Lambda s$ we have:*
   $s(a\,b) = s(a)s(b)\;,\quad s(\lambda a) = \lambda(s(a))\;,\quad s(\varphi^i_k a) = U^i_k(s(a))\;,\quad s(a\,\sigma^j b) = s(a)\{\!\{\mathtt{j} \leftarrow s(b)\}\!\}\;.$
3. *Let $a, b \in \Lambda s$, if $a \to_{\sigma-gen} b$ then $s(a) \twoheadrightarrow_\beta s(b)$.*

Corollary 1 *Let $a, b \in \Lambda s$, if $a \twoheadrightarrow_{\lambda s} b$ then $s(a) \twoheadrightarrow_\beta s(b)$.*

Corollary 2 (Soundness) *Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda s} b$ then $a \twoheadrightarrow_\beta b$.*

Lemma 3 (Simulation of β-reduction) *Let $a, b \in \Lambda$, if $a \to_\beta b$ then $a \to \to_{\lambda s} b$.*

Theorem 3 (CR of λs) *The λs-calculus is confluent on Λs.*

Theorem 4 (Preservation of SN) *Pure terms which are strongly normalising in the λ-calculus are also strongly normalising in the λs-calculus.*

1.3 The λs_e-calculus

We introduce the open terms and the rules that extend λs to obtain the λs_e-calculus.

Definition 6 *The set of* open terms*, noted Λs_{op} is given as follows:*

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

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 following counterexample:

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

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. Moreover, the above example shows that even local confluence is lost. But since $((\lambda X)\sigma^1 \mathtt{1})(Y\sigma^1 \mathtt{1}) \to\to (X\sigma^2 \mathtt{1})\sigma^1(Y\sigma^1 \mathtt{1})$, the solution to the problem seems at hand if one has in mind the properties of meta-substitutions and updating functions of the λ-calculus in the Bruijn notation (cf. Lemma 1). These properties are equalities which can be given a suitable orientation and the new rules, thus obtained, added to λs give origin to a rewriting system which happens to be locally confluent (cf. [KR95b]). For instance, the rule corresponding to the Meta-substitution lemma (Lemma 1.4) is the *σ-σ-transition* rule. The addition of this rule solves the critical pair in our counterexample, since now we have $(X\sigma^1 Y)\sigma^1 \mathtt{1} \to (X\sigma^2 \mathtt{1})\sigma^1(Y\sigma^1 \mathtt{1})$.

Definition 7 *The set of rules λs_e is obtained by adding the rules given below to the set λs.*

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* $i \le j$
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* $k+i \le j$
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* $j \le k+1$
φ-φ-transition 1	$\varphi^i_k\,(\varphi^j_l\, a) \longrightarrow \varphi^j_l\,(\varphi^i_{k+1-j}\, a)$	*if* $l+j \le k$
φ-φ-transition 2	$\varphi^i_k\,(\varphi^j_l\, a) \longrightarrow \varphi^{j+i-1}_l\, a$	*if* $l \le k < l+j$

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* calculus of substitutions associated with the λs_e-calculus *is the rewriting system generated by the set of rules* $s_e = \lambda s_e - \{\sigma\text{-}generation\}$ *and we call it* s_e-calculus.

In [KR96a] we proved the following:

Theorem 5 (WN and CR of s_e) *The s_e-calculus is weakly normalising and confluent.*

Lemma 4 (Simulation of β-reduction) *Let $a, b \in \Lambda$, if $a \to_\beta b$ then $a \to\to_{\lambda s_e} b$.*

Theorem 6 (CR of λs_e) *The λs_e-calculus is confluent on open terms.*

Theorem 7 (Soundness) *Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda s_e} b$ then $a \twoheadrightarrow_\beta b$.*

1.4 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 $\lambda\sigma$-calculus:

Definition 8 *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 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* $\mathtt{n}$ *by the term* $\mathtt{1}[\uparrow^{n-1}]$. *By so doing, we have* $\Lambda \subset \Lambda\sigma^t$.

β-reduction of the λ-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 σ-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 σ- and $\lambda\sigma$-calculi (cf. [ACCL91, HL86]):

Theorem 8 *The* σ*-calculus is strongly normalising (SN) and confluent (CR).*

Theorem 9 *The* $\lambda\sigma$*-calculus is confluent.*

It is well known that the $\lambda\sigma$-calculus is not confluent on open terms, furthermore it is not even locally confluent. To obtain local confluence four rules must be added, and the calculus thus obtained is called the $\lambda\sigma_{SP}$-calculus.

