Generalised $\beta$-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 $\lambda$-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 $\lambda_{sg}$ and shows that it is a desirable extension of the $\lambda$-calculus. In particular, we show that $\lambda_{sg}$ preserves strong normalisation, is sound and it simulates classical $\beta$-reduction. Furthermore, we study the simply typed $\lambda$-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 $\lambda$-calculus with generalised reduction

In $((\lambda_x.\lambda_y.N)P)Q$, the function starting with $\lambda_x$ and the argument $P$ result in the redex $(\lambda_x.\lambda_y.N)P$ which when contracted will turn the function starting with $\lambda_y$ 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 $\lambda_y$ and $Q$ is given the same priority as the other redex. Attempts at generalising reduction can be summarized by three axioms:

$((\lambda_x.N)P)Q \rightarrow (\lambda_x.NQ)P$,  
$(\lambda_y.\lambda_y.N)P \rightarrow \lambda_y.((\lambda_x.N)P)$,  
$((\lambda_x.\lambda_y.N)P)Q \rightarrow (\lambda_y.((\lambda_x.N)P)Q)$.

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

* This work was carried out under EPSRC grant GR/K25014.
moves the argument next to its matching $\lambda$ whereas $\gamma_C$ moves the $\lambda$ next to its matching argument. $\theta$ can be equally applied to explicitly and implicitly typed systems. The transfer of $\gamma$ or $\gamma_C$ to explicitly typed systems is not straightforward however, since in these systems, the type of $y$ may be affected by the reducible pair $\lambda\omega\eta$. E.g., it is fine to write $((\lambda\omega\eta\cdot\lambda\omega\eta\cdot y)z)u \rightarrow_{\theta} (\lambda\omega\eta\cdot (\lambda\omega\eta\cdot y)u)z$ but not to write $((\lambda\omega\eta\cdot\lambda\omega\eta\cdot y)z)u \rightarrow_{\gamma_C} (\lambda\omega\eta\cdot (\lambda\omega\eta\cdot y)u)z$. Hence, we study $\theta$-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 $\lambda\eta$ and $Q$ above (which we call generalised redex). [33] uses $\theta$ and $\gamma$ (and calls the combination $\sigma$) 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 $\theta$ (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 $\theta$, [30] and [11] use $\theta$ whereas [25] uses $\gamma$ to reduce the problem of $\beta$-strong normalisation to the problem of weak normalisation (WN) for related reductions. [23] uses $\theta$ and $\gamma$ to reduce typability in the rank-2 restriction of the 2nd order $\lambda$-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 $\theta$ (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 $\theta$ where $Q$ and $N$ are not only separated by the redex $(\lambda\omega\eta\cdot N)P$ 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 $\lambda$-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 $\lambda$-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 $\lambda$-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 $\lambda$-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 $\mathcal{C}\mathcal{L}\mathcal{C}$ (cf. [10]); the calculi of categorical combinators (cf. [8]); $\lambda\sigma$, $\lambda\sigma_+$, $\lambda\sigma_{SP}$ (cf. [1, 9, 34]) referred to as the $\lambda\sigma$-family; $\lambda\nu$ (cf. [4]), a descendant of the $\lambda\sigma$-family; $\varphi\sigma(BLT)$ (cf. [15]), $\lambda\exp$ (cf. [5]), $\lambda s$ (cf. [17]), $\lambda s_e$ (cf. [19]) and $\lambda\zeta$ (cf. [29]). All these calculi (except $\lambda\exp$) are described in a de Bruijn setting where natural numbers play the role of the classical variables.

In [17], we extended the $\lambda$-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 $\lambda s$ remains as close as possible to the $\lambda$-calculus from an intuitive point of view. The main interest in introducing the $\lambda s$-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 $\lambda\sigma_\alpha$-calculus (cf. [9]), but the non-preservation of strong normalisation for $\lambda\sigma_\alpha$, 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 $\lambda\nu$-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 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 $\lambda\nu$ on open terms. Unfortunately, $\lambda\zeta$ is not able to simulate one step $\beta$-reduction as shown in [29], it simulates only a "big step" $\beta$-reduction. This lack of the simulation property is an uncommon feature among calculi of explicit substitutions. On the other hand, $\lambda s$ has been extended to $\lambda s_e$ which is confluent on open terms (cf. [19]) and simulates one step $\beta$-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 $\lambda$-calculus with both concepts results in theories that are confluent, preserve termination, and simulate $\beta$-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 $\lambda g$-calculus, in item notation with de Bruijn indices and prove its confluence.

