# The Journal of Functional and Logic Programming

*The MIT Press*

Volume 1998, Article 5

*4 June, 1998*

ISSN 1080–5230. MIT Press Journals, Five Cambridge Center, Cambridge, MA 02142-1493, USA; (617)253-2889; *journals-orders@mit.edu, journals-info @mit.edu*. Published one article at a time in LaTeX source form on the Internet. Pagination varies from copy to copy. For more information and other articles see:

- *http://www.cs.tu-berlin.de/journal/jflp/*
- *http://mitpress.mit.edu/JFLP/*
- *gopher.mit.edu*
- *ftp://mitpress.mit.edu/pub/JFLP*

*The Journal of Functional and Logic Programming* is a peer-reviewed and electronically published scholarly journal that covers a broad scope of topics from functional and logic programming. In particular, it focuses on the integration of the functional and the logic paradigms as well as their common foundations.

# Calculi of Generalized $\beta$-Reduction and Explicit Substitutions: The Type-Free and Simply Typed Versions

Fairouz Kamareddine Alejandro Ríos J. B. Wells

4 June, 1998

**Abstract**

Extending the $\lambda$-calculus with either explicit substitution or generalized reduction has been the subject of extensive research recently, and still has many open problems. This paper is the first investigation into the properties of a calculus combining both generalized reduction and explicit substitutions. We present a calculus, $\lambda gs$, that combines a calculus of explicit substitution, $\lambda s$, and a calculus with generalized reduction, $\lambda g$. We believe that $\lambda gs$ is a useful extension of the $\lambda$-calculus, because it allows postponement of work in two different but complementary ways. Moreover, $\lambda gs$ (and also $\lambda s$) satisfies properties desirable for calculi of explicit substitutions and generalized reductions. In particular, we show that $\lambda gs$ preserves strong normalization, is a conservative extension of $\lambda g$, and simulates $\beta$-reduction of $\lambda g$ and the classical $\lambda$-calculus. Furthermore, we study the simply typed versions of $\lambda s$ and $\lambda gs$, and show that well-typed terms are strongly normalizing and that other properties, such as typing of subterms and subject reduction, hold. Our proof of the preservation of strong normalization (PSN) is based on the minimal derivation method. It is, however, much simpler, because we prove the commutation of arbitrary internal and external reductions. Moreover, we use one proof to show both the preservation of $\lambda$-strong normalization in $\lambda s$ and the preservation of $\lambda g$-strong normalization in $\lambda gs$. We remark that the technique of these proofs is not suitable for calculi without explicit substitutions (e.g., the preservation of $\lambda$-strong normalization in $\lambda g$ requires a different technique).

# 1 Introduction

## 1.1 The $\lambda$-Calculus with Generalized Reduction

In the term $((\lambda_x.\lambda_y.N)P)Q$, the abstraction starting with $\lambda_x$ and the argument $P$ form the redex $(\lambda_x.\lambda_y.N)P$. When this redex is contracted, the abstraction starting with $\lambda_y$ and $Q$ will in turn form a redex. What is important is that the only argument the abstraction starting with $\lambda_y$ (or some residual of this abstraction) can ever have is $Q$ (or some residual of $Q$). This fact has been exploited by many researchers, and reduction has been extended so that the implicit redex based on the matching $\lambda_y$ and $Q$ is given the same priority as the intervening redex.

An initial attempt to generalize the notion of redex might be to define a rule like the following:

$$(\lambda_x.\lambda_y.N)PQ \to (\lambda_x.N[y{:=}Q])P$$

It quickly becomes evident that this is not sufficient. For example, the proposed rule does not allow directly reducing the binding of $y$ to $Q$ in the term $A \equiv (\lambda_z.(\lambda_x.\lambda_y.N)P)RQ$. We shall exploit the notion of a *well-balanced segment* (sometimes known as a *$\beta$-chain*), which is the special case of one-hole contexts given by this grammar:[1]

$$S ::= [\,\cdot\,] \mid (S[\lambda_x.[\,\cdot\,]])M \mid S[S]$$

Using balanced segments, *generalized reduction* is then given by this rule:

$$S[\lambda_x.M]N \to S[M[x{:=}N]]$$

We find the above definition of well-balanced segments and generalized reduction rather cumbersome, and believe that a more elegant definition can be given. To do so, we change from the classical notation to the *item notation*. Instead of writing $\lambda_x.M$, we write $(\lambda_x)M$; and instead of $MN$, we write $(N\delta)M$. Item notation has many advantages, as shown in [KN95, KN96]. Let us illustrate here using the term $A$ given above, which we write in item notation as in Figure 1. We see immediately that the redexes

[1] Actually, this is a grammar for expressions which can then be turned into contexts by rewriting innermost subexpressions of the form $S_1[S_2]$ into $S_1$ changed by replacing its hole with $S_2$. This is part of the awkwardness of specifying balanced segments in classical notation.

$$(Q\delta)\ (R\delta)\ (\lambda_z)\ (P\delta)\ (\lambda_x)\ (\lambda_y)\quad N$$

Figure 1: Redexes in item notation in term $A$

originate from the couples $(Q\delta)(\lambda_y)$, $(R\delta)(\lambda_z)$, and $(P\delta)(\lambda_x)$. Moreover, $(Q\delta)(R\delta)(\lambda_z)(P\delta)(\lambda_x)(\lambda_y)$ is a well-balanced segment. This natural matching was not present in the classical notation. We call items of the form $(P\delta)$ and $(\lambda_x)$ application and abstraction items, respectively. With item notation, generalized reduction is written as: $(M\delta)\overline{s}(\lambda_x)N \to_{g\beta} \overline{s}\{N[x := M]\}$ for $\overline{s}$ well balanced. (Here, the brackets $\{$ and $\}$ are used for grouping purposes, so that no confusion arises.) For example,

$$(Q\delta)(R\delta)(\lambda_z)(P\delta)(\lambda_x)(\lambda_y)N \to_{g\beta} (R\delta)(\lambda_z)(P\delta)(\lambda_x)\{N[y := Q]\}$$

Surely this is clearer than writing

$$(\lambda_z.(\lambda_x.\lambda_y.N)P)RQ \to_{g\beta} (\lambda_z.(\lambda_x.N[y := Q]P)R$$

Generalized reduction was first introduced by Nederpelt in 1973 to aid in proving the strong normalization of AUTOMATH [Ned73]. Kamareddine and Nederpelt have shown how generalized reduction makes more redexes visible, allowing flexibility in reducing a term [KN95]. Bloo, Kamareddine, and Nederpelt show that with generalized reduction, one may indeed avoid size explosion without the cost of a longer reduction path; and, simultaneously, the $\lambda$-calculus can be elegantly extended with definitions that result in shorter type derivations [BKN96]. Generalized reduction is strongly normalizing [BKN96] for all systems of the $\lambda$-cube [Bar92], and preserves the strong normalization of ordinary $\beta$-reduction [Kam96]. In particular, generalized reduction allows the postponement of $K$-reductions (which discard their argument) after $I$-reductions (which use their argument in at least one place).

An alternative approach to generalized reduction which has been followed by many researchers is to use one of these two local transformations:

$$
\begin{array}{ll}
(\theta) & ((\lambda_x.N)P)Q \to (\lambda_x.NQ)P \\
(\gamma) & (\lambda_x.\lambda_y.N)P \to \lambda_y.(\lambda_x.N)P
\end{array}
$$

These rules transform terms to make more redexes visible to the ordinary notion of $\beta$-reduction. For example, the $\gamma$-rule makes sure that $\lambda_y$ and $Q$ in the example $A$ above can form a redex before the redex based on $\lambda_x$ and $P$ is contracted. Also, $((\lambda_x.\lambda_y.N)P)Q \to_\theta (\lambda_x.(\lambda_y.N)Q)P$, and hence both $\theta$ and $\gamma$ put $\lambda_y$ next to its matching argument. The $\theta$-rule moves the argument next to its matching $\lambda$, whereas $\gamma$ moves the $\lambda$ next to its matching argument.

Obviously, $\theta$ and $\gamma$ are related to generalized reduction. In fact, $\theta$ and $\gamma$ transform terms to make more potential redexes visible, and then conventional $\beta$-reduction can be used to contract those newly visible redexes. Generalized reduction, on the other hand, performs reduction on the potential redexes without having to bother to make them into classical redexes. Now, we go back to the above example, where with generalized reduction we have: $(\lambda_z.(\lambda_x.\lambda_y.N)P)RQ \to_{g\beta} (\lambda_z.(\lambda_x.N[y := Q])P)R$. We illustrate how $\theta$ and $\gamma$ work. The $\theta$ case:

$$
\begin{array}{c}
(\lambda_z.(\lambda_x.\lambda_y.N)P)RQ \to_\theta \quad (\lambda_z.(\lambda_x.\lambda_y.N)PQ)R \to_\theta \\
(\lambda_z.(\lambda_x.(\lambda_y.N)Q)P)R \to_\beta \quad (\lambda_z.(\lambda_x.N[y := Q])P)R
\end{array}
$$

The $\gamma$ case:

$$
\begin{array}{c}
(\lambda_z.(\lambda_x.\lambda_y.N)P)RQ \to_\gamma \quad (\lambda_z.\lambda_y(\lambda_x.N)P)RQ \to_\gamma \\
(\lambda_y(\lambda_z.(\lambda_x.N)P)R)Q \to_\beta \quad (\lambda_z.(\lambda_x.N[y := Q])P)R
\end{array}
$$

Finally, note that in item notation it is easier to describe $\theta$ and $\gamma$. We illustrate with $\theta$ and the above example:

The term $(Q\delta)(R\delta)(\lambda_z)(P\delta)(\lambda_x)(\lambda_y)N$ can be reshuffled to the term $(R\delta)(\lambda_z)(P\delta)(\lambda_x)(Q\delta)(\lambda_y)N$ to transform the bracketing structure $\{\{\ \}\{\ \}\}$ into $\{\ \}\{\ \}\{\ \}$, where all the redexes correspond to adjacent "$\{$" and "$\}$." In other words, Figure 1 can be redrawn using the $\theta$-reduction twice as in Figure 2.

The $\theta$-rule can be applied to both explicitly and implicitly typed systems. However, the transfer of $\gamma$ to explicitly typed systems is not straightforward, since in these systems the type of $y$ in the term $A$ may be affected by the reducible pair of $\lambda_x$ and $P$. For example, it is fine to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_\theta$

$$(R\delta)\ (\lambda_z)\ (P\delta)\ (\lambda_x)\ (Q\delta)\ (\lambda_y)\quad N$$

Figure 2: $\theta$-normal forms in item notation for term $A$

$(\lambda_{x:*}.(\lambda_{y:x}.y)u)z$, but not to write $((\lambda_{x:*}.\lambda_{y:x}.y)z)u \to_\gamma (\lambda_{y:x}.(\lambda_{x:*}.y)z)u$.[2]

Local transformations such as $\gamma$ and $\theta$ began to appear in the literature around 1989. (See [KW95a] for a summary.) Regnier [Reg92] introduced the notion of a *premier redex*, which is similar to the redex based on $\lambda_y$ and $Q$ above (which we call a *generalized redex*). Later, he used $\theta$ and $\gamma$ (and called the combination $\sigma$) to show that the perpetual reduction strategy finds the longest reduction path when the term is strongly normalizing (SN) [Reg94]. Vidal also introduced similar reductions [Vid89]. Kfoury, Tiuryn, and Urzyczyn used $\theta$ (and other reductions) to show that typability in ML is equivalent to acyclic semi-unification [KTU94]. Sabry and Felleisen described a relationship between a reduction similar to $\theta$ and a particular style of CPS [SF93]. De Groote [dG93] used $\theta$ and Kfoury and Wells [KW95b] used $\gamma$ to reduce the problem of $\beta$-strong normalization to the problem of weak normalization (WN) for related reductions. Kfoury and Wells used $\theta$ and $\gamma$ to reduce typability in the rank-2 restriction of system F to the problem of acyclic semi-unification [KW94]. Klop, Sørensen, and Xi [Klo80, Xi96, Sør97] used related reductions to reduce SN to WN. Finally, [AFM$^+$95] used $\theta$ (called "let-C") as a part of an analysis of how to represent sharing in a call-by-need language implementation in a formal calculus.

## 1.2 The $\lambda$-Calculus with Explicit Substitution

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

[2] An alternative is to apply $\gamma$ to the *type erasure* of the term, which may be quite complicated to express in terms of the type-annotated term.

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

By representing substitutions in the structure of terms and by providing (first-order) reductions to propagate the substitutions, explicit substitution provides a number of benefits. A major benefit is that explicit substitution allows more flexibility in ordering work. Propagating substitutions through a particular subterm can wait until the subterm is the focus of computation. This allows all of these substitutions to be done at once, thus improving locality of reference. Obtaining more control over the ordering of work has become an important issue in functional programming-language implementation (cf. [Jon87]). The flexibility provided by explicit substitution also allows postponing unneeded work indefinitely (i.e., avoiding it completely). This can yield profits, since implicit substitution can be an inefficient, maybe even exploding, process, owing to the many repetitions it causes. Another benefit is that explicit substitution allows formal modeling of the techniques used in real implementations, e.g., environments. Because explicit substitution is closer to real implementations, it has the potential to provide a more accurate cost model. (This possibility is particularly interesting in light of the difficulty encountered in formulating a useful cost model in terms of graph reduction [LM96, Jon87].)

Proof assistants may benefit from explicit substitution, owing to the desire to perform substitutions locally and in a formal manner. Local substitutions are needed as follows. Given $xx[x{:=}y]$, one may not be interested in having $yy$ as the result of $xx[x{:=}y]$, but rather only $yx[x{:=}y]$. In other words, one only substitutes one occurrence of $x$ by $y$, and continues the substitution later. Theorem provers such as Nuprl [CABC86] and HOL [GM93] implement substitution that allows the local replacement of some abbreviated term. This avoids a size explosion when it is necessary to replace a variable by a huge term only in specific places to prove a certain theorem.

