# Comparing Calculi of Explicit Substitutions with Eta-reduction [1]

Mauricio Ayala-Rincón, Flávio L. C. de Moura [2,3]

*Departamento de Matemática*
*Universidade de Brasília*
*Brasília D.F., Brasil*

Fairouz Kamareddine [4]

*Computing and Electrical Engineering*
*Heriot-Watt University*
*Edinburgh, Scotland*

**Abstract**

The past decade has seen an explosion of work on calculi of explicit substitutions. Numerous work has illustrated the usefulness of these calculi for practical notions like the implementation of typed functional programming languages and higher order proof assistants. Three styles of explicit substitutions are treated in this paper: the $\lambda\sigma$ and the $\lambda s_e$ which have proved useful for solving practical problems like higher order unification, and the suspension calculus related to the implementation of the language $\lambda$Prolog. We enlarge the suspension calculus with an adequate eta-reduction which we show to preserve termination and confluence of the associated substitution calculus and to correspond to the eta-reductions of the other two calculi. Additionally, we prove that $\lambda\sigma$ and $\lambda s_e$ as well as $\lambda\sigma$ and the suspension calculus are non comparable while $\lambda s_e$ is more adequate than the suspension calculus.

**Keywords:** Calculi of explicit substitutions, lambda-calculi, eta reduction.

## 1 Introduction

Recent years have witnessed an explosion of work on expliciting substitutions [1,7,9,14,15,17,19] and on establishing its usefulness to computation: e.g., to

[1] Partially supported by the Brazilian CNPq research council grant number 47488101-6.
[2] First author partially suported by the FEMAT Brazilian foundation for research in mathematics, second author supported by the CAPES Brazilian foundation.
[3] Email: {ayala,flavio}@mat.unb.br
[4] Email: fairouz@cee.hw.ac.uk

automated deduction and theorem proving [24,25], to proof theory [31], to programming languages [8,20,23,26] and to higher order unification HOU [2,13]. This paper concentrates on three different styles of substitutions:

(i) The $\lambda\sigma$-style [1] which introduces two different sets of entities: one for terms and one for substitutions.

(ii) The suspension calculus [28,26], denoted $\lambda_{\mathrm{SUSP}}$, which introduces three different sets of entities: one for terms, one for environments and one for environment terms.

(iii) The $\lambda s$-style [19] which uses a philosophy of de Bruijn's *Automath* [29] elaborated in the new item notation [18]. The philosophy states that terms are built by applications (a function applied to an argument), abstraction (a function), substitution or updating. The advantages of this philosophy include remaining as close as possible to the familiar $\lambda$-calculus (cf. [18]).

The desired properties of explicit substitution calculi are a) simulation of $\beta$-reduction, b) confluence (CR) on closed terms, c) CR on open terms, d) strong normalization (SN) of explicit substitutions and e) preservation of SN of the $\lambda$-calculus. $\lambda\sigma$ satisfies a), b) and d), $\lambda s$ satisfies a)..e) but not c). $\lambda s$ has an extension $\lambda s_e$ for which a)..c) holds, but e) fails and d) is unknown. The suspension calculus satisfies a)..d), but e) is unknown. This paper deals with two useful notions for these calculi:

- Comparing the *adequacy* of their reduction process using the efficient simulation of $\beta$-reduction of [22].
- Extending the suspension calculus with eta-reduction resulting in $\lambda_{\mathrm{SUSP}}$. Eta-reduction for $\lambda\sigma$ was used in [13] to deal with HOU and was introduced in [2] for the same purpose in $\lambda s_e$.

It was shown in [22] that $\lambda s$ and $\lambda\sigma$ are non comparable. In this paper we prove that $\lambda s_e$ and $\lambda\sigma$ as well as $\lambda\sigma$ and $\lambda_{\mathrm{SUSP}}$ are non comparable and that $\lambda s_e$ is more adequate than the $\lambda_{\mathrm{SUSP}}$. Additionally, we show that $\lambda_{\mathrm{SUSP}}$ preserves confluence and SN of the substitution calculus associated with $\lambda_{\mathrm{SUSP}}$.

## 2 Preliminaries

We assume familiarity with $\lambda$-calculus (cf. [6]) and the notion of term algebra $\mathcal{T}(\mathcal{F},\mathcal{X})$ built on a (countable) set of variables $\mathcal{X}$ and a set of operators $\mathcal{F}$. Variables in $\mathcal{X}$ are denoted by $X, Y, ...$ and for a term $a \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $var(a)$ denotes the set of variables occurring in $a$. Throughout, we take $a, b, c, \ldots$ to range over terms.

Additionally, we assume familiarity with basic notions of rewriting as in [5]. In particular, for a *reduction relation* $R$ over a set $A$, we denote with $\overset{=}{\rightarrow}_R$ the **reflexive closure** of $R$, with $\rightarrow^*_R$ or just $\rightarrow^*$ the **reflexive and transitive closure** of $R$ and with $\rightarrow^+_R$ or just $\rightarrow^+$ the **transitive closure** of $R$. When $a \to^* b$ we say that there exists a **derivation** from $a$ to $b$. By $a \to^n b$, we mean that the derivation consists of $n$ steps of reduction and call $n$ the **length of the derivation**. Syntactical identity is denoted by $a = b$. For a reduction relation $R$ over $A$, $(A, \to_R)$, we use the standard definitions of **(locally-)confluent** or (weakly) Church Rosser **(W)CR**, normal forms and **strong** and **weak normalization/termination SN** and **WN**. Suppose $R$ is a SN reduction relation and let $t$ be a term, then $R$-nf($t$) denotes its normal form. As usual we use indiscriminately either "noetherian" or "terminating" instead of SN.

A **valuation** is a mapping from $\mathcal{X}$ to $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The homeomorphic extension of a valuation, $\theta$, from its domain $\mathcal{X}$ to the domain $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is called the **grafting** of $\theta$. As usual, valuations and their corresponding graftings are denoted by the same Greek letter. The application of a valuation $\theta$ or its corresponding grafting to a term $a \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ will be written in postfix notation $a\theta$. The **domain** of a grafting $\theta$, is defined by $Dom(\theta) = \{X \mid X\theta \neq X, X \in \mathcal{X}\}$. Its **range**, is defined by $Ran(\theta) = \cup_{X \in Dom(\theta)} var(X\theta)$. We let $var(\theta) = Dom(\theta) \cup Ran(\theta)$. For explicit representations of a valuation and its corresponding grafting $\theta$, we use the notation $\theta = \{X \mapsto X\theta \mid X \in Dom(\theta)\}$. Note that the notion of grafting, usually called first order substitution, corresponds to simple syntactic substitution without renaming.

Let $\mathcal{V}$ be a (countable) set of variables denoted by lowercase last letters of the Roman alphabet $x, y, \ldots$.

**Definition 2.1** *Terms $\Lambda(\mathcal{V})$, of the* **$\lambda$-calculus with names** *are inductively defined by $\Lambda(\mathcal{V}) ::= x \mid (\Lambda(\mathcal{V})\ \ \Lambda(\mathcal{V})) \mid \lambda_x.\Lambda(\mathcal{V})$, where $x \in \mathcal{V}$.*
*$\lambda_x.a$ and $(a\ \ b)$ are called* **abstraction** *and* **application** *terms, respectively.*

Terms in $\Lambda(\mathcal{V})$ are called *closed $\lambda$-terms* or terms without substitution meta-variables. An abstraction $\lambda_x.a$ represents a function of parameter $x$, whose body is $a$. Its application $(\lambda_x.a\ \ b)$ to an argument $b$, returns the value of $a$, where the formal parameter $x$ is replaced by $b$. This replacement of formal parameters with arguments is known as **$\beta$-reduction**. In the first order context of the term algebra $\mathcal{T}(\{\lambda_x.\_ \mid x \in \mathcal{V}\} \cup \{(\_\ \ \_)\}, \mathcal{V})$ and its first order substitution or grafting, $\beta$-reduction would be defined by $(\lambda_x.a\ \ b) \to a\{x \mapsto b\}$.

But in this context problems arise forcing the use of **$\alpha$-conversion** to rename bound variables:

(i) Let $\theta = \{x \mapsto b\}$. There are no semantic differences between the abstractions $\lambda_x.x$ and $\lambda_z.z$; both abstractions represent the identity function. But $(\lambda_x.x)\theta = \lambda_x.b$ and $(\lambda_z.z)\theta = \lambda_z.z$ are different.

(ii) Let $\theta = \{x \mapsto y\}$. $(\lambda_y.x)\theta = \lambda_y.y$ and $(\lambda_z.x)\theta = \lambda_z.y$, thus a capture is possible.

Consequently, $\beta$-reduction, should be defined in a way that takes care of renaming bound variables when necessary to avoid harmful capture of variables.

The $\lambda$-calculus usually considers substitution as an atomic operation leaving implicit the computational steps needed to effectively perform computational operations based on substitution such as matching and unification. In any real higher order deductive system, the substitution required by basic operations such as $\beta$-reduction should be implemented via smaller operations. Explicit substitution is an appropriate formalism for reasoning about the operations involved in real implementations of substitution. Since explicit substitution is closer to real implementations than to the classic $\lambda$-calculus, it provides a more accurate theoretical model to analyze essential properties of real systems (termination, confluence, correctness, completeness, etc.) as well as their time/space complexity. For further details of the importance of explicit substitution see [23,4].

$\alpha$-conversion should be performed before applying the substitution in the body of an abstraction. The grafting of a fresh variable avoids the possibility of capture. It is very important to remark that renaming selects fresh variables that have not been used previously. Moreover, since fresh variables are selected randomly, the result of the application of a substitution can be conceived as a *class* of equivalence terms.