In Section 3 we extend the $\lambda s$-calculus with $\rightarrow g\beta$ into the $\lambda g s$-calculus. We show that $\lambda g s$ is sound with respect to $\lambda g$, simulates $g\beta$ and is confluent.

In Section 4 we prove that the $\lambda g s$-calculus preserves $\lambda s$-strong normalisation and conclude that $a$ is $\lambda s$-SN $\iff$ $a$ is $\lambda g s$-SN.

In Section 5 the simply typed versions of the $\lambda s$- and $\lambda g s$-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 $\lambda g$-calculus

We assume familiarity with de Bruijn notation. Since generalised $\beta$-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 $A$, is defined as follows: $A ::= \mathbb{N} | (A\delta)A | (\lambda)A$

We let $a, b, \ldots$ range over $A$ and $m, n, \ldots$ over $\mathbb{N}$ (positive natural numbers). $a = b$ means that $a$ and $b$ are syntactically identical. We write $a a b$ when $a$ is a subterm of $b$. We assume the usual definition of compatibility.

$(\lambda x y . x y)(\lambda x . y x) \rightarrow g\beta \lambda u . z (\lambda x . y x) u$ translates to $(\lambda s 521)(\lambda s 31) \rightarrow g\beta \mathcal{M}(\lambda s 41)1$.

Note that we did not simply replace 2 in 521 by 31. Instead, we decreased 5 as one $\lambda$ disappeared, and incremented the free variables of 31 as they occur within the scope of one more $\lambda$. For incrementing the free variables we need updating functions $U^i_k$, where $k$ tests for free variables and $i = 1$ is the value by which a variable, if free, must be incremented:

**Definition 2** $U^i_k : A \rightarrow A$ for $k \geq 0$ and $i \geq 1$ are defined inductively:

\[
U^i_k((a \delta)b) = (U^i_k(a) \delta)U^i_k(b) \\
U^i_k((\lambda)a) = (\lambda)(U^i_{k+1}(a))
\]

$U^i_k(n) = \begin{cases} 
  n + i - 1 & \text{if } n > k \\
  n & \text{if } n \leq k
\end{cases}$

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 $\lambda$ 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 $b \in A$ in a term $a \in A$, denoted $a[j \leftarrow b]$, is defined inductively on $a$ as follows:

$((a_1 \delta) a_2)[j \leftarrow b] = ((a_1[j \leftarrow b]) \delta)(a_2[j \leftarrow b])$

$n[j \leftarrow b] = \begin{cases} n-1 & \text{if } n > j \\ U_j^n(b) & \text{if } n = j \\ n & \text{if } n < j \end{cases}$

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

Lemma 1 Let $a, b, c \in A$. We have:
1. for $k < n < k + i : U_k^{i-1}(a) = U_k^i(a) n \leftarrow b$.
2. for $l < k < l + j : U_k^l(U_l^j(a)) = U_k^{l+j-1}(a)$.
3. for $k + i \leq n : U_k^i(a) n \leftarrow b = U_k^n(a) n \leftarrow b$.
4. for $i \leq n : a[i \leftarrow b] n \leftarrow c = a n + 1 \leftarrow c i \leftarrow b n - 1 \leftarrow c$.
5. for $l + j < k + 1 : U_k^l(U_l^j(a)) = U_k^l(U_{l+1-j}^j(a))$.
6. for $n \leq k + 1 : U_k^i(a) n \leftarrow b = U_{k+1}^i(a) n \leftarrow U_{k-n+1}^i(b)$.