Definition 9 *The $\lambda\sigma_{SP}$-calculus is obtained by adding to $\lambda\sigma$ the rules given below and by deleting the rules (VarId) and (ShiftId), since both of them are instances of the new rules.*

(Id)	$a[id] \longrightarrow a$
(IdR)	$s \circ id \longrightarrow s$
(VarShift)	$1 \cdot \uparrow \longrightarrow id$
(SCons)	$1[s] \cdot (\uparrow \circ s) \longrightarrow s$

Even the $\lambda\sigma_{SP}$-calculus is not confluent on open terms (terms which admit metavariables of both sorts), as shown in [CHL92], but it is confluent when the set of open terms is restricted to those which admit metavariables of sort `term` only [Río93].

2 The new calculi

2.1 The $\lambda\omega$-calculus

In order to express λs-terms in the $\lambda\sigma$-style we are going to split the closure operator of $\lambda\sigma$ (denoted in a semi-infix notation as $-[-]$) in a family of closures operators that shall be denoted also with a semi-infix notation as $-[-]_i$, where i ranges on the set of natural numbers.

We will admit as basic operators the iterations of $\uparrow$ and therefore we will have 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 are available in our new syntax as $-[\uparrow^n]_i$.

Finally, we introduce a *slash* operator of sort `term` $\rightarrow$ `substitution` wich transform a term a into a substitution $a/$. This operator may be considered as *consing with id* (in the $\lambda\sigma$-jargon) and has been exploited in the $\lambda\upsilon$-calculus (cf. [BBLRD95]).

Here is the formalisation of this syntax and the rewriting rules of $\lambda\omega$:

Definition 10 *The set of terms of the $\lambda\omega$-calculus, noted $\Lambda\omega$, is defined as $\Lambda\omega^t \cup \Lambda\omega^s$, where $\Lambda\omega^t$ and $\Lambda\omega^s$ are mutually defined as follows:*

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

*where $j \geq 1$ and $i \geq 0$. The set, denoted $\lambda\omega$, of rules of the $\lambda\omega$-*calculus *is given as follows:*

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

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

As we said before, the /-operator is present in $\lambda\upsilon$. Furthermore, the constant substitutions $\uparrow^i$ are exploited in the recent calculus of Muñoz [MH97] $\lambda\phi$. This calculus is so designed to avoid the non left linear rule *(SCons)* of $\lambda\sigma_{SP}$. Moreover, our indexed substitutions are reminiscent of the substitutions of the $\lambda\chi$-calculus with *levels* considered in [LRD96].

However, there is an essential difference between $\lambda\omega$ and $\lambda\chi$: in $\lambda\chi$ the terms (which are described with variable names) are stratified in levels whereas this is not the case for the $\lambda\omega$-terms. There is also an essential difference between $\lambda\phi$ and $\lambda\omega$ concerning the substitutions: composition is a basic operator in $\lambda\phi$ but it does not exist in $\lambda\omega$.

It is interesting to realize that the iterations $\uparrow^i$ as basic operators as well as the indexed substitutions are features which are tacitly embodied in λs since, as we shall prove in the next section, $\lambda\omega$ and λs are isomorphic.

2.2 The $\lambda\omega_e$-calculus

As we pointed out in Section 1.3 the λs-calculus is not even locally confluent on open terms. The same negative result can be easily transferred to the $\lambda\omega$-calculus.

By open terms in this new syntax we mean terms which admit variables (usually called metavariables) of sort `term` but not metavariables of sort `substitution`. In the $\lambda\sigma$-jargon they are often referred as *semi-closed* or pure terms (cf. [Río93]).