**Definition 2.2** $\beta$**-reduction** *is the rewriting relation defined by the rewrite rule* $(\beta)$ *and* $\eta$**-reduction** *is the rewriting relation defined by the rewrite rule* $(\eta)$*, where:*

$$(\beta)\ \ (\lambda_x.a\ \ b) \to \{x/b\}(a) \qquad \textit{and} \qquad (\eta)\ \ \lambda_x.(a\ \ x) \to a,\ \textit{if}\, x \notin \mathcal{F}var(a)$$

$\mathcal{F}var(a)$ denotes the free variables occurring in $a$. Notice that our notion of substitution is not completely satisfactory because the idea of fresh variables is implicit and depends on the history of the renaming process.

Lambda terms with meta-variables or *open lambda terms* are given by the following.

**Definition 2.3** *Terms* $\Lambda(\mathcal{V}, \mathcal{X})$*, of the* $\lambda$**-calculus with names** *are inductively defined by:* $\Lambda(\mathcal{V}, \mathcal{X}) ::= x \mid X \mid (\Lambda(\mathcal{V}, \mathcal{X})\ \ \Lambda(\mathcal{V}, \mathcal{X})) \mid \lambda_x.\Lambda(\mathcal{V}, \mathcal{X})$*, where* $x \in \mathcal{V}$ *and* $X \in \mathcal{X}$*.*

We have seen that the names of bound variables and their corresponding abstractors play a semantically irrelevant role in the $\lambda$-calculus. So any term in $\Lambda(\mathcal{V})$ or $\Lambda(\mathcal{V}, \mathcal{X})$ can be seen as a syntactical representative of its obvious equivalence class. Hence, during syntactic unification, the role that names of bound variables and their corresponding abstractors play increases the complexity of the process and creates confusion.

Avoiding names in the $\lambda$-calculus is an effective way of clarifying the meaning of $\lambda$-terms and, for the unification process, of eliminating redundant renaming. De Bruijn developed in [12] a notation where names of bound variables are replaced by indices which relate these bound variables to their corresponding abstractors.

It is clear that the correspondence between an occurrence of a bound variable and its associated abstractor operator is uniquely determined by its *depth*, that is the number of abstractors between them. Hence, $\lambda$-terms can be written in a term algebra over the natural numbers $\mathbb{N}$, representing depth indices, the application operator $(\_\ \ \_)$ and a sole abstractor operator $\lambda\_$; i.e., $\mathcal{T}(\{(\_\ \ \_), \lambda\_\} \cup \mathbb{N})$.

In de Bruijn's notation, indexing the occurrences of free variables is given by a *referential* according to a fixed enumeration of the set of variables $\mathcal{V}$, say $x, y, z, \ldots$, and prefixing all $\lambda$-terms with $\ldots \lambda_z.\lambda_y.\lambda_x.\_$.

Now we can define the $\lambda$-calculus in de Bruijn notation with open terms or meta-variables.

**Definition 2.4** *The set $\Lambda_{dB}(\mathcal{X})$ of* **$\lambda$-terms in de Bruijn notation** *is defined inductively as: $\Lambda_{dB}(\mathcal{X}) ::= \underline{\mathtt{n}} \mid X \mid (\Lambda_{dB}(\mathcal{X})\ \ \Lambda_{dB}(\mathcal{X})) \mid \lambda\Lambda_{dB}(\mathcal{X})$, where $X \in \mathcal{X}$ and $n \in \mathbb{N} \setminus \{0\}$.*

$\Lambda_{dB}(\mathcal{X})$-terms without meta-variables are called closed lambda terms.

We write de Bruijn indices as $\underline{\mathtt{1}}, \underline{\mathtt{2}}, \underline{\mathtt{3}}, \ldots, \underline{\mathtt{n}}, \ldots$, to distinguish them from scripts. Since all considered calculi of explicit substitutions are built over the language of $\Lambda_{dB}(\mathcal{X})$, we will use $\Lambda$ to denote $\Lambda_{dB}(\mathcal{X})$.

Defining $\beta$-reduction in de Bruijn notation by $(\lambda a\ \ b) \to \{\underline{\mathtt{1}}/b\}a$ (where $\{\underline{\mathtt{1}}/b\}a$ is the substitution of the index 1 in $a$ with $b$) fails because:
• when eliminating the leading abstractor all indices associated with free variable occurrences in $a$ should be decremented;
• when propagating in $a$ the substitution $\{\underline{\mathtt{1}}/b\}$ through $\lambda$s, the indices of the substitution (initially $\underline{\mathtt{1}}$) and of the free variables in $b$ should be incremented.

Hence, we need new operators for detecting, incrementing and decrementing free variables which will be used in the definition of substitution.

**Definition 2.5** *Let $a \in \Lambda_{dB}(\mathcal{X})$. The* **$i$-lift** *of $a$, denoted $a^{+i}$ is defined inductively as follows:*

*1)* $X^{+i} = X$ *, for* $X \in \mathcal{X}$ *2)* $(a_1\ \ a_2)^{+i} = (a_1^{+i}\ \ a_2^{+i})$

*3)* $(\lambda a_1)^{+i} = \lambda a_1^{+(i+1)}$ *4)* $\underline{\mathtt{n}}^{+i} = \begin{cases} \underline{\mathtt{n}} + \underline{\mathtt{1}}, & \textit{if } n > i \\ \underline{\mathtt{n}}, & \textit{if } n \leq i \end{cases}$ *for* $n \in \mathbb{N}$.

The **lift** of a term $a$ is its 0-lift and is denoted briefly as $a^+$.

**Definition 2.6** *The application of the* **substitution** *with $b$ at the depth $n - 1, n \in \mathbb{N} \setminus \{0\}$, denoted $\{\underline{\mathtt{n}}/b\}a$, on a term $a$ in $\Lambda_{dB}(\mathcal{X})$ is defined inductively as follows:*

*1)* $\{\underline{\mathtt{n}}/b\}X = X$, *for* $X \in \mathcal{X}$ *2)* $\{\underline{\mathtt{n}}/b\}(a_1\ \ a_2) = (\{\underline{\mathtt{n}}/b\}a_1\ \ \{\underline{\mathtt{n}}/b\}a_2)$

*3)* $\{\underline{\mathtt{n}}/b\}\lambda a_1 = \lambda\{\underline{\mathtt{n+1}}/b^+\}a_1$ *4)* $\{\underline{\mathtt{n}}/b\}\underline{\mathtt{m}} = \begin{cases} \underline{\mathtt{m}} - \underline{\mathtt{1}}, & \textit{if } m > n \\ b, & \textit{if } m = n \\ \underline{\mathtt{m}}, & \textit{if } m < n \end{cases}$ *if* $m \in \mathbb{N}$.

**Definition 2.7** *The* $\beta$**-reduction** *in the* $\lambda$*-calculus with de Bruijn indices is defined as* $(\lambda a\ \ b) \rightarrow \{\underline{1}/b\}a$.

Observe that the rewriting system of the sole $\beta$-reduction rule is left-linear and non overlapping (i.e. orthogonal). Consequently, the rewriting system defined over $\Lambda_{dB}(\mathcal{X})$ by the $\beta$-reduction rule is CR.

In the $\lambda$-calculus with names, the $\eta$-reduction rule is defined as $\lambda_x.(a\ \ x) \rightarrow a$, if $x \notin \mathcal{F}var(a)$. In $\Lambda_{dB}(\mathcal{X})$, the left side of this rule is written as $\lambda(a'\ \ 1)$, where $a'$ stands for the corresponding translation of $a$ under some fixed referential of variables into the language of $\Lambda_{dB}(\mathcal{X})$. "$a$ has no free occurrences of $x$" means, in $\Lambda(\mathcal{X})$, that there are neither occurrences in $a'$ of the index 1 at height zero nor of the index 2 at height one nor of the index 3 at height two etc. This means, in general, that there exists a term $b$ such that $b^+ = a$.

**Definition 2.8** *The* $\eta$**-reduction** *in the* $\lambda$*-calculus with de Bruijn indices is defined as* $\lambda(a\ \ 1) \rightarrow b$ *if* $\exists b\ b^+ = a$.

## 3 Calculi à la $\lambda\sigma$, $\lambda s_e$ and $\lambda_{\mathbf{susp}}$

Rewriting systems for the $\lambda\sigma$ and the $\lambda s_e$-calculi including the eta-rule can be found either in [13] (for the $\lambda\sigma$) [2] (for both the $\lambda\sigma$ and $\lambda s_e$) or in the full version of this work (for the three calculi).

### *3.1 The* $\lambda\sigma$*-calculus*

The $\lambda\sigma$-calculus introduced in [1] works on 2-sorted terms: *(proper) terms*, and *substitutions*.

The rewriting system $\lambda\sigma$ is locally confluent [1], CR on substitution-closed terms (i.e., terms without substitution variables) [30] and not CR on open terms (i.e., terms with term and substitution variables) [11]. The possible forms of a $\lambda\sigma$-term in $\lambda\sigma$*-normal form* were given in [30].

### *3.2 The* $\lambda s_e$*-calculus*

The $\lambda s_e$-calculus of [21] is an extension of the $\lambda s$-calculus ([19]) which is CR on open terms and insists on remaining close to the syntax of the $\lambda$-calculus. Next to abstraction and application, substitution ($\sigma$) and updating ($\varphi$) operators are introduced. A term containing neither $\sigma$ nor $\varphi$ is called a *pure lambda term.* This calculus was originally introduced without the *Eta* rule that was added in [2] to deal with higher order unification problems as originally done in [13] for the $\lambda\sigma$-calculus.

The $\lambda s_e$-calculus has been proved in [21] to be CR on open terms; to simulate $\beta$-reduction: let $a, b \in \Lambda$, if $a \rightarrow_\beta b$ then $a \rightarrow^*_{\lambda s_e} b$; to be sound: let $a, b \in \Lambda$, if $a \rightarrow^*_{\lambda s_e} b$ then $a \rightarrow^*_\beta b$; and its associated substitution calculus, that is the $s_e$-calculus, to be WN and CR. The characterization of the $\lambda s_e$-normal forms was given in [21,2].

