Generalised β-reduction and explicit substitutions 8th international conference on Programming Languages: Implementations, Logics and Programs, PLILP96, LNCS 1140, pages 378-392 *

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. Extending the λ-calculus with either explicit substitution or generalised reduction has been the subject of extensive research recently which still has many open problems. Due to this reason, the properties of a calculus combining both generalised reduction and explicit substitutions have never been studied. This paper presents such a calculus λsg and shows that it is a desirable extension of the λ-calculus. In particular, we show that λsg preserves strong normalisation, is sound and it simulates classical β-reduction. Furthermore, we study the simply typed λ-calculus extended with both generalised reduction and explicit substitution and show that well-typed terms are strongly normalising and that other properties such as subtyping and subject reduction hold.

1 Introduction

1.1 The λ-calculus with generalised reduction

In $((\lambda_x.\lambda_y.N)P)Q$, the function starting with λ_x and the argument P result in the redex $(\lambda_x.\lambda_y.N)P$ which when contracted will turn the function starting with λ_y and Q into a redex. This fact has been exploited by many researchers and reduction has been extended so that the future redex based on the matching λ_y and Q is given the same priority as the other redex. Attempts at generalising reduction can be summarized by three axioms:

$$(\theta)\quad ((\lambda_x.N)P)Q \to (\lambda_x.NQ)P, \qquad (\gamma)\quad (\lambda_x.\lambda_y.N)P \to \lambda_y.(\lambda_x.N)P,$$
$$(\gamma_C)\quad ((\lambda_x.\lambda_y.N)P)Q \to (\lambda_y.(\lambda_x.N)P)Q.$$

These rules attempt to make more redexes visible. γ_C e.g., makes sure that λ_y and Q form a redex even before the redex based on λ_x and P is contracted. By compatibility, γ implies γ_C. Moreover, $((\lambda_x.\lambda_y.N)P)Q \to_\theta (\lambda_x.(\lambda_y.N)Q)P$ and hence both θ and γ_C put λ adjacently next to its matching argument. θ

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

moves the argument next to its matching λ whereas γ_C moves the λ next to its matching argument. θ can be equally applied to explicitly and implicitly typed systems. The transfer of γ or γ_C to explicitly typed systems is not straightforward however, since in these systems, the type of y may be affected by the reducible pair λ_x, P. E.g., it is fine to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_\theta (\lambda_{x:*}.(\lambda_{y:x}.y)u)z$ but not to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_{\gamma_C} (\lambda_{y:x}.(\lambda_{x:*}.y)z)u$. Hence, we study θ-like rules in this paper. Now, we discuss where generalised reduction has been used (cf. [24]).

[32] introduces the notion of a *premier redex* which is similar to the redex based on λ_y and Q above (which we call *generalised redex*). [33] uses θ and γ (and calls the combination σ) to show that the perpetual reduction strategy finds the longest reduction path when the term is Strongly Normalising (SN). [37] also introduces reductions similar to those of [33]. Furthermore, [22] uses θ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification. [35] uses a reduction which has some common themes with θ. [30] and [11] use θ whereas [25] uses γ to reduce the problem of β-strong normalisation to the problem of weak normalisation (WN) for related reductions. [23] uses θ and γ to reduce typability in the rank-2 restriction of the 2nd order λ-calculus to the problem of acyclic semi-unification. [27, 38, 36, 26] use related reductions to reduce SN to WN and [21] uses similar notions in SN proofs. [2] uses θ (called "let-C") as a part of an analysis of how to implement sharing in a real language interpreter in a way that directly corresponds to a formal calculus. [16] uses a more extended version of θ where Q and N are not only separated by the redex $(\lambda_x.N)P$ but by many redexes (ordinary and generalised). [16] shows that generalised reduction makes more redexes visible allowing flexibility in reducing a term. [6] shows that with generalised reduction one may indeed avoid size explosion without the cost of a longer reduction path and that λ-calculus can be elegantly extended with definitions which result in shorter type derivations. Generalised reduction is strongly normalising (cf. [6]) for all systems of the cube (cf. [3]) and preserves strong normalisation of classical reduction (cf. [13]).

1.2 The λ-calculus with explicit substitution

Functional programming and in particular partial evaluation may benefit from explicit substitution. For example, given $xx[x := y]$, we may not be interested in having yy as the result of $xx[x := y]$ but rather only $yx[x := y]$. In other words, we only substitute one occurrence of x by y and continue the substitution later. This issue of being able to follow substitution and decide how much to do and how much to postpone, has become a major one in functional language implementation (cf. [31]). Another wish is to execute substitutions only when necessary. For this purpose one may decide to postpone substitutions as long as possible ("lazy evaluations"). This can yield profits, since substitution is an inefficient, maybe even exploding, process by the many repetitions it causes. This is the ground for the so-called graph reduction (cf. [31]). Most theorem provers (Nuprl [7], Coq [12]) use explicit substitutions in their implementation in order to replace locally (rather than globally) some abbreviated term. This

avoids explosion when it is necessary that a variable be replaced by a huge term only in specific places so that a certain theorem can be proved.

Most literature on the λ-calculus considers substitution as an implicit operation: the computations to perform substitution are usually described with operators which do not belong to the language of the λ-calculus. The last fifteen years have seen an interest in formalising substitution explicitly; various calculi including new operators to denote substitution have been proposed. Amongst these calculi we mention $C\lambda\xi\phi$ (cf. [10]); the calculi of categorical combinators (cf. [8]); $\lambda\sigma$, $\lambda\sigma_{\Uparrow}$, $\lambda\sigma_{SP}$ (cf. [1, 9, 34]) referred to as the $\lambda\sigma$-family; $\lambda\upsilon$ (cf. [4]), a descendant of the $\lambda\sigma$-family; $\varphi\sigma BLT$ (cf. [15]), $\lambda\mathtt{exp}$ (cf. [5]), λs (cf. [17]), λs_e (cf. [19]) and $\lambda\zeta$ (cf. [29]). All these calculi (except $\lambda\mathtt{exp}$) are described in a de Bruijn setting where natural numbers play the role of the classical variables.

In [17], we extended the λ-calculus with explicit substitutions by turning de Bruijn's meta-operators into object-operators offering a style of explicit substitution that differs from that of $\lambda\sigma$. The resulting calculus λs remains as close as possible to the λ-calculus from an intuitive point of view. The main interest in introducing the λs-calculus (cf. [17]) was to provide a calculus of explicit substitutions which would both preserve strong normalisation and have a confluent extension on open terms (cf. [19]). There are calculi of explicit substitutions which are confluent on open terms: the $\lambda\sigma_{\Uparrow}$-calculus (cf. [9]), but the non-preservation of strong normalisation for $\lambda\sigma_{\Uparrow}$, for the rest of the $\lambda\sigma$-family and for the categorical combinators, has been proved (cf. [28]). There are also calculi which satisfy the preservation property: the $\lambda\upsilon$-calculus (cf. [4]), but this calculus is not confluent on open terms. Recently, the $\lambda\zeta$-calculus (cf. [29]) has been proposed as a calculus which preserves strong normalisation and is itself confluent on open terms. It 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. Unfortunately, $\lambda\zeta$ is not able to simulate one step β-reduction as shown in [29], it simulates only a "big step" β-reduction. This lack of the simulation property is an uncommon feature among calculi of explicit substitutions. On the other hand, λs has been extended to λs_e which is confluent on open terms (cf. [19]) and simulates one step β-reduction but the preservation of strong normalisation is still an open problem.