Now, we define formally what we mean by open terms in our new syntax and give the $\lambda\omega_e$-rules:

Definition 11 *The set of* open terms*, noted* $\Lambda\omega_{op}$ *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:*

Open Terms $\Lambda\omega^t_{op} ::= \mathbf{V} \mid \mathbb{N} \mid \Lambda\omega^t_{op}\Lambda\omega^t_{op} \mid \lambda\Lambda\omega^t_{op} \mid \Lambda\omega^t_{op}[\Lambda\omega^s_{op}]_j$
Substitutions $\Lambda\omega^s_{op} ::= \uparrow^i \mid \Lambda\omega^t_{op}/$

where $j \geq 1$ and $i \geq 0$, and where $\mathbf{V}$ *stands for a set of variables, over which* X, Y, ... *range. We take* a, b, c *to range over* $\Lambda\omega^t_{op}$ *and* s, t, $\ldots$ *over* $\Lambda\omega^s_{op}$. *Furthermore,* closures, pure terms *and* compatibility *are defined as for* $\Lambda\omega$. *The set, denoted* $\lambda\omega_e$, *of rules of the* $\lambda\omega_e$-calculus *is obtained by adding to the set of rules* $\lambda\omega$ *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* $\lambda\omega_e - \{\sigma-generation\}$.

Remark that the rule schemes /-$\uparrow$ and $\uparrow$-$\uparrow$ can be merged into the single scheme

$$a\,[\uparrow^i]_k[s]_j \to a[s]_{j-i}[\uparrow^i]_k \qquad \text{for } k+i<j$$

but they must be kept distinct for the case $k+i=j$ if SN is to be preserved. In fact, the $\uparrow$-$\uparrow$-scheme, if admited in the case $k+i=j$, may generate an infinite loop by itself (take for instance $i=k=l=1$ and $j=2$).

3 The isomorphisms

We define in this section two functions, that are inverse of each other, and that establish an isomorphism between λs_e and $\lambda\omega_e$. Furthermore, their restriction to ground terms also establish an isomorphism between λs and $\lambda\omega$. These isomorphisms translate all the properties of λs and λs_e to $\lambda\omega$ and $\lambda\omega_e$, respectively. We remark that the sets of terms Λs and Λs_{op} correspond with the sets of terms $\Lambda\omega^t$ and $\Lambda\omega^t_{op}$, respectively, rather than $\Lambda\omega$ and $\Lambda\omega_{op}$. Thus, it is only the sort `term` that is involved in the isomorphism.

Definition 12 *The functions* $T : \Lambda s_{op} \to \Lambda\omega^t_{op}$ *and* $S : \Lambda\omega^t_{op} \to \Lambda s_{op}$ *are defined inductively by:*

$T(X) = X$	$S(X) = X$
$T(\mathtt{n}) = \mathtt{n}$	$S(\mathtt{n}) = \mathtt{n}$
$T(a\,b) = T(a)T(b)$	$S(a\,b) = S(a)S(b)$
$T(\lambda a) = \lambda T(a)$	$S(\lambda a) = \lambda S(a)$
$T(a\,\sigma^j b) = T(a)[T(b)/]_j$	$S(a\,[b/]_j) = S(a)\,\sigma^j S(b)$
$T(\varphi^i_k a) = T(a)[\uparrow^{i-1}]_{k+1}$	$S(a\,[\uparrow^i]_k) = \varphi^{i+1}_{k-1}(S(a))$

We make an "abus de notation" and use the same names T and S for the trivial restrictions of these functions to ground terms. The context will be always clear enough in order to avoid ambiguities.

Theorem 10 *The following hold:*

1. *For all $a, b \in \Lambda s$, if $a \to_s b$ then $T(a) \to_\omega T(b)$.*
2. *For all $a, b \in \Lambda s$, if $a \to_{\lambda s} b$ then $T(a) \to_{\lambda\omega} T(b)$.*
3. *For all $a, b \in \Lambda s_{op}$, if $a \to_{s_e} b$ then $T(a) \to_{\omega_e} T(b)$.*
4. *For all $a, b \in \Lambda s_{op}$, if $a \to_{\lambda s_e} b$ then $T(a) \to_{\lambda\omega_e} T(b)$.*

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

Theorem 11 *The following hold:*

1. *For all $a, b \in \Lambda\omega^t$, if $a \to_\omega b$ then $S(a) \to_s S(b)$.*
2. *For all $a, b \in \Lambda\omega^t$, if $a \to_{\lambda\omega} b$ then $S(a) \to_{\lambda s} S(b)$.*
3. *For all $a, b \in \Lambda\omega^t_{op}$, if $a \to_{\omega_e} b$ then $S(a) \to_{s_e} S(b)$.*
4. *For all $a, b \in \Lambda\omega^t_{op}$, if $a \to_{\lambda\omega_e} b$ then $S(a) \to_{\lambda s_e} S(b)$.*

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

We verify finally that T and S are in fact inverse of each other.