Formalization helps in studying the termination and confluence properties of systems. Without formalization, important properties such as the correctness of substitutions often remain unestablished, causing mistrust in the implementation. As the implementation of substitution in many theorem provers is not based on a formal system, it is not clear what properties their underlying substitution has, nor can their implementations be compared. Thus, it helps to have a choice of explicit substitution systems whose properties have already been established. This is witnessed by the recent theorem prover ALF, which is formally based on Martin-Löf's type theory with explicit substitution [Mag95]. Another justification for explicit substitution in theorem proving is that some researchers believe "tactics" can be replaced by the notion of incomplete proofs, which are believed to need explicit substitutions [Hur96b, Mag95].

The last 15 years have seen an increasing interest in formalizing substitution explicitly; various calculi, including new operators to denote substitution, have been proposed. Among these calculi we mention $C\lambda\xi\phi$ [dB78]; the calculi of categorical combinators [Cur86]; $\lambda\sigma$ [ACCL91], $\lambda\sigma_{\Uparrow}$ [CHL96], and $\lambda\sigma_{SP}$ [Río93], referred to as the $\lambda\sigma$-family; $\lambda\upsilon$ [BBLRD96], a descendant of the $\lambda\sigma$-family; $\varphi\sigma BLT$ [KN93]; $\lambda\mathsf{exp}$ [Blo95]; $\lambda s$ [KR95a]; $\lambda s_e$ [KR97]; and $\lambda\zeta$ [Hur96a]. All of these calculi (except $\lambda\mathsf{exp}$) are described in a de Bruijn setting, where natural numbers play the role of variables.

In [KR95a], we extended the $\lambda$-calculus with explicit substitutions by turning de Bruijn's meta operators into object operators, thus offering a style of explicit substitution that differs from that of $\lambda\sigma$. The resulting calculus $\lambda s$ remains intuitively as close to the $\lambda$-calculus as possible for a calculus of explicit substitution. An important motivation for introducing the $\lambda s$-calculus [KR95a] was to provide a calculus of explicit substitutions which would both preserve strong normalization and have a confluent extension on open terms [KR97]. There are calculi of explicit substitutions that are confluent on open terms, e.g., $\lambda\sigma_{\Uparrow}$ [CHL96] and $\lambda\zeta$ [Hur96a], but they also have important disadvantages. Melliès proved that $\lambda\sigma_{\Uparrow}$ (as well as the rest of the $\lambda\sigma$-family and the categorical combinators) does not preserve strong normalization [Mel95]. There are also calculi that preserve strong normalization, e.g., the $\lambda\upsilon$-calculus [BBLRD96], but this calculus is not confluent on open terms. Recently, the $\lambda\zeta$-calculus (cf. [Hur96a]) has been proposed as a calculus that preserves strong normalization and is itself confluent on open terms. The $\lambda\zeta$-calculus works with two new applications that allow the passage of substitutions within classical applications only if these applications have a head variable. This is done to cut the branch of the critical pair that is responsible for the nonconfluence of $\lambda\upsilon$ on open terms. Hence, $\lambda\zeta$ preserves strong normalization, and is itself confluent on open terms. Unfortunately, $\lambda\zeta$ is not able to simulate one-step $\beta$-reduction, as shown in [Hur96a]. Instead, it simulates only a "big-step" $\beta$-reduction. On the other hand, $\lambda s$

has been extended to $\lambda s_e$, which is confluent on open terms (cf. [KR97]) and simulates one-step $\beta$-reduction, but the preservation of strong normalization for the extension $\lambda s_e$ remained an open problem until it was shown at the end of November 1997 by Bruno Guillome that the property does not hold.

## 1.3 Combining Generalized Reduction and Explicit Substitution

We have already explained the separate usefulness of generalized reduction and explicit substitutions. The main benefits of these concepts are similar: both emphasize flexibility in the ordering of operations. In particular, generalized reduction and explicit substitution allow the postponement of work, but in different, complementary ways. On one side, generalized reduction always allows unnecessary $K$-redexes to be bypassed. Explicit substitution will not, in general, allow this, because reducing the $K$-redex might be necessary to expose an essential $I$-redex. Similarly, on the other side, explicit substitution allows bypassing any work inside a subterm that will be discarded later. However, generalized reduction does not provide any means for performing only those parts of a substitution that will be used later. Thus, we can see that their benefits are complementary.

We claim that a system with the combination of generalized reduction and explicit substitution is more advantageous than a system containing each concept separately. Obviously, if the benefits of both are desired simultaneously, it is important to study the combination, a task which this paper performs. Before the combination can be safely used, it must be checked that this combination is sound and safe, exactly as it has been checked that each of explicit substitutions and generalized reductions separately 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.

Generalized reduction, $(g\beta)$, has never before been introduced in a de Bruijn setting. Explicit substitution has almost always been presented in a de Bruijn setting. Since explicit-substitution calculi are usually written with de Bruijn indices, we combine $g\beta$-reduction and explicit substitution in a de Bruijn setting, giving the first calculus of generalized reduction à la de Bruijn.[3] As we need to describe generalized redexes in an elegant way,

[3]The main advantages of de Bruijn's notation is that it allows us to get rid of Baren-

we use a notation for $\lambda$-terms that is suitable for this purpose, the *item notation* [KN96].

In Section 2, we introduce the calculus of generalized reduction, the $\lambda g$-calculus, in item notation with de Bruijn indices, and prove its confluence. In Section 3 we introduce the $\lambda s$-calculus and extend it into the $\lambda gs$-calculus by adding the necessary reductions to simulate $\to_{g\beta}$. We show that $\lambda gs$ is a conservative extension of $\lambda g$, it simulates $g\beta$, and is confluent. In Section 4, we prove that the $\lambda gs$-calculus preserves $\lambda g$-strong normalization (i.e., $a$ is $\lambda g$-SN $\Rightarrow$ $a$ is $\lambda gs$-SN), and that the $\lambda s$-calculus preserves the $\lambda$-strong normalization. We conclude that $a$ is $\lambda$-SN $\Leftrightarrow$ $a$ is $\lambda s$-SN $\Leftrightarrow$ $a$ is $\lambda g$-SN $\Leftrightarrow$ $a$ is $\lambda gs$-SN. In Section 5, the simply typed versions of the $\lambda s$- and $\lambda gs$-calculi are presented and subject reduction, typing of subterms, strong normalization of well-typed terms, and other properties are proved.

## 2 The $\lambda g$-Calculus

We assume familiarity with the $\lambda$-calculus and its various notions such as reduction, contexts, etc. Where not otherwise defined, we follow the conventions of Barendregt [Bar84, Bar92]. Nevertheless, we present some basic needed definitions in what follows:

**Definition 1 (Reduction Notations)**
*Let $A$ be a set, and $r$ a binary relation on $A$. We denote the fact $(a,b) \in r$ by $a \to_r b$ or $a \to b$ when the context is clear enough. We denote:*

1. *$r^\epsilon$ or $\stackrel{\epsilon}{\to}_r$ or just $\stackrel{\epsilon}{\to}$, the reflexive closure of $r$;*
2. *$r^+$ or $\to_r^+$ or just $\to^+$, the transitive closure of $r$;*
3. *$r^*$ or $\twoheadrightarrow_r$ or just $\twoheadrightarrow$, the reflexive and transitive closure of $r$. When $a \twoheadrightarrow b$, we say there exists a* reduction sequence *from $a$ to $b$;*
4. *$=_r$, the reflexive, symmetric and transitive closure of $\to_r$; that is, $=_r$ is the least-equivalence relation containing $\to_r$; and*
5. *$=$ for syntactic identity, and write $a = b$ when $a$ and $b$ are syntactically identical.*

dregt's variable convention (which insists that free variables be different from bound ones, and that if $\lambda_x$ and $\lambda_y$ occur in a term, then $x$ must be distinct from $y$), since $\alpha$-congruent terms are syntactically identical.

**Definition 2 (Reduction Relations and Systems)** *For a given set of rewrite rules $r$ on a set $A$, we define $r$-reduction to be the reduction relation of the $r$-calculus (i.e., the least-compatible relation containing the rules of $r$). If $R$ is a reduction relation on a set $A$, we say that $(A, R)$ is a reduction system.*[4]

**Definition 3 (Confluence and Church Rosser)** *Let $R$ be a reduction relation on $A$. For $R$, we define local confluence (or weak Church Rosser, WCR); confluence (or Church Rosser, CR); and strong confluence (or strong Church Rosser, SCR) respectively as follows:*

1. *WCR:*

$$\forall a, b, c \in A \ \exists d \in A \ : (a \to_R, b \ \wedge \ a \to_R \ c) \Rightarrow (b \twoheadrightarrow_R \ d \ \wedge \ c \twoheadrightarrow_R \ d)\,.$$

2. *CR:*

$$\forall a, b, c \in A \ \exists d \in A \ : (a \twoheadrightarrow_R \ b \ \wedge \ a \twoheadrightarrow_R \ c) \Rightarrow (b \twoheadrightarrow_R \ d \ \wedge \ c \twoheadrightarrow_R \ d)\,.$$

3. *SCR:*

$$\forall a, b, c \in A \ \exists d \in A \ : (a \to_R \ b \ \wedge \ a \to_R \ c) \Rightarrow (b \to_R \ d \ \wedge \ c \to_R \ d)\,.$$

**Definition 4 (Normal Forms and Normalization)** *Let $R$ be a reduction relation on $A$.*

- *We say that $a \in A$ is an $R$-*normal form *($R$-nf for short) if there exists no $b \in A$ such that $a \to_R b$.*
- *We say that $b$* has an $R$-normal form *if there exists an $R$-normal form $a$ such that $b \twoheadrightarrow_R a$. In this case, we say $b$ is $R$-normalizing.*
- *We say that $R$ is* weakly normalizing *(WN) if every $a \in A$ has an $R$-normal form.*
- *We say that $R$ is* strongly normalizing *(SN) if there is no infinite sequence $(a_i)_{i \geq 0}$ in $A$ such that $a_i \to_R a_{i+1}$ for all $i \geq 0$.*

[4] Note that we depart from [Bar84, definition 3.1.1], where a reduction relation is not only compatible, but also reflexive and transitive. Our reason for doing so is that we want to keep the notation for the reduction system simpler.

- *We say that a term $M$ is strongly $R$-normalizing if there are no infinite $R$-reduction sequences starting at $M$.*
- *When no confusion arises, then $R$ may be omitted, and we speak simply of normal forms or normalization.*

Note that confluence of $R$ guarantees unicity of $R$-normal forms. In that case, the $R$-normal form of $a$, if it exists, is denoted by $R(a)$. Strong normalization implies weak normalization and, therefore, the existence of normal forms. The following lemma is an important connection between strong normalization and confluence (its proof can be found in [Bar84], proposition 3.1.25):

**Lemma 1 (Newman[New42])** *Every strongly normalizing, locally confluent reduction relation is confluent.*

We assume familiarity with de Bruijn notation; e.g., $\lambda_x.\lambda_y.(x(\lambda_z.zx))y$ is written in ordinary de Bruijn notation as $\lambda(\lambda(2(\lambda(13))1))$ and $\lambda_x.\lambda_y.xy$ as $\lambda\lambda(21)$. To translate free variables, we assume a fixed-ordered list of binders (written from left to right) $\ldots, \lambda_z, \lambda_y, \lambda_x$, and prefix it to the term to be translated. Hence, $\lambda_x.yz$ translates as $\lambda 34$, whereas $\lambda_x.zy$ translates as $\lambda 43$. Since generalized $\beta$-reduction is better described in item notation, we adopt the item syntax (see [KN95, KN96] for the advantages of item notation) and write $a\,b$ as $(b\,\delta)a$, and $\lambda a$ as $(\lambda)a$. The $\delta$ symbol informs us that we are dealing with an application, just as $\lambda$ informs us that there is an abstraction.

**Definition 5** *The* set of terms, $\Lambda$, *is defined by the grammar* $\Lambda ::= \mathbb{N} \mid (\Lambda\,\delta)\Lambda \mid (\lambda)\Lambda$. *We let $a, b, \ldots$ range over $\Lambda$ and $m, n, \ldots$ over $\mathbb{N}$ (positive natural numbers).*[5] *We write $a \lhd b$ when $a$ is a subterm of $b$. A reduction $\to$ is* compatible on $\Lambda$ *when for all $a$, $b$, $c \in \Lambda$, it holds that $a \to b$ implies $(a\,\delta)c \to (b\,\delta)c$, $(c\,\delta)a \to (c\,\delta)b$, and $(\lambda)a \to (\lambda)b$.*

For example, $(\lambda_x\lambda_y.zxy)(\lambda_x.yx) \to_\beta \lambda_u.z(\lambda_x.yx)u$, which in de Bruijn notation is $(\lambda\lambda 521)(\lambda 31) \to_\beta \lambda 4(\lambda 41)1$, is expressed in de Bruijn *item* notation as $((\lambda)(1\delta)3\delta)(\lambda)(\lambda)(1\delta)(2\delta)5 \to_\beta (\lambda)(1\delta)((\lambda)(1\delta)4\delta)4$. Note that we did not simply replace $2$ in $(\lambda)(1\delta)(2\delta)5$ by $(\lambda)(1\delta)3$. Instead, we decreased $5$ as one $\lambda$ disappeared, and incremented the free variables of $(\lambda)(1\delta)3$ as they

[5] Our use of $\mathbb{N}$ as the set of *positive* natural numbers may be considered nonstandard by computer scientists who insist on having the number 0 as an element of $\mathbb{N}$.

occurred within the scope of one more $\lambda$. 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 6** *The* updating functions $U_k^i : \Lambda \to \Lambda$ *for* $k \geq 0$ *and* $i \geq 1$ *are defined inductively:*

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

In the following, 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 7** *The* meta substitutions at level $j$, *for* $j \geq 1$, *of a term* $b \in \Lambda$ *in a term* $a \in \Lambda$, *denoted* $a\{\!\{\mathsf{j} \leftarrow b\}\!\}$, *are defined inductively on* $a$ *as follows:*

$$\begin{array}{l} ((a_1\delta)a_2)\{\!\{\mathsf{j}\leftarrow b\}\!\} = ((a_1\{\!\{\mathsf{j}\leftarrow b\}\!\})\delta)(a_2\{\!\{\mathsf{j}\leftarrow b\}\!\}) \\ ((\lambda)c)\{\!\{\mathsf{j}\leftarrow b\}\!\} = (\lambda)(c\{\!\{\mathsf{j+1}\leftarrow b\}\!\}) \end{array} \qquad \mathsf{n}\{\!\{\mathsf{j}\leftarrow b\}\!\} = \begin{cases} \mathsf{n-1} & \textit{if } n > j, \\ U_0^j(b) & \textit{if } n = j, \\ \mathsf{n} & \textit{if } n < j. \end{cases}$$

The following lemma establishes the properties of meta substitution and updating.