In order to introduce generalised $\beta$-reduction we need some definitions (cf. [14]):

Definition 4 Items, segments and well-balanced segments (w.b.) are defined respectively by $I := (A \delta) | (\lambda) S := \phi | IS = (A \delta)(\lambda) | WW$ where $\phi$ is the empty segment. Hence, a segment is a sequence of items. $(A \delta)$ and $(\lambda)$ are called $\delta$- and $\lambda$-item respectively. We let $I, J, \ldots$ range over $I$; $S, S', \ldots$ over $S$ and $W, U, \ldots$ over $W$. For a segment $S$, $\lg S$, is given by $\lg S = 1 + \lg S$. The number of main $\lambda$-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 $\lambda$-calculus is the reduction system $(A, \rightarrow_\beta)$, where $\rightarrow_\beta$ is the least compatible reduction on $A$ generated by the $\beta$-rule: $(a \delta)(\lambda) b \rightarrow a[1 \leftarrow b]$.

Definition 6 Generalised $\beta$, $\rightarrow_{g\beta}$, is the least compatible reduction on $A$ generated by the $g\beta$-rule: $(a \delta) W(\lambda) b \rightarrow W(b[1 \leftarrow U_0^n(W)+1(a)])$ where $W$ is w.b. The $\lambda g\beta$-calculus is the reduction system $(A, \rightarrow_{g\beta})$.

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

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

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

$(Q \delta)(P \delta)(\lambda_x) N \rightarrow (P \delta)(\lambda_x)(Q \delta) N \rightarrow (P \delta)(\lambda_x)(\lambda_y) N \rightarrow (\lambda_y)(P \delta)(\lambda_x) N$

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

We define segments’ updating and meta-substitution and prove some properties.
Definition 7 Let $S \in S, a, b \in A, k \geq 0$ and $n, i \geq 1$. We define $U^i_k(S)$ and $S[n \leftarrow a]$ by:

\[
U^i_k(\phi) = \phi \quad \phi \{n \leftarrow a\} = \phi
\]

\[
U^i_k((b \delta)S) = (U^i_k(b) \delta)U^i_k(S) \quad ((b \delta)S)[n \leftarrow a] = (b \{n \leftarrow a\} \delta)(S\{n \leftarrow a\})
\]

\[
U^i_k((\lambda)S) = (\lambda)(U^i_{k+1}(S)) \quad ((\lambda)S)[n \leftarrow a] = (\lambda)(S[n+1 \leftarrow a])
\]

Lemma 2 Let $S, T$ be segments and $a, b \in A$. The following hold:

1. $U^i_k(ST) = U^i_k(S)U^i_k(T)$ and $U^i_k(Sa) = U^i_k(S)U^i_{k+1}(a)$
2. $\lg(S) = \lg(U^i_k(S))$, $N(S) = N(U^i_k(S))$ and if $S$ w.b. then $U^i_k(S)$ w.b.
3. $(S \xi)[n \leftarrow a] = S[n \leftarrow a] \xi[n + N(S) \leftarrow a]$ for $\xi$ a segment or a term
4. If $r \in \{\lg, N\}$ then $r(S) = r(S[n \leftarrow a])$. If $S$ w.b. then $S[n \leftarrow a]$ w.b.

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

Lemma 3 Let $a, b \in A$. If $a \rightarrow_{g\beta} b$ then $a =_{\beta} b$.

Proof: First prove by induction on $a$ that $a \rightarrow_{g\beta} b$ implies $a =_{\beta} b$. To show the case $c\delta W(\lambda)d \rightarrow_{g\beta} W(d1 \leftarrow U^i_0(W+1)(c))$ use induction on $\lg W$. □

Theorem 1 (Confluence of $\lambda g$) The $\lambda g$-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 $U^i_k$.

Lemma 4 Let $a, b, c, d \in A$. The following hold:

1. If $c \rightarrow_{g\beta} d$ then $U^i_k(c) \rightarrow_{g\beta} U^i_k(d)$.
2. If $c \rightarrow_{g\beta} d$ then $a[n \leftarrow c] \rightarrow_{g\beta} a[n \leftarrow d]$
3. If $a \rightarrow_{g\beta} b$ then $a[n \leftarrow c] \rightarrow_{g\beta} b[n \leftarrow c]$


3 The $\lambda$s- and $\lambda sg$-calculi

The idea is to handle explicitly the meta-operators of definitions 2 and 3. Hence, the syntax of the $\lambda$s-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 $\{\phi_k\}_{k \geq 0, i \geq 1}$ needed for working with de Bruijn indices.