Theorem 12 *The following hold:*

1. *For all $a \in \Lambda\omega^t$, we have $T(S(a)) = a$.*
2. *For all $a \in \Lambda\omega^t_{op}$, we have $T(S(a)) = a$.*
3. *For all $a \in \Lambda s$, we have $S(T(a)) = a$.*
4. *For all $a \in \Lambda s_{op}$, we have $S(T(a)) = a$.*

Proof: By an easy induction on a. □

Now that the calculi have been proved isomorphic, all the results of sections 1.2 and 1.3 concerning λs and λs_e translate into corresponding results for the sort `term` to $\lambda\omega$ and $\lambda\omega_e$.

Theorem 13 *The following hold:*

1. *The ω-calculus is SN and confluent on $\Lambda\omega^t$.*
2. *Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda\omega} b$ then $a \twoheadrightarrow_\beta b$.*
3. *Let $a, b \in \Lambda$, if $a \to_\beta b$ then $a \twoheadrightarrow_{\lambda\omega} b$.*
4. *The $\lambda\omega$-calculus is confluent on $\Lambda\omega^t$.*
5. *Pure terms which are SN in the λ-calculus are also SN in the $\lambda\omega$-calculus.*

Theorem 14 *The following hold:*

1. *The ω_e-calculus is weakly normalising and confluent.*

2. *The $\lambda\omega_e$-calculus is confluent on open terms.*
3. *Let $a, b \in \Lambda$, if $a \twoheadrightarrow_{\lambda\omega_e} b$ then $a \twoheadrightarrow_\beta b$.*
4. *Let $a, b \in \Lambda$, if $a \rightarrow_\beta b$ then $a \twoheadrightarrow_{\lambda\omega_e} b$.*

Remark that the schemes *σ-σ-tr.* and *φ-σ-tr.* of λs_e both translate in the same scheme of $\lambda\omega_e$, namely *σ-/-transition*. We shall exploit this fact in the next section.

4 Strong normalisation of subcalculi of s_e

As we said in the Introduction the strong normalisation of s_e is still an open problem. However, between the positive answer to strong normalisation for s and the unknown answer to the strong normalisation of s_e, partial results can be stated. In this section we show two subcalculi of s_e which contain s and which are strongly normalising.

Let us summarize first the main problems that we face when trying to establish SN for s_e. Only by taking a quick look at the rules 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 from establishing an order on the set of operators in order to solve the normalisation problem with a recursive path ordering. Furthermore, these rules "look like" associative rules but unfortunately they are not; e.g. in *σ-σ-transition* one could think as the σ^j-operator distributing 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 prevent us to use Zantema's distribution elimination method [Zan94] to obtain SN. Finally, another technique to show SN is modularity, i.e. establish SN for certain subcalculi and afterwards prove that these subcalculi satisfy a commutation property to conclude SN for the whole calculus. At the end of this section we will come back to this point and show that the necessary commutation results do not hold.

Let us say here that, even if *σ-σ-transition* seems responsible for the difficulties in establishing SN, Zantema succeded in establishing that the *σ-σ-transition* scheme on its own is SN (personal communication cited in [KR96a]). Here we shall go a step further: we shall prove that *σ-σ-tr.+φ-σ-tr.* is SN and also that the calculus obtained with the rest of the rules is SN as well.

In this section we shall frequently use the following nomenclature:

Definition 13 *We define the following sets of rules:*
$*\varphi = \{$*σ-φ-tr.1, σ-φ-tr.2, φ-φ-tr.1, φ-φ-tr.2*$\}$,
$*\sigma = \{$*σ-σ-tr., φ-σ-tr.*$\}$,
$*\varphi^- = \{$*σ-φ-tr.1, φ-φ-tr.2*$\}$, $*\varphi^{--} = \{$*σ-φ-tr.2, φ-φ-tr.1*$\}$.

We shall prove in this section that both calculi generated by the set of rules $s + *\varphi$ and $*\sigma$ are SN. Unfortunately, these calculi do not possess the property of commutation needed to ensure that their union is SN. We show counterexamples at the end of the section.