## *3.3 The suspension calculus*

The suspension calculus [28,26] deals with $\lambda$-terms as computational mechanisms. This was motivated by implementational questions related to $\lambda$Prolog, a logic programming language that uses typed $\lambda$-terms as data structures [27]. The suspension calculus works with three different types of entities:

$$\begin{array}{lll}
\text{SUSPENDED TERMS} & M,\ N & ::= \ \ Cons \mid \underline{\mathtt{n}} \mid \lambda M \mid (M\ N) \mid [\![M, i, j, e_1]\!] \\
\text{ENVIRONMENTS} & e_1,\ e_2 & ::= \ \ nil \mid et :: e_1 \mid \{\!\{e_1, i, j, e_2\}\!\} \\
\text{ENVIRONMENT TERMS} & et & ::= \ \ @i \mid (M, i) \mid \langle\!\langle et, i, j, e_1 \rangle\!\rangle
\end{array}$$

where *Cons* denotes any constant and $i, j$ are non negative natural numbers.

As constants and de Bruijn indices are suspended terms, the suspension calculus has open terms.

The suspension calculus owns a *generation* rule $\beta_s$, that initiates the simulation of a $\beta$-reduction (as for the $\lambda\sigma$ and the $\lambda s_e$, respectively, the *Beta* and the $\sigma$*-generation* rules do) and two sets of rules for handling the suspended terms. The first set, the $r$ rules, for reading suspensions and the second set, the $m$ rules, for merging suspensions are given in Table 1.

As in [28] we denote by $\rhd_{rm}$ the reduction relation defined by the $r$- and $m$-rules in Table 1. The associated substitution calculus, denoted as SUSP, is the one given by the congruence $=_{rm}$.

**Definition 3.1 ([28])** *The* length $len(e)$ *of an environment* $e$ *is given by:*
$len(nil) := 0$; $len(et :: e') := len(e') + 1$ *and*
$len(\{\!\{e_1, i, j, e_2\}\!\}) := len(e_1) + (len(e_2) \dot{-} i)$.
*The* index $ind(et)$ *of an environment term* $et$*, and the* $l$-th index $ind_l(e)$ *of environment* $e$ *and natural number* $l$*, are simultaneously defined by induction on the structure of expressions:*

$$ind(@m) = m + 1 \qquad ind((t', m)) = m$$

$$ind(\langle\!\langle et', j, k, e\rangle\!\rangle) = \begin{cases} ind_m(e) + (j \dot{-} k) & \text{if } len(e) > j \dot{-} ind(et') = m \\ ind(et') & \textit{otherwise} \end{cases}$$

$$ind_l(nil) = 0 \qquad ind_0(et :: e') = ind(et) \ \textit{and} \ ind_{l+1}(et :: e') = ind_l(e')$$

$$ind_l(\{\!\{e_1, j, k, e_2\}\!\}) = \begin{cases} ind_m(e_2) + (j \dot{-} k) & \text{if } l < len(e_1) \text{ and} \\ & len(e_2) > m = j \dot{-} ind_l(e_1) \\ ind_l(e_1) & \text{if } l < len(e_1) \text{ and} \\ & len(e_2) \leq m = j \dot{-} ind_l(e_1) \\ ind_{l-l_1+j}(e_2) & \text{if } l \geq l_1 = len(e_1) \end{cases}$$

*The* index *of an environment* $e$*, denoted as* $ind(e)$*, is* $ind_0(e)$*.*

**Definition 3.2 ([28])** *An expression of the suspension calculus is said to be*

Table 1
Rewriting rules of the suspension calculus

| | | |
|---|---|---|
| $(\beta_s)$ | $((\lambda t_1\ \ t_2)$ | $\longrightarrow [\![ t_1, 1, 0, (t_2, 0) :: nil ]\!]$ |
| $(r_1)$ | $[\![ c, ol, nl, e ]\!]$ | $\longrightarrow c$, where $c$ is a constant |
| $(r_2)$ | $[\![ \underline{\mathtt{i}}, 0, nl, nil ]\!]$ | $\longrightarrow \underline{\mathtt{i+nl}}$ |
| $(r_3)$ | $[\![ \underline{\mathtt{1}}, ol, nl, @l :: e ]\!]$ | $\longrightarrow \underline{\mathtt{nl\text{-}l}}$ |
| $(r_4)$ | $[\![ \underline{\mathtt{1}}, ol, nl, (t, l) :: e ]\!]$ | $\longrightarrow [\![ t, 0, (nl\text{-}l), nil ]\!]$ |
| $(r_5)$ | $[\![ \underline{\mathtt{i}}, ol, nl, et :: e ]\!]$ | $\longrightarrow [\![ \underline{\mathtt{i\text{-}1}}, (ol\text{-}1), nl, e ]\!]$, for $i > 1$ |
| $(r_6)$ | $[\![ (t_1\ t_2), ol, nl, e ]\!]$ | $\longrightarrow ([\![ t_1, ol, nl, e ]\!]\ [\![ t_2, ol, nl, e ]\!])$ |
| $(r_7)$ | $[\![ \lambda\, t, ol, nl, e ]\!]$ | $\longrightarrow \lambda\, [\![ t, (ol+1), (nl+1), @nl :: e ]\!]$ |
| $(m_1)$ | $[\![ [\![ t, ol_1, nl_1, e_1 ]\!], ol_2, nl_2, e_2 ]\!]$ | $\longrightarrow [\![ t, ol', nl', \{\!\{ e_1, nl_1, ol_2, e_2 \}\!\} ]\!]$, where |
| | | $ol' = ol_1 + (ol_2 \dot{-} nl_1)$ and |
| | | $nl' = nl_2 + (nl_1 \dot{-} ol_2)$ |
| $(m_2)$ | $\{\!\{ nil, nl, 0, nil \}\!\}$ | $\longrightarrow nil$ |
| $(m_3)$ | $\{\!\{ nil, nl, ol, et :: e \}\!\}$ | $\longrightarrow \{\!\{ nil, (nl\text{-}1), (ol\text{-}1), e \}\!\}$, for $nl, ol \geq 1$ |
| $(m_4)$ | $\{\!\{ nil, 0, ol, e \}\!\}$ | $\longrightarrow e$ |
| $(m_5)$ | $\{\!\{ et :: e_1, nl, ol, e_2 \}\!\}$ | $\longrightarrow \langle\!\langle et, nl, ol, e_2 \rangle\!\rangle :: \{\!\{ e_1, nl, ol, e_2 \}\!\}$ |
| $(m_6)$ | $\langle\!\langle et, nl, 0, nil \rangle\!\rangle$ | $\longrightarrow et$ |
| $(m_7)$ | $\langle\!\langle @m, nl, ol, @l :: e \rangle\!\rangle$ | $\longrightarrow @(l + (nl \dot{-} ol))$, for $nl = m+1$ |
| $(m_8)$ | $\langle\!\langle @m, nl, ol, (t, l) :: e \rangle\!\rangle$ | $\longrightarrow (t, (l + (nl \dot{-} ol)))$, for $nl = m+1$ |
| $(m_9)$ | $\langle\!\langle (t, nl), nl, ol, et :: e \rangle\!\rangle$ | $\longrightarrow ([\![ t, ol, l', et :: e ]\!], m)$, where |
| | | $l' = ind(et)$ and $m = l' + (nl \dot{-} ol)$ |
| $(m_{10})$ | $\langle\!\langle et, nl, ol, et' :: e \rangle\!\rangle$ | $\longrightarrow \langle\!\langle et, (nl\text{-}1), (ol\text{-}1), e \rangle\!\rangle$, for $nl \neq ind(et)$ |

well-formed *if the following conditions hold over all its subexpressions s:*
- *if s is* $[\![ t, ol, nl, e ]\!]$ *then* $len(e) = ol$ *and* $ind(e) \leq nl$
- *if s is* $et :: e$ *then* $ind(e) \leq ind(et)$
- *if s is* $\langle\!\langle et, j, k, e \rangle\!\rangle$ *then* $len(e) = k$ *and* $ind(et) \leq j$
- *if s is* $\{\!\{ e_1, j, k, e_2 \}\!\}$ *then* $len(e_2) = k$ *and* $ind(e_1) \leq j$.

In the sequel, we only deal with well-formed expressions of the suspension calculus.

The suspension calculus simulates $\beta$-reduction and its associated substitution calculus SUSP is CR (over closed and open terms) and SN [28]. In [26] Nadathur conjectures that the suspension calculus preserves strong normalization too. The following lemma characterizes the $\rhd_{rm}$-normal forms.

**Lemma 3.3 ([28])** *A well-formed expression of the suspension calculus $x$ is in its $\rhd_{rm}$-nf if and only if one of the following affirmations holds:*
*1) $x$ is a pure $\lambda$-term in de Bruijn notation;*
*2) $x$ is an environment term of the form $@l$ or $(t, l)$, where $t$ is a term in its $\rhd_{rm}$-nf;*
*3) $x$ is the environment $nil$ or $et :: e$ for $et$ and $e$ resp. an environment term and an environment in $\rhd_{rm}$-nf.*

## 4 The suspension calculus enlarged with the $\eta$-reduction: the $\lambda_{\mathbf{susp}}$-calculus

The suspension calculus was initially formulated without $\eta$-reduction. Here we introduce an adequate *Eta* rule that enlarges the suspension calculus preserving correctness, confluence, and termination of the associated substitution calculus. The suspension calculus enlarged with this *Eta* rule is denoted by $\lambda_{\text{SUSP}}$ and its associated substitution calculus remains as SUSP. The *Eta* rule is formulated as follows:

$$(\mathit{Eta}) \quad (\lambda\ (t_1\ \underline{1})) \longrightarrow t_2, \quad \text{if} \quad t_1 =_{rm} [\![t_2, 0, 1, nil]\!]$$

Intuitively *Eta* may be interpreted as: when it is possible to apply the $\eta$-reduction to the redex $\lambda(t_1\ \underline{1})$ we obtain a term $t_2$ that has the same structure as $t_1$ with all its free de Bruijn indices decremented by one. This is possible whenever there are no free occurrences of the variable corresponding to $\underline{1}$ in $t_1$. Proposition 4.2 proves the correctness of *Eta* according to this interpretation. We follow [10] and [3] for $\lambda\sigma$ and $\lambda s_e$ respectively, and implement the *Eta* rule of the $\lambda_{\text{SUSP}}$-calculus by introducing a dummy symbol $\diamond$, by:

$$\lambda(M\ \underline{1}) \longrightarrow_{Eta} N$$
$$\text{if } N = \rhd_{rm}\text{-nf}([\![M, 1, 0, (\diamond, 0) :: nil]\!]) \text{ and } \diamond \text{ does not occur in } N.$$

The correctness of this implementation is explained because an $\eta$-reduction $\lambda(M\ \underline{1}) \to_\eta N$ gives us a term $N$, which is obtained from $M$ by decrementing by one all free occurrences of de Bruijn indices, as previously mentioned, and which corresponds exactly to the $\rhd_{rm}$-normalization of the term $((\lambda M)\ \diamond) \to_{\beta_s} [\![M, 1, 0, (\diamond, 0) :: nil]\!]$, whenever $\diamond$ does not appear in this normalized term.