Definition 8 The set of terms, noted $As$, of the $\lambda$s-calculus is given as follows:

\[
As := N \mid (As \delta)As \mid (\lambda)As \mid (As \sigma_j)As \mid (\phi_k)As \quad \text{where} \quad j, i \geq 1, \quad k \geq 0.
\]

We let $a, b, c$ range over $As$. A term $(a \sigma_j)b$ is called a closure. Furthermore, a term containing neither $\sigma$’s nor $\phi$’s is called a pure term. $\delta$ denotes the set of pure terms, $\delta$-segments are those whose main items are either $\delta$- or $\lambda$-items, i.e. $DL ::= \phi \mid (As \delta)DL \mid (\lambda)DL$. Compatibility is extended by adding:

$(a \sigma_j)c \rightarrow (b \sigma_j)c$, $(c \sigma_j)a \rightarrow (c \sigma_j)b$ and $(\phi_k)a \rightarrow (\phi_k)b$ whenever $a \rightarrow b$. 

Theorem 2 The \( s \)-calculus is strongly normalising and confluent on \( \Lambda s \), hence \( s \)-normal forms are unique. The set of \( s \)-normal forms is exactly \( \Lambda \). If \( s(a) \)
denotes the s-normal form of a, then for $a, b \in \Lambda$: $s((a \delta) b) = (s(a) \delta)s(b)$, $s((\lambda a) = (\lambda)(s(a))$, $s((\varphi^a_b) a) = U^a_b(s(a))$ and $s((b \sigma^a) a) = s(a)\{b \leftarrow s(b)\}$.

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

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 $a, b \in \Lambda$, if $a \rightarrow_{\lambda sg} b$ then $s(a) \rightarrow_{g\beta}s(b)$.

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

Hence, the $\lambda sg$-calculus is correct w.r.t. the $\lambda g$-calculus, i.e. $\lambda sg$-derivations of pure terms ending with pure terms can also be derived in the $\lambda g$-calculus.

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

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


Theorem 3 (Confluence of $\lambda sg$) The $\lambda sg$-calculus is confluent on $\Lambda$.

Proof: Use the interpretation method (cf. [9]), Corollary 1, confluence of the $\lambda g$-calculus and Lemma 6.

4 The $\lambda sg$-calculus preserves $\lambda s$-SN

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

Notation 1 We write $a \in \lambda$-SN resp. $a \in \lambda r$-SN when $a$ is strongly normalising in the $\lambda$-calculus resp. in the $\lambda r$-calculus for $r \in \{g, sg, s\}$. We write $a \rightarrow b$ to denote that $p$ is the occurrence of the redex which is contracted. Therefore $a \rightarrow b$ means that the reduction takes place at the root. If no specification is made the reduction must be understood as a $\lambda sg$-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 $a = 2\sigma^3((\lambda l)A)$, we have $a_1 = 2$, $a_2 = (\lambda l)A$, $a_{21} = \lambda l$, $a_{211} = 1$, $a_{22} = 4$. Since $2 \prec 21$, $a_{21}$ is a subterm of $a_{22}$.

The following three lemmas assert that all the $\sigma$'s in the last term of a derivation beginning with a $\lambda$-term must have been created at some previous step by a (generalised) $\sigma$-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 \( a \to C[(e \sigma^i)y] \) then one of the following must hold:
1. \( a = (e \delta)(\lambda y), C = \Box \) and \( i = 1 \).
2. \( a = (e \delta)W(\lambda)y, W \neq \phi, C = W \Box, e = (\varphi_0^{N(W)+1})e' \) and \( i = 1 \).
3. \( a = C'[e(e \sigma^j)y'] \) 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 \( \lambda sg \) 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. \( a = C'[e(e \delta)W(\lambda)d], C = C'[W \Box], e = (\varphi_0^{N(W)+1})e' \) and \( i = 1 \).
3. \( a = C'[e(e \sigma^j)d'] \) where \( e = e' \) or \( e' \to e \).