It is not difficult to prove that $s + *\varphi$ is SN by giving a weight that decreases through reduction. We begin by definig two weight functions we will need for the final weight:

Definition 14 *Let $P : \Lambda s_{op} \to \mathbb{N}$ and $W : \Lambda s_{op} \to \mathbb{N}$ be defined inductively by:*

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

Lemma 5 *For $a, b \in \Lambda s_{op}$ the following hold:*

1. *If $a \to_{s+*\varphi} b$ then $W(a) \geq W(b)$.*
2. *If $a \to_{s+*\varphi^-} b$ then $W(a) > W(b)$.*
3. *If $a \to_{*\varphi^{--}} b$ then $P(a) > P(b)$.*

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

An immediate consequence of the previous lemma is:

Theorem 15 *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. □

Now, to prove SN for $*\sigma$ we are going to use the isomorphism presented in the previous section and the technique that Zantema used to prove SN for the calculus whose only rule is *σ-σ-transition* (cf. [KR96a]). Zantema uses the following lemma (cf. [Klo92]):

Lemma 6 *Any reduction relation $\to$ on a set T satisfying 1,2, and 3 is strongly normalising:*

1. *$\to$ is weakly normalising.*
2. *$\to$ is locally confluent.*
3. *$\to$ is increasing, i.e., $\exists$ a function $f : T \longrightarrow \mathbb{N}$ where $a \to b \Rightarrow f(a) < f(b)$.*

We use the previous lemma to prove that the calculus whose only rule is *σ-/-transition*, let us call it σ-/-calculus, is strongly normalising.

For σ-/, 2 follows from a simple critical pair analysis and 3 can be easily established by choosing $f(a)$ to be the size of a. To show weak normalisation of σ-/ the technique used by Zantema (cf. [KR96a]) can be adapted to our calculus:

Definition 15 *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 $a \neq c[d/]_k$ and if $s_k = b_k/$ then $i_k > i_{k+1}$.*

We denote the set of external normal forms ENF.

Lemma 7 *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 8 *Let* $c = a[s_1]_{i_1} \cdots [s_n]_{i_n}$ *such that* $a \neq c[d/]_k$. *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 9 *The* σ*-/-calculus is weakly normalising.*

Proof: Suppose there is 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 $a \neq c[d/]_k$. Applying Lemma 8, we get $c \twoheadrightarrow a[t_1]_{j_1} \cdots [t_n]_{j_n} \in ENF$. Note that $a, t_1, \cdots t_n$ are all smaller than c and hence admit a normal form. Now replacing each of them by its normal form in $a[t_1]_{j_1} \cdots [t_n]_{j_n}$ we have a normal form for c which is a contradiction. □

Therefore we can finally apply Lemma 6 to conclude:

Theorem 16 *The* σ*-/-calculus is strongly normalising on* $\Lambda\omega^t$.

Now, using the isomorphism, since, as we mentioned before, both rule schemes in $*\sigma$ translate into the single σ-/ rule scheme, we have:

Theorem 17 *The* $*\sigma$*-calculus is strongly normalising.*

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

Definition 16 *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 18 (Bachmair-Dershowitz-85) *Let* R *commute over* S. *The combined system* $R \cup S$ *is SN iff* R *and* S *both are SN.*

Now, here is an example which shows that $*\sigma$ does not commute over $s + *\varphi$:

Let $k + i \leq j$, $h \leq j - i + 1$ and $h > k + 1$. Let us consider the following derivation:

$$(\varphi^i_k(a\,\sigma^h b))\,\sigma^j c \to_{*\varphi} \varphi^i_k((a\,\sigma^h b)\,\sigma^{j-i+1} c) \to_{\sigma-\sigma-tr} \varphi^i_k((a\,\sigma^{j-i+2} c)\,\sigma^h (b\,\sigma^{j-i-h+2} c))$$

But it is easy to see that $(\varphi_k^i(a\,\sigma^h b))\,\sigma^j c$ 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, whose reduction 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.

We have thus proved that no commutation is possible between $s + *\varphi$ and $*\sigma$ and therefore the Bachmair-Dershowitz's Theorem cannot be applied to get SN for s_e.

5 Typed calculi

We begin with a brief survey of the typed versions of λs and $\lambda\sigma$. From the point of view of syntax the only difference is that the abstractions are marked with types. We have thus $\lambda A.a$ where A is a simple type, that is a type obtained from a set of basic types using the only binary infix constructor of types $\to$. In the case of $\lambda\sigma$ we also have the conses marked with types: $a : A \cdot s$.