**Lemma 2** *Let* $a, b, c \in \Lambda$. *The following properties hold:*

$$\begin{array}{lll}
\textit{1. for } k < n < k+i\textit{:} & (U_k^i(a))\{\!\{\mathsf{n}\leftarrow b\}\!\} & = U_k^{i-1}(a) \\
\textit{2. for } l \leq k < l+j\textit{:} & U_k^i(U_l^j(a)) & = U_l^{j+i-1}(a) \\
\textit{3. for } l+j \leq k+1\textit{:} & U_k^i(U_l^j(a)) & = U_l^j(U_{k+1-j}^i(a)) \\
\textit{4. for } k+i \leq n\textit{:} & (U_k^i(a))\{\!\{\mathsf{n}\leftarrow b\}\!\} & = U_k^i(a\{\!\{\mathsf{n-i+1}\leftarrow b\}\!\}) \\
\textit{5. for } n \leq k+1\textit{:} & U_k^i(a\{\!\{\mathsf{n}\leftarrow b\}\!\}) & = (U_{k+1}^i(a))\{\!\{\mathsf{n}\leftarrow U_{k-n+1}^i(b)\}\!\} \\
\textit{6. for } i \leq n\textit{:} & a\{\!\{\mathsf{i}\leftarrow b\}\!\}\{\!\{\mathsf{n}\leftarrow c\}\!\} & = a\{\!\{\mathsf{n+1}\leftarrow c\}\!\}\{\!\{\mathsf{i}\leftarrow b\{\!\{\mathsf{n-i+1}\leftarrow c\}\!\}\}\!\}
\end{array}$$

**Proof of Lemma 2** The proof is by induction on $a$. The proof of property 4 requires property 2 with $l = 0$; the proof of property 6 uses properties 1 and 4, both with $k = 0$; and finally, property 3 with $l = 0$ is needed to prove property 5.

**Proof of Lemma 2** □

To introduce generalized $\beta$-reduction, we need some definitions (cf. [KN96]).

**Definition 8 (Items and Segments)** Items*,* segments*, and* well-balanced segments *(w.b.) are defined, respectively, by:*

$$\mathcal{I} ::= (\Lambda\,\delta) \mid (\lambda)$$

$$\mathcal{S} ::= \phi \mid \mathcal{I}\,\mathcal{S}$$

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

*where $\phi$ is the empty segment. Hence, a segment is a sequence of items. Items $(a\,\delta)$ and $(\lambda)$ are called a* $\delta$-item *and a* $\lambda$-item*, respectively. We let $I$, $J$, ... range over $\mathcal{I}$; $S$, $S'$, ... over $\mathcal{S}$; and $W$, $U$, ... over $\mathcal{W}$. For a segment $S$, $\operatorname{len} S$ is given by $\operatorname{len}\phi = 0$, and $\operatorname{len}(I\,S) = 1 + \operatorname{len} S$. The number of main $\lambda$-items in $S$, $\#_\lambda(S)$, is given by $\#_\lambda(\phi) = 0$; $\#_\lambda((a\,\delta)S) = \#_\lambda(S)$; and $\#_\lambda((\lambda)S) = 1 + \#_\lambda(S)$.*

**Definition 9 ($\boldsymbol{\lambda}$-Calculus)** *The* $\lambda$-calculus *(à la de Bruijn) is the reduction system $(\Lambda, \to_\beta)$, where $\to_\beta$ is the least-compatible reduction on $\Lambda$ generated by the $\beta$-rule: $(a\delta)(\lambda)b \to b\{\!\{1 \leftarrow a\}\!\}$.*

**Definition 10 ($\boldsymbol{\lambda g}$-Calculus)** *The* $\lambda g$-calculus *is the reduction system $(\Lambda, \to_{g\beta})$, where $\to_{g\beta}$ denotes* generalized $\beta$-reduction*, the least-compatible reduction on $\Lambda$ generated by the $g\beta$-rule:*

$$(a\delta)W(\lambda)b \to W(b\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(a)\}\!\}) \quad \textit{where } W \textit{ is well balanced}$$

**Remark 1** *The $\beta$-rule is an instance of the $g\beta$-rule. (Take $W = \phi$, and check that $U_0^1(a) = a$.)*

Now, let us briefly explain the relation between $\to_{g\beta}$ and $\to_\theta$ and $\to_\gamma$, given in the introduction. It is helpful to write $\to_\theta$ and $\to_\gamma$ in item notation:

$$(Q\delta)(P\delta)(\lambda_x)N \to_\theta (P\delta)(\lambda_x)(Q\delta)N$$

$$(P\delta)(\lambda_x)(\lambda_y)N \to_\gamma (\lambda_y)(P\delta)(\lambda_x)N$$

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 $(\lambda)$ in $N$. In $\to_\gamma$, the same happens but now it is $(\lambda_y)$ that is given the chance to look for its matching $(-\delta)$. Only after reshuffling has taken place can the newly found redex be contracted. On the other hand, $\to_{g\beta}$ avoids reshuffling and contracts the redex as soon as it sees the matching of $\delta$ and $\lambda$.

In the following, we extend the definitions of updating and meta substitution to cover segments, and prove some useful properties.

**Definition 11** *Let $S \in \mathcal{S}$, $a, b \in \Lambda$, $k \geq 0$, and $n, i \geq 1$. We define $U_k^i(S)$ and $S\{\!\{\mathsf{n} \leftarrow a\}\!\}$ by:*

$$\begin{array}{llll} U_k^i(\phi) & = \phi & \phi\{\!\{\mathsf{n}\leftarrow a\}\!\} & = \phi \\ U_k^i((b\,\delta)S) & = (U_k^i(b)\,\delta)U_k^i(S) & ((b\,\delta)S)\{\!\{\mathsf{n}\leftarrow a\}\!\} & = (b\{\!\{\mathsf{n}\leftarrow a\}\!\}\,\delta)(S\{\!\{\mathsf{n}\leftarrow a\}\!\}) \\ U_k^i((\lambda)S) & = (\lambda)(U_{k+1}^i(S)) & ((\lambda)S)\{\!\{\mathsf{n}\leftarrow a\}\!\} & = (\lambda)(S\{\!\{\mathsf{n+1}\leftarrow a\}\!\}) \end{array}$$

**Lemma 3** *Let $S$, $T$ be segments, and $a, b \in \Lambda$. The following hold:*

1. $U_k^i(S\,T) = U_k^i(S)U_{k+\#_\lambda(S)}^i(T)$ *and* $U_k^i(S\,a) = U_k^i(S)U_{k+\#_\lambda(S)}^i(a)$.
2. $\mathrm{len}(S) = \mathrm{len}(U_k^i(S))$, *and* $\#_\lambda(S) = \#_\lambda(U_k^i(S))$, *and if $S$ is w.b., then $U_k^i(S)$ is w.b.*
3. $(S\,\xi)\{\!\{\mathsf{n} \leftarrow a\}\!\} = S\{\!\{\mathsf{n} \leftarrow a\}\!\}\,\xi\{\!\{\mathsf{n} + \#_\lambda(S) \leftarrow a\}\!\}$ *for $\xi$ a segment or a term.*
4. $\mathrm{len}(S) = \mathrm{len}(S\{\!\{\mathsf{n} \leftarrow a\}\!\})$, *and* $\#_\lambda(S) = \#_\lambda(S\{\!\{\mathsf{n} \leftarrow a\}\!\})$. *If $S$ is w.b., then $S\{\!\{\mathsf{n}\leftarrow a\}\!\}$ is w.b.*

**Proof of Lemma 3** Points 1 and 3, by induction on $S$. Points 2 and 4, by induction on $S$ using points 1 and 3, respectively.

**Proof of Lemma 3** □

**Lemma 4 (Preservation of $\boldsymbol{\beta}$-Equivalence)** *Let $a$, $b \in \Lambda$. If $a \twoheadrightarrow_{g\beta} b$, then $a =_\beta b$.*

**Proof of Lemma 4** It is sufficient to prove by induction on $a$ that $a \to_{g\beta} b$ implies $a =_\beta b$. We will only prove the particular base case $a = (c\delta)W(\lambda)d \to_{g\beta} W(d\{\!\{\mathsf{1} \leftarrow U_0^{\#_\lambda(W)+1}(c)\}\!\}) = b$, with $W \neq \phi$, since the other cases are simpler. We prove this case by a nested induction on $\operatorname{len} W$. Observe that $W = (e\delta)W_1(\lambda)W_2$, where $W_1$ and $W_2$ are well balanced, because $W \neq \phi$. Let $w_1 = \#_\lambda(W_1)$ and $w_2 = \#_\lambda(W_2)$. We have the following equalities, where in the justifications "IH" means the induction hypothesis and the numbers are lemmas:

$$
\begin{array}{lll}
 & & (c\delta)W(\lambda)d \\
 & = & (c\delta)(e\delta)W_1(\lambda)W_2(\lambda)d \\
\text{(IH)} & =_\beta & (c\delta)W_1((W_2(\lambda)d)\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\}) \\
\text{(3.3)} & = & (c\delta)W_1(W_2\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\})(\lambda)(d\{\!\{\mathsf{2}+\mathsf{w_2}\leftarrow U_0^{w_1+1}(e)\}\!\}) \\
\text{(IH \& 3.4)} & =_\beta & W_1(W_2\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\})(d\{\!\{\mathsf{2}+\mathsf{w_2}\leftarrow U_0^{w_1+1}(e)\}\!\}\{\!\{\mathsf{1}\leftarrow U_0^{w_1+w_2+1}(c)\}\!\}) \\
\text{(2.1)} & = & W_1(W_2\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\}) \\
 & & (d\{\!\{\mathsf{2}+\mathsf{w_2}\leftarrow U_0^{w_1+1}(e)\}\!\}\{\!\{\mathsf{1}\leftarrow U_0^{w_1+w_2+2}(c)\{\!\{\mathsf{1}+\mathsf{w_2}\leftarrow U_0^{w_1+1}(e)\}\!\}\}\!\}) \\
\text{(2.6)} & = & W_1(W_2\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\})(d\{\!\{\mathsf{1}\leftarrow U_0^{w_1+w_2+2}(c)\}\!\}\{\!\{\mathsf{1}+\mathsf{w_2}\leftarrow U_0^{w_1+1}(e)\}\!\}) \\
\text{(3.3 \& 3.4)} & = & W_1((W_2(d\{\!\{\mathsf{1}\leftarrow U_0^{w_1+w_2+2}(c)\}\!\}))\{\!\{\mathsf{1}\leftarrow U_0^{w_1+1}(e)\}\!\}) \\
\text{(IH)} & =_\beta & (e\delta)W_1(\lambda)W_2(d\{\!\{\mathsf{1}\leftarrow U_0^{w_1+w_2+2}(c)\}\!\}) \\
 & = & W(d\{\!\{\mathsf{1}\leftarrow U_0^{\#_\lambda(W)+1}(c)\}\!\})
\end{array}
$$

**Proof of Lemma 4** □

**Theorem 1 (Confluence of $\boldsymbol{\lambda g}$)** *The $\lambda g$-calculus is confluent.*

**Proof of Theorem 1** This proof is the de Bruijn version of the proof given in [KN95]. Let $a \twoheadrightarrow_{g\beta} b$, and $a \twoheadrightarrow_{g\beta} c$. By Lemma 4, $a =_\beta b$, and $a =_\beta c$; hence $b =_\beta c$. By confluence of $\beta$, $\exists d \in \Lambda$ where $b \twoheadrightarrow_\beta d$, and $c \twoheadrightarrow_\beta d$. By Remark 1, $b \twoheadrightarrow_{g\beta} d$, and $c \twoheadrightarrow_{g\beta} d$.

There are, as we mentioned in the introduction, various notions of generalized reduction. For other proofs of confluence of some of these notions, we refer the reader to [AFM$^+$95, dG93, Kam96, KW95b, Klo80].