Proof: Induction on \( a \), using lemma 7 for the reductions at the root.

Lemma 9 If \( a_1 \to \ldots \to a_{n+1} = C[(e \sigma^j)d] \), there exist \( e',d \in A_s \) with \( e' \not \to e \) and, either \( a_1 = C'[e(e \sigma^j)d] \) or for some \( k < n \) and \( W \) w.b., \( a_k = C'[e(e \sigma^j)W(\lambda)d'] \) and \( a_{k+1} = C'[W((\varphi_0^{N(W)+1})\sigma^j)d'] \) or, if \( W = \phi \), \( a_{k+1} = C'[e(e \sigma^j)d'] \).

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 \( \sigma^i \) operator. An external reduction is a non-internal one. Our definition is inductive rather than starting from the notion of internal atom and external position as in [4].

Definition 11 The reduction \( \int_{\lambda sg} \) is defined by the following rules:

\[
\begin{align*}
\lambda a & \int_{\lambda sg} \lambda b, \\
(a \sigma) & \int_{\lambda sg} (b \sigma)c \\
(a \delta) & \int_{\lambda sg} (b \delta)c \\
(c \delta) & \int_{\lambda sg} (c \delta)b \\
(c \sigma) & \int_{\lambda sg} (c \sigma)b \\
(\varphi_0^j) & \int_{\lambda sg} (\varphi_0^j)b
\end{align*}
\]

Definition 12 The reduction \( \ext_s \) is defined by induction. The axioms are the rules of the s-calculus and the inference rules are the following:

\[
\begin{align*}
(a \delta) & \ext_s (b \delta)c \\
(c \sigma) & \ext_s (c \sigma)b \\
(\varphi_0^j) & \ext_s (\varphi_0^j)b
\end{align*}
\]

An external (generalised) \( \sigma \)-generation is defined by the rule \( (g)\sigma \)-generation and the five inference rules above where \( \ext_s \) is replaced by \( \ext_{\langle g \rangle \sigma-gen} \).

Remark 2 By inspection of the inference rules, \( \int_{\lambda sg} \) is impossible and:
- If \( a \int_{\lambda sg} \lambda b \) then \( a = (\lambda)c \) and \( c \int_{\lambda sg} b \).
- If \( a \int_{\lambda sg} (c \delta)b \) then \( a = (e \delta)d \) and
  \((d \int_{\lambda sg} b \text{ and } e = c) \text{ or } (e \int_{\lambda sg} c \text{ and } d = b)) \).
Note that $\frac{a \xrightarrow{\text{ext}} b}{\langle a \sigma^i \rangle c \xrightarrow{\text{ext}} (b \sigma^i)c}$ and $\frac{a \xrightarrow{\text{ext}} b}{\langle a \sigma^i \rangle c \xrightarrow{\text{ext}} (b \sigma^i)c}$ are excluded from the definitions of external $\sigma$-reduction and external (generalised) $\sigma$-generation, respectively. Thus external reductions will not occur at the left of a $\sigma^i$ operator and we write $\xrightarrow{\text{+}}$ instead of $\xrightarrow{\beta}$ in the following (compare with Lemma 5):

**Proposition 1** Let $a, b \in As$, if $a \xrightarrow{\langle g \rangle \sigma - \text{gen}} b$ then $s(a) \xrightarrow{\langle g \rangle \beta} s(b)$.

**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 $a, b \in As$ such that $s(a) \in \Lambda$ and $s(a) = s(b)$. If $a \xrightarrow{\text{int}} b$ then $a \xrightarrow{\text{ext}} b$.