**Lemma 4.1** *Let $A$ be a well-formed term of the suspension calculus. Then the* SUSP*-normalization of the term $[\![A, k, k+1, @k :: @k-1 :: \ldots :: @1 :: nil]\!]$ gives a term obtained from $A$ by incrementing by one all its de Bruijn free indices greater than $k$ and preserving unaltered all other de Bruijn indices.*

**Proof.** By induction on the structure of $A$. The constant case is trivial.

- $\boldsymbol{A = \underline{\mathtt{n}}}$. If $n > k$ then $[\![\underline{\mathtt{n}}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_5}^{k}$ $[\![\underline{\mathtt{n-k}}, 0, k+1, nil]\!] \rightarrow_{r_2} \underline{\mathtt{n+1}}$.
  If $n \leq k$ then $[\![\underline{\mathtt{n}}, k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_5}^{n-1}$ $[\![\underline{\mathtt{1}}, k-n+1, k+1, @k-n+1 :: \ldots :: @1 :: nil]\!] \rightarrow_{r_3} \underline{\mathtt{n}}$;
- $\boldsymbol{A = (B\ C)}$. we apply $r_6$ and induction hypothesis for $B$ and $C$;
- $\boldsymbol{A = (\lambda B)}$. Since $B$ is bounded by an abstractor just its free variables greater than $k+1$ should be incremented by one, while the other variables should remain unchanged. Since $[\![(\lambda B), k, k+1, @k :: \ldots :: @1 :: nil]\!] \rightarrow_{r_7}$ $\lambda[\![B, k+1, k+2, @k+1 :: \ldots :: @1 :: nil]\!]$, by applying induction hypothesis over the previous term we obtain the desired result.
- $\boldsymbol{A = [\![t, ol, nl, e]\!]}$. Without loss of generality $A$ may be $\triangleright_{rm}$-normalized and by Lemma 3.3 the obtained term is of one of the forms analysed in the previous cases.

□

**Proposition 4.2 (Soundness of the** ***Eta*** **rule)** *Every application of the Eta rule of $\lambda_{\text{SUSP}}$ to the redex $\lambda(t_1\ \underline{\mathtt{1}})$ gives effectively the term $t_2$ obtained from $t_1$ by decrementing all its de Bruijn free indices by one.*

**Proof.** The proof is by induction over the structure of $t_2$ considering the premise $t_1 =_{rm} [\![t_2, 0, 1, nil]\!]$. The effect of normalizing $[\![t_2, 0, 1, nil]\!]$ is to increment by one all de Bruijn free indices occurring at $t_2$:

- $\boldsymbol{t_2 = \underline{\mathtt{n}}}$. $[\![\underline{\mathtt{n}}, 0, 1, nil]\!] \rightarrow_{r_2} \underline{\mathtt{n+1}} =_{rm} t_1$.
- $\boldsymbol{t_2 = (A\ B)}$. Without loss of generality we can assume that both $A$ and $B$ are in $\triangleright_{rm}$-nf. Observe that $[\![(A\ B), 0, 1, nil]\!] \rightarrow_{r_6} [\![A, 0, 1, nil]\!]\ [\![B, 0, 1, nil]\!]$. Now, by induction hypothesis over $A$ and $B$, we have that the normalization of the suspended terms $[\![A, 0, 1, nil]\!]$ and $[\![B, 0, 1, nil]\!]$ have the desired effect and consequently the same happens with the normalization of the suspended term $[\![(A\ B), 0, 1, nil]\!]$.
- $\boldsymbol{t_2 = (\lambda A)}$. As before, assume $A$ is in $\triangleright_{rm}$-nf. Note that $[\![(\lambda A), 0, 1, nil]\!]$ $\rightarrow_{r_7} (\lambda[\![A, 1, 2, @1 :: nil]\!])$. By applying Lemma 4.1 to the term $[\![A, 1, 2, @1 :: nil]\!]$ we conclude that all free occurrences of de Bruijn indices greater than 1 at $A$ are incremented by one while the other indices are unchanged.
- $\boldsymbol{t_2 = [\![t, i, j, e]\!]}$. If $t$ is in $\triangleright_{rm}$-nf then $[\![t, i, j, e]\!] \triangleright_{rm}^{*} t'$, where $t'$ is a pure $\lambda$-term in de Bruijn notation by Lemma 3.3. Hence, the analysis given in the previous three cases applies here too.

□

Noetherianity of SUSP plus the *Eta* rule enables us to apply the Newman diamond lemma and the Knuth-Bendix critical pair criterion for proving its confluence.

**Lemma 4.3 (susp plus** ***Eta*** **is SN)** *The rewriting system associated to* SUSP *and the Eta rule is noetherian.*

**Proof.** (Sketch) This is proved by showing that the *Eta* rule is also compatible with the well-founded partial ordering $\prec$ that is defined and proved compatible with $\rhd_{rm}$ in [28]. $\square$

A simple environment is an environment without subexpressions of the form $\{\!\{\_,\_,\_,\_\}\!\}$ or $\langle\!\langle\_,\_,\_,\_\rangle\!\rangle$.

**Lemma 4.4 ([28])** *Let $e_1$ be a simple environment and suppose that $nl$ and $ol$ are naturals such that $(nl - ind(e_1)) \geq ol$. Then $\{\!\{e_1, nl, ol, e_2\}\!\} \rhd^*_{rm} e_1$.*

**Lemma 4.5 (Local-confluence of susp plus *Eta*)** *The rewriting system of the substitution calculus* SUSP *plus the Eta rule is locally-confluent.*

**Proof.** The rewrite relation $\rhd_{rm}$, i.e., SUSP, was shown in [28] to be (locally) confluent. Thus for proving that the associated rewriting system enlarged with the *Eta* rule is locally-confluent, it is enough to show that all additional critical pairs built by overlapping between the *Eta* rule and the other rules of SUSP are joinable.

Note that no critical pairs are generated from the rule *Eta* and itself. Also, note that there is a unique overlapping between the set of rules in Table 1 (minus $(\beta_s)$) and *Eta*: namely, the one between *Eta* and $(r_7)$.

This critical pair is $\langle \llbracket t_2, ol, nl, e\rrbracket, \lambda\llbracket (t_1\ \underline{\mathtt{1}}), ol+1, nl+1, @nl :: e\rrbracket\rangle$, where $t_1 =_{rm} \llbracket t_2, 0, 1, nil\rrbracket$. After applying the rules $r_6$ and $r_3$ the right-side term of this critical pair reduces to $\lambda(\llbracket t_1, ol+1, nl+1, @nl :: e\rrbracket\ \underline{\mathtt{1}})$.

We prove by analyzing the structure of the term $t_1$ that this critical pair is joinable. As usual we can consider the terms $t_1$ and $t_2$ as $\rhd_{rm}$-nf's.

- $\boldsymbol{t_1} = \underline{\mathtt{n}}$. For making possible the *Eta* application, we need that $n > 1$. According to the length of the environment $@nl :: e$ (i.e., $ol+1$) we have the following cases:
  - $ol + 1 < n$. On the one side, $\lambda(\llbracket \underline{\mathtt{n}}, ol+1, nl+1, @nl :: e\rrbracket\ \underline{\mathtt{1}}) \rightarrow^{ol+1}_{r_5}$ $\lambda(\llbracket \underline{\mathtt{n\text{-}ol\text{-}1}}, 0, nl+1, nil\rrbracket\ \underline{\mathtt{1}}) \rightarrow_{r_2} \lambda(\underline{\mathtt{n\text{-}ol\text{+}nl}}\ \underline{\mathtt{1}}) \rightarrow_{Eta} \underline{\mathtt{n\text{-}ol\text{+}nl\text{-}1}}$. On the other side, $t_1 =_{rm} \llbracket t_2, 0, 1, nil\rrbracket$, hence $t_2 = \underline{\mathtt{n\text{-}1}}$ and we have $\llbracket \underline{\mathtt{n\text{-}1}}, ol, nl, e\rrbracket$ $\rightarrow^{ol}_{r_5} \llbracket \underline{\mathtt{n\text{-}1\text{-}ol}}, 0, nl, nil\rrbracket \rightarrow_{r_2} \underline{\mathtt{n\text{-}ol\text{+}nl\text{-}1}}$.
  - $ol + 1 \geq n$. On the one side, $\lambda(\llbracket \underline{\mathtt{n}}, ol+1, nl+1, @nl :: e\rrbracket\ \underline{\mathtt{1}}) \rightarrow^{n-1}_{r_5}$ $\lambda(\llbracket \underline{\mathtt{1}}, ol-n+2, nl+1, e_1 :: e'\rrbracket\ \underline{\mathtt{1}})$ and the subsequent derivation depends on the structure of $e_1$: when $e_1 = @l$ we apply $r_3$ obtaining $\lambda(\underline{\mathtt{nl\text{+}1\text{-}l}}\ \underline{\mathtt{1}})$ $\rightarrow_{Eta} \underline{\mathtt{nl\text{-}l}}$ and on the other side, $\llbracket \underline{\mathtt{n\text{-}1}}, ol, nl, e\rrbracket \rightarrow^{n-2}_{r_5}$ $\llbracket \underline{\mathtt{1}}, ol-n+2, nl, @l :: e'\rrbracket \rightarrow_{r_3} \underline{\mathtt{nl\text{-}l}}$; when $e_1 = (t, l)$, where without loss of generality $t$ is suppossed to be in $\rhd_{rm}$-nf, we have $\lambda(\llbracket \underline{\mathtt{1}}, ol-n+2, nl+1, (t,l) :: e'\rrbracket\ \underline{\mathtt{1}}) \rightarrow_{r_4} \lambda(\llbracket t, 0, nl-l+1, nil\rrbracket\ \underline{\mathtt{1}}) \rightarrow_{Eta}$ $\rhd_{rm}\text{-}nf(\llbracket\llbracket t, 0, nl{+}1{-}l, nil\rrbracket, 1, 0, (\diamond, 0) :: nil\rrbracket) \rightarrow_{m_1}$ $\rhd_{rm}\text{-}nf(\llbracket t, 0, nl{-}l, \{\!\{nil, nl{+}1{-}l, 1, (\diamond, 0) :: nil\}\!\}\rrbracket) \rightarrow_{m_3}$ $\rhd_{rm}\text{-}nf(\llbracket t, 0, nl-l, \{\!\{nil, nl-l, 0, nil\}\!\}\rrbracket) \rightarrow_{m_2} \rhd_{rm}\text{-}nf(\llbracket t, 0, nl-l, nil\rrbracket)$ and on the other side, $\llbracket \underline{\mathtt{1}}, ol-n+2, nl, (t,l) :: e'\rrbracket \rightarrow_{r_4} \llbracket t, 0, nl-l, nil\rrbracket$.