**Proof of Theorem 1** □

Finally, the following ensures the good passage of $g\beta$-reduction through $\{\!\{\leftarrow\}\!\}$ and $U_k^i$:

**Lemma 5** *Let $a$, $b$, $c$, $d \in \Lambda$. The following hold:*

1. *If $c \rightarrow_{g\beta} d$, then $U_k^i(c) \rightarrow_{g\beta} U_k^i(d)$.*
2. *If $c \rightarrow_{g\beta} d$, then $a\{\!\{\mathsf{n}\leftarrow c\}\!\} \twoheadrightarrow_{g\beta} a\{\!\{\mathsf{n}\leftarrow d\}\!\}$.*
3. *If $a \rightarrow_{g\beta} b$, then $a\{\!\{\mathsf{n}\leftarrow c\}\!\} \rightarrow_{g\beta} b\{\!\{\mathsf{n}\leftarrow c\}\!\}$.*

**Proof of Lemma 5**

1. By induction on $c$. We only prove the base case where $c = (c_1\delta)W(\lambda)c_3$, $W$ is well balanced, and $d = W(c_3\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(c_1)\}\!\})$.

$$
\begin{array}{lll}
 & & U_k^i(c) \\
 & = & U_k^i((c_1\delta)W(\lambda)c_3) \\
(3.1) & = & (U_k^i(c_1)\delta)(U_k^i(W))(\lambda)U^i_{k+\#_\lambda(W)+1}(c_3) \\
(3.2) & \to_{g\beta} & (U_k^i(W))((U^i_{k+\#_\lambda(W)+1}(c_3))\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(U_k^i(c_1))\}\!\}) \\
(2.3) & = & (U_k^i(W))((U^i_{k+\#_\lambda(W)+1}(c_3))\{\!\{1 \leftarrow U^i_{k+\#_\lambda(W)}(U_0^{\#_\lambda(W)+1}(c_1))\}\!\}) \\
(2.5) & = & (U_k^i(W))U^i_{k+\#_\lambda(W)}(c_3\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(c_1)\}\!\}) \\
(3.1) & = & U_k^i(W(c_3\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(c_1)\}\!\})) \\
 & = & U_k^i(d)
\end{array}
$$

2. By induction on $a$ using point 1.

3. By induction on $a$. We only prove the base case: $a = (a_1\delta)W(\lambda)a_2$, and $b = W(a_2\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(a_1)\}\!\})$.

$$
\begin{array}{lll}
 & & a\{\!\{\mathsf{i} \leftarrow c\}\!\} \\
 & = & ((a_1\delta)W(\lambda)a_2)\{\!\{\mathsf{i} \leftarrow c\}\!\} \\
(3.3) & = & (a_1\{\!\{\mathsf{i} \leftarrow c\}\!\}\delta)(W\{\!\{\mathsf{i} \leftarrow c\}\!\})(\lambda)(a_2\{\!\{\mathsf{i} + \#_\lambda(W) + 1 \leftarrow c\}\!\}) \\
(3.4) & \to_{g\beta} & W\{\!\{\mathsf{i} \leftarrow c\}\!\}(a_2\{\!\{\mathsf{i} + \#_\lambda(W) + 1 \leftarrow c\}\!\}\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(a_1\{\!\{\mathsf{i} \leftarrow c\}\!\})\}\!\}) \\
(2.4) & = & W\{\!\{\mathsf{i} \leftarrow c\}\!\}(a_2\{\!\{\mathsf{i} + \#_\lambda(W) + 1 \leftarrow c\}\!\}\{\!\{1 \leftarrow (U_0^{\#_\lambda(W)+1}(a_1))\{\!\{\mathsf{i} + \#_\lambda(W) \leftarrow c\}\!\}\}\!\}) \\
(2.6) & = & W\{\!\{\mathsf{i} \leftarrow c\}\!\}(a_2\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(a_1)\}\!\}\{\!\{\mathsf{i} + \#_\lambda(W) \leftarrow c\}\!\}) \\
(3.3) & = & (W(a_2\{\!\{1 \leftarrow U_0^{\#_\lambda(W)+1}(a_1)\}\!\}))\{\!\{\mathsf{i} \leftarrow c\}\!\} \\
 & = & b\{\!\{\mathsf{i} \leftarrow c\}\!\}
\end{array}
$$

**Proof of Lemma 5** □

# 3 The $\boldsymbol{\lambda s}$- and $\boldsymbol{\lambda gs}$-Calculi

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

A calculus accommodating explicit substitution via explicit rewrite rules in the $\lambda$-calculus was first presented in [KN93]. In that article, the intention was to introduce the philosophy in general, and the calculus obtained did not possess either confluence or preservation of strong normalization. In [KR95a], the part of the calculus that was confluent and preserved strong normalization was singled out. In this paper, we take that part ($\lambda s$) and extend it with generalized reduction. We start this section by presenting the $\lambda s$- and $\lambda gs$-calculi, and then by studying their properties.

The $\lambda s$-calculus is obtained by internalizing the meta operators of Definitions 6 and 7 in order to handle substitutions explicitly. Therefore, the syntax of the $\lambda s$-calculus is obtained by adding to $\Lambda$ 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 the index $j$ are to be substituted by the appropriately updated $b$, and
2. updating operators $\{\varphi^i_k\}_{\substack{k\geq 0\\ i\geq 1}}$, which are necessary for working with de Bruijn indices.

**Definition 12** *The set of terms of the $\lambda s$-calculus, denoted $\Lambda s$, is given as follows:*

$$\Lambda s ::= \mathbb{N} \mid (\Lambda s\,\delta)\Lambda s \mid (\lambda)\Lambda s \mid (\Lambda s\,\sigma^j)\Lambda s \mid (\varphi^i_k)\Lambda s$$

*where $j, i \geq 1$ and $k \geq 0$. We let $a$, $b$, and $c$ range over $\Lambda s$. A term $(a\,\sigma^j)b$ is called a* closure. *Furthermore, a term containing neither $\sigma$s nor $\varphi$s is called a* pure term. *The symbol $\Lambda$ denotes the set of pure terms. The set $\mathcal{DL}$ of $\delta\lambda$*-segments *is the set whose main items are either $\delta$-items or $\lambda$-items, i.e., $\mathcal{DL} ::= \phi \mid (\Lambda s\,\delta)\mathcal{DL} \mid (\lambda)\mathcal{DL}$. As usual, a reduction $\rightarrow$ on $\Lambda s$ is* compatible *if for all $a, b, c \in \Lambda s$, if $a \rightarrow b$, then $(a\,\delta)c \rightarrow (b\,\delta)c$, $(c\,\delta)a \rightarrow (c\,\delta)b$, $(\lambda)a \rightarrow (\lambda)b$, $(a\,\sigma^j)c \rightarrow (b\,\sigma^j)c$, $(c\,\sigma^j)a \rightarrow (c\,\sigma^j)b$, and $(\varphi^i_k)a \rightarrow (\varphi^i_k)b$.*

$$
\begin{array}{lrcl}
(\sigma\textit{-generation}) & (b\,\delta)(\lambda)a & \longrightarrow & (b\,\sigma^1)a \\
(\sigma\text{-}\lambda\textit{-transition}) & (b\,\sigma^j)(\lambda)a & \longrightarrow & (\lambda)(b\,\sigma^{j+1})a \\
(\sigma\text{-}\delta\textit{-transition}) & (b\,\sigma^j)(a_1\delta)a_2 & \longrightarrow & ((b\,\sigma^j)a_1\delta)\,(b\,\sigma^j)a_2 \\
(\sigma\textit{-destruction}) & (b\,\sigma^j)\mathsf{n} & \longrightarrow & \begin{cases} \mathsf{n-1} & \text{if } n > j \\ (\varphi_0^j)b & \text{if } n = j \\ \mathsf{n} & \text{if } n < j \end{cases} \\
(\varphi\text{-}\lambda\textit{-transition}) & (\varphi_k^i)(\lambda)a & \longrightarrow & (\lambda)(\varphi_{k+1}^i)a \\
(\varphi\text{-}\delta\textit{-transition}) & (\varphi_k^i)(a_1\delta)a_2 & \longrightarrow & ((\varphi_k^i)a_1\delta)(\varphi_k^i)a_2 \\
(\varphi\textit{-destruction}) & (\varphi_k^i)\mathsf{n} & \longrightarrow & \begin{cases} \mathsf{n+i-1} & \text{if } n > k \\ \mathsf{n} & \text{if } n \leq k \end{cases}
\end{array}
$$

Figure 3: The $\lambda s$-calculus

**Definition 13** Items, segments, *and* well-balanced segments *for* $\Lambda s$ *are defined as follows:*

$$
\begin{aligned}
\mathcal{I}s &::= (\Lambda s\,\delta) \mid (\lambda) \mid (\Lambda s\,\sigma^j) \mid (\varphi_k^i) \\
\mathcal{S}s &::= \phi \mid \mathcal{I}s\,\mathcal{S}s \\
\mathcal{W}s &::= \phi \mid (\Lambda s\,\delta)\mathcal{W}s(\lambda) \mid \mathcal{W}s\,\mathcal{W}s
\end{aligned}
$$

*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 $(\varphi_k^i)$ a* $\sigma$-item *and a* $\varphi$-item, *respectively. The notion* $\text{len}(S)$ *is trivially extended to $S \in \mathcal{S}s$ in the obvious way, and $\#_\lambda(S)$ is extended by declaring that $\#_\lambda((a\,\sigma^j)S) = \#_\lambda(S)$ and $\#_\lambda((\varphi_k^i)S) = \#_\lambda(S)$.*

As the $\lambda s$-calculus updates and substitutes explicitly, we include a set of rules that are the equations in Definitions 6 and 7 oriented from left to right.

**Definition 14 ($\boldsymbol{\lambda s}$-Calculus)** *The* $\lambda s$-calculus *is the reduction system $(\Lambda s, \rightarrow_{\lambda s})$, where $\rightarrow_{\lambda s}$ is the least-compatible reduction on $\Lambda s$ generated by the rules given in Figure 3. We use $\lambda s$ to denote this set of rules.*

**Definition 15 ($\boldsymbol{s}$-Calculus)** *The* $s$-calculus, *the calculus of substitutions associated with the $\lambda s$-calculus, is the reduction system generated by the set of rules $s = \lambda s - \{(\sigma\textit{-generation})\}$.*

**Definition 16 (Notation for $s$-Normal Forms)** *We use $s(a)$ to denote the s-normal form of $a$. Define $s(S)$ for a $\delta\lambda$-segment by $s(\phi) = \phi$, and $s((a\,\delta)S) = (s(a)\,\delta)s(S)$, and $s((\lambda)S) = (\lambda)s(S)$.*

**Definition 17 ($\lambda gs$-Calculus)** *The $\lambda gs$-*calculus *is the calculus whose set of rules consists of $\lambda gs$ where $\lambda gs = \lambda s \cup \{(g\sigma\text{-generation})\}$ and (we write w.b. for "well balanced"):*

$$(g\sigma\text{-}generation): \qquad (b\,\delta)W(\lambda)a \longrightarrow W((\varphi_0^{\#_\lambda(W)+1})b\,\sigma^1)a$$

*where $W \neq \phi$ is w.b.*

Note that in the $\lambda gs$-calculus we do not merge ($\sigma$-*generation*) and ($g\sigma$-*generation*) into the following:

$$(new\ g\sigma\text{-}generation): \qquad (b\,\delta)W(\lambda)a \longrightarrow W((\varphi_0^{\#_\lambda(W)+1})b\,\sigma^1)a$$

where $W$ is w.b. The reason for this lies in the fact that (*new $g\sigma$-generation*) does not generalize ($\sigma$-*generation*) of the $\lambda s$-calculus. That is, $(b\delta)(\lambda)a \to_{\sigma\text{-}gen} (b\sigma^1)a$, yet $(b\delta)(\lambda)a \to_{new\ g\sigma\text{-}gen} ((\varphi_0^1 b)\sigma^1)a$.

The ($\sigma$-*generation*) rule starts the simulation of a $\beta$-reduction by generating a substitution operator ($\sigma^1$). The ($\sigma$-$\delta$-*transition*) and ($\sigma$-$\lambda$-*transition*) rules propagate copies of this operator throughout the term until they arrive at the variable occurrences. If a variable should be affected by the substitution, the ($\sigma$-*destruction*) rule (case $j = n$) carries out the substitution by the updated term, thus introducing the updating operators. Finally, the $\varphi$-*rules* compute the updating.

We state now the following theorem of the $\lambda s$-calculus.

**Theorem 2** *The following hold:*

1. *The s-calculus is strongly normalizing and confluent on $\Lambda s$.*
2. *All s-normal forms are unique.*
3. *The set of s-normal forms is exactly $\Lambda$.*
4. *For every $a, b \in \Lambda s$, the following hold:*
   (a) $s((a\,\delta)b) = (s(a)\,\delta)s(b)$

*(b)* $s((\lambda)a) = (\lambda)(s(a))$

*(c)* $s((\varphi^i_k)a) = U^i_k(s(a))$

*(d)* $s((b\,\sigma^j)a) = s(a)\{\!\{\mathsf{j} \leftarrow s(b)\}\!\}$

**Proof of Theorem 2**

1. We define recursively a weight function $W$:
   $W(\mathsf{n}) = 1$ $\quad W((a\delta)b) = W(a) + W(b) + 1$
   $W((\varphi^i_k)a) = 2W(a)$ $\quad W((b\,\sigma^j)a) = 2W(a)(W(b) + 1)$
   $W((\lambda)a) = W(a) + 1$

   It is easy to show by induction on $a$ that $a \to_s b$ implies $W(a) > W(b)$; hence the $s$-calculus is strongly normalizing.

   As for confluence, note first that the reduction $\to_s$ is locally confluent because there are no critical pairs and the theorem of Knuth-Bendix applies trivially. Finally, Newman's lemma (see Lemma 1) guarantees confluence.

2. The existence and unicity of $s$-normal forms ($s$-nf) is guaranteed by point 1.

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

4. Cases a and b hold because there is no $s$-rule whose left-hand side is an application or an abstraction. Case c is shown as follows: first show the equality for terms in $s$-nf, i.e., use an inductive argument on $c \in \Lambda$ to show $s((\varphi^i_k)c) = U^i_k(s(c))$. Then let $a \in \Lambda s$, $s((\varphi^i_k)a) = s((\varphi^i_k)s(a)) = U^i_k(s(s(a))) = U^i_k(s(a))$. Case d is shown similarly to (and using) Case c.

**Proof of Theorem 2** $\square$

**Lemma 6** *Let $a, b \in \Lambda s$. Then both of these statements hold:*

- *if $a \to_{\sigma\text{-}gen} b$, then $s(a) \twoheadrightarrow_\beta s(b)$; and*
- *if $a \to_{g\sigma\text{-}gen} b$, then $s(a) \twoheadrightarrow_{g\beta} s(b)$.*

**Proof of Lemma 6** The first claim is proved by induction on $a$ using Lemma 5 and Theorem 2. For the second claim, we need the following additional argument. Observe that for any $\delta\lambda$-segment $S$, it holds that $s(S\,a) = s(S)s(a)$. Then note that if $W$ is well balanced, then it is a $\delta\lambda$-segment, and thus $s(W\,a) = s(W)s(a)$.