1.3 Combining generalised reduction and explicit substitution

All the research mentioned above is a living proof for the importance and usefulness of generalised reduction and explicit substitutions. Moreover, a system where reduction is generalised and substitution is explicit, gives a more flexible way of evaluating programs thanks to the advantages of step-wise substitution and the ability of reducing more redexes.

Before such a combination can be used as a powerful basis for programming, we need to check that this combination is sound and safe exactly like we checked that each of explicit substitutions and generalised reductions are sound and safe.

This paper shows that extending the λ-calculus with both concepts results in theories that are confluent, preserve termination, and simulate β-reduction.

Generalised reduction $g\beta$, has never been introduced in a de Bruijn setting. Explicit substitution, has almost always been presented in a de Bruijn setting. For this reason, we combine $g\beta$-reduction and explicit substitution in a de Bruijn setting giving the first calculus of generalised reduction à la de Bruijn. As we need to describe generalised redexes in an elegant way, we use a notation suitable for this purpose *the item notation* (cf. [14]).

In Section 2 we introduce the calculus of generalised reduction, the λg-calculus, in item notation with de Bruijn indices and prove its confluence.

In Section 3 we extend the λs-calculus with $\rightarrow_{g\beta}$ into the λsg-calculus. We show that λsg is sound with respect to λg, simulates $g\beta$ and is confluent.

In Section 4 we prove that the λsg-calculus preserves λs-strong normalisation and conclude that a is λ-SN $\Leftrightarrow$ a is λs-SN $\Leftrightarrow$ a is λg-SN $\Leftrightarrow$ a is λsg-SN.

In Section 5 the simply typed versions of the λs- and λsg-calculi are presented and subject reduction, subtyping, and SN of well typed terms are proved.

This article is an abridged version of [20], where more detailed proofs are given.

2 The λg-calculus

We assume familiarity with de Bruijn notation. Since generalised β-reduction is easily described in item notation, we adopt the item syntax (cf. [16, 14] for the advantages of item notation) and write $a\,b$ as $(b\,\delta)a$ and $\lambda.a$ as $(\lambda)a$.

Definition 1 *The* set of terms Λ*, is defined as follows:* $\Lambda ::= \mathbb{N} | (\Lambda\,\delta)\Lambda | (\lambda)\Lambda$
We let $a, b, \ldots$ *range over* Λ *and* $m, n, \ldots$ *over* $\mathbb{N}$ *(positive natural numbers).* $a = b$ *means that* a *and* b *are syntactically identical. We write* $a \lhd b$ *when* a *is a subterm of* b*. We assume the usual definition of compatibility.*

$(\lambda x\lambda y.zxy)(\lambda x.yx) \rightarrow_\beta \lambda u.z(\lambda x.yx)u$ translates to $(\lambda\lambda\mathtt{521})(\lambda\mathtt{31}) \rightarrow_\beta \lambda\mathtt{4}(\lambda\mathtt{41})\mathtt{1}$. Note that we did not simply replace $\mathtt{2}$ in $\lambda\mathtt{521}$ by $\lambda\mathtt{31}$. Instead, we decreased $\mathtt{5}$ as one λ disappeared, and incremented the free variables of $\lambda\mathtt{31}$ as they occur within the scope of one more λ. For incrementing the free variables we need updating functions U_k^i, where k tests for free variables and $i-1$ is the value by which a variable, if free, must be incremented:

Definition 2 $U_k^i : \Lambda \rightarrow \Lambda$ *for* $k \geq 0$ *and* $i \geq 1$ *are defined inductively:*

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

Now we define meta-substitution. The last equality substitutes the intended variable (when $n = j$) by the updated term. If n is not the intended variable, it is decreased by 1 if it is free (case $n > j$) as one λ has disappeared and if it is bound (case $n < j$) it remains unaltered.

Definition 3 *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:*

$$((a_1\delta)a_2)\{\!\{\mathtt{j} \leftarrow b\}\!\} = ((a_1\{\!\{\mathtt{j} \leftarrow b\}\!\})\delta)(a_2\{\!\{\mathtt{j} \leftarrow b\}\!\})$$
$$((\lambda)c)\{\!\{\mathtt{j} \leftarrow b\}\!\} = (\lambda)(c\{\!\{\mathtt{j+1} \leftarrow b\}\!\})$$
$$\mathtt{n}\{\!\{\mathtt{j} \leftarrow b\}\!\} = \begin{cases} \mathtt{n}-1 & \text{if } n > j \\ U_0^j(b) & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases}$$

The following gives the properties of meta-substitution and updating (cf. [17]):

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* $k+i \leq n$: $U_k^i(a)\{\!\{\mathtt{n} \leftarrow b\}\!\} = U_k^i(a\{\!\{\mathtt{n}-\mathtt{i}+1 \leftarrow b\}\!\})$.
4. *for* $i \leq n$: $a\{\!\{\mathtt{i} \leftarrow b\}\!\}\{\!\{\mathtt{n} \leftarrow c\}\!\} = a\{\!\{\mathtt{n}+1 \leftarrow c\}\!\}\{\!\{\mathtt{i} \leftarrow b\{\!\{\mathtt{n}-\mathtt{i}+1 \leftarrow c\}\!\}\}\!\}$.
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* $n \leq k+1$: $U_k^i(a\{\!\{\mathtt{n} \leftarrow b\}\!\}) = U_{k+1}^i(a)\{\!\{\mathtt{n} \leftarrow U_{k-n+1}^i(b)\}\!\}$.

In order to introduce generalised β-reduction we need some defintions (cf. [14]).

Definition 4 Items, segments *and* well-balanced segments (w.b.) *are defined respectively by:* $\mathcal{I} ::= (\Lambda\,\delta) \mid (\lambda) \quad \mathcal{S} ::= \phi \mid \mathcal{I}\,\mathcal{S} \quad \mathcal{W} ::= \phi \mid (\Lambda\delta)\mathcal{W}(\lambda) \mid \mathcal{W}\,\mathcal{W}$ *where* ϕ *is the empty segment. Hence, a segment is a sequence of items.* $(a\,\delta)$ *and* (λ) *are called* δ- *and* λ-item *respectively. We let* I, J, ... *range over* $\mathcal{I}$; S, S', ... *over* $\mathcal{S}$ *and* W, U, ... *over* $\mathcal{W}$. *For a segment* S, $\lg S$, *is given by:* $\lg\phi = 0$, $\lg(I\,S) = 1 + \lg S$. *The number of main* λ*-items in* S, $N(S)$, *is given by:* $N(\phi) = 0$, $N((a\,\delta)S) = N(S)$ *and* $N((\lambda)S) = 1 + N(S)$.