Since $\triangleright_{rm}$-$nf([\![t,0,nl-l,nil]\!])$ and $[\![t,0,nl-l,nil]\!]$ are joinable we obtain the confluence.

- $\boldsymbol{t_1 = (A\ B)}$. Since the sole rule of the $\lambda_{\text{SUSP}}$ that truly “applies” applications is the $\beta_s$, we can separately consider *Eta* reductions for $A$ and $B$ and then apply the induction hypothesis. That is, suppose inductively that $\lambda([\![A, ol+1, nl+1, @nl :: e]\!]\ \underline{1}) \rightarrow_{Eta} A''$ and $[\![A', ol, nl, e]\!]$, where $[\![A', 0, 1, nil]\!] =_{rm} A$ as well as $\lambda([\![B, ol+1, nl+1, @nl :: e]\!]\ \underline{1}) \rightarrow_{Eta} B''$ and $[\![B', ol, nl, e]\!]$, where $[\![B', 0, 1, nil]\!] =_{rm} B$ are joinable. Then since $\lambda([\![(A\ B), ol+1, nl+1, @nl :: e]\!]\ \underline{1}) \rightarrow_{r_6}$ $\lambda(([\![A, ol+1, nl+1, @nl :: e]\!]\ [\![B, ol+1, nl+1, @nl :: e]\!])\ \underline{1}) \rightarrow_{Eta} (A''\ B'')$ and $[\![(A'\ B'), ol, nl, e]\!] \rightarrow_{r_6} ([\![A', ol, nl, e]\!]\ [\![B', ol, nl, e]\!])$ we can conclude the confluence.

- $\boldsymbol{t_1 = (\lambda A)}$. By the *Eta* rule implementation, it is enough to show the joinability of the *Eta* reduction of the term $\lambda([\![(\lambda A), ol+1, nl+1, @nl :: e]\!]\ \underline{1})$ that is $\triangleright_{rm}$-nf($[\![[\![(\lambda A), ol+1, nl+1, @nl :: e]\!], 1, 0, (\diamond, 0) :: nil]\!]$) and the term $[\![\ \triangleright_{rm}$ -nf($[\![(\lambda A), 1, 0, (\diamond, 0) :: nil]\!]$)$, ol, nl, e]\!]$.

  On the one side, $[\![\ \triangleright_{rm}$ -nf($[\![(\lambda A), 1, 0, (\diamond, 0) :: nil]\!]$)$, ol, nl, e]\!] \triangleright^*_{rm}$ $\triangleright_{rm}$-nf($[\![[\![(\lambda A), 1, 0, (\diamond, 0) :: nil]\!], ol, nl, e]\!]$) $\rightarrow_{r_7, r_7}$ $\triangleright_{rm}$-nf($(\lambda [\![[\![A, 2, 1, @0 :: (\diamond, 0) :: nil]\!], ol+1, nl+1, @nl :: e]\!])$) $\triangleright^*_{rm}$ $(\lambda \triangleright_{rm}$ -nf($[\![[\![A, 2, 1, @0 :: (\diamond, 0) :: nil]\!], ol+1, nl+1, @nl :: e]\!]$)$) \rightarrow_{m_1}$ $(\lambda \triangleright_{rm}$ -nf($[\![A, ol+2, nl+1, \{\!\{@0 :: (\diamond, 0) :: nil, 1, ol+1, @nl :: e\}\!\}]\!]$)$)$ and we have that $\{\!\{@0 :: (\diamond, 0) :: nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_5, m_5}$ $\langle\!\langle @0, 1, ol+1, @nl :: e\rangle\!\rangle :: \langle\!\langle (\diamond, 0), 1, ol+1, @nl :: e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_7}$ $@nl :: \langle\!\langle (\diamond, 0), 1, ol+1, @nl :: e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_{10}}$ $@nl :: \langle\!\langle (\diamond, 0), 0, ol, e\rangle\!\rangle :: \{\!\{nil, 1, ol+1, @nl :: e\}\!\} \rightarrow_{m_3, m_4}$ $@nl :: \langle\!\langle (\diamond, 0), 0, ol, e\rangle\!\rangle :: e$. Then we obtain the term $(\lambda \triangleright_{rm}$-nf($[\![A, ol+2, nl+1, @nl :: \langle\!\langle (\diamond, 0), 0, ol, e\rangle\!\rangle :: e]\!]$)$)$. On the other side, $\triangleright_{rm}$-nf($[\![[\![(\lambda A), ol+1, nl+1, @nl :: e]\!], 1, 0, (\diamond, 0) :: nil]\!]$) $\rightarrow_{r_7, r_7}$ $\triangleright_{rm}$-nf($(\lambda [\![[\![A, ol+2, nl+2, @nl+1 :: @nl :: e]\!], 2, 1, @0 :: (\diamond, 0) :: nil]\!])$) $\triangleright^*_{rm}$ $(\lambda \triangleright_{rm}$ -nf($[\![[\![A, ol+2, nl+2, @nl+1 :: @nl :: e]\!], 2, 1, @0 :: (\diamond, 0) :: nil]\!]$)$) \rightarrow_{m_1}$ $(\lambda \triangleright_{rm}$-nf$[\![A, ol+2, nl+1, \{\!\{@nl+1 :: @nl :: e, nl+2, 2, @0 :: (\diamond, 0) :: nil]\!])$ and we have that $\{\!\{@nl+1 :: @nl :: e, nl+2, 2, @0 :: (\diamond, 0) :: nil\}\!\} \rightarrow_{m_5, m_5}$ $\langle\!\langle @nl+1, nl+2, 2, @0 :: (\diamond, 0) :: nil\rangle\!\rangle :: \langle\!\langle @nl, nl+2, 2, @0 :: (\diamond, 0) :: nil\rangle\!\rangle ::$ $\{\!\{e, nl+2, 2, @0 :: (\diamond, 0) :: nil\}\!\} \rightarrow_{m_7} @nl :: \langle\!\langle @nl, nl+2, 2, @0 :: (\diamond, 0) :: nil\rangle\!\rangle :: \{\!\{e, nl+2, 2, @0 :: (\diamond, 0) :: nil\}\!\} \triangleright^*_{rm}$ (By Lemma 4.4, since we are working with well-formed terms and then) $ind(e) \leq nl$) $@nl :: \langle\!\langle @nl, nl+2, 2, @0 :: (\diamond, 0) :: nil\rangle\!\rangle :: e \rightarrow_{m_{10}}$ $@nl :: \langle\!\langle @nl, nl+1, 1, (\diamond, 0) :: nil\rangle\!\rangle :: e \rightarrow_{m_8} @nl :: (\diamond, nl) :: e$.

  Then we obtain the term $(\lambda \triangleright_{rm}$-nf($[\![A, ol+2, nl+1, @nl :: (\diamond, nl) :: e]\!]$)$)$.

  The sole difference of the obtained suspended terms is the second environment term of their environments, that is $\langle\!\langle (\diamond, 0), 0, ol, e\rangle\!\rangle$ and $(\diamond, nl)$. But since the *Eta* rule applies, when propagating the substitution between these suspended terms, the dummy symbol and hence these second environment terms should disapear. Now we can conclude that these terms are joinable.

□

Finally, since the rewriting system associated to SUSP enlarged with the *Eta* rule is locally-confluent and noetherian, we can apply the Newman diamond lemma for concluding its confluence.

**Theorem 4.6 (Confluence of susp plus** *Eta***)** *The calculus* SUSP *jointly with the Eta rule, is confluent.*

## 5 Comparing the adequacy of the calculi

According to the criterion of adequacy introduced in [22] we prove that the $\lambda\sigma$ and the $\lambda_{\text{SUSP}}$ as well as the $\lambda\sigma$ and the $\lambda s_e$ are non comparable. Additionally, we prove that the $\lambda s_e$ is more adequate than the $\lambda_{\text{SUSP}}$.