**Proof of Lemma 6** $\square$

**Corollary 1** *Let $a, b \in \Lambda s$. Then both of these statements hold:*

- *if $a \twoheadrightarrow_{\lambda s} b$, then $s(a) \twoheadrightarrow_{\beta} s(b)$; and*
- *if $a \twoheadrightarrow_{\lambda gs} b$, then $s(a) \twoheadrightarrow_{g\beta} s(b)$.*

**Corollary 2 (Conservative Extension)** *Let $a, b \in \Lambda$. Then both of these statements hold:*

- *if $a \twoheadrightarrow_{\lambda s} b$, then $a \twoheadrightarrow_{\beta} b$; and*
- *if $a \twoheadrightarrow_{\lambda gs} b$, then $a \twoheadrightarrow_{g\beta} b$.*

This last corollary says that the $\lambda(g)s$-calculus is correct with respect to the $\lambda(g)$-calculus, i.e., if a $\lambda(g)s$-reduction sequence begins and ends with pure terms, there is a $\lambda(g)$-reduction sequence beginning and ending with the same terms.

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

**Lemma 7 (Simulation of $(g)\beta$-Reduction)** *Let $a, b \in \Lambda$. Then the following statements hold:*

- *if $a \rightarrow_{\beta} b$, then $a \rightarrow^{+}_{\lambda s} b$; and*
- *if $a \rightarrow_{g\beta} b$, then $a \rightarrow^{+}_{\lambda gs} b$.*

**Proof of Lemma 7** The first case is by induction on $a$. As usual, the interesting case is when $a = (\lambda c)d$ and $b = c\{\!\{1 \leftarrow d\}\!\}$. In this case, $(\lambda c)d \rightarrow_{\sigma-gen} c\sigma^1 d \twoheadrightarrow_s s(c\sigma^1 d) \stackrel{T2}{=} s(c)\{\!\{1 \leftarrow s(d)\}\!\} \stackrel{c,d\in\Lambda}{=} c\{\!\{1 \leftarrow d\}\!\}$. The second case is by induction on $a$ using Theorem 2.

**Proof of Lemma 7** $\square$

**Corollary 3** *Let $a \in \Lambda$. The following hold:*

- *if $a$ is strongly normalizing in the $\lambda s$-calculus, then $a$ is strongly normalizing in the $\lambda$-calculus; and*
- *if $a$ is strongly normalizing in the $\lambda gs$-calculus, then $a$ is strongly normalizing in the $\lambda g$-calculus.*

We prove now the confluence of $\lambda s$ and $\lambda gs$ on ground terms. We remark that not even $\lambda s$ is confluent on open terms. As a matter of fact, to obtain confluence on open terms, certain rules must be added. The calculus thus obtained, $\lambda s_e$ has been shown confluent (cf. [KR97]). The combination of $\lambda s_e$ with generalized reduction has not yet been studied.

**Theorem 3 (Confluence of $\boldsymbol{\lambda s}$ and $\boldsymbol{\lambda gs}$)** *The $\lambda s$ and $\lambda gs$-calculi are confluent on $\Lambda s$.*

**Proof of Theorem 3** We use the interpretation method (cf. [Har89, CHL96]). To prove confluence of the $\lambda s$-calculus, remove each $(g)$ from the proof below. For the confluence of the $\lambda gs$-calculus, leave each $(g)$ but remove the parentheses that embrace the $g$s. The proof goes as follows:

We interpret the $\lambda(g)s$-calculus into the $\lambda(g)$-calculus via $s$-normalization:

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

The existence of the arrows $s(a) \twoheadrightarrow_{(g)\beta} s(b)$ and $s(a) \twoheadrightarrow_{(g)\beta} s(c)$ is guaranteed by Corollary 1. We can close the diamond thanks to the confluence of the $\lambda(g)$-calculus. Finally, Lemma 7 ensures $s(b) \twoheadrightarrow_{\lambda(g)s} d$ and $s(b) \twoheadrightarrow_{\lambda(g)s} d$, proving thus the confluence for the $\lambda(g)s$-calculus.

**Proof of Theorem 3** $\square$

# 4 Preservation of Strong Normalization

We show in this section that the $\lambda s$-calculus preserves the $\lambda$-calculus strong normalization and that the $\lambda gs$-calculus preserves the $\lambda g$-calculus strong normalization.

The technique used in this section to prove preservation of strong normalization (PSN) is an adaptation of the minimal derivation method used in [BBLRD96] to prove PSN for $\lambda\upsilon$ and in [KR95a] to prove PSN for $\lambda s$. Our proof includes the first proof of the commutation of arbitrary external and internal reduction. Moreover, we give an inductive and elegant definition of internal/external reduction, instead of the one that depends on internal and external positions as in [BBLRD96]. Finally, we introduce a syntactic notion of *skeletons* that will be very informative about internal and external reduction. The elegance of our presentation is reflected by the fact that one proof is enough to achieve both preservation results above.

**Notation 1** *We write $a \in \lambda$-SN, respectively, $a \in \lambda r$-SN, when $a$ is strongly normalizing in the $\lambda$-calculus (respectively in the $\lambda r$-calculus) for $r \in \{g, gs, s\}$. We write $a \underset{p}{\to} b$ to denote that $p$ is the position of the redex that is contracted. Therefore, $a \underset{\epsilon}{\to} b$ means that the reduction takes place at the root. We denote by $\prec$ the prefix order between positions in a term. Hence if $p$ and $q$ are positions in the term $a$ such that $p \prec q$, and we write $a_p$ (respectively $a_q$) for the subterm of $a$ at position $p$ (respectively $q$), then $a_q$ is a subterm of $a_p$.*

For example, if $a = ((4\delta)(\lambda)1\sigma^3)2$, we have $a_1 = 2$, $a_2 = (4\delta)(\lambda)1$, $a_{21} = (\lambda)1$, $a_{211} = 1$, and $a_{22} = 4$. For example, since $2 \prec 21$, it must hold that $a_{21}$ is a subterm of $a_2$.

The following three lemmas assert that every $\sigma$ in the last term of a reduction sequence beginning with a $\lambda$-term must have been created at some previous step by a (generalized) ($(g)\sigma$-*generation*), and trace the history of these closures. The first lemma deals with one-step reduction where the redex is at the root; the second generalizes the first; and the third treats arbitrary reduction sequences.

**Lemma 8** *Let $\to \in \{\to_{\lambda s}, \to_{\lambda gs}\}$. If $a \underset{\epsilon}{\to} C[(e\,\sigma^i)d]$, then one of the following must hold:*

1. *$a = (e\,\delta)(\lambda)d$, $C = [\,\cdot\,]$, and $i = 1$;*

2. $\to = \to_{\lambda gs}$, $a = (e'\,\delta)W(\lambda)d$, $W \neq \phi$, $C = W[\,\cdot\,]$, $e = (\varphi_0^{\#_\lambda(W)+1})e'$, *and* $i = 1$; *or*

3. $a = C'[(e\,\sigma^j)d']$ *for some context* $C'$, *some term* $d'$, *and* $j \in \{i-1, i\}$.

**Proof of Lemma 8** Since the reduction is at the root, we must check for every rule $a \to a'$ that if $(e\,\sigma^i)d$ occurs in $a'$, then one of the three possibilities follows. We supply proofs only for the interesting rules:

($\sigma$-*generation*)**:** $a = (c\,\delta)(\lambda)b$, and $a' = (c\,\sigma^1)b$. If $(e\,\sigma^i)d$ matches $(c\,\sigma^1)d$, then rule 1 holds; else $(e\,\sigma^i)d$ must occur within $b$ or $c$, and hence rule 3 holds, with $j = i$ and $d' = d$.

($g\sigma$-*generation*)**:** Occurs only if $\to = \to_{\lambda gs}$. $a = (c\,\delta)W(\lambda)b$, $W \neq \phi$, and $a' = W((\varphi_0^{\#_\lambda(W)+1})c\,\sigma^1)b$. If $(e\,\sigma^i)d$ is $((\varphi_0^{\#_\lambda(W)+1})c\,\sigma^1)d$, then rule 2 holds; else $(e\,\sigma^i)d$ occurs in $b$, $c$, or $W$, hence rule 3 holds, with $j = i$, and $d' = d$.

($\sigma$-$\lambda$-*transition*)**:** $a = (c\,\sigma^h)(\lambda)b$, and $a' = (\lambda)(c\,\sigma^{h+1})b$. If $(e\,\sigma^i)d$ matches $(c\,\sigma^{h+1})b$, then rule 3 holds, with $j = i-1$, and $d' = (\lambda)d$; else $(e\,\sigma^i)d$ occurs in $b$ or $c$, hence rule 3 holds, with $j = i$, and $d' = d$.

**Proof of Lemma 8** $\square$

**Lemma 9** *Let* $\to \in \{\to_{\lambda s}, \to_{\lambda gs}\}$. *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. $\to = \to_{\lambda gs}$, $a = C'[(e'\,\delta)W(\lambda)d]$, $C = C'[W[\,\cdot\,]]$, $e = (\varphi_0^{\#_\lambda(W)+1})e'$, *and* $i = 1$; *or*

3. $a = C'[(e'\,\sigma^j)d']$ *where* $e' = e$ *or* $e' \to e$ *and* $j \in \{i-1, i\}$.

**Proof of Lemma 9** Induction on $a$, using Lemma 8 for the reductions at the root.

**Proof of Lemma 9** $\square$

**Lemma 10** *Let* $\to \in \{\to_{\lambda s}, \to_{\lambda gs}\}$. *Let* $a_1 \to \cdots \to a_n \to a_{n+1} = C[(e\,\sigma^i)d]$. *There exist* $e', d' \in \Lambda s$, *and a context* $C'[\,\cdot\,]$ *such that* $e' \twoheadrightarrow e$ *and one of the following holds:*

1. $a_k = C'[(e'\,\delta)(\lambda)d']$ *and* $a_{k+1} = C'[(e'\,\sigma^1)d']$ *for some* $k \le n$;

2. $\to=\to_{\lambda gs}$, $a_k = C'[(e'\,\delta)W(\lambda)d']$, $a_{k+1} = C'[W(e''\,\sigma^1)d']$, *and* $e'' = (\varphi_0^{\#_\lambda(W)+1})e'$ *for* $k \le n$ *and w.b.* $W$; *or*

3. $a_1 = C'[(e'\,\sigma^j)d']$, *where* $j \le i$.

**Proof of Lemma 10** Induction on $n$ and use of Lemma 9.

**Proof of Lemma 10** □

We now define internal and external reductions. An internal reduction takes place somewhere at the left of a $\sigma^i$-operator. An external reduction is a noninternal one. Our definition is inductive rather than starting from the notions of internal and external position, as in [BBLRD96].

**Definition 18 (Internal Reduction)**
*For any notion of reduction* $r$, *the reduction* $\xrightarrow{\text{int}}_r$ *is defined by the following rules:*

$$\frac{a \longrightarrow_r b}{(a\,\sigma^i)c \xrightarrow{\text{int}}_r (b\,\sigma^i)c} \qquad \frac{a \xrightarrow{\text{int}}_r b}{(a\,\delta)c \xrightarrow{\text{int}}_r (b\,\delta)c} \qquad \frac{a \xrightarrow{\text{int}}_r b}{(c\,\delta)a \xrightarrow{\text{int}}_r (c\,\delta)b}$$

$$\frac{a \xrightarrow{\text{int}}_r b}{(\lambda)a \xrightarrow{\text{int}}_r (\lambda)b} \qquad \frac{a \xrightarrow{\text{int}}_r b}{(c\,\sigma^i)a \xrightarrow{\text{int}}_r (c\,\sigma^i)b} \qquad \frac{a \xrightarrow{\text{int}}_r b}{(\varphi_k^i)a \xrightarrow{\text{int}}_r (\varphi_k^i)b}$$

*Therefore,* $\xrightarrow{\text{int}}_r$ *is the least-compatible relation closed under* $\dfrac{a \longrightarrow_r b}{(a\,\sigma^i)c \xrightarrow{\text{int}}_r (b\,\sigma^i)c}$.

**Remark 2** *By inspecting the inference rules, one can check that:*

1. *If* $a \xrightarrow{\text{int}}_r (\lambda)b$, *then* $a = (\lambda)c$, *and* $c \xrightarrow{\text{int}}_r b$.

2. *If* $a \xrightarrow{\text{int}}_r (c\,\delta)b$, *then* $a = (e\,\delta)d$, *and ((*$d \xrightarrow{\text{int}}_r b$ *and* $e = c$*) or (*$e \xrightarrow{\text{int}}_r c$ *and* $d = b$*)).*

3. *If* $a \xrightarrow{\text{int}}_r W(\lambda)b$ *with* $W$ *well balanced, then one of the following holds:*

   - $a = W(\lambda)b'$ *with* $b' \xrightarrow{\text{int}}_r b$, *or*

- $W = W_1(b_1'\delta)W_2(\lambda)W_3$ *where* $W_1$, $W_2$, *and* $W_3$ *are well balanced,* $a = W_1(b_1\delta)W_2(\lambda)W_3(\lambda)b$, *and* $b_1 \xrightarrow{\mathsf{int}}_r b_1'$.

4. $a \xrightarrow{\mathsf{int}}_r \mathsf{n}$ *is impossible.*

**Definition 19 (External Reduction)** *For any notion of reduction* $r$, *the reduction* $\xrightarrow{\mathsf{ext}}_r$ *is defined by induction. The axioms are the rules of* $r$, *and the inference rules are the following:*

$$\frac{a \xrightarrow{\mathsf{ext}}_r b}{(a\,\delta)c \xrightarrow{\mathsf{ext}}_r (b\,\delta)c} \qquad \frac{a \xrightarrow{\mathsf{ext}}_r b}{(c\,\delta)a \xrightarrow{\mathsf{ext}}_r (c\,\delta)b}$$

$$\frac{a \xrightarrow{\mathsf{ext}}_r b}{(\lambda)a \xrightarrow{\mathsf{ext}}_r (\lambda)b} \qquad \frac{a \xrightarrow{\mathsf{ext}}_r b}{(c\,\sigma^i)a \xrightarrow{\mathsf{ext}}_r (c\,\sigma^i)b} \qquad \frac{a \xrightarrow{\mathsf{ext}}_r b}{(\varphi^i_k)a \xrightarrow{\mathsf{ext}}_r (\varphi^i_k)b}$$

Note that the potential rule $\dfrac{a \xrightarrow{\mathsf{ext}}_r b}{(a\,\sigma^i)c \xrightarrow{\mathsf{ext}}_r (b\,\sigma^i)c}$ is excluded from the definition of external reduction. Thus, as expected, external reductions will not occur at the left of a $\sigma^i$-operator. This enables us to write $\to^+{}_\beta$ instead of $\twoheadrightarrow_\beta$ in the following proposition (compare with Lemma 6).