**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 $a \in \lambda g$-SN $\cap \Lambda$ and $a \xrightarrow{\lambda g} b_1 \xrightarrow{\lambda g} \cdots \xrightarrow{\lambda g} b_n \xrightarrow{\lambda g} \cdots$, an infinite derivation. There exists $N$ such that for every $i \geq N$, the reductions $b_i \xrightarrow{\lambda g} b_{i+1}$ 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 $\lambda g$-derivation $a_1 \rightarrow \cdots \rightarrow a_n \rightarrow \cdots$ is minimal if for every step $a_i \rightarrow a_{i+1}$, any derivation starting with $a_i \rightarrow a'_i$, if $p < q$, 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**

$K := \mathbb{N} \mid (K \delta)K \mid (\lambda)K \mid (\phi^i)K \mid (\phi^i_k)K$

$Sk(n) = n$ \hspace{1cm} $Sk((a \delta)b) = (Sk(a) \delta)Sk(b)$

$Sk((\lambda)a) = (\lambda)Sk(a)$ \hspace{1cm} $Sk((\phi^i_k)a) = (\phi^i_k)Sk(a)$

**Remark 3** Let $a, b \in As$. If $a \xrightarrow{\text{int}} b$ then $Sk(a) = Sk(b)$.

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

**Proof:** Assume $a \in \lambda g$-SN, $a \not\in \lambda g$-SN and take a minimal infinite $\lambda g$-derivation $D : a \rightarrow a_1 \rightarrow \cdots \rightarrow a_n \rightarrow \cdots$. Lemma 11 gives $N$ such that for $i \geq N$, $a_i \rightarrow a_{i+1}$ 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 $D'$ of $D$ must take place within
the same and unique closure in $Sk(a_N)$ and $D'$ 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 $D'$ takes place.

$D' : a_N = C[(d \sigma^i)c] \overset{\text{inj}_{\lambda \sigma}}{\rightarrow} C[(d_1 \sigma^i)c] \overset{\text{inj}_{\lambda \sigma}}{\rightarrow} \cdots \overset{\text{inj}_{\lambda \sigma}}{\rightarrow} C[(d_n \sigma^i)c] \overset{\text{inj}_{\lambda \sigma}}{\cdots}$

Since $a$ is a pure term, Lemma 9 ensures the existence of $I \leq N$ such that either

$$a_I = C'[d \delta)(\lambda)c'] \rightarrow a_{I+1} = C'[d \delta)(\lambda)c']$$

and $d' \rightarrow d$ or

$$a_I = C'[d \delta)W(\lambda)c'] \rightarrow a_{I+1} = C'[W((\phi_0^{(W)+1})d \delta)(\lambda)c']$$

and $d \rightarrow d$.

Let us consider in the first and second cases respectively, the infinite derivations:

$D'' : a \rightarrow a_I \rightarrow a_{I+1} \rightarrow C'[d \delta)(\lambda)c'] \rightarrow C'[d_1 \delta)(\lambda)c'] \rightarrow \cdots \rightarrow C'[d_n \delta)(\lambda)c'] \cdots$

$D''' : a \rightarrow a_I \rightarrow a_{I+1} \rightarrow C'[d \delta)W(\lambda)c'] \rightarrow C'[d_1 \delta)W(\lambda)c'] \rightarrow \cdots \rightarrow C'[d_n \delta)W(\lambda)c'] \cdots$

In $D''$ and $D'''$, 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 $D$ 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 $D$. □

**Corollary 3** For every $a \in \Lambda$, the following equivalences hold:

$a \in \lambda_{g-SN}$ iff $a \in \lambda_{g-SN}$

$a \in \lambda-SN$ iff $a \in \lambda-SN$

$a \in \lambda_{g-SN}$ iff $a \in \lambda_{g-SN}$. Due to [13], $a \in \lambda-SN$ iff $a \in \lambda_{g-SN}$. Due to [17], $a \in \lambda-SN$ iff $a \in \lambda_{g-SN}$. □

## 5 The typed $\lambda$- and $\lambda g$-calculi

We prove $\lambda g$-SN of well typed terms using the technique developed in [18] to prove $\lambda g$-SN and suggested to us by P.-A. Melliès as a successful technique to prove $\lambda$-SN (personal communication). We recall the syntax and typing rules for the simply typed $\lambda$-calculus in de Bruijn notation. The types are generated from a set of basic types $T$ with the binary type operator $\rightarrow$. 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 $\lambda$-terms is given as follows:

<table>
<thead>
<tr>
<th>Types</th>
<th>$T ::= T \mid T \rightarrow T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environments</td>
<td>$E ::= nil \mid T, E$</td>
</tr>
<tr>
<td>Terms</td>
<td>$A_t ::= n \mid (A_t \delta)A_t \mid (T \lambda)A_t$</td>
</tr>
</tbody>
</table>

We let $A, B, \ldots$ range over $T$; $E, E_1, \ldots$ over $E$ and $a, b, \ldots$ over $A_t$.