Let $a, b \in \Lambda$ such that $a \to_\beta b$. A *simulation* of this $\beta$-reduction in $\lambda\xi$, for $\xi \in \{\sigma, s_e, \text{SUSP}\}$ is a $\lambda\xi$-derivation $a \to_r c \to_\xi^* \xi(c) = b$, where $r$ is the rule starting $\beta$ (*beta* for $\lambda\sigma$, *$\sigma$-generation* for $\lambda s_e$, $\beta_s$ for $\lambda_{\text{SUSP}}$) applied to the same redex as the redex in $a \to_\beta b$. The criterion of adequacy is defined as follow:

**Definition 5.1 (Adequacy)** *Let $\xi_1, \xi_2 \in \{\sigma, s_e, \text{SUSP}\}$. The $\lambda\xi_1$-calculus is* more adequate *(in simulating one step of $\beta$-reduction) than the $\lambda\xi_2$-calculus, denoted $\lambda\xi_1 \prec \lambda\xi_2$, if:*

- *for every $\beta$-reduction $a \to_\beta b$ and every $\lambda\xi_2$-simulation $a \to_{\lambda\xi_2}^n b$ there exists a $\lambda\xi_1$-simulation $a \to_{\lambda\xi_1}^m b$ such that $m \leq n$;*
- *there exists a $\beta$-reduction $a \to_\beta b$ and a $\lambda\xi_1$-simulation $a \to_{\lambda\xi_1}^m b$ such that for every $\lambda\xi_2$-simulation $a \to_{\lambda\xi_2}^n b$ we have $m < n$.*

*If neither $\lambda\xi_1 \prec \lambda\xi_2$ nor $\lambda\xi_2 \prec \lambda\xi_1$, then we say that $\lambda\xi_1$ and $\lambda\xi_2$ are* non comparable.

The counterexamples proving that $\lambda\sigma$ and $\lambda s$ are non comparable presented in [22] apply for the incomparability of $\lambda\sigma$ and $\lambda s_e$ since $\lambda s_e$ is an extension of $\lambda s$ for open terms.

**Proposition 5.2** *The $\lambda\sigma$- and the $\lambda s_e$-calculi are non comparable.*

**Lemma 5.3** *Every $\lambda\sigma$-derivation of* $((\lambda\lambda\underline{2})\ \underline{1})$ *to its $\lambda\sigma$-nf has length greater than or equal to* 6.

**Proof.** In fact, all possible derivations are of one of the following forms.

- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Clos}$ $\lambda\underline{1}[\uparrow\circ(\underline{1}.((\underline{1}.id)\circ\uparrow))] \to_{ShiftCons} \lambda\underline{1}[(\underline{1}.id)\circ\uparrow] \to_{Map} \lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{VarCons}$ $\lambda\underline{1}[\uparrow] = \lambda\underline{2}$;
- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Clos}$ $\lambda\underline{1}[\uparrow\circ(\underline{1}.((\underline{1}.id)\circ\uparrow))] \to_{ShiftCons} \lambda\underline{1}[(\underline{1}.id)\circ\uparrow] \to_{Map} \lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{IdL}$ $\lambda\underline{1}[\underline{1}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2}$;
- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Clos}$

$\lambda\underline{1}[\uparrow\circ(\underline{1}.((\underline{1}.id)\circ\uparrow))] \to_{Map} \lambda\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
$\lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2};$

- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Clos}$
  $\lambda\underline{1}[\uparrow\circ(\underline{1}.((\underline{1}.id)\circ\uparrow))] \to_{Map} \lambda\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{IdL} \lambda\underline{1}[\underline{1}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2};$
- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Map}$
  $\lambda\underline{1}[\uparrow][\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow))] \to_{Clos} \lambda\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2};$
- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Map}$
  $\lambda\underline{1}[\uparrow][\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow))] \to_{Clos} \lambda\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow)))] \to_{ShiftCons}$
  $\lambda\underline{1}[\underline{1}[\uparrow].(id\circ\uparrow)] \to_{IdL} \lambda\underline{1}[\underline{1}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2};$
- $(\lambda\lambda\underline{1}[\uparrow])\ \underline{1} \to_{Beta} (\lambda\underline{1}[\uparrow])[\underline{1}.id] \to_{Abs} \lambda\underline{1}[\uparrow][\underline{1}.((\underline{1}.id)\circ\uparrow)] \to_{Map}$
  $\lambda\underline{1}[\uparrow][\underline{1}.(\underline{1}[\uparrow].(id\circ\uparrow))] \to_{IdL} \lambda\underline{1}[\uparrow][\underline{1}.(\underline{1}[\uparrow].\uparrow)] \to_{Clos}$
  $\lambda\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow].\uparrow))] \to_{ShiftCons} \lambda\underline{1}[\underline{1}[\uparrow].\uparrow] \to_{VarCons} \lambda\underline{1}[\uparrow] = \lambda\underline{2}.$

□

**Lemma 5.4** *Every $\lambda_{\text{SUSP}}$-derivation of $(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n$ to its $\lambda_{\text{SUSP}}$-nf has length $4n+5$.*

**Proof.** In fact, note that the sole possible derivation is:
$(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n \to_{\beta_s} \llbracket(\lambda(\underline{2}\ \underline{2})), 1, 0, (\underline{1}^n, 0) :: nil\rrbracket \to_{r_7}$
$\lambda\llbracket(\underline{2}\ \underline{2}), 2, 1, @0 :: (\underline{1}^n, 0) :: nil\rrbracket \to_{r_6}$
$\lambda(\llbracket\underline{2}, 2, 1, @0 :: (\underline{1}^n, 0) :: nil\rrbracket\ \llbracket\underline{2}, 2, 1, @0 :: (\underline{1}^n, 0) :: nil\rrbracket) \to^2_{r_5}$
$\lambda(\llbracket\underline{1}, 1, 1, (\underline{1}^n, 0) :: nil\rrbracket\ \llbracket\underline{1}, 1, 1, (\underline{1}^n, 0) :: nil\rrbracket) \to^2_{r_4}$
$\lambda(\llbracket\underline{1}^n, 0, 1, nil\rrbracket\ \llbracket\underline{1}^n, 0, 1, nil\rrbracket) \to^{2(n-1)}_{r_6} \lambda((\llbracket\underline{1}, 0, 1, nil\rrbracket)^n\ (\llbracket\underline{1}, 0, 1, nil\rrbracket)^n) \to^{2n}_{r_2}$
$\lambda(\underline{2}^n\ \underline{2}^n).$ □

**Lemma 5.5** ( [22]) *There exists a derivation of $(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n$ to its $\lambda\sigma$-nf whose length is $n+9$.*

**Proof.** Consider the following derivation:
$(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n = (\lambda\lambda(\underline{1}[\uparrow]\ \underline{1}[\uparrow]))\ \underline{1}^n \to_{Beta} (\lambda(\underline{1}[\uparrow]\ \underline{1}[\uparrow]))[\underline{1}^n.id] \to_{Abs}$
$\lambda((\underline{1}[\uparrow]\ \underline{1}[\uparrow])[\underline{1}.((\underline{1}^n.id)\circ\uparrow)]) \to_{Map}$
$\lambda((\underline{1}[\uparrow]\ \underline{1}[\uparrow])[\underline{1}.(\underline{1}^n[\uparrow].(id\circ\uparrow))]) \to^{n-1}_{App} \lambda((\underline{1}[\uparrow]\ \underline{1}[\uparrow])[\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))]) \to_{App}$
$\lambda((\underline{1}[\uparrow][\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))])\ (\underline{1}[\uparrow][\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))])) \to_{Clos}$
$\lambda((\underline{1}[\uparrow\circ(\underline{1}.(\underline{1}[\uparrow])^n.(id\circ\uparrow))])\ (\underline{1}[\uparrow][\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))])) \to_{ShiftCons}$
$\lambda((\underline{1}[(\underline{1}[\uparrow])^n.(id\circ\uparrow)])\ (\underline{1}[\uparrow][\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))])) \to_{VarCons}$
$\lambda((\underline{1}[\uparrow])^n\ (\underline{1}[\uparrow][\underline{1}.((\underline{1}[\uparrow])^n.(id\circ\uparrow))])) \to^3 \lambda((\underline{1}[\uparrow])^n\ (\underline{1}[\uparrow])^n) = \lambda(\underline{2}^n\ \underline{2}^n).$ □

**Proposition 5.6** *The $\lambda\sigma$- and $\lambda_{\text{SUSP}}$-calculi are non comparable.*

**Proof.** On the one side, by Lemmas 5.4 and 5.5, there exists a simulation $(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n \twoheadrightarrow_{\lambda\sigma} \lambda(\underline{2}\ \underline{2})$ shorter than the shortest of the simulations $(\lambda\lambda(\underline{2}\ \underline{2}))\ \underline{1}^n \twoheadrightarrow_{\lambda_{\text{SUSP}}} \lambda(\underline{2}\ \underline{2})$. Then $\lambda_{\text{SUSP}} \not\preceq \lambda\sigma$.

On the other side, consider the following simulation in $\lambda_{\text{SUSP}}$:
$((\lambda\lambda\underline{2})\ \underline{1}) \to_{\beta_s} \llbracket(\lambda\underline{2}), 1, 0, (\underline{1}, 0) :: nil\rrbracket \to_{r_7} \lambda\llbracket\underline{2}, 2, 1, @0 :: (\underline{1}, 0) :: nil\rrbracket \to_{r_5}$

$\lambda[\![\underline{1}, 1, 1, (\underline{1}, 0) :: nil]\!] \to_{r_4} \lambda[\![\underline{1}, 0, 1, nil]\!] \to_{r_2} \lambda\underline{2}$.

This simulation together with Lemma 5.3 allows us to conclude that: $\lambda\sigma \not\preceq \lambda_{\text{SUSP}}$. $\square$

To prove that $\lambda s_e$ is more adequate than $\lambda_{\text{SUSP}}$ we need to estimate the lengths of derivations.