Definition 5 λ-calculus *is the reduction system* $(\Lambda, \to_\beta)$, *where* $\to_\beta$ *is the least compatible reduction on* Λ *generated by the* β*-rule:* $(a\delta)(\lambda)b \to a\{\!\{\mathtt{1} \leftarrow b\}\!\}$.

Definition 6 Generalised β, $\to_{g\beta}$, *is the least compatible reduction on* Λ *generated by the* $g\beta$*-rule:* $(a\delta)W\,(\lambda)b \to W\,(b\{\!\{\mathtt{1} \leftarrow U_0^{N(W)+1}(a)\}\!\})$ *where* W *is w.b. The* λg-calculus *is the reduction system* $(\Lambda, \to_{g\beta})$.

Remark 1 *The* β*-rule is an instance of the* $g\beta$*-rule.*

Proof: Take $W = \phi$ and check $U_0^1(a) = a$. □

Now, let us briefly explain the relation between $\to_{g\beta}$ and $\to_\theta$, $\to_\gamma$, $\to_{\gamma C}$ given in the introduction. As $\to_\gamma$ implies $\to_{\gamma C}$, we ignore the latter. It would be helpful if we write $\to_\theta$ and $\to_\gamma$ in item notation:

$(Q\delta)(P\delta)(\lambda_x)N \to_\theta (P\delta)(\lambda_x)(Q\delta)N \qquad (P\delta)(\lambda_x)(\lambda_y)N \to_\gamma (\lambda_y)(P\delta)(\lambda_x)N$

Note how in $\to_\theta$, the start of a redex $(P\delta)(\lambda_x)$ is moved (or reshuffled) giving $(Q\delta)$ the chance to find its matching (λ) in N. In $\to_\gamma$ the same happens but now it is (λ_y) which is given the chance to look for its matching $(-\delta)$. Only once reshuffling has taken place, can the newly found redex be contracted. $\to_{g\beta}$ on the other hand avoids reshuffling and contracts the redex as soon as it sees the matching of δ and λ.

We define segments' updating and meta-substitution and prove some properties.

Definition 7 *Let $S \in \mathcal{S}$, $a, b \in \Lambda$, $k \geq 0$ and $n, i \geq 1$.*
We define $U_k^i(S)$ and $S\{\!\{\mathtt{n} \leftarrow a\}\!\}$ by:

$$U_k^i(\phi) = \phi \qquad \phi\{\!\{\mathtt{n} \leftarrow a\}\!\} = \phi$$
$$U_k^i((b\,\delta)S) = (U_k^i(b)\,\delta)U_k^i(S) \qquad ((b\,\delta)S)\{\!\{\mathtt{n} \leftarrow a\}\!\} = (b\{\!\{\mathtt{n} \leftarrow a\}\!\}\,\delta)(S\{\!\{\mathtt{n} \leftarrow a\}\!\})$$
$$U_k^i((\lambda)S) = (\lambda)(U_{k+1}^i(S)) \qquad ((\lambda)S)\{\!\{\mathtt{n} \leftarrow a\}\!\} = (\lambda)(S\{\!\{\mathtt{n+1} \leftarrow a\}\!\})$$

Lemma 2 *Let S, T be segments and $a, b \in \Lambda$. The following hold:*
1. *$U_k^i(S\,T) = U_k^i(S)U_{k+N(S)}^i(T)$ and $U_k^i(S\,a) = U_k^i(S)U_{k+N(S)}^i(a)$*
2. *$\lg(S) = \lg(U_k^i(S))$, $N(S) = N(U_k^i(S))$ and if S w.b. then $U_k^i(S)$ w.b.*
3. *$(S\,\xi)\{\!\{\mathtt{n} \leftarrow a\}\!\} = S\{\!\{\mathtt{n} \leftarrow a\}\!\}\,\xi\{\!\{\mathtt{n} + N(S) \leftarrow a\}\!\}$ for ξ a segment or a term*
4. *If $r \in \{\lg, N\}$ then $r(S) = r(S\{\!\{\mathtt{n} \leftarrow a\}\!\})$. If S w.b. then $S\{\!\{\mathtt{n} \leftarrow a\}\!\}$ w.b.*

Proof: All by induction on S. For 2. and 4. use 1. and 3. respectively. □

Lemma 3 *Let $a, b \in \Lambda$. If $a \twoheadrightarrow_{g\beta} b$ then $a =_\beta b$.*

Proof: First prove by induction on a that $a \to_{g\beta} b$ implies $a =_\beta b$. To show the case $(c\delta)W(\lambda)d \to_{g\beta} W(d\{\!\{\mathtt{1} \leftarrow U_0^{N(W)+1}(c)\}\!\})$ use induction on $\lg W$. □

Theorem 1 (Confluence of λg) *The λg-calculus is confluent.*

Proof: Use Lemma 3 and Remark 1 (cf. [16]). □

Next, we ensure the good passage of $g\beta$-reduction through $\{\!\{ \leftarrow \}\!\}$ and U_k^i:

Lemma 4 *Let a, b, c, $d \in \Lambda$. The following hold:*
1. *If $c \to_{g\beta} d$ then $U_k^i(c) \to_{g\beta} U_k^i(d)$.*
2. *If $c \to_{g\beta} d$ then $a\{\!\{\mathtt{n} \leftarrow c\}\!\} \twoheadrightarrow_{g\beta} a\{\!\{\mathtt{n} \leftarrow d\}\!\}$.*
3. *If $a \to_{g\beta} b$ then $a\{\!\{\mathtt{n} \leftarrow c\}\!\} \to_{g\beta} b\{\!\{\mathtt{n} \leftarrow c\}\!\}$.*

Proof: 1. By induction on c. 2. and 3. By induction on a. □

3 The λs- and λsg-calculi

The idea is to handle explicitly the meta-operators of definitions 2 and 3. Hence, the syntax of the λs-calculus is obtained by adding two families of operators:
1. Explicit substitution operators $\{\sigma^j\}_{j\geq 1}$ where $(b\,\sigma^j)a$ stands for a where all free occurrences of the variable representing j are to be substituted by b.
2. Updating operators $\{\varphi_k^i\}_{k\geq 0\ i\geq 1}$ needed for working with de Bruijn indices.

Definition 8 *The* set of terms, *noted Λs*, of the λs-calculus *is given as follows:*

$\Lambda s ::= \mathbb{N} \mid (\Lambda s\,\delta)\Lambda s \mid (\lambda)\Lambda s \mid (\Lambda s\,\sigma^j)\Lambda s \mid (\varphi_k^i)\Lambda s$ *where* $j, i \geq 1$, $k \geq 0$.