The typing rules are given by the typing system **L1** as follows:

$$(\text{L1} - \text{var}) \quad A, E \vdash 1 : A \quad (\text{L1} - \lambda) \quad A, E \vdash b : B \quad E \vdash (A \lambda b : A \rightarrow B)$$

$$(\text{L1} - \text{var}) \quad E \vdash n : B \quad A, E \vdash n + 1 : B \quad (\text{L1} - \text{app}) \quad E \vdash b : A \rightarrow B \quad E \vdash a : A \quad E \vdash (a \delta)b : B$$

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 $E_{<i}$ stands for $E_1, \ldots, E_{i-1}$, etc.
Definition 16 The syntax for the simply typed λs-terms is given as follows:

\[ \mathcal{A}_s := \mathbf{N} \mid (\mathcal{A}_s \delta).\mathcal{A}_s \mid ((\mathcal{T} \lambda).\mathcal{A}_s) \mid ((\mathcal{A}_s \sigma^i).\mathcal{A}_s) \mid (\varphi_0^i).\mathcal{A}_s \quad i \geq 1, \; k \geq 0. \]

Types and environments are as above. The typing rules of the system \(\mathbf{Ls1}\) are:

The rules \(\mathbf{Ls1}\)-var, \(\mathbf{Ls1}\)-var, \(\mathbf{Ls1}\)-λ and \(\mathbf{Ls1}\)-app are exactly the same as \(\mathbf{L1}\)-var, \(\mathbf{L1}\)-var, \(\mathbf{L1}\)-λ and \(\mathbf{L1}\)-app, respectively. The new rules are:

\[
\begin{align*}
\text{\(\mathbf{Ls1} - \sigma\)} & \quad \frac{E \vdash b : B}{E \vdash (b \sigma^i)\alpha : A} \quad (\text{\(\mathbf{Ls1} - \phi\)}) \quad \frac{E \vdash b \alpha : A}{E \vdash \varphi_0^i \alpha}\end{align*}
\]

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

Definition 17 \(a \in \mathcal{A}_s\) is a well-typed term if for some environment \(E\) and type \(A\), \(E \vdash_{\mathbf{Ls1}} a : A\). We note \(\mathcal{A}_{s\text{ntd}}\) the set of well-typed terms.

The aim of this section is to prove that every well-typed λs-term \(a\) is λs\(g\)-SN (and hence λs-SN). To do so, we show \(\mathcal{A}_{s\text{ntd}} \subseteq \Xi \subseteq \mathcal{A}_{s\text{g}}\), where

\[ \Xi = \{ a \in \mathcal{A}_s : \text{for every subterm } b \text{ of } a, \; s(b) \in \lambda s-SN \}. \]

To prove \(\mathcal{A}_{s\text{ntd}} \subseteq \Xi\) (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 \(a, b, c \in \mathcal{A}_s\). We have:

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

Proof: All by induction on \(S\).

\[ \square \]

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

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

\[ E \vdash S(a \delta)(c \delta)U(B \lambda)\nu b : A\] iff (IH)

\[ E \vdash S(a \delta)U((\varphi_0^{N(U)+1})\alpha \delta)(B \lambda)\nu b : A\] iff (IH)

\[ E \vdash S((\varphi_0^{N(U)+1})\alpha \delta)(B \lambda)(\varphi_0^{N(U)+1})\alpha \delta)b : A\] iff (Lemma 12.1)

\[ E \vdash S((\varphi_0^{N(U)+1})\alpha \delta)(B \lambda)(\varphi_0^{N(U)+2})\alpha \delta)b : A\] iff (IH, twice)

\[ E \vdash S(c \delta)U(B \lambda)V((\varphi_0^{N(V)+1})\alpha \delta)b : A\] iff (Lemma 12.2)

\[ E \vdash S(c \delta)U(B \lambda)V((\varphi_0^{N(V)+N(U)+2})\alpha \delta)b : A\]

\[ \square \]