We recall that environments in de Bruijn setting are simply lists of types and in the case of $\lambda\sigma$, substitutions receive environments as types. We introduce the following notation concerning environments. If E is the environment $E_1, E_2, \ldots, E_n$, we shall use the notation $E_{\geq i}$ for the environment $E_i, E_{i+1}, \ldots, E_n$, analogously $E_{\leq i}$ stands for $E_1, \ldots, E_i$, etc.

The rewriting rules of the corresponding typed calculi are exactly the same (except that rules involving abstractions are now typed).

We concentrate now on the typing rules of these calculi. We begin by recalling the typing rules for the simply typed λ-calculus in de Bruijn notation. We call the typing system **L1**:

$$(\mathbf{L1} - var) \quad A, E \vdash \mathtt{1} : A \qquad\qquad (\mathbf{L1} - \lambda) \quad \frac{A, E \vdash b : B}{E \vdash \lambda A.b : A \to B}$$

$$(\mathbf{L1} - varn) \; \frac{E \vdash \mathtt{n} : B}{A, E \vdash \mathtt{n} + \mathtt{1} : B} \qquad\qquad (\mathbf{L1} - app) \; \frac{E \vdash b : A \to B \quad E \vdash a : A}{E \vdash b\,a : B}$$

We recall now the typing rules for λs and λs_e. The typing system **Ls1** is defined as follows:

The rules **Ls1**-*var*, **Ls1**-*varn*, **Ls1**-λ and **Ls1**-*app* are exactly the same as **L1**-*var*, **L1**-*varn*, **L1**-λ and **L1**-*app*, respectively. The new rules are:

$$(\mathbf{Ls1} - \sigma) \; \frac{E_{\geq i} \vdash b : B \quad E_{<i}, B, E_{\geq i} \vdash a : A}{E \vdash a\,\sigma^i b : A} \qquad\qquad (\mathbf{Ls1} - \varphi) \; \frac{E_{\leq k}, E_{\geq k+i} \vdash a : A}{E \vdash \varphi_k^i a : A}$$

We recall now the main results concerning typed λs and λs_e [KR95a]:

Theorem 19 *The following hold:*

1. **Subject Reduction of λs:** *If $E \vdash a : A$ and $a \rightarrow_{\lambda s} b$ then $E \vdash b : A$.*
2. **SN of λs:** *Every well typed term is SN in the simply typed λs-calculus.*
3. **Subject Reduction of λs_e:** *If $E \vdash a : A$ and $a \rightarrow_{\lambda s_e} b$ then $E \vdash b : A$.*

Finally, in order that the reader could compare we the typing system $\mathbf{L\sigma 1}$ of $\lambda\sigma$, we recall $\mathbf{L\sigma 1}$:

The rules $\mathbf{L\sigma 1}$-*var*, $\mathbf{L\sigma 1}$-λ and $\mathbf{L\sigma 1}$-*app* are exactly the same as $\mathbf{L1}$-*var*, $\mathbf{L1}$-λ and $\mathbf{L1}$-*app*, respectively. The new rules are:

$$(\mathbf{L\sigma 1} - \mathit{clos})\ \frac{E \vdash s \triangleright E' \quad E' \vdash a : A}{E \vdash a[s] : A} \qquad (\mathbf{L\sigma 1} - \mathit{cons})\ \frac{E \vdash a : A \quad E \vdash s \triangleright E'}{E \vdash a : A \cdot s \triangleright A, E'}$$

$$(\mathbf{L\sigma 1} - \mathit{id})\quad E \vdash id \triangleright E \qquad (\mathbf{L\sigma 1} - \mathit{comp})\ \frac{E \vdash s'' \triangleright E'' \quad E'' \vdash s' \triangleright E'}{E \vdash s' \circ s'' \triangleright E'}$$

$$(\mathbf{L\sigma 1} - \mathit{shift})\quad A, E \vdash \uparrow \triangleright E \qquad (\mathbf{L\sigma 1} - \mathit{Mtv})\quad E_X \vdash X : A_X$$

The last rule is added to type open terms and should be understood as follows: for every metavariable X, there exists an environment E_X and a type A_X such that the rule holds.