We let a, b, c range over Λs. A term $(a\,\sigma^j)b$ 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. $\delta\lambda$-*segments *are those whose main items are either δ- or λ-items, i.e. $\mathcal{DL} ::= \phi \mid (\Lambda s\,\delta)\mathcal{DL} \mid (\lambda)\mathcal{DL}$. Compatibility is extended by adding:*
$(a\,\sigma^j)c \to (b\,\sigma^j)c$, $(c\,\sigma^j)a \to (c\,\sigma^j)b$ and $(\varphi_k^i)a \to (\varphi_k^i)b$ whenever $a \to b$.

Definition 9 Items, segments *and* well-balanced segments *for Λs are defined as follows:* $\mathcal{I}s ::= (\Lambda s\,\delta) \mid (\lambda) \mid (\Lambda s\,\sigma^j) \mid (\varphi_k^i) \qquad \mathcal{S}s ::= \phi \mid \mathcal{I}s\,\mathcal{S}s$

$$\mathcal{W}s ::= \phi \mid (\Lambda s\,\delta)\mathcal{W}s(\lambda) \mid \mathcal{W}s\,\mathcal{W}s$$

*We let I, J, ... range over $\mathcal{I}s$; S, S', ... over $\mathcal{S}s$ and W, U, ... over $\mathcal{W}s$. We call $(a\,\sigma^j)$ and (φ_k^i), σ- and φ-*item *respectively.* $\lg(S)$ *is trivially extended to $S \in \mathcal{S}s$ and $N(S)$ is extended by: $N((a\,\sigma^j)S) = N(S)$ and $N((\varphi_k^i)S) = N(S)$.*

As the λs-calculus updates and substitutes explicitly, we include a set of rules which are the equations in definitions 2 and 3 oriented from left to right.

Definition 10 *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	$(b\,\delta)(\lambda)a \longrightarrow (b\,\sigma^1)a$
σ-λ-transition	$(b\,\sigma^j)(\lambda)a \longrightarrow (\lambda)(b\,\sigma^{j+1})a$
σ-app-transition	$(b\,\sigma^j)(a_1\delta)a_2 \longrightarrow ((b\,\sigma^j)a_1\delta)\,(b\,\sigma^j)a_2$
σ-destruction	$(b\,\sigma^j)\mathtt{n} \longrightarrow \begin{cases} \mathtt{n-1} & \text{if } n > j \\ (\varphi_0^j)b & \text{if } n = j \\ \mathtt{n} & \text{if } n < j \end{cases}$
φ-λ-transition	$(\varphi_k^i)(\lambda)a \longrightarrow (\lambda)(\varphi_{k+1}^i)a$
φ-app-transition	$(\varphi_k^i)(a_1\delta)a_2 \longrightarrow ((\varphi_k^i)a_1\delta)(\varphi_k^i)a_2$
φ-destruction	$(\varphi_k^i)\mathtt{n} \longrightarrow \begin{cases} \mathtt{n+i-1} & \text{if } n > k \\ \mathtt{n} & \text{if } n \le k \end{cases}$

We use λs to denote this set of rules. The calculus of substitutions associated with the λs-calculus *is the reduction system generated by the set of rules $s = \lambda s - \{\sigma\text{-generation}\}$ and we call it the* s-calculus.
The λsg-calculus *is the calculus whose set of rules is $\lambda sg = \lambda s + \{g\sigma\text{-generation}\}$:*

$$g\sigma\text{-generation} \qquad (b\,\delta)W(\lambda)a \longrightarrow W((\varphi_0^{N(W)+1})b\,\sigma^1)a \qquad W \text{ w.b, } W \neq \phi$$

Note that in the λsg-calculus we do not merge *σ-generation* and *$g\sigma$-generation* in a new *$g\sigma$-generation* which admits $W = \phi$ because in that case we would obtain, when $W = \phi$, the rule $(b\delta)(\lambda)a \to ((\varphi_0^1 b)\sigma^1)a$, and this is not a generalisation of the original *σ-generation* of the λs-calculus.

σ-generation starts β-reduction by generating a substitution operator (σ^1). *σ-app* and *σ-λ* allow this operator to travel throughout the term until its arrival to the variables. If a variable should be affected by the substitution, *σ-destruction* (case $j = n$) carries out the substitution by the updated term, thus introducing the updating operators. Finally the *φ-rules* compute the updating. We state now the following theorem of the λs-calculus (cf. [19]).

Theorem 2 *The s-calculus is strongly normalising and confluent on Λs, hence s-normal forms are unique. The set of s-normal forms is exactly Λ. If $s(a)$*

denotes the s-normal form of a, then for $a, b \in \Lambda s$: $s((a\,\delta)b) = (s(a)\,\delta)s(b)$, $s((\lambda)a) = (\lambda)(s(a))$, $s((\varphi^i_k)a) = U^i_k(s(a))$ and $s((b\,\sigma^j)a) = s(a)\{\!\{\mathtt{j} \leftarrow s(b)\}\!\}$.

Lemma 5 *Let $a, b \in \Lambda s$, if $a \to_{(g)\sigma-gen} b$ then $s(a) \twoheadrightarrow_{(g)\beta} s(b)$.*

Proof: Induction on a using Lemma 4 and Theorem. 2. For the case with g, note that if W is w.b then $s(W\,a) = s(W)s(a)$, where the s-nf of a $\delta\lambda$-segment is given by: $s(\phi) = \phi$, $s((a\,\delta)S) = (s(a)\,\delta)s(S)$ and $s((\lambda)S) = (\lambda)s(S)$. □

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

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

Hence, the λsg-calculus is correct w.r.t. the λg-calculus, i.e. λsg-derivations of pure terms ending with pure terms can also be derived in the λg-calculus.

Moreover, the λsg-calculus is powerful enough to simulate $g\beta$-reduction.

Lemma 6 (Simulation of $\to_{g\beta}$) *Let $a, b \in \Lambda$, if $a \to_{g\beta} b$ then $a \twoheadrightarrow_{\lambda sg} b$.*

Proof: Induction on a using Lemma 4. □

Theorem 3 (Confluence of λsg) *The λsg-calculus is confluent on Λs.*

Proof: Use the interpretation method (cf. [9]), Corollary 1, confluence of the λg-calculus and Lemma 6. □

4 The λsg-calculus preserves λs-SN

The technique used here to prove preservation of strong normalisation (PSN) is the same used in [4] to prove PSN for $\lambda\upsilon$ and in [17] to prove PSN for λs.