Lemma 14 (Subject reduction) If \(E \vdash_{\mathbf{Ls1}} a : A, \; a \to_{\lambda s\lambda} b\) 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 \(\sigma\text{-gen}\), use Lemma 12.3. Case \(\eta\text{-gen}\): If \(E \vdash ((a \delta)\nu)(B \lambda)b : A\), then, by Lemma 13, we have \(E \vdash W(\varphi_0^{N(W)+1})\alpha \delta)(B \lambda)b : A\) and, by Lemma 12.3, we conclude \(E \vdash W((\varphi_0^{N(W)+1})\alpha \sigma^i)b : A\).
Corollary 4 Let \( E \vdash_{\text{L}_1} a : A \), if \( a \rightarrow_{\text{sg}} b \) then \( E \vdash_{\text{L}_1} b : A \).

Lemma 15 (Typing of subterms) If \( a \in \text{As}_{\text{ut}} \) and \( b \triangleleft a \) then \( b \in \text{As}_{\text{ut}} \).

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. \( \square \)

Lemma 16 (Soundness of typing) If \( a \in A \), \( E \vdash_{\text{L}_1} a : A \) then \( E \vdash_{\text{L}_1} a : A \).

Proof: Easy induction on \( a \). \( \square \)

Proposition 2 \( \text{As}_{\text{ut}} \subseteq \Xi \).

Proof: Let \( a \in \text{As}_{\text{ut}} \) and \( b \) a subterm of \( a \). By Lemma 15, \( b \in \text{As}_{\text{ut}} \) and by Corollary 4, \( s(b) \in \text{As}_{\text{ut}} \). Since \( s(b) \in A \) (Thm. 2), Lemma 16 gives \( s(b) \) is \( \text{L}_1 \)-typable. But classical typable \( \lambda \)-terms are strongly normalising in the \( \lambda \)-calculus, Hence, \( s(b) \in \lambda \text{-SN} \) and, by Corollary 3, \( s(b) \in \lambda g \text{-SN} \). Therefore \( a \in \Xi \). \( \square \)

We prove now \( \Xi \subseteq \lambda g \text{-SN} \).

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

Proof: The proof is almost the same as the proof of lemma 11. \( \square \)

Proposition 3 For every \( a \in \text{As}_{\text{t}} \), if \( a \in \Xi \) then \( a \in \lambda g \text{-SN} \).

Proof: Assume \( a' \in \Xi \) and \( a' \not\in \lambda g \text{-SN} \), then there exists a term \( a \) of minimal size such that \( a \in \Xi \) and \( a \not\in \lambda g \text{-SN} \). Let \( D : a \rightarrow a_1 \rightarrow \cdots \rightarrow a_n \rightarrow \cdots \) be a minimal infinite \( \lambda g \)-derivation and follow the proof of Theorem 4 to obtain:

\[
D' : a_N = C[(d^i )c] \underset{\text{int}_{\lambda g}}{\rightarrow} C[(d_1^i )c] \underset{\text{int}_{\lambda g}}{\rightarrow} \cdots \underset{\text{int}_{\lambda g}}{\rightarrow} C[(d_n^i )c] \underset{\text{int}_{\lambda g}}{\rightarrow} \cdots
\]

Now three possibilities arise from Lemma 9. Two of them have been considered in the proof of Theorem 4 and contradicted the minimality of \( D \). Take the third one:

\( a = C'[(d^i )c] \) where \( d' \rightarrow d \). Now we have \( d' \rightarrow d \rightarrow d_1 \rightarrow \cdots \rightarrow d_n \rightarrow \cdots \). As \( d' \) is a subterm of \( a \), \( d' \in \Xi \), contradicting that \( a \) has minimal size. \( \square \)

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

Theorem 5 Well typed \( \lambda s \)-term are strongly normalising in the \( \lambda g \text{-calculus} \).

Corollary 5 Well typed \( \lambda s \)-term are strongly normalising in the \( \lambda s \)-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 \( \lambda \)-calculus but have never been combined together. We commented that the combination might indeed join both benefits and hence a \( \lambda \)-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