We introduce now the typing rules for $\lambda\omega$ and $\lambda\omega_e$. The typing system is called $\mathbf{L\omega 1}$. The rules $\mathbf{L\omega 1}$-*var*, $\mathbf{L\omega 1}$-*varn*, $\mathbf{L\omega 1}$-λ and $\mathbf{L\omega 1}$-*app* are exactly the same as $\mathbf{L1}$-*var*, $\mathbf{L1}$-*varn*, $\mathbf{L1}$-λ and $\mathbf{L1}$-*app*, respectively. The new rules are:

$$(\mathbf{L\omega 1} - \mathit{id})\quad E \vdash \uparrow^0 \triangleright E \qquad (\mathbf{L\omega 1} - \mathit{slash})\quad \frac{E \vdash a : A}{E \vdash a/ : A, E}$$

$$(\mathbf{L\omega 1} - \mathit{shift})\ \frac{E \vdash \uparrow^i \triangleright E'}{A, E \vdash \uparrow^{i+1} : E'} \qquad (\mathbf{L\omega 1} - \mathit{clos})\ \frac{E_{\geq j} \vdash s \triangleright E' \quad E_{<j}, E' \vdash a : A}{E \vdash a[s]_j : A}$$

$$(\mathbf{L\sigma 1} - \mathit{Mtv})\quad E_X \vdash X : A_X$$

We prove now that the isomorphism defined in Section 3 preserves typing. For the definition of T and S in the next theorem we refer to Section 3.

Theorem 20 *The following hold:*

1. *For $a \in \Lambda s$, if $E \vdash a : A$ then $E \vdash T(a) : A$.*
2. *For $a \in \Lambda s_{op}$, if $E \vdash a : A$ then $E \vdash T(a) : A$.*
3. *For $a \in \Lambda\omega^t$, if $E \vdash a : A$ then $E \vdash S(a) : A$.*
4. *For $a \in \Lambda\omega^t_{op}$, if $E \vdash a : A$ then $E \vdash S(a) : A$.*

Proof: The four items are proved by an easy induction on the inference of $E \vdash a : A$. $\square$

Now that we have established that the isomorphism preserves types we can transfer our results concerning typed λs and λs_e (cf. Theorem 19) to typed $\lambda\omega$ and $\lambda\omega_e$:

Theorem 21 *The following hold:*

1. **Subject Reduction of $\lambda\omega$:** *If $E \vdash a : A$ and $a \to_{\lambda\omega} b$ then $E \vdash b : A$.*
2. **SN of $\lambda\omega$:** *Every well typed term is SN in the simply typed $\lambda\omega$-calculus.*
3. **Subject Reduction of $\lambda\omega_e$:** *If $E \vdash a : A$ and $a \to_{\lambda\omega_e} b$ then $E \vdash b : A$.*

Finally, [KR97] shows the following (which implies weak normalisation of λs_e):

Theorem 22 (Weak Normalisation of $\lambda\omega_e$)
Every well typed term in the simply typed $\lambda\omega_e$-calculus is weakly normalising.

Conclusion

In this paper, we introduced two new calculi $\lambda\omega$ and $\lambda\omega_e$ that can bridge the two styles of calculi of explicit substitutions. Our motivation for doing so comes from the fact that the two different styles of substitutions provide complementary properties and that it remains an open problem whether a calculus of explicit substitution that satisfies the union of these properties exist. Another reason is that, although open problems concerning one of the styles (the $\lambda\sigma$-style) have been solved (some with a negative answer and others with a positive one), the second style (the λs-style) still has two puzzling open problems: the termination of the substitution calculus s_e and the preservation of termination of λs_e.

Although, the $\lambda\omega$ and $\lambda\omega_e$-calculi are calculi in the $\lambda\sigma$-style, their stratified substitutions are more λs-style than $\lambda\sigma$-style. The main new feature towards the $\lambda\sigma$-style is the introduction of $\uparrow^i$ and hence the availability of the updated term $\varphi^i_k(a)$ as $a[\uparrow^i]_k$. Hence the stratified substitutions play a double role: as *real substitutions* as in $a[b/]_k$ and as *updatings* as in $a[\uparrow^i]_k$.

We believe that this two-sorted presentation of λs may be useful to gain a new insight on the open problems for this calculus, mainly the strong normalisation of s_e. We have exploited the above mentioned double role of stratified substitutions to show that the subcalculus $*\sigma$ of s_e is strongly normalising; in fact, both rule schemes of $*\sigma$ translate into a single scheme of ω_e thanks to this duality.