Notation 1 *We write $a \in \lambda$-SN resp. $a \in \lambda r$-SN when a is strongly normalising in the λ-calculus resp. in the λr-calculus for $r \in \{g, sg, s\}$. We write $a \underset{p}{\to} b$ to denote that p is the occurrence of the redex which is contracted. Therefore $a \underset{\epsilon}{\to} b$ means that the reduction takes place at the root. If no specification is made the reduction must be understood as a λsg-reduction. We denote by $\prec$ the prefix order between occurrences of a term. Hence if p, q are occurrences of the term a such that $p \prec q$, and we write a_p (resp. a_q) for the subterm of a at occurrence p (resp. q), then a_q is a subterm of a_p. E.g., if $a = 2\sigma^3((\lambda 1)4)$, we have $a_1 = 2$, $a_2 = (\lambda 1)4$, $a_{21} = \lambda 1$, $a_{211} = 1$, $a_{22} = 4$. Since $2 \prec 21$, a_{21} is a subterm of a_2.*

The following three lemmas assert that all the σ's in the last term of a derivation beginning with a λ-term must have been created at some previous step by a (generalised) σ-generation and trace the history of these closures. The first lemma deals with one-step derivation where the redex is at the root; the second generalises the first; the third treats arbitrary derivations.

Lemma 7 *If $a \xrightarrow{\epsilon} C[(e\,\sigma^i)d]$ then one of the following must hold:*

1. $a = (e\,\delta)(\lambda)d$, $C = \Box$ *and* $i = 1$.
2. $a = (e'\,\delta)W(\lambda)d$, $W \neq \phi$, $C = W\Box$, $e = (\varphi_0^{N(W)+1})e'$ *and* $i = 1$.
3. $a = C'[(e\,\sigma^j)d']$ *for some context* C', *some term* d' *and some natural* j.

Proof: Since the reduction is at the root, check for every rule $a \to a'$ in λsg that if $(e\,\sigma^i)d$ occurs in a' then either 1. or 2. or 3. follows. □

Lemma 8 *If $a \to C[(e\,\sigma^i)d]$ then one of the following must hold:*

1. $a = C[(e\,\delta)(\lambda)d]$ *and* $i = 1$.
2. $a = C'[(e'\,\delta)W(\lambda)d]$, $C = C'[W\Box]$, $e = (\varphi_0^{N(W)+1})e'$ *and* $i = 1$.
3. $a = C'[(e'\,\sigma^i)d']$ *where* $e' = e$ *or* $e' \to e$.

Proof: Induction on a, using lemma 7 for the reductions at the root. □

Lemma 9 *If $a_1 \to \ldots \to a_{n+1} = C[(e\,\sigma^i)d]$, there exist $e', d' \in \Lambda s$ with $e' \twoheadrightarrow e$ and, either $a_1 = C'[(e'\,\sigma^j)d']$ or for some $k \leq n$ and W w.b., $a_k = C'[(e'\,\delta)W(\lambda)d']$ and $a_{k+1} = C'[W((\varphi_0^{N(W)+1})e'\,\sigma^1)d']$ or, if $W = \phi$, $a_{k+1} = C'[(e'\,\sigma^1)d']$.*

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

We define now internal and external reductions. An internal reduction takes place at the left of a σ^i operator. An external reduction is a non-internal one. Our definition is inductive rather than starting from the notion of internal and external position as in [4].

Definition 11 *The reduction $\xrightarrow{\mathtt{int}}_{\lambda sg}$ is defined by the following rules:*

$$\frac{a \longrightarrow_{\lambda sg} b}{(a\,\sigma^i)c \xrightarrow{\mathtt{int}}_{\lambda sg} (b\,\sigma^i)c} \qquad \frac{a \xrightarrow{\mathtt{int}}_{\lambda sg} b}{(a\,\delta)c \xrightarrow{\mathtt{int}}_{\lambda sg} (b\,\delta)c} \qquad \frac{a \xrightarrow{\mathtt{int}}_{\lambda sg} b}{(c\,\delta)a \xrightarrow{\mathtt{int}}_{\lambda sg} (c\,\delta)b}$$

$$\frac{a \xrightarrow{\mathtt{int}}_{\lambda sg} b}{(\lambda)a \xrightarrow{\mathtt{int}}_{\lambda sg} (\lambda)b} \qquad \frac{a \xrightarrow{\mathtt{int}}_{\lambda sg} b}{(c\,\sigma^i)a \xrightarrow{\mathtt{int}}_{\lambda sg} (c\,\sigma^i)b} \qquad \frac{a \xrightarrow{\mathtt{int}}_{\lambda sg} b}{(\varphi_k^i)a \xrightarrow{\mathtt{int}}_{\lambda sg} (\varphi_k^i)b}$$

Definition 12 *The reduction $\xrightarrow{\mathtt{ext}}_s$ is defined by induction. The axioms are the rules of the s-calculus and the inference rules are the following:*

$$\frac{a \xrightarrow{\mathtt{ext}}_s b}{(a\,\delta)c \xrightarrow{\mathtt{ext}}_s (b\,\delta)c} \qquad \frac{a \xrightarrow{\mathtt{ext}}_s b}{(c\,\delta)a \xrightarrow{\mathtt{ext}}_s (c\,\delta)b} \qquad \frac{a \xrightarrow{\mathtt{ext}}_s b}{(\lambda)a \xrightarrow{\mathtt{ext}}_s (\lambda)b}$$

$$\frac{a \xrightarrow{\mathtt{ext}}_s b}{(c\,\sigma^i)a \xrightarrow{\mathtt{ext}}_s (c\,\sigma^i)b} \qquad \frac{a \xrightarrow{\mathtt{ext}}_s b}{(\varphi_k^i)a \xrightarrow{\mathtt{ext}}_s (\varphi_k^i)b}$$

An external (generalised) σ-generation is defined by the rule $(g)\sigma$-generation and the five inference rules above where $\xrightarrow{\mathtt{ext}}_s$ is replaced by $\xrightarrow{\mathtt{ext}}_{(g)\sigma-gen}$.

Remark 2 *By inspection of the inference rules, $a \xrightarrow{\mathtt{int}}_{\lambda sg} \mathtt{n}$ is impossible and:*

- *If $a \xrightarrow{\mathtt{int}}_{\lambda sg} (\lambda)b$ then $a = (\lambda)c$ and $c \xrightarrow{\mathtt{int}}_{\lambda sg} b$.*
- *If $a \xrightarrow{\mathtt{int}}_{\lambda sg} (c\,\delta)b$ then $a = (e\,\delta)d$ and*
  $((d \xrightarrow{\mathtt{int}}_{\lambda sg} b$ and $e = c)$ or $(e \xrightarrow{\mathtt{int}}_{\lambda sg} c$ and $d = b))$.

Note that $\dfrac{a \xrightarrow{\mathtt{ext}}_s b}{(a\,\sigma^i)c \xrightarrow{\mathtt{ext}}_s (b\,\sigma^i)c}$ and $\dfrac{a \xrightarrow{\mathtt{ext}}_{(g)\sigma-gen} b}{(a\,\sigma^i)c \xrightarrow{\mathtt{ext}}_{(g)\sigma-gen} (b\,\sigma^i)c}$ are excluded from the definitions of external s-reduction and external (generalised) σ-generation, respectively. Thus external reductions will not occur at the left of a σ^i operator and we write $\overset{+}{\to}_\beta$ instead of $\twoheadrightarrow_\beta$ in the following (compare with Lemma 5):

Proposition 1 *Let $a, b \in \Lambda s$, if $a \xrightarrow{\mathtt{ext}}_{(g)\sigma-gen} b$ then $s(a) \overset{+}{\to}_{(g)\beta} s(b)$.*

Proof: Induction on a (as in Lemma 5). Note that when $a = c\,\sigma^i d$, the reduction cannot take place within d because it is external, and this is the only case that forced us to consider the reflexive-transitive closure because of lemma 4.2. □

The following is needed in Lemma 11 and hence in the Preservation Theorem.