**Proposition 1** *Let* $a$, $b \in \Lambda s$. $a \xrightarrow{\mathsf{ext}}_{\sigma\text{-}gen} b \Rightarrow s(a) \to^+{}_\beta s(b)$ , *and* $a \xrightarrow{\mathsf{ext}}_{g\sigma\text{-}gen} b \Rightarrow s(a) \to^+{}_{g\beta}\, s(b)$ .

**Proof of Proposition 1** By induction on $a$ (as in Lemma 6). Note that when $a = (d\,\sigma^i)c$, 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 5.2.

**Proof of Proposition 1** □

The following is needed in Lemma 12 and hence in the preservation theorem. Note that we depart from the traditional *minimal derivation method* (which we call here the *minimal reduction-sequence method*) which commutes internal $\lambda r$-steps and external $s$-steps and assumes that $s(a)$ is $\lambda$-SN and that $s(a) = s(b)$. Instead, we commute arbitrary internal and external reduction, and drop the extra assumptions concerning SN and the $s$-normal forms. Our generality enables us to simplify the proof of the commutation lemma (no need to always check during the induction that terms are $\lambda$-SN, and evaluate the $s$-normal forms). Moreover, our commutation of arbitrary internal and external reduction simplifies Lemma 12, which is needed in the proof of PSN. In particular, Lemma 12 drops the condition that the term is strongly normalizing, and its proof is very simple.

**Lemma 11 (Commutation of Internal/External Reduction)** *Let $a, b \in \Lambda s$, and $r \in \{s, gs\}$. If $a \xrightarrow{\mathsf{int}}_{\lambda r} \cdot \xrightarrow{\mathsf{ext}}_{\lambda r} b$, then $a \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda r} \cdot \xrightarrow{\mathsf{int}}\!\!\!\!\to_{\lambda r} b$.*

**Remark 3** *The relation $\xrightarrow{\mathsf{ext}}{}^{+}_{\lambda r}$ can represent one or two steps. The two-step use is necessary when the internal step changes an external redex from $(a\,\sigma^n)\mathsf{n}$ to $(a'\sigma^n)\mathsf{n}$ and the external step uses ($\sigma$-destruction) to destroy this redex, producing $\varphi_0^n a'$. The relation $\xrightarrow{\mathsf{int}}\!\!\!\!\to_{\lambda r}$ can represent zero, one, or two steps. The zero-step case is necessary when the two-step case of $\xrightarrow{\mathsf{ext}}{}^{+}_{\lambda r}$ occurs (already mentioned), or when the internal step changes an external redex from $(a\,\sigma^i)\mathsf{n}$ to $(a'\sigma^i)\mathsf{n}$ and $i \neq n$, and the external step uses ($\sigma$-destruction) to destroy this redex, discarding the subterm $a'$. The two-step case happens when the internal step changes an external redex from $(a\;\sigma^i)(b\,\delta)c$ to $(a'\,\sigma^i)(b\,\delta)c$, and the external step uses the ($\sigma$-$\delta$-transition) rule to duplicate the $\sigma$-item, producing $((a'\,\sigma^i)b\,\delta)(a'\,\sigma^i)c$.*

**Proof of Lemma 11** By induction on $a$, analyzing the positions of the redexes. We give the proof for $r = gs$. The basic case, which is $a = \mathsf{n}$, is trivial.

$a = (a_2\delta)a_1$**:** Since we are dealing with an internal reduction, there are only two possibilities: $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$, or $a_2 \xrightarrow{\mathsf{int}}_{\lambda gs} a_2'$. Let us study, for instance, the first one. There are four cases:

- $a = (a_2\delta)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\delta)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2\delta)a_1''$, and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} a_1''$. Therefore, by induction hypothesis, $a_1 \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda gs} \cdot \xrightarrow{\mathsf{int}}\!\!\!\!\to_{\lambda gs} a_1''$, and then $(a_2\delta)a_1 \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda gs} \cdot \xrightarrow{\mathsf{int}}\!\!\!\!\to_{\lambda gs} (a_2\delta)a_1''$.
- $a = (a_2\delta)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\delta)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2'\delta)a_1'$, with $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$ and $a_2 \xrightarrow{\mathsf{ext}}_{\lambda gs} a_2'$. We can simply commute the reductions: $a = (a_2\delta)a_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2'\delta)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2'\delta)a_1'$.
- $a = (a_2\delta)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\delta)(\lambda)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2\sigma^1)a_1'$, with $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (\lambda)a_1'$. Hence, by Remark 2, $a_1 = (\lambda)b_1$ with $b_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$. Now, $(a_2\delta)a_1 = (a_2\delta)(\lambda)b_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2\sigma^1)b_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\sigma^1)a_1'$.

- $a = (a_2\delta)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\delta)W(\lambda)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} W((\varphi^{\#_\lambda(W)+1})a_2\sigma^1)a_1'$, with $W$ well balanced and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} W(\lambda)a_1'$. Hence, by Remark 2.3, there are two cases:
  - case $a_1 = W(\lambda)b_1$, where $b_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$. Then $a = (a_2\delta)W(\lambda)b_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} W((\varphi^{\#_\lambda(W)+1})a_2\sigma^1)b_1 \xrightarrow{\mathsf{int}}_{\lambda gs}$, and $W((\varphi^{\#_\lambda(W)+1})a_2\sigma^1)a_1'$, and
  - case $a_1 = W_1(b_1\delta)W_2(\lambda)W_3(\lambda)a_1'$ and $W = W_1(b_1'\delta)W_2(\lambda)W_3$, where $W_1$, $W_2$, and $W_3$, are well balanced, and $b_1 \xrightarrow{\mathsf{int}}_{\lambda gs} b_1'$. Then,

$$a = (a_2\delta)W_1(b_1\delta)W_2(\lambda)W_3(\lambda)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs}$$

$$W_1(b_1\delta)W_2(\lambda)W_3((\varphi^{\#_\lambda(W)+1})a_2\sigma^1)a_1' \xrightarrow{\mathsf{int}}_{\lambda gs}$$

$$W_1(b_1'\delta)W_2(\lambda)W_3((\varphi^{\#_\lambda(W)+1})a_2\sigma^1)a_1' = W((\varphi^{\#_\lambda(W)+1}a_2)\sigma^1)a_1'.$$

$a = (\lambda)a_1$**:** The reduction must take place within $a_1$, and we use the induction hypothesis.

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

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

- $a = (a_2\,\sigma^i)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\,\sigma^i)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2\,\sigma^i)a_1''$, and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} a_1''$. We can now apply the induction hypothesis to $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} a_1''$ to obtain $a_1 \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda gs} \cdot \overset{\mathsf{int}}{\twoheadrightarrow}_{\lambda gs} a_1''$, and hence $(a_2\,\sigma^i)a_1 \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda gs} \cdot \overset{\mathsf{int}}{\twoheadrightarrow}_{\lambda gs} (a_2\,\sigma^i)a_1''$.
- $a = (a_2\,\sigma^i)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2\,\sigma^i)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} b$, and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$. The external reduction takes place at the root. We study the three possible rules:
  - ($\sigma$-$\lambda$-*transition*): We have $a_1' = (\lambda)c'$, and $b = (\lambda)(a_2\sigma^{i+1})c'$. Remark 2.1 ensures that $a_1 = (\lambda)c$ and $c \xrightarrow{\mathsf{int}}_{\lambda gs} c'$. We can then commute: $a = (a_2\,\sigma^i)a_1 = (a_2\,\sigma^i)(\lambda)c \xrightarrow{\mathsf{ext}}_{\lambda gs} (\lambda)(a_2\sigma^{i+1})c \xrightarrow{\mathsf{int}}_{\lambda gs} (\lambda)(a_2\sigma^{i+1})c' = b$.

- ($\sigma$-$\delta$-*transition*): $a_1' = (c'\delta)d'$, and $b = ((a_2\,\sigma^i)c'\delta)(a_2\,\sigma^i)d'$. Remark 2.2 ensures that $a_1 = (c\delta)d$ and either $c \xrightarrow{\mathsf{int}}_{\lambda gs} c'$ and $d = d'$, or $d \xrightarrow{\mathsf{int}}_{\lambda gs} d'$ and $c = c'$. In both cases we can commute, as in the previous case.
- ($\sigma$-*destruction*): We have $a_1' = \mathsf{n}$, and this is impossible by Remark 2.4.

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

- $a = (a_2\,\sigma^i)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2'\,\sigma^i)a_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2'\,\sigma^i)a_1'$, and $a_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} a_1'$. We can commute to obtain:

$$a = (a_2\,\sigma^i)a_1 \xrightarrow{\mathsf{ext}}_{\lambda gs} (a_2\,\sigma^i)a_1' \xrightarrow{\mathsf{int}}_{\lambda gs} (a_2'\,\sigma^i)a_1'.$$

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

$$\begin{aligned} a = (a_2\,\sigma^i)a_1 &= (a_2\,\sigma^i)(\lambda)c \\ &\xrightarrow{\mathsf{ext}}_{\lambda gs} (\lambda)(a_2\sigma^{i+1})c \\ &\xrightarrow{\mathsf{int}}_{\lambda gs} (\lambda)(a_2'\sigma^{i+1})c = b \end{aligned}$$

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

$$\begin{aligned} a = (a_2\,\sigma^i)a_1 &= (a_2\,\sigma^i)(c\delta)d \\ &\xrightarrow{\mathsf{ext}}_{\lambda gs} ((a_2\,\sigma^i)c\delta)(a_2\,\sigma^i)d \\ &\xrightarrow{\mathsf{int}}_{\lambda gs} ((a_2'\,\sigma^i)c\delta)(a_2\,\sigma^i)d \\ &\xrightarrow{\mathsf{int}}_{\lambda gs} ((a_2'\,\sigma^i)c)(a_2'\,\sigma^i)d = b \end{aligned}$$

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

$$a = (a_2\,\sigma^i)a_1 = (a_2\,\sigma^i)\mathsf{n} \xrightarrow{\mathsf{ext}}_{\lambda gs} (\varphi_0^i)a_2 \xrightarrow{\mathsf{int}}_{\lambda gs} (\varphi_0^i)a_2' = b$$

If it is external, we get:

$$a = (a_2\,\sigma^i)a_1 = (a_2\,\sigma^i)\mathsf{n} \xrightarrow{\mathsf{ext}}_{\lambda gs} (\varphi_0^i)a_2 \xrightarrow{\mathsf{ext}}_{\lambda gs} (\varphi_0^i)a_2' = b$$

giving us $a \xrightarrow{\mathsf{ext}}{}^{+}_{\lambda gs} b \xrightarrow{\mathsf{int}}\!\!\!\!\rightarrow_{\lambda gs} b$.

$a = (\varphi_k^i)a_1$: Two possibilities exist, according to the position of the external reduction:

- $(\varphi_k^i)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (\varphi_k^i)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} (\varphi_k^i)a_1''$, and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} a_1''$. Use the induction hypothesis.
- $(\varphi_k^i)a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} (\varphi_k^i)a_1' \xrightarrow{\mathsf{ext}}_{\lambda gs} b$, and $a_1 \xrightarrow{\mathsf{int}}_{\lambda gs} a_1'$. The external reduction takes place at the root. Three rules are possible:
  - *($\varphi$-$\lambda$-transition)*: We have $a_1' = (\lambda)c'$, and $b = (\lambda)(\varphi_{k+1}^i)c'$. Remark 2.1 ensures that $a_1 = (\lambda)c$ and $c \xrightarrow{\mathsf{int}}_{\lambda gs} c'$. We can then commute:

    $$a = (\varphi_k^i)a_1 = (\varphi_k^i)(\lambda)c \xrightarrow{\mathsf{ext}}_{\lambda gs} (\lambda)(\varphi_{k+1}^i)c \xrightarrow{\mathsf{int}}_{\lambda gs} (\lambda)(\varphi_{k+1}^i)c' = b$$

  - *($\varphi$-$\delta$-transition)*: We have $a_1' = (c'\delta)d'$, and $b = ((\varphi_k^i c')\delta)(\varphi_k^i)d'$. Remark 2.2 ensures that $a_1 = (c\delta)d$, and either $c \xrightarrow{\mathsf{int}}_{\lambda gs} c'$ and $d = d'$, or $d \xrightarrow{\mathsf{int}}_{\lambda gs} d'$ and $c = c'$. In both cases, we can commute as in the previous case.
  - *($\varphi$-destruction)*: We have $a_1' = \mathsf{n}$, and this is impossible by Remark 2.4.

**Proof of Remark 3** □

**Lemma 12** *Let $a \in \Lambda s$, and $r \in \{s, gs\}$. For every infinite $\lambda r$-reduction sequence $a \rightarrow_{\lambda r} b_1 \rightarrow_{\lambda r} \cdots \rightarrow_{\lambda r} b_n \rightarrow_{\lambda r} \ldots$, one of these two possibilities holds:*

1. *there exists $N$ such that for $i \geq N$, it holds that $b_i \xrightarrow{\mathsf{int}}_{\lambda r} b_{i+1}$, i.e., all the reductions beyond the $N$th step are internal; or*
2. *there exists an infinite external $\lambda r$-reduction sequence:*

$$a \xrightarrow{\mathsf{ext}}_{\lambda r} c_1 \xrightarrow{\mathsf{ext}}_{\lambda r} \ldots \xrightarrow{\mathsf{ext}}_{\lambda r} c_n \xrightarrow{\mathsf{ext}}_{\lambda r} \ldots$$

**Proof of Lemma 12** Suppose there are an infinite number of external steps in the given reduction sequence. Then by repeated use of the commutation lemma (Lemma 11), we construct an infinite external reduction sequence starting from $a$. Otherwise, there is some $N$ such that all steps past the $N$th step are internal.