**Definition 5.7** *Let $A, B, C \in \Lambda$ and $k \geq 0$. We define the functions $M : \Lambda \to \mathbb{N}$ and $Q_k : \Lambda \times \Lambda \to \mathbb{N}$ by:*

- $M(\underline{\mathtt{n}}) = 1$
- $M(\lambda A) = M(A) + 1$
- $M(A\ B) = M(A) + M(B) + 1$
- $Q_k(\underline{\mathtt{n}}, B) = \begin{cases} n & \textit{if } n < k \\ n + M(B) & \textit{if } n = k \\ k+1 & \textit{if } n > k \end{cases}$
- $Q_k((A\ B), C) = Q_k(A, C) + Q_k(B, C) + 1$
- $Q_k(\lambda A, B) = Q_{k+1}(A, B) + 1$

**Lemma 5.8** *Let $A \in \Lambda$. Then all $s_e$-derivations of $\varphi^i_k A$ to its $s_e$-nf have length $M(A)$.*

**Proof.** By simple induction over the structure of $A$. This is an easy extension of the same lemma formulated for the $\lambda s$-calculus in [22]. $\square$

**Lemma 5.9** *Let $A \in \Lambda$. Then all* SUSP*-derivations of the well-formed term $[\![A, i, i, @i-1 :: \ldots :: @0 :: nil]\!]$ to its* SUSP*-nf have length greater than or equal to $M(A)$.*

**Proof.** By induction over the structure of terms.

- $\boldsymbol{A = \underline{\mathtt{n}}}$. If $n > i$ then $[\![\underline{\mathtt{n}}, i, i, @i-1 :: \ldots :: @0 :: nil]\!] \to^i_{r_5} [\![\underline{\mathtt{n-i}}, 0, i, nil]\!] \to_{r_2} \underline{\mathtt{n}}$. The length of the derivation is $i + 1 \geq M(A)$. If $n \leq i$ then $[\![\underline{\mathtt{n}}, i, i, @i-1 :: \ldots :: @0 :: nil]\!] \to^{n-1}_{r_5} [\![\underline{1}, i-n+1, i, @i-n :: \ldots :: @0 :: nil]\!] \to_{r_3} \underline{\mathtt{n}}$. The length of the derivation is $n \geq M(A)$.
- $\boldsymbol{A = (B\ C)}$. We have that $[\![(B\ C), i, i, @i-1 :: \ldots :: @0 :: nil]\!] \to_{r_6} ([\![B, i, i, @i-1 :: \ldots :: @0 :: nil]\!]\ [\![C, i, i, @i-1 :: \ldots :: @0 :: nil]\!])$. By the induction hypothesis we conclude that the length of the derivation is greater than or equal to $1 + M(B) + M(C) = M(B\ C) = M(A)$.
- $\boldsymbol{A = (\lambda B)}$. We have that $[\![(\lambda B), i, i, @i-1 :: \ldots :: @0 :: nil]\!] \to_{r_7} \lambda[\![B, i+1, i+1, @i :: \ldots :: @0 :: nil]\!]$. By induction hypothesis we conclude that the length of the derivation is greater than or equal to $1 + M(B) = M(\lambda B) = M(A)$.

$\square$

**Lemma 5.10** *Let $B \in \Lambda$ and $i, j \geq 0$. The derivation of the* SUSP*-term $[\![B, i, j, @j-1 :: e]\!]$ to its* SUSP*-nf has length greater than or equal to $M(B)$.*

**Proof.** – Case $\boldsymbol{B = \underline{\mathtt{n}}}$, $[\![\underline{\mathtt{n}}, i, j, @j-1 :: e]\!]$ rewrites to its SUSP-nf in one or more steps depending on $n$.

- Case $\boldsymbol{B = (C\ D)}$, we have $[\![(C\ D), i, j, @j-1 :: e]\!] \to_{r_6} [\![C, i, j, @j-1 :: e]\!]$

$[\![D, i, j, @j-1 :: e]\!]$. By the induction hypothesis we obtain the desired result.

- Case $\boldsymbol{B = (\lambda C)}$, we have $[\![(\lambda C), i, j, @j-1 :: e]\!] \rightarrow_{r_7}$ $\lambda[\![C, i+1, j+1, @j :: e']\!]$, that by induction hypothesis completes the proof.

$\square$

**Proposition 5.11** *Let $A, B \in \Lambda$ and $k \geq 0$. Then every* SUSP*-derivation of $[\![A, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!]$ to its* SUSP*-nf has length greater than or equal to $Q_k(A, B)$.*

**Proof.** By structural induction over $A$.

- $\boldsymbol{A} = \underline{\mathtt{n}}$. If $n < k$ then $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{n-1}$ $[\![\underline{\mathtt{1}}, k-n+1, k-1, @k-n-1 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow r_3\ \underline{\mathtt{n}}$. This derivation has length $n \geq Q_k(\underline{\mathtt{n}}, B)$.
  If $n = k$ then $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{n-1}$ $[\![\underline{\mathtt{1}}, 1, k-1, (B, l) :: nil]\!] \rightarrow r_4\ [\![B, 0, k-1-l, nil]\!]$. By Lemma 5.10 the last term rewrites to its SUSP-nf in $M(B)$ or more rewrite steps. The whole derivation has length greater than or equal to $n + M(B) = Q_k(\underline{\mathtt{n}}, B) = Q_k(A, B)$.
  If $n > k$ then $[\![\underline{\mathtt{n}}, k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_5}^{k} [\![\underline{\mathtt{n\text{-}k}}, 0, k\text{-}1, nil]\!]$ $\rightarrow_{r_2} \underline{\mathtt{n-1}}$. Derivation whose length is $k + 1 \geq Q_k(\underline{\mathtt{n}}, B) = Q_k(A, B)$.
- $\boldsymbol{A = (C\ D)}$. $[\![(C\ D), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_6}$ $([\![C, k, k\text{-}1, @k\text{-}2 :: \ldots :: @0 :: (B,0) :: nil]\!]\ [\![D, k, k\text{-}1, @k\text{-}2 :: \ldots :: @0 :: (B,0) :: nil]\!])$. By the induction hypothesis the derivation has length greater than or equal to $1 + Q_k(C, B) + Q_k(D, B) = Q_k((C\ D), B) = Q_k(A, B)$.
- $\boldsymbol{A = (\lambda C)}$. $[\![(\lambda C), k, k-1, @k-2 :: \ldots :: @0 :: (B, l) :: nil]\!] \rightarrow_{r_7}$ $\lambda[\![C, k+1, k, @k-1 :: \ldots :: @0 :: (B, l) :: nil]\!]$. By the induction hypothesis we can conclude that this derivation has length greater than or equal to $1 + Q_{k+1}(C, B) = Q_k(\lambda C, B) = Q_k(A, B)$.

$\square$

**Proposition 5.12** *Let $A, B \in \Lambda$ and $k \geq 1$. $s_e$-derivations of $A\sigma^k B$ to its $s_e$-nf have length $\leq Q_k(A, B)$.*

**Proof.** By structural induction over the pure lambda term $A$.

- $\boldsymbol{A} = \underline{\mathtt{n}}$. By applying the *$\sigma$-destruction* rule, in the case $n \neq k$, we obtain either $\underline{\mathtt{n-1}}$ or $\underline{\mathtt{n}}$ and in the case $n = k$, $\varphi_0^k B$. In the case that $n \neq k$, the derivation has length equal to $1 \leq Q_k(\underline{\mathtt{n}}, B)$. In the other case, we apply Lemma 5.8 obtaining that the complete $s_e$-normalization has length $1 + M(B)$. In both cases the derivation has length less than or equal to $Q_k(\underline{\mathtt{n}}, B)$.
- $\boldsymbol{A = (C\ D)}$. $(C\ D)\sigma^k B \rightarrow (C\sigma^k B\ D\sigma^k B)$. By applying the induction hypothesis we conclude that the complete derivation has length less than or equal to $1 + Q_k(C, B) + Q_k(D, B) = Q_k((C\ D), B)$.

- $\boldsymbol{A = (\lambda C)}$. $(\lambda C)\sigma^k B \to \lambda(C\sigma^{k+1}B)$. By the induction hypothesis we conclude that the whole derivation has length less than or equal to $1 + Q_{k+1}(C, B) = Q_k(\lambda C, B)$.

$\square$

**Theorem 5.13** ($\lambda s_e \prec \lambda_{\mathbf{susp}}$) *The $\lambda s_e$ is more adequate than the $\lambda_{\text{SUSP}}$-calculus.*

**Proof.** We prove the stronger result that if $A \in \Lambda$ and $A \to_{\beta_s} B \to^m_{\text{SUSP}}$ SUSP-nf$(B)$ is a $\lambda_{\text{SUSP}}$-simulation of a $\beta$-reduction then: $A \to_{\sigma-generation} C \to^n_{s_e} s_e$-nf$(C)$ has length $n + 1 \leq m + 1$ .
In $\lambda_{\text{SUSP}}$, for any redex of $\beta_s$ we have $(\lambda D)\ E \to_{\beta_s} [\![D, 1, 0, (E, 0) :: nil]\!] \to^m_{\text{SUSP}}$ SUSP-nf$([\![D, 1, 0, (E, 0) :: nil]\!])$. In the $\lambda s_e$, $(\lambda D)\ E \to_{\sigma-generation} D\sigma^1 E \to^n_{s_e} s_e$-nf$(D\sigma^1 E)$. By Propositions 5.11 and 5.12, $m \geq Q_1(D, E) \geq n$. Hence, the length of a $\lambda_{\text{SUSP}}$-simulation of a $\beta$-contraction is not shorter than that of some $\lambda s_e$-simulation.