Lemma 10 (Commutation Lemma) *Let $a, b \in \Lambda s$ such that $s(a) \in \lambda$-SN and $s(a) = s(b)$. If $a \xrightarrow{\mathtt{int}}_{\lambda sg} . \xrightarrow{\mathtt{ext}}_s b$ then $a \xrightarrow{\mathtt{ext}}_s{}^{+} . \xrightarrow{\mathtt{int}}{\!\!\!\!\twoheadrightarrow}_{\lambda sg} b$.*

Proof: By a careful induction on a analysing the positions of the redexes. The proof is exactly the same as that of the Commutation Lemma in [17] □

Lemma 11 *Let $a \in \lambda g\text{-}SN \cap \Lambda$ and $a \to_{\lambda sg} b_1 \to_{\lambda sg} \cdots \to_{\lambda sg} b_n \to_{\lambda sg} \cdots$, an infinite derivation. There exists N such that for every $i \geq N$, the reductions $b_i \to_{\lambda sg} b_{i+1}$ are internal.*

Proof: Analogous to the proof of the corresponding lemma in [17]. □

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

Definition 13 *An infinite λsg-derivation $a_1 \to \cdots \to a_n \to \cdots$ is* minimal *if for every step $a_i \underset{p}{\to} a_{i+1}$, any derivation starting with $a_i \underset{q}{\to} a'_{i+1}$, if $p \prec q$, is finite.*

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

Definition 14 *The syntax of* skeletons *and the* skeleton of a term *are as follows:*

Skeletons $K ::= \mathbb{N} \mid (K\,\delta)K \mid (\lambda)K \mid ([.]\,\sigma^j)K \mid (\varphi^i_k)K$

$$Sk(\mathtt{n}) = \mathtt{n} \quad Sk((a\,\delta)b) = (Sk(a)\,\delta)Sk(b) \quad Sk((b\,\sigma^i)a) = ([.]\,\sigma^i)Sk(a)$$
$$Sk((\lambda)a) = (\lambda)Sk(a) \qquad Sk((\varphi^i_k)a) = (\varphi^i_k)Sk(a)$$

Remark 3 *Let $a, b \in \Lambda s$. If $a \xrightarrow{\mathtt{int}}_{\lambda sg} b$ then $Sk(a) = Sk(b)$.*

Theorem 4 (Preservation of λs-SN) *For every $a \in \Lambda$, if a is strongly normalising in the λs-calculus then a is strongly normalising in the λsg-calculus.*

Proof: Assume $a \in \lambda s$-SN, $a \notin \lambda sg$-SN and take a minimal infinite λsg-derivation $\mathcal{D} : a \to a_1 \to \cdots \to a_n \to \cdots$. Lemma 11 gives N such that for $i \geq N$, $a_i \to a_{i+1}$ is internal. By Remark 3, $Sk(a_i) = Sk(a_{i+1})$ for $i \geq N$. As there are only a finite number of closures in $Sk(a_N)$ and as the reductions within these closures are independent, an infinite subderivation $\mathcal{D}'$ of $\mathcal{D}$ must take place within

the same and unique closure in $Sk(a_N)$ and $\mathcal{D}'$ is also minimal. Let C be the context such that $a_N = C[(d\,\sigma^i)c]$ and $(d\,\sigma^i)c$ is the closure where $\mathcal{D}'$ takes place:
$\mathcal{D}' \,:\, a_N = C[(d\,\sigma^i)c] \xrightarrow{\mathtt{int}}_{\lambda sg} C[(d_1\,\sigma^i)c] \xrightarrow{\mathtt{int}}_{\lambda sg} \cdots \xrightarrow{\mathtt{int}}_{\lambda sg} C[(d_n\,\sigma^i)c] \xrightarrow{\mathtt{int}}_{\lambda sg} \cdots$
Since a is a pure term, Lemma 9 ensures the existence of $I \leq N$ such that either

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

$a_I = C'[(d'\,\delta)W\,(\lambda)c'] \to a_{I+1} = C'[W\,((\varphi_0^{N(W)+1})d'\sigma^1)c']$ and $d' \twoheadrightarrow d$.

Let us consider in the first and second cases respectively, the infinite derivations:
$\mathcal{D}'' : a \twoheadrightarrow a_I \twoheadrightarrow C'[(d\delta)(\lambda)c'] \to C'[(d_1\delta)(\lambda)c'] \to \cdots \to C'[(d_n\delta)(\lambda)c'] \cdots$
$\mathcal{D}''' : a \twoheadrightarrow a_I \twoheadrightarrow C'[(d\delta)W\,(\lambda)c'] \to C'[(d_1\delta)W\,(\lambda)c'] \to \cdots \to C'[(d_n\delta)W\,(\lambda)c'] \cdots$
In $\mathcal{D}''$ and $\mathcal{D}'''$, the redex in a_I is within d' which is a proper subterm of $(d'\,\delta)(\lambda)c'$ (of $(d'\,\delta)W\,(\lambda)c'$ in the second case), whereas in $\mathcal{D}$ the redex in a_I is $(d'\,\delta)(\lambda)c'$ (in the second case $(d'\,\delta)W\,(\lambda)c'$) and this contradicts the minimality of $\mathcal{D}$. □

Corollary 3 *For every $a \in \Lambda$, the following equivalences hold:*
$a \in \lambda g$-SN iff $a \in \lambda sg$-SN iff $a \in \lambda$-SN iff $a \in \lambda s$-SN

Proof: By Remark 1 and Theorem 4, $a \in \lambda s$-SN iff $a \in \lambda sg$-SN. Due to [13], $a \in \lambda$-SN iff $a \in \lambda g$-SN. Due to [17], $a \in \lambda$-SN iff $a \in \lambda s$-SN. □

5 The typed λs- and λsg-calculi

We prove λsg-SN of well typed terms using the technique developped in [18] to prove λs-SN and suggested to us by P.-A. Melliès as a successful technique to prove $\lambda\upsilon$-SN (personal communication). We recall the syntax and typing rules for the simply typed λ-calculus in de Bruijn notation. The types are generated from a set of basic types T with the binary type operator $\to$. Environments are lists of types. Typed terms differ from the untyped ones only in the abstractions which are now marked with the type of the abstracted variable.