**Proof of Lemma 12** $\square$

To prove the preservation theorem (Theorem 4), we need two definitions.

**Definition 20** *Let $r \in \{s, gs\}$. An infinite $\lambda r$-reduction sequence $a_1 \to \cdots \to a_n \to \ldots$ is* minimal *if for every step $a_i \underset{p}{\to}_{\lambda r} a_{i+1}$, every other reduction sequence beginning with $a_i \underset{q}{\to}_{\lambda r} a'_{i+1}$ where $p \prec q$ is finite.*

The idea of a minimal reduction sequence is that at every step, it picks a redex as deeply nested as possible without preventing an infinite reduction. If one changes any one of its steps to rewrite a redex within a subterm of the original redex, then an infinite reduction sequence is impossible.

**Definition 21** *The syntax of skeletons and the* skeleton *of a term are defined as follows:*

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

$$\begin{array}{llll} Sk(\mathsf{n}) & = \mathsf{n} & Sk((a\,\delta)b) & = (Sk(a)\,\delta)Sk(b) \\ & & Sk((\lambda)a) & = (\lambda)Sk(a) \\ & & Sk((b\,\sigma^i)a) & = ([\,\cdot\,]\,\sigma^i)Sk(a) \\ & & Sk((\varphi^i_k)a) & = (\varphi^i_k)Sk(a) \end{array}$$

**Remark 4** *A definition of internal and external reduction equivalent to Definitions 18 and 18 is the following. Let $a, b \in \Lambda s$. Then define:*

$$\begin{array}{l} a \xrightarrow{\mathsf{int}}_r b \Leftrightarrow \; (a \to_r b \text{ and } Sk(a) = Sk(b)) \\ a \xrightarrow{\mathsf{ext}}_r b \Leftrightarrow \; (a \to_r b \text{ and } Sk(a) \neq Sk(b)) \end{array}$$

In other words, skeletons provide a syntax that is informative regarding what kind of $r$-reduction takes place. In particular, the following two properties hold:

1. each occurrence of $[\cdot]$ in $Sk(a)$ corresponds to an external closure of $a$ (i.e., a closure that is not at the left of any other closure), and this correspondence is a bijection; and
2. internal closures (those which are at the left of another closure) vanish in the skeleton.

**Theorem 4 (Preservation of Strong Normalization)** *Let $a \in \Lambda$. The following hold:*

1. *if $a$ is strongly normalizing in the $\lambda$-calculus, then $a$ is strongly normalizing in the $\lambda s$-calculus; and*
2. *if $a$ is strongly normalizing in the $\lambda g$-calculus, then $a$ is strongly normalizing in the $\lambda gs$-calculus.*

**Proof of Theorem 4** The proof of rule 1 is obtained by replacing, in the proof below, $\lambda g$ by $\lambda$ and $\lambda gs$ by $\lambda s$, and by dropping the second case given in Lemma 10. We prove rule 2.

Assume $a \in \lambda g$-SN, $a \notin \lambda gs$-SN, and take a minimal infinite $\lambda gs$-reduction sequence:

$$\mathcal{D} : a \rightarrow_{\lambda gs} a_1 \rightarrow_{\lambda gs} \cdots \rightarrow a_n \rightarrow_{\lambda gs} \ldots$$

Lemma 12 gives $N$ such that for $i \geq N$, $a_i \rightarrow_{\lambda gs} a_{i+1}$ is internal. (Note that case 2 of Lemma 12 cannot hold. Otherwise, by Proposition 1, there would be an infinite $\lambda g$-reduction sequence starting at $a$ and hence $a \notin \lambda g$-SN—contradiction.) By Remark 4, $Sk(a_i) = Sk(a_{i+1})$ 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 reduction sequence $\mathcal{D}'$ can be formed by taking steps from $\mathcal{D}$ such that all steps take place within a single 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:

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

Since $a$ is a pure term, Lemma 10 ensures the existence of $I \leq N$ such that one of the following holds:

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

2. $a_I = C'[(d'\,\delta)W(\lambda)c'] \to_{\lambda gs} a_{I+1} = C'[W((\varphi_0^{\#_\lambda(W)+1})d'\sigma^1)c']$ and $d' \twoheadrightarrow_{\lambda gs} d$.

Let us consider in the first and second cases, respectively, the following infinite $\lambda gs$-reduction sequences:

$$\mathcal{D}'' : a \twoheadrightarrow_{\lambda gs} a_I \twoheadrightarrow_{\lambda gs} C'[(d\delta)(\lambda)c'] \to_{\lambda gs} C'[(d_1\delta)(\lambda)c'] \to_{\lambda gs} \cdots \to_{\lambda gs} C'[(d_n\delta)(\lambda)c'] \to_{\lambda gs} \ldots$$

$$\mathcal{D}''' : a \twoheadrightarrow_{\lambda gs} a_I \twoheadrightarrow_{\lambda gs} C'[(d\delta)W(\lambda)c'] \to_{\lambda gs} C'[(d_1\delta)W(\lambda)c'] \to_{\lambda gs} \cdots \to_{\lambda gs} C'[(d_n\delta)W(\lambda)c'] \to_{\lambda gs} \ldots$$

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 $\mathcal{D}'$.

**Proof of Theorem 4** □

**Theorem 5** *For every $a \in \Lambda$, the following equivalences hold:*

$$a \in \lambda\text{-}SN \Leftrightarrow a \in \lambda s\text{-}SN$$

*and*

$$a \in \lambda g\text{-}SN \Leftrightarrow a \in \lambda gs\text{-}SN$$

**Proof of Theorem 5** By Lemma 3 and Theorem 4.

**Proof of Theorem 5** □

To complete the picture, we need to use a result of [Kam96]:

**Theorem 6** *Let $a \in \Lambda$. It holds that $a \in \lambda\text{-}SN \Leftrightarrow a \in \lambda g\text{-}SN$.*

**Corollary 4** *For every $a \in \Lambda$, the following equivalences hold:*

$$a \in \lambda g\text{-}SN \Leftrightarrow a \in \lambda\text{-}SN \Leftrightarrow a \in \lambda s\text{-}SN \Leftrightarrow a \in \lambda gs\text{-}SN$$

Note that the main preservation results that we show in this paper (Theorem 4) are concerned with substitution calculi. That is, we show that if $a \in \lambda r$-SN, then $a \in \lambda rs$-SN for $\lambda r \in \{\lambda, \lambda g\}$. What we do not show in this paper is the preservation result concerned with generalized reduction. That is, we do not prove $a \in \lambda$-SN $\Rightarrow a \in \lambda g$-SN; rather, we take the result of [Kam96]. The reason for this is that the minimal derivation method and even our adaptation of it are not suited to prove PSN for calculi that do not have explicit substitutions. In fact, the whole idea of internal and external reduction and of skeletons is based around substitutions. It is also fair to say that generalized reduction did not play any role in the proof of PSN (despite its role in proofs of SN as shown in [KW95b, Klo80, Ned73]).

# 5 The Typed $\lambda s$- and $\lambda gs$-Calculi

Our calculi of explicit substitutions $\lambda s$ and $\lambda gs$ possess a very nice property that other calculi of explicit substitutions do not possess: namely, the simply typed versions of $\lambda s$ and $\lambda gs$ are strongly normalizing. The $\lambda\sigma$-calculus of [GL97] does not possess this property, as is shown by Melliès in [Mel95], and only very recently has its weak normalization on open terms been shown to hold, in [GL97]. The simply typed $\lambda\upsilon$-calculus of [BBLRD96] is strongly normalizing, however, it is not confluent on open terms. In fact, our $\lambda s$- and $\lambda gs$-calculi are the first calculi of explicit substitutions whose simply typed versions are strongly normalizing (cf. [KR95b, KR96]) and which possess a confluent extension on open terms (we have shown the confluence of the extension of $\lambda s$ on open terms; although the extension for $\lambda gs$ on open terms has not yet been investigated, we believe that the details are similar to those for $\lambda s$).

In this section, we present the simply typed versions of $\lambda s$ and $\lambda gs$, and prove the strong normalization of the well-typed terms using the technique developed in [KR95b] to prove $\lambda s$-SN, which was 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 $\lambda$-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 22 ($\mathbf{\Lambda_t}$ and L1)** *The syntax for the simply typed $\lambda$-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 | $\Lambda_t ::= \mathbb{N} \mid (\Lambda_t\, \delta)\Lambda_t \mid (\mathcal{T}\, \lambda)\Lambda_t$ |

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

$$\begin{array}{ll}
(\mathrm{L1}\text{-}\mathit{var}) & A, E \vdash \mathsf{1} : A \\
(\mathrm{L1}\text{-}\mathit{varn}) & \dfrac{E \vdash \mathsf{n} : B}{A, E \vdash \mathsf{n+1} : B} \\
(\mathrm{L1}\text{-}\mathit{abs}) & \dfrac{A, E \vdash b : B}{E \vdash (A\,\lambda)b : A \to B} \\
(\mathrm{L1}\text{-}\mathit{app}) & \dfrac{E \vdash b : A \to B, \quad E \vdash a : A}{E \vdash (a\,\delta)b : B}
\end{array}$$

Before presenting the simply typed $\lambda s$-calculus and $\lambda sg$-calculus, we introduce the following notation concerning environments. If $E$ is the environment $A_1, A_2, \ldots, A_n$, we shall use the notation $E_{\geq i}$ for the environment $A_i, A_{i+1}, \ldots, A_n$. Analogously, $E_{\leq i}$ stands for $A_1, \ldots, A_i$. The notations $E_{<i}$ and $E_{>i}$ are defined similarly.

**Definition 23 ($\Lambda s_t$ and Ls1)** *The syntax for the simply typed $\lambda s$-terms is given as follows:*

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

*Types and environments are as above. The typing rules of the system* Ls1 *are as follows. The rules* Ls1-*var,* Ls1-*varn,* Ls1-abs*, and* Ls1-*app are exactly the same as* L1-*var,* L1-*varn,* L1-abs*, and* L1-*app, respectively. The new rules are:*

$$(\mathrm{Ls1}\text{-}\sigma) \quad \frac{E_{\geq i} \vdash b : B, \quad E_{<i}, B, E_{\geq i} \vdash a : A}{E \vdash (b\,\sigma^i)a : A}$$

$$(\mathrm{Ls1}\text{-}\varphi) \quad \frac{E_{\leq k}, E_{\geq k+i} \vdash a : A}{E \vdash (\varphi^i_k)a : A}$$

*The reduction rules of the* simply typed $\lambda s$- *and* $\lambda sg$-calculi *are given by the same rules of the corresponding untyped versions, except that abstractions in the typed versions are marked with types.*

We say that $a : A$ is derivable in some type system $\mathrm{X} \in \{\mathrm{L1}, \mathrm{Ls1}\}$ from an environment $E$, notation $E \vdash_{\mathrm{X}} a : A$ if and only if $E \vdash a : A$ can be produced by the typing rules of the system X. We say that $a \in \Lambda s_t$ is *well* *typed* if there exists an environment $E$ and a type $A$ such that $E \vdash_{\mathrm{Ls1}} a : A$. The symbol $\Lambda s_{wt}$ denotes the set of well-typed terms.

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

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

To prove $\Lambda s_{wt} \subseteq \Xi$ (Proposition 2), we need to establish some useful results such as subject reduction, soundness of typing (i.e., Ls1 is a conservative extension of L1), and typing of subterms:

**Lemma 13** *Let $E$ be a type environment; $A$ and $B$ be types; and $a, b, c \in \Lambda s_t$. The following hold:*

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

**Proof of Lemma 13** We supply only the proof of the first item, since the others are similar.

$$\begin{array}{lllllll}
 & \multicolumn{1}{l}{\quad E \vdash ((\varphi_0^i)a\,\delta)(c\,\delta)(B\,\lambda)b : A} & & & & & \\
\text{iff} & \exists C.\,(E \vdash (c\,\delta)(B\,\lambda)b : C \to A & & & \textit{and} & E \vdash (\varphi_0^i)a : C) \\
\text{iff} & \exists C.\,(E \vdash (B\,\lambda)b : B \to (C \to A) & \textit{and} & E \vdash c : B & \textit{and} & E_{\geq i} \vdash a : C) \\
\text{iff} & \exists C.\,(B, E \vdash b : C \to A & \textit{and} & E \vdash c : B & \textit{and} & (B, E)_{\geq i+1} \vdash a : C) \\
\text{iff} & \exists C.\,(B, E \vdash b : C \to A & \textit{and} & E \vdash c : B & \textit{and} & B, E \vdash (\varphi_0^{i+1})a : C) \\
\text{iff} & \quad (B, E \vdash ((\varphi_0^{i+1})a\,\delta)b : A & \textit{and} & E \vdash c : B) & & \\
\text{iff} & \quad (E \vdash (B\,\lambda)((\varphi_0^{i+1})a\,\delta)b : B \to A & \textit{and} & E \vdash c : B) & & \\
\text{iff} & \quad E \vdash (c\,\delta)(B\,\lambda)((\varphi_0^{i+1})a\,\delta)b : A & & & &
\end{array}$$

**Proof of Lemma 13** $\square$

**Lemma 14** *Let $C$ be a context, and $a, b \in \Lambda s_t$. $E$ will range over type environments, and $A$ will range over types. The following holds:*

$$(\forall E, A.\ (E \vdash a : A \Leftrightarrow E \vdash b : A)) \Rightarrow (\forall E, A.\ (E \vdash C[a] : A \Leftrightarrow E \vdash C[b] : A))$$

**Proof of Lemma 14** By induction on $C$.

**Proof of Lemma 14** $\square$

The following lemma is necessary in the case of generalized reduction.