The 2nd part of being *more adequate* is shown by comparing the length of simulations. E.g., let $(\lambda \underline{2})\ \underline{1} \to_\beta \underline{1}$. In $\lambda_{\text{SUSP}}$ the only possible three steps simulation is: $(\lambda \underline{2})\ \underline{1} \to_{\beta_s} [\![\underline{2}, 1, 0, (\underline{1}, 0) :: nil]\!] \to_{r_5} [\![\underline{1}, 0, 0, nil]\!] \to_{r_2} \underline{1}$. In $\lambda s_e$ the only possible two steps simulation is: $(\lambda \underline{2})\ \underline{1} \to_{\sigma-generation} \underline{2}\sigma^1\underline{1} \to_{\sigma-destruction} \underline{1}$. $\square$

As mentioned in the above proof, we prove a stronger result than simple better adequacy of $\lambda s_e$ as in [22]. In fact, we prove that the length of all $\lambda s_e$-simulations are shorter than the length of any $\lambda_{\text{SUSP}}$-simulation. Examining the proofs of Propositions 5.11 and 5.12 which relate the length of derivations with the measure operator $Q_k$, it appears evident that both calculi work similarly except that after having propagated suspended terms between the body of abstractors, $\lambda_{\text{SUSP}}$ deals with the substitutions in a less efficient way. To explain that, compare the simulations of $\beta$-reduction from the term $(\lambda(\lambda^n \underline{\mathtt{i}}))\ \underline{\mathtt{j}}$, where $n \geq 0$:
$(\lambda(\lambda^n \underline{\mathtt{i}}))\underline{\mathtt{j}} \to_{\sigma-gen} (\lambda^n \underline{\mathtt{i}})\sigma^1 \underline{\mathtt{j}} \to^n_{\sigma-\lambda-trans} \lambda^n(\underline{\mathtt{i}}\sigma^{n+1}\underline{\mathtt{j}}) =: t_1$
$(\lambda(\lambda^n \underline{\mathtt{i}}))\underline{\mathtt{j}} \to_{\beta_s} [\![\lambda^n \underline{\mathtt{i}}, 1, 0, (\underline{\mathtt{j}}, 0) :: nil]\!] \to^n_{r_7} \lambda^n [\![\underline{\mathtt{i}}, n+1, n, @n\text{-}1 :: \ldots :: @0 :: (\underline{\mathtt{j}}, 0) :: nil]\!]$ $=: t_2$

After that the $\lambda s_e$ complete the simulation in one or two steps by checking arithmetic inequations:

$$t_1 \to_{\sigma-dest} \begin{cases} \lambda^n \underline{\mathtt{i}}, & \text{if } i < n+1 \\ \lambda^n \underline{\mathtt{i} - \mathtt{1}}, & \text{if } i > n+1 \\ \lambda^n(\varphi^{n+1}_0 \underline{\mathtt{j}}) \to_{\varphi-dest} \lambda^n \underline{\mathtt{j} + \mathtt{n}}, & \text{if } i = n+1 \end{cases}$$

But in the $\lambda_{\text{SUSP}}$ we have to destruct the environment list, environment by environment:

$$t_2 \begin{cases} \to_{r_5}^{i-1} \lambda^n[\![\underline{\mathtt{1}}, n\text{-}i+2, n, @n\text{-}i :: \ldots :: @0 :: (\underline{\mathtt{j}}, 0) :: nil]\!] \to_{r_3} \lambda^n \underline{\mathtt{i}}, & \text{if } i < n+1 \\ \to_{r_5}^{n+1} \lambda^n[\![\underline{\mathtt{i}-\mathtt{n}-\mathtt{1}}, 0, n, nil]\!] \to_{r_2} \lambda^n \underline{\mathtt{i}-\mathtt{1}}, & \text{if } i > n+1 \\ \to_{r_5}^{i-1} \lambda^n[\![\underline{\mathtt{1}}, 1, n, (\underline{\mathtt{j}}, 0) :: nil]\!] \to_{r_4} \lambda^n[\![\underline{\mathtt{j}}, 0, n, nil]\!] \to_{r_2} \lambda^n \underline{\mathtt{j}+\mathtt{n}}, & \text{if } i = n+1 \end{cases}$$

These simple considerations lead us to believe that the main difference of the two calculus (at least in the simulation of $\beta$-reduction) is given by the manipulation of indices: although $\lambda_{\text{SUSP}}$ includes all de Bruijn indices, it does not profit from the existence of the built-in arithmetic for indices. These observations may be relevant for the treatment of the open question of preservation of strong normalization of $\lambda_{\text{SUSP}}$ (conjectured positively in [26]), since the $\lambda s_e$ has been proved to answer this question negatively in [16].

## 6 Future Work and Conclusion

[13,2] showed that $\eta$-reduction is of great interest for adapting substitution calculi ($\lambda\sigma$ and $\lambda s_e$) for important practical problems like higher order unification. In this paper we have enlarged the suspension calculus of [28,26] with an adequate *Eta* rule for $\eta$-reduction and showed that this extended suspension calculus $\lambda_{\text{SUSP}}$ enjoys confluence and termination of the associated substitution calculus SUSP.

Additionally, we used the notion of adequacy of [22] for comparing these three calculi when simulating one step $\beta$-reduction. We concluded that $\lambda\sigma$ and $\lambda\xi$ are mutually non comparable for $\xi \in \{s_e, \text{SUSP}\}$ but that $\lambda s_e$ is more adequate than $\lambda_{\text{SUSP}}$. After all, although $\lambda\sigma$ is a first order calculus and the other two calculi are second order, comparing them is not unfair since the use of (built-in) arithmetic is standard in all modern programming environments.

An immediate work to be done is to study two open questions: 1) whether the $s_e$-calculus has strong normalization (SN), 2) whether $\lambda_{\text{SUSP}}$ preserves SN. Interesting points arise in this context since: a) $\lambda s_e$ is more adequate than $\lambda_{\text{SUSP}}$, b) $\lambda s_e$ does not preserves SN [16] and c) the substitution calculus of $\lambda_{\text{SUSP}}$ has SN.

## References

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

[2] M. Ayala-Rincón and F. Kamareddine. Unification via $\lambda s_e$-Style of Explicit Substitution. In Journal of the IGPL 9(4):521-555, 2001.

[3] M. Ayala-Rincón and F. Kamareddine. On Applying the $\lambda s_e$-Style of Unification for Simply-Typed Higher Order Unification in the Pure lambda Calculus. In *Pre-Proceedings Eighth Workshop on Logic, Language, Information and Computation - WoLLIC 2001*, pages 41–54, 2001.

[4] M. Ayala-Rincón and C. Muñoz. Explicit Substitutions and All That. *Revista Colombiana de Computación*, 1(1):47–71, 2000.

[5] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge University Press, 1998.

[6] H. Barendregt. *The Lambda Calculus : Its Syntax and Semantics (revised edition).* North Holland, 1984.

[7] Z.-el-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda v$, a Calculus of Explicit Substitutions which Preserves Strong Normalization. *Journal of Functional Programming*, 6(5):699–722, 1996.

[8] Z.-el-A. Benaissa, P. Lescanne, and K. H. Rose. Modeling Sharing and Recursion for Weak Reduction Strategies using Explicit Substitution. In *PLILP'96,LNCS* 1140, 393–407. Springer, 1996.

[9] R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1997.

[10] P. Borovanský. Implementation of Higher-Order Unification Based on Calculus of Explicit Substitutions. In M. Bartošek, J. Staudek, and J. Wiedermann, editors, *Proceedings of the SOFSEM'95: Theory and Practice of Informatics*, volume 1012 of *Lecture Notes on Computer Science*, pages 363–368. Springer Verlag, 1995.

[11] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence Properties of Weak and Strong Calculi of Explicit Substitutions. *Journal of the ACM*, 43(2):362–397, 1996. Also as *Rapport de Recherche* INRIA 1617, 1992.

[12] N. G. de Bruijn. Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[13] G. Dowek, T. Hardin, and C. Kirchner. Higher-order Unification via Explicit Substitutions. *Information and Computation*, 157(1/2):183–235, 2000.

[14] M. C. F. Ferreira, D. Kesner, and L. Puel. $\lambda$-Calculi with Explicit Substitutions and Composition which Preserve $\beta$-Strong Normalisation. In *Algebraic and Logic Programming, ALP'96*, *LNCS* 1139, 284–298. Springer, 1996.

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

[16] B. Guillaume. The $\lambda s_e$-calculus Does Not Preserve Strong Normalization. *Journal of Functional Programming*, 10(4):321–325, 2000.

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

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

[19] F. Kamareddine and A. Ríos. A $\lambda$-calculus à la de Bruijn with Explicit Substitutions. In *Proc. of PLILP'95*, *LNCS* 982 , 45–62. Springer, 1995.

[20] T. Hardin, L. Maranget, and B. Pagano. Functional runtime systems within the lambda-sigma calculus. *Functional Programming*, 8(2):131–176, 1998.

[21] 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.

[22] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and $\lambda s$-Styles of Explicit Substitutions. *Journal of Logic and Computation*, 10(3):349–380, 2000.

[23] F. Kamareddine, A. Ríos, and J.B. Wells. Calculi of Generalised $\beta$-reduction and explicit substitution: Type Free and Simply Typed Versions. *Journal of Functional and Logic Programming*, 1998(Article 5):1–44, 1998.

[24] L. Magnusson. *The implementation of ALF - a proof editor based on Martin Löf's Type Theory with explicit substitutions.* PhD thesis, Chalmers, 1995.

[25] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie de types.* PhD thesis, Université Paris 7, 1997. English version in *Rapport de recherche* INRIA RR-3309, 1997.

[26] G. Nadathur. A Fine-Grained Notation for Lambda Terms and Its Use in Intensional Operations. *The Journal of Functional and Logic Programming*, 1999(2):1–62, 1999.

[27] G. Nadathur and D. Miller. An Overview of $\lambda$Prolog. In K.A. Bowen and R.A. Kowalski, editors, *Proc. 5th Int. Logic Programming Conference*, pages 810–827. MIT Press, 1988.

[28] G. Nadathur and D. S. Wilson. A Notation for Lambda Terms A Generalization of Environments. *Theoretical Computer Science*, 198:49–98, 1998.

[29] R. Nederpelt, J. H. Geuvers and R. de Vrijer. *Selected Papers on Automath.* North-Holland, Amsterdam, 1994.

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

[31] R. Vestergaard and J. B. Wells. Cut Rules and Explicit Substitutions. *Mathematical Structures in Computer Science 11(1)*, pages 131–168, 2001.