Definition 15 *The syntax for the simply typed λ-terms is given as follows:*

Types	$\mathcal{T} ::= T \mid \mathcal{T} \to \mathcal{T}$
Environments	$\mathcal{E} ::= nil \mid \mathcal{T}, \mathcal{E}$
Terms	$\Lambda_t ::= \mathtt{n} \mid (\Lambda_t\,\delta)\Lambda_t \mid (\mathcal{T}\,\lambda)\Lambda_t$

We let A, B, ... range over $\mathcal{T}$; E, E_1, ... over $\mathcal{E}$ and a, b, ... over Λ_t.
The typing rules are given by the typing system **L1** *as follows:*

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

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

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.

Definition 16 *The syntax for the* simply typed λs-terms *is given as follows:*

$$\Lambda s_t ::= \mathbb{N} \mid (\Lambda s_t\, \delta)\Lambda s_t \mid (\mathcal{T}\, \lambda)\Lambda s_t \mid (\Lambda s_t\, \sigma^i)\Lambda s_t \mid (\varphi_k^i)\Lambda s_t \quad i \geq 1\,,\ k \geq 0.$$

Types and environments are as above. The typing rules of the system **Ls1** *are: 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:*

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

The simply typed λs- *and* λsg-calculi *are defined by the same rules of the untyped versions, except that abstractions in the typed versions are marked with types.*

Definition 17 $a \in \Lambda s_t$ *is a* well typed term *if for some environment* E *and type* A, $E \vdash_{\mathbf{Ls1}} a : A$. *We note* Λs_{wt} *the set of well typed terms.*

The aim of this section is to prove that every well typed λs-term a is λsg-SN (and hence λs-SN). To do so, we show $\Lambda s_{wt} \subseteq \Xi \subseteq \lambda sg$-SN, where

$\Xi = \{a \in \Lambda s_t :$ for every subterm b of a, $s(b) \in \lambda g\text{-}SN\}$.

To prove $\Lambda s_{wt} \subseteq \Xi$ (Proposition 2) we need to establish some useful results such as subject reduction, soundness of typing and typing of subterms:

Lemma 12 *Let* S *be a segment,* A, B *types and* a, b, $c \in \Lambda s_t$. *We have:*

1. $E \vdash S((\varphi_0^i)a\,\delta)(c\,\delta)(B\,\lambda)b : A$ *iff* $E \vdash S(c\,\delta)(B\,\lambda)((\varphi_0^{i+1})a\,\delta)b : A$
2. $E \vdash S((\varphi_0^i)(\varphi_0^j)a\,\delta)b : A$ *iff* $E \vdash S((\varphi_0^{i+j-1})a\,\delta)b : A$
3. $E \vdash S(a\,\delta)(B\,\lambda)b : A$ *iff* $E \vdash S(a\,\sigma^1)b : A$

Proof: All by induction on S. □

Lemma 13 (Shuffle Lemma) *Let* S *be an arbitrary segment,* W *a w.b. segment and* a, $b \in \Lambda s_t$, *then* $E \vdash S(a\,\delta)W\,b : A$ *iff* $E \vdash SW((\varphi_0^{N(W)+1})a\,\delta)\,b : A$.

Proof: By induction on W using Lemma 12. If $W = \phi$, it is immediate since $E' \vdash d : D$ iff $E' \vdash (\varphi_0^1)d : D$. Let us assume $W = (c\,\delta)U(B\,\lambda)V$, with U, V w.b..

$E \vdash S(a\,\delta)(c\,\delta)U(B\,\lambda)V\,b : A$ iff (IH)
$E \vdash S(a\,\delta)U((\varphi_0^{N(U)+1})c\,\delta)(B\,\lambda)V\,b : A$ iff (IH)
$E \vdash SU((\varphi_0^{N(U)+1})a\,\delta)((\varphi_0^{N(U)+1})c\,\delta)(B\,\lambda)V\,b : A$ iff (Lemma 12.1)
$E \vdash SU((\varphi_0^{N(U)+1})c\,\delta)(B\,\lambda)((\varphi_0^{N(U)+2})a\,\delta)V\,b : A$ iff (IH, twice)
$E \vdash S(c\,\delta)U(B\,\lambda)V((\varphi_0^{N(V)+1})(\varphi_0^{N(U)+2})a\,\delta)b : A$ iff (Lemma 12.2)
$E \vdash S(c\,\delta)U(B\,\lambda)V((\varphi_0^{N(V)+N(U)+2})a\,\delta)b : A$ □

Lemma 14 (Subject reduction) *If* $E \vdash_{\mathbf{Ls1}} a : A$, $a \to_{\lambda sg} b$ *then* $E \vdash_{\mathbf{Ls1}} b : A$.

Proof: Induction on a. If the reduction is not at the root, use IH. Else, show for every rule $a \to b$ that $E \vdash_{\mathbf{Ls1}} a : A$ implies $E \vdash_{\mathbf{Ls1}} b : A$. Case σ-*gen*, use Lemma 12.3. Case $g\sigma$-*gen*: If $E \vdash (a\,\delta)W(B\,\lambda)b : A$ then, by Lemma 13, we have $E \vdash W((\varphi_0^{N(W)+1})a\,\delta)(B\,\lambda)b : A$ and, by Lemma 12.3, we conclude $E \vdash W((\varphi_0^{N(W)+1})a\,\sigma^1)b : A$. □

Corollary 4 *Let* $E \vdash_{\mathbf{Ls1}} a : A$, *if* $a \twoheadrightarrow_{\lambda sg} b$ *then* $E \vdash_{\mathbf{Ls1}} b : A$.

Lemma 15 (Typing of subterms) *If* $a \in \Lambda s_{wt}$ *and* $b \lhd a$ *then* $b \in \Lambda s_{wt}$.

Proof: By induction on a. If b is not an immediate subterm of a, use IH. Else, the last rule used to type a has a premise in which b is typed. □

Lemma 16 (Soundness of typing) *If* $a \in \Lambda_t$, $E \vdash_{\mathbf{Ls1}} a : A$ *then* $E \vdash_{\mathbf{L1}} a : A$.

Proof: Easy induction on a. □

Proposition 2 $\Lambda s_{wt} \subseteq \Xi$.

Proof: Let $a \in \Lambda s_{wt}$ and b a subterm of a. By Lemma 15, $b \in \Lambda s_{wt}$ and by Corollary 4, $s(b) \in \Lambda s_{wt}$. Since $s(b) \in \Lambda$ (Thm. 2), Lemma 16 gives $s(b)$ is **L1**-typable. But classical typable λ-terms are strongly normalising in the λ-calculus. Hence, $s(b) \in \lambda$-SN and, by Corollary 3, $s(b) \in \lambda g$-SN. Therefore $a \in \Xi$. □

We prove now $\Xi \subseteq \lambda sg$-SN.

Lemma 17 *Let* $a \in \Xi$ *and* $a \to_{\lambda s} b_1 \to_{\lambda s} \cdots \to_{\lambda s} b_n \to_{\lambda s} \cdots$, *an infinite* λs-*derivation. There exists* N *such that for* $i \geq N$ *all the reductions* $b_i \to_{\lambda s} b_{i+1}$ *are internal.*

Proof: The proof is almost the same as the proof of lemma 11. □

Proposition 3 *For every* $a \in \Lambda s_t$, *if* $a \in \Xi$ *then* $a \in \lambda sg$-SN.