**Lemma 15 (Shuffle Lemma)** *Let $S$ be an arbitrary segment, $W$ a well-balanced segment, and $a, b \in \Lambda s_t$. Then:*

$$E \vdash S(a\,\delta)W\,b : A \text{ if and only if } E \vdash SW((\varphi_0^{\#_\lambda(W)+1})a\,\delta)\,b : A$$

**Proof of Lemma 15** By induction on $W$, using Lemmas 13 and 14. If $W = \phi$, it is immediate since $E' \vdash d : D$ if and only if $E' \vdash (\varphi_0^1)d : D$. Let us assume $W = (c\,\delta)U(B\,\lambda)V$, with $U$ and $V$ well balanced. The following equivalences hold using the indicated lemmas:

$$\begin{array}{lll}
 & & E \vdash S(a\,\delta)(c\,\delta)U(B\,\lambda)V\,b : A \\
\text{(IH)} & \text{iff} & E \vdash S(a\,\delta)U((\varphi_0^{\#_\lambda(U)+1})c\,\delta)(B\,\lambda)V\,b : A \\
\text{(IH)} & \text{iff} & E \vdash SU((\varphi_0^{\#_\lambda(U)+1})a\,\delta)((\varphi_0^{\#_\lambda(U)+1})c\,\delta)(B\,\lambda)V\,b : A \\
\text{(13.1, 14)} & \text{iff} & E \vdash SU((\varphi_0^{\#_\lambda(U)+1})c\,\delta)(B\,\lambda)((\varphi_0^{\#_\lambda(U)+2})a\,\delta)V\,b : A \\
\text{(IH, twice)} & \text{iff} & E \vdash S(c\,\delta)U(B\,\lambda)V((\varphi_0^{\#_\lambda(V)+1})(\varphi_0^{\#_\lambda(U)+2})a\,\delta)b : A \\
\text{(13.2, 14)} & \text{iff} & E \vdash S(c\,\delta)U(B\,\lambda)V((\varphi_0^{\#_\lambda(V)+\#_\lambda(U)+2})a\,\delta)b : A
\end{array}$$

**Proof of Lemma 15** □

**Lemma 16 (Subject Reduction)** *Let $r \in \{s, gs\}$. If $E \vdash_{\mathrm{Ls1}} a : A$ and $a \to_{\lambda r} b$, then $E \vdash_{\mathrm{Ls1}} b : A$.*

**Proof of Lemma 16** By induction on $a$. If the reduction is not at the root, use the induction hypothesis. If it is, check that for each rule $a \to b$ we have $E \vdash_{\mathrm{Ls1}} a : A$ implies $E \vdash_{\mathrm{Ls1}} b : A$. For the case of ($\sigma$-*generation*), use Lemma 13i, rule 3. For the case of ($g\sigma$-*generation*) and $r = gs$, if $E \vdash (a\,\delta)W(B\,\lambda)b : A$, then by Lemma 15, we have $E \vdash W((\varphi_0^{\#_\lambda(W)+1})a\,\delta)(B\,\lambda)b : A$, and by Lemma 13, rule 3, we conclude $E \vdash W((\varphi_0^{\#_\lambda(W)+1})a\,\sigma^1)b : A$. The other rules are proven by similar reasoning.

**Proof of Lemma 16** □

**Corollary 5** *Let $r \in \{s, gs\}$ and $E \vdash_{\mathrm{Ls1}} a : A$. If $a \twoheadrightarrow_{\lambda r} b$, then $E \vdash_{\mathrm{Ls1}} b : A$.*

**Lemma 17 (Typing of Subterms)** *If $a \in \Lambda s_{wt}$ and $b \triangleleft a$, then $b \in \Lambda s_{wt}$.*

**Proof of Lemma 17** By induction on $a$. If $b$ is not an immediate subterm of $a$, use the induction hypothesis. Otherwise, the last rule used to type $a$ must contain a premise in which $b$ is typed.

**Proof of Lemma 17** □

**Lemma 18 (Conservative Extension of Typing)**

$$\text{If } a \in \Lambda_t \text{ and } E \vdash_{\mathrm{Ls1}} a : A, \quad \text{then } E \vdash_{\mathrm{L1}} a : A$$

**Proof of Lemma 18** Easy induction on $a$.

**Proof of Lemma 18** □

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

**Proof of Proposition 2** Let $a \in \Lambda s_{wt}$, and $b$ be a subterm of $a$. By Lemma 17, $b \in \Lambda s_{wt}$, and by Corollary 5, $s(b) \in \Lambda s_{wt}$. Since $s(b) \in \Lambda$ (Theorem 2), Lemma l8 yields that $s(b)$ is L1-typable, and it is well known that classical typable $\lambda$-terms are strongly normalizing in the $\lambda$-calculus. Hence, $s(b) \in \lambda$-SN, and by preservation (Corollary 4), $s(b) \in \lambda g$-SN. Therefore $a \in \Xi$.

**Proof of Proposition 2** □

**Proposition 3** $\Xi \subseteq \lambda gs$*-SN.*

**Proof of Proposition 3** Suppose there exist $a' \in \Xi$ and $a' \notin \lambda gs$-SN. Then there must exist a term $a$ of minimal size such that $a \in \Xi$ and $a \notin \lambda gs$-SN. Let us consider a minimal infinite $\lambda gs$-derivation $\mathcal{D} : a \to a_1 \to \cdots \to a_n \to \ldots$, and follow the proof of Theorem 4 to obtain:

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

(Again, as we argued in Theorem 4, case 2 of Lemma 12 is discarded. This is because $a \in \Xi \Rightarrow s(a) \in \lambda g$-SN, and Proposition 1 would yield a contradiction if case 2 of Lemma 12 holds.) Now three possibilities arise from Lemma 10. Two of them have been considered in the proof of Theorem 4 and have contradicted the minimality of $\mathcal{D}$. Use the third one, that $a = C'[(d'\,\sigma^i)c']$ where $d' \twoheadrightarrow d$. Now we have $d' \twoheadrightarrow d \to d_1 \to \cdots \to d_n \to \ldots$. Since $d'$ is a subterm of $a$, it must be the case that $d' \in \Xi$, contradicting our choice of $a$ with minimal size.

**Proof of Proposition 3** □

Therefore we conclude, using Propositions 2 and 3 and Corollary 4:

**Theorem 7** *Every well-typed $\lambda s$-term is strongly normalizing in the $\lambda gs$-calculus.*

**Corollary 6** *Every well-typed $\lambda s$-term is strongly normalizing in the $\lambda s$-calculus.*

# 6 Conclusion

In this paper, we first explained the relevance and benefits of both generalized reduction and explicit substitution. We discussed alternatives and presented the research history behind the two concepts. We explained that they have never been combined, and we commented that the combination might indeed join both benefits, hence a $\lambda$-calculus extended with both needed to be studied.

Then we introduced $\lambda g$, the first system of generalized reduction using de Bruijn indices. We proved $\lambda g$ confluent, sound (i.e., it preserves $\beta$-equivalence), and complete (it properly contains $\beta$-reduction) with respect to the ordinary $\lambda$-calculus with de Bruijn indices.

Building on this success, we then introduced $\lambda gs$, the first system combining generalized reduction with explicit substitution. For this combination, we relied on the proven explicit substitution technology of the $\lambda s$-calculus. We proved $\lambda gs$ sound (a conservative extension) and complete (simulating $g\beta$-reduction) with respect to $\lambda g$. We proved that $\lambda gs$ preserves the strong normalization (PSN) of terms that are terminating in $\lambda g$, $\lambda s$, and the ordinary $\lambda$-calculus (all with de Bruijn indices). Our proof of PSN included the first proof of the commutation of arbitrary external and internal reductions, and is simpler than the standard proof of PSN using the traditional method of minimal derivations.

We proceeded further and added simple types to both $\lambda s$ and $\lambda gs$ (simultaneously) in the form of the type system Ls1. By proving the strong normalization (SN) of well-typed terms under $\lambda gs$-reduction (and therefore also under $\lambda s$-reduction), we have provided the first typed systems of explicit substitution and generalized reduction with the SN property. We proved that $\lambda gs$ (and therefore also $\lambda s$) has the essential property of subject reduction with respect to Ls1 typing. We also proved that typing for Ls1 is sound (a conservative extension) and complete (a proper extension) with respect to the simply typed $\lambda$-calculus, and that it has typing of subterms.

Now that a calculus combining both concepts has been shown to be theoretically correct, it would be interesting to extend our $\lambda gs$-calculus to one that is confluent on open terms, as is the tradition with calculi of explicit substitution. It would also be interesting to study the polymorphically (rather than the simply) typed version of $\lambda gs$. These are issues we are investigating at the moment. We are also investigating the correspondence of our calculus to methods that implement sharing, to test if the analysis of sharing given in [AFM+95] can be recast in an elegant fashion in our calculus.

**Acknowledgement of support:** This work is supported by EPSRC grants GR/K25014 and GR/L36963, and by NSF grant CCR-9417382.

# References

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

[AFM+95] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *Proceedings of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, New York, 1995. ACM.

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

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 2, pages 117–309. Oxford University Press, Oxford, 1992.

[BBLRD96] Z.-E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6, 1996.

[BKN96] R. Bloo, F. Kamareddine, and R. Nederpelt. The Barendregt cube with definitions and generalised reduction. *Information & Computation*, 126:123–143, May 1996.

[Blo95] R. Bloo. Preservation of strong normalisation for explicit substitution. Technical Report 95-08, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, Netherlands, 1995.

[CABC86] R. L. Constable, S. Allen, H. Bromely, and W. Cleveland. *Implementing Mathematics with the NUPRL Development System.* Prentice-Hall, Englewood Cliffs, NJ, 1986.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43:362–397, March 1996.

[Cur86] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Wiley, New York, 1986.

[dB78] N. G. de Bruijn. A name-free lambda calculus with facilities for internal definition of expressions and segments. Technical Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978.

[dG93] P. de Groote. The conservation theorem revisited. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 163–178, Berlin, March 1993. Springer-Verlag.

[GL97] Jean Goubault-Larrecq. A proof of weak termination of the simply typed $\lambda\sigma$-calculus. Technical Report 3090, INRIA, January 1997.

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

[Har89] T. Hardin. Confluence results for the pure strong categorical logic CCL: $\lambda$-calculi as subsystems of CCL. *Theoretical Computer Science*, 65:291–342, 1989.

[Hur96a] C. A. Muñoz Hurtado. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 440–447, Los Alomitos, 1996. IEEE.

[Hur96b] C. A. Muñoz Hurtado. Proof representation in type theory: State of the art. In *Proceedings, XXII Latin-American Conference of Informatics CLEI Panel '96*, June 1996.

[Jon87] S. Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[Kam96] F. Kamareddine. A reduction relation for which postponement of k-contractions, conservation, and preservation of strong normalisation hold. Technical Report TR-1996-11, University of Glasgow, Scotland, March 1996.

[Klo80] J. W. Klop. *Combinatory Reduction Systems.* PhD thesis, Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1980.

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

[KN95] F. Kamareddine and R. Nederpelt. Generalising reduction in the $\lambda$-calculus. *Journal of Functional Programming*, 5:637–651, 1995.

[KN96] F. Kamareddine and R. Nederpelt. A useful $\lambda$-notation. *Theoretical Computer Science*, 155:85–109, 1996.

[KR95a] F. Kamareddine and A. Ríos. A $\lambda$-calculus à la de Bruijn with explicit substitution. In *Proceedings of the 7th International Symposium of Programming Languages: Implementation, Logics and Programs PLILP '95*, volume 982 of *Lecture Notes in Computer Science*, pages 45–62, Berlin, 1995. Springer-Verlag.

[KR95b] F. Kamareddine and A. Ríos. The $\lambda s$-calculus: Its typed and its extended versions. Technical Report TR-95-13, Department of Computing Science, University of Glasgow, 1995.

[KR96] F. Kamareddine and A. Ríos. A generalised $\beta$-reduction and explicit substitutions. In *Proceedings of the 8th International Symposium of Programming Languages: Implementation, Logics and Programs, PLILP '96*, volume 1140 of *Lecture Notes in Computer Science*, pages 378–392, Berlin, 1996. Springer-Verlag.

[KR97] F. Kamareddine and A. Ríos. Extending a $\lambda$-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7:395–420, 1997.

[KTU94] A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. An analysis of ML typability. *Journal of the ACM*, 41:368–398, March 1994.

[KW94] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order $\lambda$-calculus. In *Proceedings of the 1994 ACM Conference on LISP Functional Programming*, New York, 1994. ACM.

[KW95a] A. J. Kfoury and J. B. Wells. Addendum to "New notions of reduction and non-semantic proofs of $\beta$-strong normalization in typed $\lambda$-calculi". Technical Report 95-007, Department of Computer Science, Boston University, 1995.

[KW95b] A. J. Kfoury and J. B. Wells. New notions of reduction and non-semantic proofs of $\beta$-strong normalization in typed $\lambda$-calculi. In *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 311–321, Los Alamitos, CA, 1995. IEEE.

[LM96] J. L. Lawall and H. Mairson. Optimality and inefficiency: What isn't a cost model of the lambda calculus? In *Proceedings of the 1996 ACM SIGPLAN International Conference on Functional Programming*, pages 92–101, New York, 1996. ACM.

[Mag95] L. Magnusson. *The implementation of ALF—a proof editor based on Martin-Löf's monomorphic type theory with explicit substitution.* PhD thesis, Chalmers University of Technology and Goteborg University, January 1995.

[Mel95] P.-A. Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate. In *Second International Conference on Typed Lambda Calculi and Applications*, Berlin, April 1995. Springer-Verlag.

[Ned73] R. P. Nederpelt. *Strong normalization for a typed lambda calculus with lambda structured types.* PhD thesis, Technische Hogeschool Eindhoven, 1973.

[New42] M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[Reg92] L. Regnier. *Lambda calcul et réseaux*. PhD thesis, Université de Paris VII, 1992.

[Reg94] L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126:281–292, 1994. (In French).

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

[SF93] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *LISP and Symbolic Computation*, 6:289, November 1993.

[Sør97] M. H. Sørensen. Strong normalization from weak normalization in typed $\lambda$-calculi. *Information and Computation*, 133(1):35–71, 25 February 1997.

[Vid89] D. Vidal. *Nouvelles notions de réduction en lambda-calcul*. PhD thesis, Université de Nancy, February 1989.

[Xi96] H. Xi. On weak and strong normalizations. Technical Report 96-187, Carnegie Mellon University, 1996.