Apart from their role as bridging calculi betwee the $\lambda\sigma$- and λs-styles of explicit substitutions, the $\lambda\omega$- and $\lambda\omega_e$-calculi are interesting on their own for the following reasons:

1. $\lambda\omega$ is confluent on closed terms and preserves strong normalisation,
2. the associated calculus of substitutions of $\lambda\omega$ is SN,
3. the simply typed version of $\lambda\omega$ is SN,

4. $\lambda\omega$ possesses an extension $\lambda\omega_e$ that is confluent on open terms, simulates β-reduction, and whose simply typed version is weakly normalising (on open term).

As far as we know, the $\lambda\omega$-calculus is the first calculus in the $\lambda\sigma$-style that has all those properties. However, the two open problems of the SN of the associated calculus of substitution of $\lambda\omega_e$ and of the preservation of strong normalisation of $\lambda\omega_e$ remain unsolved.

References

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

[AS86] H. Abelson and G. Sussman. *Structure and Interpretation of Computer Programs.* MIT Press, 1986.

[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.

[Blo95] R. Bloo. Preservation of Strong Normalisation for Explicit Substitution. Technical Report 95-08, Department of Mathematics and Computing Science, Eindhoven University of Technology, 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.

[Con86] R. Constable et al. *Implementing Mathematics with the NUPRL Development System.* Prentice-Hall, 1986.

[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.

[dB78] 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.

[DHK95] G. Dowek, T. Hardin, and C. Kirchner. Higher order unification via explicit substitutions (extended abstract). In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science,* San Diego, pages 366–374, 1995.

[FKP96] Ferreira, Kesner, and Puel. λ-calculi with explicit substitutions and composition which preserve β-strong normalisation. *ALP*, 1996.

[GL97] Jean Goubault-Larrecq. A proof of weak termination of the simply typed $\lambda\sigma$-calculus. Technical report, INRIA, January 1997.

[GM93] M.J.C. Gordon and T.F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, 1993.

[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.

[Klo92] J.-W. Klop. Term rewriting systems. *Handbook of Logic in Computer Science*, II, 1992.

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

[KN95] F. Kamareddine and R. P. Nederpelt. Generalising reduction in the λ-calculus. *Journal of Functional Programming*, 5(4):637–651, 1995.

[KR95a] F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *LNCS*, 982:45–62, 1995.

[KR95b] F. Kamareddine and A. Ríos. The λs-calculus: its typed and its extended versions. Technical report, Department of Computing Science, University of Glasgow, 1995.

[KR96a] 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*, 1996. To appear.

[KR96b] F. Kamareddine and A. Ríos. The confluence of the λs_e-calculus via a generalized interpretation method. Technical report, Glasgow University, 1996.

[KR97] F. Kamareddine and A. Rios. Weak normalisation of the simply typed λs_e-calculus. Technical report, Glasgow University, 1997. In preparation.

[KR96] F. Kamareddine and A. Ríos. Generalised β_e-reduction and explicit substitution. *PLILP*, '96.

[KRW97] F. Kamareddine, A. Ríos, and J.B. Wells. Calculi of generalised β_e-reduction and explicit substitution: Type free and untyped versions. *Journal of Functional and Programming Languages*, 1997.

[LM96] J. L. Lawall and H. Mairson. Optimality and inefficiency: What isn't a cost model of the lambda calculus? Proc. 1996 ACM SIGPLAN Int'l Conf. Functional Programming, pages 92–101, 1996.

[LRD96] Lescanne and Rouyer-Degli. Explicit substitutions with de bruijn's levels. Technical report, CNRS and INRIA, 1996.

[Mag95] Magnusson. *The implementation of ALF - a proof editor based on Martin Löf's Type Theory with explicit substitutions.* PhD thesis, Chalmers, '95.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate in Proceedings of TLCA'95. *LNCS*, 902, 1995.

[MH95] C. A. Muñoz Hurtado. Confluence and preservation of strong normalisation in an explicit substitutions calculus. Technical Report 2762, INRIA, Rocquencourt, December 1995.

[MH97] C. A. Muñoz Hurtado. Meta-theoretical properties of dependent types systems with explicit substitutions. Technical report, Personal communication, 1997.

[NGdV94] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath.* North-Holland, Amsterdam, 1994.

[Pau90] L. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, pages 361–386, 1990.

[Pey87] S.L. Peyton-Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

[Río93] A. Ríos. *Contribution à l'étude des λ-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.

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