Proof: Assume $a' \in \Xi$ and $a' \notin \lambda sg$-SN, then there exists a term a of minimal size such that $a \in \Xi$ and $a \notin \lambda sg$-SN. Let $\mathcal{D} : a \to a_1 \to \cdots \to a_n \to \cdots$ be a minimal infinite λsg-derivation and follow the proof of Theorem 4 to obtain:

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

Now three possibilities arise from lemma 9. Two of them have been considered in the proof of Theorem 4 and contradicted the minimality of $\mathcal{D}$. Take the third one: $a = C'[(d'\,\sigma^i)c']$ where $d' \twoheadrightarrow d$. Now we have $d' \twoheadrightarrow d \to d_1 \to \cdots \to d_n \to \cdots$. As d' is a subterm of a, $d' \in \Xi$, contradicting that a has minimal size. □

Therefore we conclude, using Propositions 2 and 3 and Corollary 3:

Theorem 5 *Well typed* λs-*term are strongly normalising in the* λsg-*calculus.*

Corollary 5 *Well typed* λs-*term are strongly normalising in the* λs-*calculus.*

6 Conclusion

In this paper, we started from the fact that generalised reduction and explicit substitution play a vital role in useful extensions of the λ-calculus but have never been combined together. We commented that the combination might indeed join both benefits and hence a λ-calculus extended with both needs to be studied.

We presented such a calculus and showed that it possesses the important properties that have been the center of research for each concept on its own. In particular, we showed that the resulting calculus is confluent, sound and simulates β-reduction. We showed moreover that it preserves strong normalisation of the unextended λ-calculus and of the λ-calculus extended with each of the two concepts independently. We studied furthermore, the simply typed version of our calculus of explicit substitution and generalised reduction and showed that it has again the important properties such as subject reduction, soundness of subtyping, typing of subterms and strong normalisation of well typed terms.

Now that a calculus combining both concepts have been shown to be theoretically correct, it would be interesting to extend our calculus λsg to one that is confluent on open terms as is the tradition with calculi of explicit substitution. It would be also interesting to study the polymorphically (rather than the simply) typed version of λsg. These are issues we are investigating at the moment. We are also investigating the correspondence of our calculus to methods that implement sharing and parallelism to test if the analysis of sharing given in [2] can be recast in an elegant fashion in our calculus.

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.M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call by need lambda calculus. *Conf. Rec. 22nd Ann. ACM Symp. Princ. Program. Lang. ACM*, 1995.
3. H. Barendregt. λ-calculi with types. *Handbook of Logic in Computer Science*, II, 1992.
4. Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Personal communication*, 1995.
5. R. Bloo. Preservation of Strong Normalisation for Explicit Substitution. Technical Report 95-08, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1995.
6. R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt Cube with Definitions and Generalised Reduction. *Information and Computation*, 126 (2):123–143, 1996.
7. R. Constable et al. *Implementing Mathematics with the NUPRL Development System*. Prentice-Hall, 1986.
8. P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming*. Pitman, 1986. Revised edition : Birkhäuser (1993).
9. 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.
10. 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.
11. P. de Groote. The conservation theorem revisited. *Int'l Conf. Typed Lambda Calculi and Applications LNCS*, 664, 1993.

12. G. Dowek et al. The coq proof assistant version 5.6, users guide. Technical Report 134, INRIA, 1991.
13. F. Kamareddine. A reduction relation for which postponement of k-contractions, conservation and preservation of strong normalisation hold. Technical report, Glasgow University, 1996.
14. F. Kamareddine and R. Nederpelt. A useful λ-notation. *Theoretical Computer Science*, 155:85–109, 1996.
15. F. Kamareddine and R. P. Nederpelt. On stepwise explicit substitution. *International Journal of Foundations of Computer Science*, 4(3):197–240, 1993.
16. F. Kamareddine and R. P. Nederpelt. Generalising reduction in the λ-calculus. *Journal of Functional Programming*, 5(4):637–651, 1995.
17. F. Kamareddine and A. Ríos. A λ-calculus à la de Bruijn with explicit substitutions. Proceedings of PLILP'95. *LNCS*, 982:45–62, 1995.
18. F. Kamareddine and A. Ríos. The λs-calculus: its typed and its extended versions. Technical report, Department of Computing Science, University of Glasgow, 1995.
19. 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.
20. F. Kamareddine and A. Ríos. Generalised β-reduction and explicit substitutions. Technical Report TR-1996-21, Department of Computing Science, University of Glasgow, 1996.
21. M. Karr. Delayability in proofs of strong normalizability in the typed λ-calculus. *Mathematical Foundations of Computer Software, LNCS*, 185, 1985.
22. A.J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *ACM*, 41(2):368–398, 1994.
23. A.J. Kfoury and J.B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second order λ-calculus. *Proc. 1994 ACM Conf. LISP Funct. Program.*, 1994.
24. A.J. Kfoury and J.B. Wells. Addendum to new notions of reduction and non-semantic proofs of β-strong normalisation in typed λ-calculi. Technical report, Boston University, 1995.
25. A.J. Kfoury and J.B. Wells. New notions of reductions and non-semantic proofs of β-strong normalisation in typed λ-calculi. *LICS*, 1995.
26. Z. Khasidashvili. The longest perpetual reductions in orthogonal expression reduction systems. *Proc. of the* 3^{rd} *International Conference on Logical Foundations of Computer Science, Logic at St Petersburg*, 813, 1994.
27. J. W. Klop. Combinatory Reduction Systems. *Mathematical Center Tracts*, 27, 1980.
28. P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate in Proceedings of TLCA'95. *LNCS*, 902, 1995.
29. C. A. Muñoz Hurtado. Confluence and preservation of strong normalisation in an explicit substitutions calculus. Technical Report 2762, INRIA, Rocquencourt, December 1995.
30. R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer. *Selected papers on Automath.* North-Holland, Amsterdam, 1994.
31. S.L. Peyton-Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.
32. L. Regnier. *Lambda calcul et réseaux.* PhD thesis, Paris 7, 1992.
33. L. Regnier. Une équivalence sur les lambda termes. *Theoretical Computer Science*, 126:281–292, 1994.

34. A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites.* PhD thesis, Université de Paris 7, 1993.
35. A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Proc. 1992 ACM Conf. LISP Funct. Program.*, pages 288–298, 1992.
36. M. Sϕrensen. Strong normalisation from weak normalisation in typed λ-calculi. *Submitted.*
37. D. Vidal. *Nouvelles notions de réduction en lambda calcul.* PhD thesis, Université de Nancy 1, 1989.
38. H. Xi. On weak and strong normalisations. Technical Report 96-187, Carnegie Mellon University, 1996.

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